JOURNAL OF CHEMICAL INFORMATION AND MODELING

Application Note

pubs.acs.org/jcim

HackaMol: An Object-Oriented Modern Perl Library for Molecular Hacking on Multiple Scales

Demian Riccardi,*[†,‡] Jerry M. Parks,[§] Alexander Johs,[∥] and Jeremy C. Smith[‡,⊥]

[†]Department of Chemistry, Earlham College, 801 National Road West, Richmond, Indiana 47374-4095, United States

[‡]UT/ORNL Center for Molecular Biophysics, Oak Ridge National Laboratory, 1 Bethel Valley Road, Oak Ridge, Tennessee 37831-6309, United States

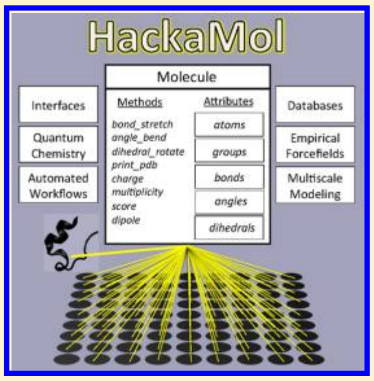
[§]Biosciences Division, Oak Ridge National Laboratory, 1 Bethel Valley Road, Oak Ridge, Tennessee 37831-6309, United States

[∥]Environmental Sciences Division, Oak Ridge National Laboratory, 1 Bethel Valley Road, Oak Ridge, Tennessee 37831-6309, United States

[⊥]Department of Biochemistry and Cellular and Molecular Biology, University of Tennessee, Knoxville, Tennessee 37996, United States

Supporting Information

ABSTRACT: HackaMol is an open source, object-oriented toolkit written in Modern Perl that organizes atoms within molecules and provides chemically intuitive attributes and methods. The library consists of two components: HackaMol, the core that contains classes for storing and manipulating molecular information, and HackaMol::X, the extensions that use the core. The core is well-tested, well-documented, and easy to install across computational platforms. The goal of the extensions is to provide a more flexible space for researchers to develop and share new methods. In this application note, we provide a description of the core classes and two extensions: HackaMol::X::Calculator, an abstract calculator that uses code references to generalize interfaces with external programs, and HackaMol::X::Vina, a structured class that provides an interface with the AutoDock Vina docking program.



■ INTRODUCTION

The daily, and often mundane, tasks of computational work in the molecular sciences generally require the mastery of a dynamic language to effectively leverage numerically intensive packages. Researchers typically write "scripts" in the languages of their mentors and co-workers to generate inputs, run calculations, and extract information from outputs for analysis. Shell programming, Perl, Python, Tcl, R, Javascript, and Ruby are widely used examples of these dynamic languages. Complete with advanced data structures and functions and not requiring a compilation phase such languages are ideal for the fast development of scripts to carry out a specific task or "glue" together methods into novel workflows.

However, generally, the writing and sharing of scripts localized within independent research groups leads to significant redundancy and wasted effort in the work. Fundamentally, all computational chemistry software maps input (coordinates, keywords, and additional information) into output, and the development of competing software packages that carry out similar functions with various inputs and outputs will continue for the foreseeable future. Therefore, libraries written in dynamic languages that abstract overlap in different software packages have the potential to simplify and accelerate scripted workflows, thus reducing human labor and error. More importantly, such libraries enhance scientific analysis, facilitate reproducibility by providing frameworks for testing methods, and prototype and create new multidisciplinary methods that mix unrelated software packages.

There are a multitude of full-featured programs that take molecular coordinates and input to generate useful information, such as that from geometry optimization or molecular dynamics simulations. Typically, there are several competing programs at each system scale. Numerically intensive tasks are CPU-bound and require implementation in fast compiled languages, such as Fortran, C, and C++. Often, programming languages are mixed to take advantage of different capabilities. For example, the Molecular Modeling Toolkit (MMTK),[1] introduced in 1997, was one of the first programs to implement object-oriented molecular objects alongside standard algorithms for molecular simulation in Python. MMTK uses C for implementing the more computationally demanding components.

All full-featured software packages attempt to include scripting capabilities, user-friendly inputs, or specialized subprograms. However, the input and output formats vary highly between programs. Using different programs, the user often invokes similar tasks, which may even use the same algorithms, in different ways. Thus, detailed computational

Received: June 18, 2014
Published: March 20, 2015

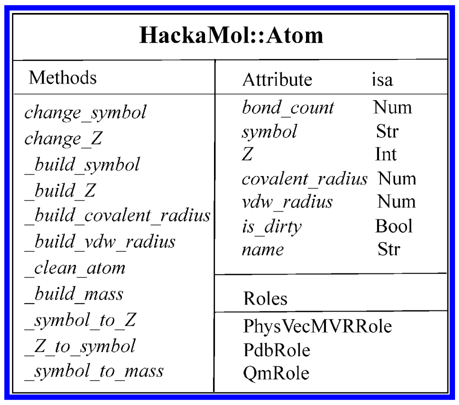
work often requires the user to become a specialist in a given program, which sometimes leads to user contributions that extend the program's capabilities, a win−win for the program. An alternative to building full-featured programs is to interface or "wrap" the functionality of various software or compiled libraries within a unified layer of a high-level language or GUI. Often, large packages wrap other programs to extend or create capabilities. One example of a large software package that wraps other programs to extend functionality is ChemShell, which provides a unified Tcl scripting interface to a large number of quantum chemistry and molecular mechanics packages to create QM/MM capabilities.[2] Another example is the BioPerl toolkit,[3] an actively developed open-source library that provides interfaces to external programs and object-oriented data structures for writing bioinformatics applications. Similar bioinformatics projects in Python (BioPython[4]) and Ruby (BioRuby[5]) are also being actively developed.

Several open-source libraries have been developed for the analysis of molecular trajectories and structures. Examples of these are VMD (written in C++ with Python and Tcl scripting capabilities and a GUI),[6] MDAnalysis (Python),[7] Lightweight Object-Oriented Structure libraries (LOOS, C++ with Python bindings),[8] Pteros (C++ with Python bindings),[9] and the Molecular Modeling Toolkit for Structural Biology (MMTSB, object-oriented Perl packaged with a large collection of scripts).[10] These provide advanced and user-friendly tools for analyzing molecular structures and trajectories in various formats generated by simulation programs. Another example is Avogadro, an open-source program (C++ with Python bindings) developed for multiscale molecular model building that has an advanced GUI and IO management capabilities for many quantum chemistry programs.[11] Avogadro depends on Open Babel,[12] a C++ program for cheminformatics that can translate between many molecular formats and search for molecular substructures and fingerprints using Daylight SMARTS pattern matching.[13] Open Babel has SWIG (Simplified Wrapper and Interface Generator) bindings to Python, Perl, Ruby, and other languages. The software mentioned above is not an exhaustive list. Building external interfaces to software is essentially an act of editing where a slice of a program's functionality can be extracted and mixed with other external interfaces to create new methods with different contexts.

Here, we introduce an open-source library that combines object-oriented atoms and molecules with functional programming to create a powerful approach to solving molecular problems. We focus on molecular classes and generalized interfaces to software rather than carrying out specific tasks or wrapping specific programs. We call this approach "Molecular Hacking", which is a term intended to evoke the harnessing of a programmable meta layer to abstract the capabilities of varied computational software. The library is written in Perl 5 because it is a flexible language that has the best support for regular expressions and a large, mature set of libraries available on the Central Perl Archive Network (CPAN), including a hypermodern library for functional object-oriented programming (Moose[14]). In the following work, we describe the core of the library, as well as the HackaMol::X::Calculator and HackaMol::X::Vina extensions. More extensive descriptions of the implementation and library use are provided as Supporting Information. All source code is available on GitHub along with additional documentation and examples. The library and extensions can be installed directly from CPAN. A case-study of disulfide bonds in the Protein Data Bank is also available at GitHub.

HackaMol Core. *Object-Oriented Atoms and Molecules.* In the HackaMol object system, atoms are represented with some metadata that usually remains fixed, such as the atomic symbol, and other attributes, such as the Cartesian coordinates, which are variable. Molecules have a collection of atoms with associated methods that may be organized into groups, bonds, angles, and dihedrals, each constructed from classes that have associated attributes and methods. In the following, we briefly describe the organization, attributes, and methods of the core classes.

HackaMol::Atom. The HackaMol::Atom class provides methods and attributes for a given atom object. This class adds two methods (*change_Z* and *change_symbol*) and several attributes, such as atomic number (Z) and symbol, to those provided by the consumed roles (PhysVecMVRRole, PdbRole, and QmRole) (Figure 1). All attributes are specified as "lazy", a



HackaMol::Atom

Methods	Attribute	isa
change_symbol	*bond_count*	Num
change_Z	*symbol*	Str
_build_symbol	*Z*	Int
_build_Z	*covalent_radius*	Num
_build_covalent_radius	*vdw_radius*	Num
_build_vdw_radius	*is_dirty*	Bool
_clean_atom	*name*	Str
_build_mass	Roles	
_symbol_to_Z	PhysVecMVRRole	
_Z_to_symbol	PdbRole	
_symbol_to_mass	QmRole	

Figure 1. Representative diagram for the Atom class. The Atom class has attributes and methods provided by the consumed roles: PhysVecMVRRole, PdbRole, and QmRole. Methods proceeded by an underscore are private to the class, by convention. See Figure 2 for more information about PhysVecMVRRole.

feature of Moose (Supporting Information) that enables attribute defaults not to be set unless accessed. Thus, lazy attributes allow classes to provide general attributes from which different subsets may be used in different contexts without adding CPU cycles for those unused. For example, atom attributes suitable for PDB protein structures or for those suitable for electronic structure calculations can coexist in the class without coexisting in the objects. Many attribute defaults are provided by builder methods that can be overridden in new subclasses. HackaMol attributes are generally read-write. An alternative approach is to constrain the API by setting attributes to be read-only and require the user to rebuild objects as needed. This approach will continue to be explored, and future releases may reduce the flexibility of the HackaMol API. The physical attributes of the Atom class (such as charge, forces, and coordinates) are provided by the PhysVecMVRRole, which is described next.

HackaMol::PhysVecMVRRole. In Moose, roles are defined as encapsulated behavior or states that can be shared between classes; roles provide an approach code reuse and interface definition that reduces the need for subclassing (see the Supporting Information for more detailed discussions). The physical vector role (PhysVecMVRRole, Figure 2) provides the core attributes and methods shared between Atom and

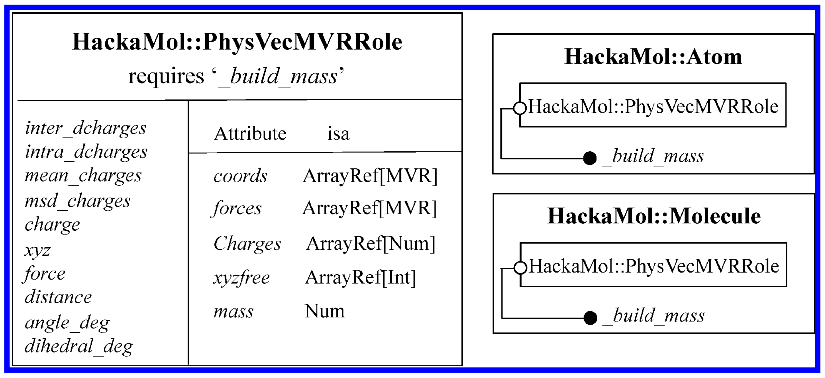


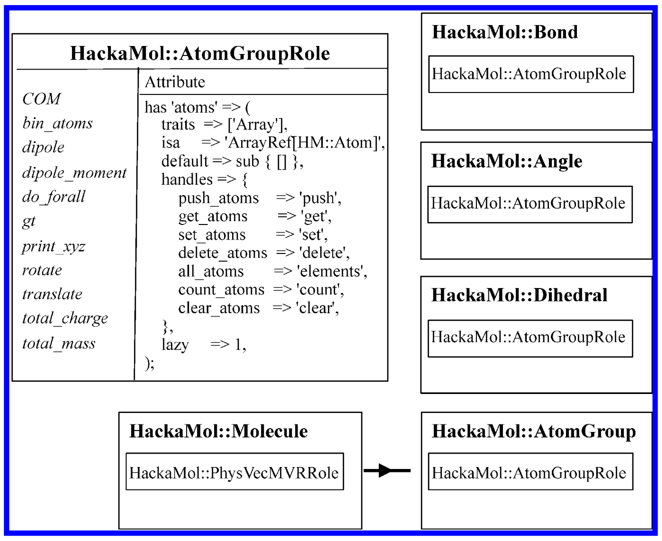
Figure 2. Representative methods and attributes of the PhysVecMVRRole role. Methods for coords and forces analogous to those shown for charges are also included in the role. In the diagram representing the Atom and Molecule classes, the open and closed circles denote an interface defined by PhysVecMVRRole with a required method (*_build_mass*) that is provided by the consuming class. The *coords*, *forces*, and *charges* attributes have delegation methods with Array traits (see Figure 3 for code defining the atoms attribute in AtomGroupRole).

Molecule classes. Consuming this role gives the Atom and Molecule classes the ability to store coordinates, forces, and charges, perhaps, over the course of a simulation or for a collection of configurations for which all other object metadata (name, mass, etc.) remains fixed. As such, the concept of "time" (t) is important in HackaMol. PhysVecMVRRole provides the attribute t, which indexes the arrays used to store charges (*charges*), XYZ coordinates (*coords*), and forces (*forces*) for analysis. These attributes are array references with array traits provided by Moose for interacting with the data structure.

The Atom and Molecule classes consume the PhysVecMVRRole (Figure 2). As mentioned above, coords gives an atom (with fixed metadata) the ability to store multiple [*x,y,z*] positions (as a function of time, symmetry, distribution, etc.). Thus, the Molecule class also has the same capabilities. Used within the molecular context, coords can be used to track the center of mass or other information as a function of the t attribute. The PhysVecMVRRole defines a simple interface by requiring a builder of the mass attribute for the consuming class; the mass is built from the relative atomic mass for the Atom and from the sum of the relative atomic masses for the Molecule.

HackaMol::AtomGroupRole. The HackaMol AtomGroupRole class provides methods and attributes for AtomGroup, Bond, Angle, and Dihedral classes, which constitute groups of atoms. The AtomGroupRole provides the *atoms* attribute (Figure 3), which is an array reference with array traits analogous (*push_atoms*, *get_atoms*, *delete_atoms*, etc.) with those discussed above for *coords* and *charges*. The AtomGroupRole also provides methods for calculating molecular properties from groups of atoms, such as the center of mass, dipole, and dipole moment. All AtomGroupRole properties are calculated by accessing attributes using the t attribute provided from the state of the atom. Utility methods are also included. The general *do_forall* method takes a function with an array of arguments and invokes the function for each atom in the group. The AtomGroupRole methods will be discussed further below for the consuming classes.

HackaMol::Bond, HackaMol::Angle, and HackaMol::Dihedral. The Bond, Angle, and Dihedral classes provide attributes and methods for internal coordinates. These classes each build on functionality introduced by the AtomGroupRole, adding methods such as *bond_length*, *ang_deg*, and *dihe_deg*, which characterize the internal coordinates, and *ang_deg* and *dihe_deg* returns the respective angles in degrees. There are also other



```
has 'atoms' => (
    traits  => ['Array'],
    isa     => 'ArrayRef[HM::Atom]',
    default => sub { [] },
    handles => {
        push_atoms   => 'push',
        get_atoms    => 'get',
        set_atoms    => 'set',
        delete_atoms => 'delete',
        all_atoms    => 'elements',
        count_atoms  => 'count',
        clear_atoms  => 'clear',
    },
    lazy    => 1,
);
```

Figure 3. AtomGroupRole and the classes (AtomGroup, Bond, Angle, Dihedral) that consume it. The AtomGroupRole adds the atoms attribute (shown with representative code), which is an ArrayRef containing HackaMol::Atom objects that uses handles to access methods provided by the Perl object system. Minimal representations showing the classes that consume the AtomGroupRole are displayed. The Molecule class is a subclass of the AtomGroup class. The Molecule class has *bonds*, *angles*, *dihedrals*, and *groups* constructed analogously to that shown for the atoms.

methods, such as *bond_vector*, which returns the vector between two bonded atoms as an MVR object, and *ang_normvec*, which returns the normalized vector from the cross product of the two interatomic vectors in the Angle object. These three classes also introduce attributes and methods for calculating energies. Each energy function is a code reference attribute that may be provided by the user with any function. Generic molecular mechanics functions are built by default.

HackaMol::AtomGroup and HackaMol::Molecule. The AtomGroup class consumes the AtomGroupRole and adds a method to calculate the radius of gyration. AtomGroup is the parent class for the Molecule class (Figure 3), which is presently the only subclass in HackaMol. Arrays of AtomGroup, Bond, Angle, and Dihedral objects are added to the Molecule class. The Molecule class is designed to be a collection of atoms whose references can be added into lists of Bond, Angle, and Dihedral objects as well as into flexible subgroups. The atom references point to the same object, so operations that change the state of a given atom in the group will also be reflected anywhere else the reference exists. Thus, cloning is required to copy an object into a new distinct object with the same state. The Molecule class provides methods and attributes for interacting with its object groupings but does not provide methods for defining and redefining those groups. Molecular groups can be cleared using *clear_groups* before assigning new atom groups. While the Molecule object has no algorithms to convert atomic coordinates into bonds, angles, etc., it does have methods that will return a list of bonds, angles, etc. associated with a subset of atoms. The building of internal coordinates and clustering of atoms into groups using sophisticated algorithms or by loading structure files created by other programs is more suitable for HackaMol extensions.

The Molecule class provides methods to stretch bonds, bend angles, and rotate dihedrals for which the coordinates of groups of atoms are transformed. For example, an extension specializing in proteins could group the atoms by side chains and backbones, populate bonds, angles, and dihedrals, and then use Molecule methods to enable sampling of alternative side chain conformations.[15]

Building Molecular Objects. The HackaMol class is a builder class that provides several methods for building molecular objects from files and other objects. For example, this class has methods to build HackaMol::Atom or HackaMol::Molecule objects from coordinate files (XYZ, PDB, and PDBQT are currently supported). The coordinates for multiple models, common for NMR structures, are loaded into the same atoms with file error checking based on the atom metadata (see the IPerl Notebooks included in the Supporting Information). This class also has methods for creating instances of the HackaMol::Bond, HackaMol::Angle, and HackaMol::Dihedral classes from lists of Atom objects.

References pointing to the same Atom object are commonly shared among the different groupings of atoms. Thus, operations that change the state of an atom in a given AtomGroup, Bond, etc. will also be reflected anywhere else the reference exists. Sharing memory references among several objects is a powerful tool for multiscale modeling, such as mixed quantum mechanics and molecular mechanics potentials. A large system can be defined as one molecule with references copied into multiple smaller subsystems (or molecules). Changing the state of one of these subsystems (perturbation of coordinates, quantum calculations of partial charges, etc.) will affect the state of the corresponding region within the larger system. The behavior of the subsystem can be observed within the context of the full system, which in turn can provide effects from the subsystem surroundings. Thus, modeling regions on multiple scales, each governed by different external programs, is possible (see HackaMol::X::Calculator).

HackaMol Extensions. Both HackaMol extensions described here are classes that consume the HackaMol::X::ExtensionRole. This role provides the *map_in* and *map_out* attributes, which require the consuming class to provide functions to build the defaults. The role consumes the core HackaMol::PathRole, which provides the ability to manipulate files and directories in the local file system. Described first, the HackaMol::X::Calculator extension provides an abstract class for interfacing external programs, and described second, the HackaMol::X::Vina extension provides a more rigid interface with a specific program, Autodock Vina,[16] described below.

HackaMol::X::Calculator. The HackaMol::X::Calculator extension generalizes molecular calculations using external programs. The Calculator class consumes a role provided by the core that manages the running of executables on input (*in_fn*) to generate output (*out_fn*) within the local filesystem (*scratch*). For example, suppose we want to compute empirical dispersion corrections to a DFT calculation with the program DFT-D3.[17] A Calculator object (*$dftd3*) can be created,

```
my $dftd3 = HackaMol::X::Calculator->new(
    molecule => $mol,
    in_fn    => $fxyz,
    out_fn   => $out,
    map_in   => sub { shift->mol->print_xyz( $fxyz ) },
    map_out  => \&extract_energy,
    scratch  => 'dispersion_calcs',
    exe      => 'dftd3',
    exe_endops => '-func b3pw91 -bj',
);
$dftd3->map_input;
$dftd3->capture_sys_command;
my $energy = $dftd3->map_output (627.51);
```

that creates a scratch directory (*dispersion_calcs*) and sets up all the input and output parameters. The *map_input* method passes *$dftd3* and any other arguments (none in the script above) to the *map_in* code reference to generate the input file. The *capture_sys_command* method runs the command (*dftd3 $xyz -func b3pw91 -bj > $out*) using a system call to the DFT-D3 program[17,18] and captures standard error and output (ignored in void context above). The *map_output* method is analogous to *map_input*, described above, where *$dftd3* along with the conversion from hartree to kcal mol^{-1} are passed as arguments to the *map_out* code reference (*\&extract_energy*),

```
sub extract_energy {
    my $calc = shift;
    my $conv = shift;
    my $re   = qr/-\d+.\d+/;
    my $_    = $calc->out_fn->slurp;
    my ($energy) =~ m /\sEdisp\.+${re}\s+(${re})/g;
    return ( $energy * $conv );
}
```

where *$energy* is pulled from the output using a regular expression that returns a list of all captured matches, using the parentheses, found within *$_*. Because each step is carried out separately, the Calculator class can be used to create scripts limited to writing inputs, reading outputs, or running command line applications with no input and no output files. For example, the Open Babel program can be used to convert a series of SMILES[19] strings on the command line into HackaMol::Atom objects with 3D coordinates.

The Calculator class is sufficiently flexible to allow the local preparation of inputs that may be transferred over to other computer clusters for conversion to output, which can in turn be transferred back and read into the HackaMol objects for analysis. This approach was taken recently in an extensive investigation of divalent metal and anion hydration free energies, for which all the data was made available using YAML.[20] Multiple instances of the Calculator class allow distinct software to be interfaced. The Calculator class was created in around 50 lines of code by consuming attributes and methods from the HackaMol::X::ExtensionRole. More encapsulated and rigid APIs between HackaMol and external computational software (such as HackaMol::X::Vina) are also straightforward to construct using the Calculator pattern.

HackaMol::X::Vina. The HackaMol::X::Vina extension provides an interface to AutoDock Vina,[16] which is a widely used program for docking small molecules (*ligands*) into biological molecules (*receptors*). This class provides methods for writing configuration files and for processing output. The input/output associated with running Vina is relatively simple, but the application of Vina to a large number of centers, ligands, and receptors must be implemented externally in scripts. This extension enables the convenient scripting of loops over multiple centers, ligands, and receptors. For example, the following script creates an instance of the class and docks three configurations of a *ligand* (contained in the file "ligand.pdbqt") into a *receptor*,

```
my $vina = HackaMol::X::Vina -> new(
        receptor => ‘receptor.pdbqt’,
        ligand   => ‘ligand.pdbqt’,
        center   => $center,
        cpu      => 4,
        exe      => ‘~/bin/vina’,
        scratch  => ‘tmp’,
);

my $mol = $vina->dock_mol(3);
```

(contained in the "receptor.pdbqt" file) centered at the variable *$center*. The *exe* attribute is the path to the Autodock Vina binary, which runs on multiple cores (four cores are set via the *CPU* attribute). This interface simplifies virtual screens and adds analytical capabilities that can be mixed in during screening via class methods or external subroutines, which in turn could be constructed to call other programs. Centers can be defined on the fly from molecular objects, but ligands and receptors are more limited because HackaMol does not yet have PDBQT writing capabilities (although it can read PDBQT files). The user is required to provide those files, which can be generated using OpenBabel[12] or MGLTools.[21] Large screens on a typical high-performance computer cluster require splitting the collection of ligands or receptors into a number of subsets that can be run independently at the same time using a batch scheduler. Estimating the time to screen each subset along with maximum number of jobs that the queue can process provides simple criteria for determining the number of subsets to generate. A Perl or shell script can be used to generate and submit the batch scheduling scripts. For large screens, the HackaMol::X::Vina interface improves control of what is written into files that can quickly accumulate. The culling of data at runtime greatly improves the efficiency of analysis. Examples are included in the GitHub repository, which use editable YAML configuration files for parameters, and then JavaScript Object Notation (JSON) to serialize streamlined databases of docking results for each subset.

Virtual screens of about 80,000 ligands have been run using multiple receptors and centers. Comparing the time consumed by Autodock Vina to the total time, using HackaMol::X::Vina with additional structural analyses (e.g., determining the receptor residues located within 4 Å of the ligand) adds a slight overhead of around 0.2%, which will go up or down slightly depending on the parameters passed to Vina (number of CPUs, exhaustiveness, etc.). The amount of data written to the hard disk was reduced by at least an order of magnitude from the several gigabytes generated using naive scripting. More massive screens using cloud-computing or supercomputers will benefit from more streamlined docking programs or interfaces to Autodock Vina, such as VinaMPI.[22] Development of a new extension for virtual screening, which will be agnostic to the docking program, is underway to encapsulate the scripted functionality, simplify use, and provide interfaces to database interfaces.

CONCLUDING REMARKS

HackaMol is a new open-source library for multiscale molecular modeling. The library is written, in modern object-oriented Perl, to be flexible and easy to extend. The core of the library is designed to provide a robust molecular object system that is able to consume external capabilities. This abstracted approach has the potential to mix numerically intensive areas of expertise that are most often developed orthogonally, such as those of quantum chemistry and bioinformatics. The application of multiscale physics models to biological problems is well-regarded,[23] and the utility of hierarchical classifications and specific attributes to the organized application of abstract theories is also clear.

Computational technologies developed by the open-source community are evolving rapidly, often outside the scope of the scientific communities. These technologies evolve and hop between programming languages, and as a result, there is a convergence of language capabilities. The HackaMol library harnesses several libraries available from the Perl community at CPAN to design rich molecular classes, serialize data, manipulate the filesystem, and perform automated testing and distribution. The current version of the library supports the reading and writing of PDB and XYZ formats. These formats are commonly used but not the most robust available. The power of Perl regular expressions together with the ability to plug in other programs, as described above, increases the number of formats that are accessible. Future developments of the core will focus on implementing more robust, modern formats such as PDBx/mmCIF[24] and MOSAIC.[25] Future extensions will seek to tap into frameworks for visualizing and analyzing data, managing large databases, and developing web applications.

HackaMol scripts can gain significant performance enhancements using the Perl Data Language (PDL)[26] and Many-core Engine (MCE) libraries that are available on CPAN. PDL gives standard Perl access to compact storage of data arrays and fast numerical methods. MCE provides methods for parallelizing loops and drop-in replacements for the grep (mce grep) and map (mce map) functions in Perl. MCE can also be used to conveniently run several instances of a serial program via interfaces in HackaMol extensions.

HackaMol is free software; all may redistribute it and/or modify it under the same terms as the Perl 5 programming language system itself. The core currently contains around 2000 lines of code and has a unit test coverage around 99%. The test suite passes on versions of Perl (from 5.8.9 to the most recent stable release of 5.20.1) across all platforms, including Windows and OS X. The HackaMol API is currently in the alpha stage of development and will continue to evolve. With this work, the core and extensions are released as version 0.017. We hope to attract users and developers to accelerate the maturation of the core API to version 1.0 and introduce new extensions and capabilities. Above all, this library is intended to reduce the labor and increase the fun of molecular hacking.

ASSOCIATED CONTENT

Supporting Information

Source code and example scripts are available online at http://github.com/demianriccardi/HackaMol. Additional descriptions of the library implementation, a full class diagram, and a PDF rendering of an IPerl Notebook, which provides detailed examples of using the library, are provided as Supporting Information. This material is available free of charge via the Internet at http://pubs.acs.org.

AUTHOR INFORMATION

Corresponding Author

*E-mail: riccade@earlham.edu.

Notes

The authors declare no competing financial interest.

ACKNOWLEDGMENTS

We are grateful to the Perl community, and more specifically the CPAN contributors that created the modules upon which HackaMol depends (Moose, Math::Vector::Real, Path::Tiny, and others). D.R. acknowledges many helpful discussions with John Eblen, Misha Wolfson, Mario Roy, and Salvador Fandino. This work was supported in part by Grant DE-SC0004895 from the U.S. Department of Energy (DOE), Office of Science, Office of Biological and Environmental Research, Subsurface Biogeochemical Research Program. ORNL is managed by UT-Battelle, LLC for the U.S. DOE under Contract DE-AC05-00OR22725. This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. DOE under Contract DE-AC02-05CH11231.

REFERENCES

(1) Hinsen, K. The molecular modeling toolkit: A new approach to molecular simulations. *J. Comput. Chem.* **2000**, *21*, 79−85.

(2) ChemShell, a Computational Chemistry Shell. STFC, Scientific Computing Department. www.chemshell.org (accessed March 2015).

(3) Stajich, J. E.; et al. The Bioperl toolkit: Perl modules for the life sciences. *Genome Res.* **2002**, *12*, 1611−8.

(4) Cock, P. J. A.; Antao, T.; Chang, J. T.; Chapman, B. A.; Cox, C. J.; Dalke, A.; Friedberg, I.; Hamelryck, T.; Kauff, F.; Wilczynski, B.; de Hoon, M. J. L. Biopython: Freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics* **2009**, *25*, 1422−3.

(5) Goto, N.; Prins, P.; Nakao, M.; Bonnal, R.; Aerts, J.; Katayama, T. BioRuby: Bioinformatics software for the Ruby programming language. *Bioinformatics* **2010**, *26*, 2617−9.

(6) Humphrey, W.; Dalke, A.; Schulten, K. VMD - Visual Molecular Dynamics. *J. Mol. Graphics* **1996**, *14*, 33−38.

(7) Michaud-Agrawal, N.; Denning, E. J.; Woolf, T. B.; Beckstein, O. MDAnalysis: A toolkit for the analysis of molecular dynamics simulations. *J. Comput. Chem.* **2011**, *32*, 2319−2327.

(8) Romo, T. D.; Grossfield, A. LOOS: An extensible platform for the structural analysis of simulations. *Conf. Proc. IEEE Eng. Med. Biol. Soc.* **2009**, *2009*, 2332−5.

(9) Yesylevskyy, S. O. Pteros: Fast and easy to use open-source C++ library for molecular analysis. *J. Comput. Chem.* **2012**, *33*, 1632−6.

(10) Feig, M.; Karanicolas, J.; Brooks, C. L., III MMTSB Tool Set: Enhanced sampling and multiscale modeling methods for applications in structural biology. *J. Mol. Graph Mod.* **2004**, *22*, 377−395.

(11) Hanwell, M. D.; Curtis, D. E.; Lonie, D. C.; Vandermeersch, T.; Zurek, E.; Hutchison, G. R. Avogadro: An advanced semantic chemical editor, visualization, and analysis platform. *J. Cheminf.* **2012**, *4*, 1−17.

(12) O'Boyle, N. M.; Banck, M.; James, C. A.; Morley, C.; Vandermeersch, T.; Hutchison, G. R. Open Babel: An open chemical toolbox. *J. Chem. Inf Model* **2011**, *3*, 33.

(13) SMARTS: A Language for Describing Molecular Patterns. Daylight Chemical Information Systems, Inc. http://www.daylight.com/dayhtml/doc/theory/theory.smarts.html (accessed March 2015).

(14) MOOSE: A Postmodern Object System for Perl. http://moose.iinteractive.com/en/ (accessed March 2015).

(15) Dunbrack, R. L.; Cohen, F. E. Bayesian statistical analysis of protein side-chain rotamer preferences. *Protein Sci.* **1997**, *6*, 1661−81.

(16) Trott, O.; Olson, A. J. AutoDock Vina: Improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading. *J. Comput. Chem.* **2010**, *31*, 455−61.

(17) Grimme, S.; Antony, J.; Ehrlich, S.; Krieg, H. A consistent and accurate ab initio parametrization of density functional dispersion correction (DFT-D) for the 94 elements H-Pu. *J. Chem. Phys.* **2010**, *132*, 154104.

(18) Grimme, S.; Ehrlich, S.; Goerigk, L. Effect of the damping function in dispersion corrected density functional theory. *J. Comput. Chem.* **2011**, *32*, 1456−1465.

(19) SMILES: A Simplified Chemical Language. Daylight Chemical Information Systems, Inc. http://www.daylight.com/dayhtml/doc/theory/theory.smiles.html (accessed March 2015).

(20) Riccardi, D.; Guo, H.-B.; Parks, J. M.; Gu, B.; Liang, L.; Smith, J. C. Cluster-continuum calculations of hydration free energies of anions and group 12 divalent cations. *J. Chem. Theory Comput* **2013**, *9*, 555−569.

(21) MGLTools. http://mgltools.scripps.edu (accessed March 2015).

(22) Ellingson, S. R.; Smith, J. C.; Baudry, J. VinaMPI: Facilitating multiple receptor high-throughput virtual docking on high-performance computers. *J. Comput. Chem.* **2013**, *34*, 2212−21.

(23) Smith, J. C.; Roux, B. Eppur si muove! The 2013 Nobel Prize in Chemistry. *Structure* **2013**, *21*, 2102−5.

(24) PDBx/mmCIF Dictionary Resources. http://mmcif.wwpdb.org (accessed March 2015).

(25) Hinsen, K. MOSAIC: A data model and file formats for molecular simulations. *J. Chem. Inf. Model.* **2014**, *54*, 131−137.

(26) Glazebrook, K.; Economou, F. PDL: The Perl Data Language. *Dr. Dobb's Journal* **1997**, Software Careers Special Issue..