An Eclipse-Based Environment for Molecular Simulation

Henrique F. Bucher
Center for Computational Research
University at Buffalo
9 Norton Hall
Buffalo, NY 14260-1800
011-716-807-2543
henrique@bucher.com

Andrew J. Schultz
Dept of Chem.& Biol. Engineering
University at Buffalo
303 Furnas Hall
Buffalo, NY 14260-4200
011-716-645-2911 x2224
ajs42@eng.buffalo.edu

David A. Kofke
Dept of Chem.& Biol. Engineering
University at Buffalo
303 Furnas Hall
Buffalo, NY 14260-4200
011-716-645-2911 x2209
kofke@buffalo.edu

ABSTRACT
Etomica is an extensible framework for conducting molecular simulations, and it comprises an API and a graphical IDE based on the *Eclipse* framework. It is written almost entirely in Java, and presently comprises about 1200 classes. In this paper we describe the general structure of the *Etomica* API, and discuss how we have integrated it into the *Eclipse* framework. Topics discussed include the design of the simulation framework and the handling of OpenGL graphics that permit real-time visualization of the simulations.

Categories and Subject Descriptors
J.2 [**Physical Sciences and Engineering**]: *chemistry*, *physics*, *engineering*.

General Terms
Algorithms, Measurement, Performance, Design, Experimentation.

Keywords
Molecular simulation, API, IDE

1. INTRODUCTION
Molecular simulation [1] describes a collection of methods for doing computer "experiments" on molecules, which are modeled by postulating the force or energy that they feel due to each other's presence. Put simply, the aim of molecular simulation is to generate many arrangements of such molecules in ways that are consistent with the laws of statistical mechanics. Observations and measurements made on these configurations of model molecules can help us understand the behavior of real molecules and the materials they form.

The range of materials and phenomena that can be studied by molecular simulation is vast and diverse. Examples include small molecular systems such as water; macromolecules such as polymers and proteins; lattice models such as those used to model magnetic systems and surfactants; electronic and semiconductor materials such as metals and silicon; complex nanostructured phases, and so on. The methods used are primarily molecular dynamics and Monte Carlo algorithms, applied with a diverse set of specialized techniques that are each targeted to certain classes of systems and phenomena.

For the past three decades molecular simulation has steadily grown as a tool of engineering researchers and, more recently, practitioners. This trend has not yet reached maturity; on the contrary, it is likely that it will continue at least to some point beyond the maturation of computing power, a time which itself is not yet in sight. Interest in molecular simulation is expanding also because it is crucial to activities conducted under the broad label of "nanotechnology", which aims to perform chemical and materials engineering by manipulating matter at the molecular scale. Advancement of this field is now a major national research priority. In a different arena, molecular simulation has also become increasingly appreciated as a powerful pedagogical tool, inasmuch as it enables students to understand abstract material behaviors and properties (such as entropy and viscosity) by opening the window to their molecular origins.

Despite the need and promise of the broad use of molecular simulation in science and engineering, there exist significant obstacles to this outcome. The barriers were identified in the recently-completed Vision 2020 Roadmap for Computational Chemistry, (www.ccrhq.org/vision/index/roadmaps/ns15.html) and include

- limited ability to obtain results of sufficient quality for practical problem resolution
- insufficient practical usability of molecular simulation codes for non-experts
- non-transferability of codes among experts
- narrowness, inextensibility, and close-source nature of commercial software
- the lack of acquaintance of non-experts with the methods of molecular simulation.

The first item, inability of simulation to consistently provide reliable quantitative information, is a very prominent focus of the molecular simulation research community, and great progress is being made toward its resolution. As an example of the interest in this issue, NIST in 2002 initiated the first of an biennial open competition for scientists and engineers to calculate results for a

set of problems (www.cstl.nist.gov/FluidSimulationChallenge). The aim is "driving improvements in the practice of molecular modeling, formalizing methods for the evaluation and validation of simulation results with experimental data, and ensuring relevance of simulation activities to industrial requirements." Real progress has been made in enabling simulation to describe increasingly complex systems, such as polymers, electrolytes, glasses, and water, and properties such as phase coexistence and viscoelasticity. Over the same period, comparatively little attention has been put toward resolving the other obstacles to widespread adoption of molecular simulation; even worse, there is nothing on the horizon indicating that they will be addressed soon.

These remaining barriers can be remedied in differing degrees through the systematic application of modern programming tools to molecular simulation software development. The obstacle here is the inexperience and/or disinterest among front-line developers/practitioners of molecular simulation methods in the use of state-of-the-art programming tools, particularly object-oriented approaches. This situation is understandable. Most research supervisors cut their teeth on Fortran, which is fast and extremely well suited for application to narrow computational problems; for these same reasons it (or C) is often the language of choice for current students, who have little incentive to perform development that has broader impact. Any graduate students who have more exposure to newer approaches lack the broad experience in molecular simulation to use them well, and are discouraged from pursuing such development because it really isn't the most effective way to solve their immediate research problems. Of course, these are broad statements for which one can find exceptions. The point is that current research practices do not encourage activities that address the "less scholarly" obstacles to broader adoption of molecular simulation. Moreover, it is not a problem that can be delegated to computer scientists, because effective and robust application of object-oriented methods requires a deep understanding of the fundamentals and techniques of molecular simulation.

This project aims to use the Eclipse application framework to develop an advanced, interactive, GUI-based computational environment for molecular and mesoscale modeling. We believe that such a framework could nucleate an open-source movement in molecular simulation, and make molecular simulation accessible to a broader community for research and education purposes. The key issues and considerations involved in this development are

- Extensibility. We cannot expect to develop by ourselves the components for every possible application of molecular simulation, but if we develop an excellent framework we hope that others will find it useful and add to it.
- Efficiency. Molecular simulations are computationally intensive. A typical simulation can run for hours or days before yielding a result. Simulated systems can range in size from tens of atoms to hundreds of thousands of atoms. And all of this might yield information about a few nanoseconds of the life of the material! (which in many cases is plenty long enough). Although it is important to get acceptable performance, this consideration does not trump everything else. No general-purpose code can outperform one that is highly specialized to the problem at hand. We expect that some users will be willing to trade off performance for ease of use and other features that can be offered by an integrated environment of reusable components.
- Versatility. The software should be suitable for interactive or batch use, and work across a variety of platforms. Presently molecular simulations are typically conducted by submitting batch jobs to a remote machine, with results obtained some time later. We want to support this use, but also enable the ability to interact with the simulation, so that the user can guide its direction or explore behaviors. Ideally it would be possible to switch between batch and interactive use as needed. We are also interested in educational applications, in which we construct self-contained interactive simulations that permit students to explore the molecular origins of interesting physical concepts such as entropy and viscosity.

2. THE ETOMICA API

2.1 Design

Etomica is in part an Application Programming Interface (API) for the construction of molecular simulations. In developing it, we have identified the following major elements of a molecular simulation

- Simulation: serves as a point of reference for all elements.
- Space: defines features of the physical space (*e.g.*, whether it is 1-, 2-, or 3-dimensional, a lattice or a continuum) in which the simulation is conducted.
- Controller: the element through which all activities of the simulation are managed.
- Species: defines the structure of the molecules; the number of atoms and how they are arranged in the molecule.
- Potential: defines how the atoms interact.
- Phase: collects all the atoms that interact with each other. It is not unusual for a simulation to involve multiple loosely coupled Phase instances.
- Integrator: codes the algorithm used to generate configurations of the molecules; an event model is used to permit other elements to react to progress made in this activity.
- DataSource: performs the calculations that yield property measurements.
- Display: presents information about the simulation to the user.
- Device: graphical element that permits the user to interact with the simulation.

Data handling uses a model in which data flows from a source to a sink, passing through transformations and accumulators along the way. Physical units are handled in a consistent way; all quantities are represented in a common unit system based on the picosecond, the Dalton, and the Angstrom. Physical quantities for input and output have associated dimensions (*e.g.*, length, mass, time) that can be used to identify choices of units (*e.g.*, meters, kilograms, seconds) used when inputting our outputting results. Many data structures and behaviors are compartmentalized. Elementary actions and activities are defined, and can be invoked via the user interface or programmatically. A generic neighbor-listing facility is developed for computational efficiency and scalability. The *Etomica* structure overall has evolved in line with the generally accepted object-oriented design patterns, with significant roles for iterators, builders, factories, strategies, and so on [2].

Presently *Etomica* comprises more than 1200 classes.

2.2 Performance

Given the computationally intensive nature of molecular simulation, and the premium placed on good computational performance, we always have concern about how fast the codes run. Java may be a liability in this regard, but we find the performance to be acceptable for many purposes. Comparisons with dedicated Fortran codes running equivalent algorithms find that *Etomica* about 2 to 4 times slower. Depending on the problem and the urgency that the results are needed, this may be just fine or it may be prohibitive. We do find that performance scales well with the size of the simulated system, and is no worse than Fortran in this regard. Our Tinderbox benchmarks run simulations of up to 40,000 atoms, and they take only about 9 times longer than an equivalent simulation having 1/8 as many atoms (5000).

2.3 Applications

Etomica is the primary platform for research in molecular thermodynamics performed in our research group. It has been used in the study of liquid-vapor surface properties, the behavior of associating fluids, effects of point defects and strain in solids, miscibility of InGaN semiconductors, and the development and understanding of free-energy methods. It has also seen many uses in educational settings. In 1999 we used the codes as part of a two-week workshop introducing high-school students to molecular simulation. We did this again in 2000, and in 2001 we used it as the basis for an honors course for freshmen. We use it now in some undergraduate and graduate courses in Chemical and Biological Engineering at the University at Buffalo, and have plans to expand these applications further. Standalone modules made with *Etomica* have (to our knowledge) been used in courses in perhaps 10 universities.

3. THE ETOMICA IDE

3.1 General Features

Presently we are developing a molecular simulation Integrated Development Environment (IDE) that is based on the *Etomica* classes. We are building this on the *Eclipse* open-source application development platform. The aim is to produce an Office-like environment for building, running, monitoring, viewing and analyzing molecular simulations. We envision being able to cut-and-paste entire simulations or their components (*e.g.* an individual phase), so that it becomes easy to save its state and make copies to explore variations. Java's serialization capabilities provides the key functionality for these purposes. The same technology can underlie remote-monitoring capabilities, in which the IDE can attach to a remote process running a batch simulation on a high-performance server, to monitor and perhaps tune its behavior on-the-fly.

Features that we aim to incorporate in the IDE are as follows.

- views and/or editors generally designed to present and configure each of the high-level components of a simulation. These include Simulation, Phase, Species, Controller, Integrator, Activity, Accumulator, Meter, and Atom classes, among others. We have already developed Eclipse views for the Simulation and Phase classes, as well as a general-purpose reflection-based drillable property sheet that permits editing of the properties of any object.
- interactive data analysis capabilities for plotting, tabulating, transforming, and exporting the results of a simulation.
- the use of serialization to save and restore simulations, and even to copy and paste entire simulations so variations on them can be easily explored. As a sub-element of this, we envision the ability to cut, copy, paste, and delete complete simulations elements, so that (for example) a molecular dynamics simulation could be easily transformed, on the fly, into a Monte Carlo simulation (this involves swapping out one Integrator for another, following the Strategy design principle [2]). We would like to see the GUI capabilities familiar to everyone using word-processing applications brought to the realm of high-performance computing.
- wizards that guide the user through the process of assembling a new simulation and analyzing or exporting the results. This ensures, for example, that a hard-potential dynamics integrator is not applied with a soft intermolecular potential model. This functionality must adhere to the reflection method of discovery, so that compatibility rules are brought in with the classes, and are not hard-coded into the IDE.
- a complete help facility with documentation suitable for both developers and users. We use Javadoc-formatted comments extensively through all our codes, so to better facilitate the creation of a thorough documentation of the API.
- the ability to import and export simulations as source code, so that developers can easily move between graphical and text-based modes, seamlessly integrating with the existing Java-development and debugging capabilities of Eclipse.
- the ability to monitor and interact with simulations that are running elsewhere as batch process. The idea is that a user can launch a simulation on a high-performance machine, and occasionally check up on it and perhaps modify its operation by temporarily attaching the IDE.

A key principle in the Eclipse+Etomica integration effort is ensuring the extensibility of the IDE. All Etomica API classes are to be incorporated into the IDE via discovery in the classpath, and Java's reflection capabilities are used to discover the editable features of any simulation component. In this manner the IDE capabilities grow naturally as the API expands. It should even be possible to add functionality to the IDE while simulations are in progress.

3.2 Visualization and OpenGL

When creating a plugin for creating chemistry simulations within *Eclipse*, we need to include the ability to display the internals of the ongoing simulation as a 3D display. Many applications already do this as the graphics display plays an important role on capturing critical aspects of the simulation. Visualizing initial and final configuration states of such molecular systems is of great help but having a means to understand how the system evolved to reach such final configuration gives much more insight on the underlying physics, especially for molecular dynamics problems.

Therefore we needed a 3D display that would present the instantaneous state of the ongoing simulation and which could be embedded in a SWT control.

As first choice for this work, we have used OpenGL as the 3D graphics API of choice since it is fully supported on most platforms. There is at least one full SWT/OpenGL wrapper available

http://www.eclipse-plugins.info/eclipse/plugin_details.jsp?id=562

It uses Java's JNI (Java Native Interface) to wrap every OpenGL call and enumeration therefore enabling any Java program access to the 3D world.

Because every OpenGL call is wrapped in a JNI routine, and typically thousands of such calls are performed at every frame update, we observed a noticeable latency added by the additional software layer. Additionally, building complex 3D scenes writing directly code in OpenGL may remind some of the old times of writing simple programs using plain assembler – the final code is often large and very difficult to maintain.

Still, there were desired features not available on this pure OpenGL approach, as:

- Mouse and keyboard manipulators to allow an interactive user experience in 3D
- Support to importing from common 3D file formats as 3D Studio Max or OpenFlight
- Support to image file formats as JPEG, GIF, BMP
- Support for special effects (particles, cartooning)

The conclusion was that the wrapping was performed in a very low level and what we needed was a scene builder with easy-to-grasp functions as createSphere() or createBox().

This conclusion brought us to OpenSceneGraph, which is an open source 3D scene builder (www.openscenegraph.org). It is available for most unices and also Windows and MacOS, which fulfills our cross-platform requirement. This package was before successfully integrated with StreetScenes, a Qt-based traffic simulation package developed at the Center for Computational Research/State University of New York at Buffalo.

http://www.ccr.buffalo.edu/viz/content/streetscenes.htm.

However, OpenSceneGraph is itself a very large API, with hundreds of C++ classes. And as JNI requires pure C, we had to create a new software layer using C++ to bind both together, which we call the OSGWrapper. The architecture is summarized in Figure 1.

The OSGWrapper implementation is the central piece of this binding. It translates calls from the OSGWrapper/java classes, which are very high level as createObject() or object.setPosition(), into the more complex C++ statements that set all necessary OpenGL parameters according to a predefined set of defaults. The ease with which OpenGL images can be rendered in the *Eclipse* framework is illustrated by the code shown in Figure 2, which renders an image (not shown here) in an *Eclipse* control.

The effect of offloading java from all latency-critical calls and performing them at a low level language had the effect of increasing the performance from roughly 10 frames per second to around 200 frames per second for a small size simulation (~1000 atoms).

All graphics information was stored in the implementation layer using a flyweight pattern [2], which allowed for efficient and scalable use of memory, directly affecting performance.

3.3 Graphical Interface

So as to follow Eclipse's user interface guidelines [3], we chose to represent *Etomica's* main data container with an Eclipse editor (guideline 6.1), which was associated with the .etom file suffix. This EtomicaEditor class is primarily responsible for:

1. setting up the physical initial configuration with atoms (molecules) and phases

2. associating families of atoms with their respective potentials

3. planning activities to be performed on configurations as time-stepping integrations and Monte Carlo simulations.

4. adding data collectors and displays for compiling and recording the results of activities

5. starting a group of activities

6. visualizing the progress of those activities with 2D or 3D graphical interfaces

One of the major challenges we faced was regarding guidelines 6.3 and 6.4, for historical reasons. *Etomica* was first designed to run batch jobs. This means that for every Java process there would never be more than a single simulation instance running. Several *Etomica* classes took advantage of this premise to boost performance but also to achieve cleaner code by storing defaults and highly accessed objects as statically allocated instances.

Under Eclipse, simulations were run as separated threads to allow for easy retrieval of information for display. These simulations running simultaneously sometimes crashed or (less often) only produced wrong results. This required some work to remove all statically allocated instances, cleaning up default constructors, which eventually led to a less coupled system.

No special views were created for *Etomica*. The design reused already existent views as the property and outline views (guideline 6.20) to help in displaying and configuring the simulation. A special perspective for Etomica was also created to make easier the initial set up of the environment.

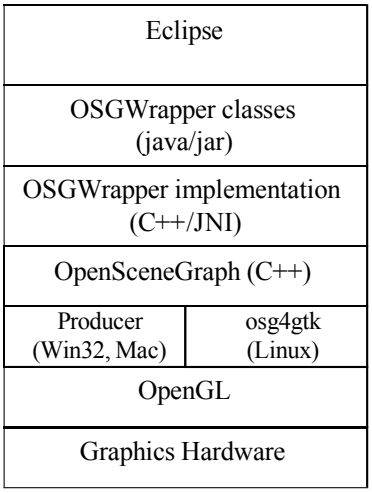

Figure 1 Software layers implementing the SWT/OpenGL binding.

3.4 Discovery

Two specialized wizards were created - a new project wizard and a new simulation wizard. The new project wizard is quite simple and only sets up a new folder for containing further new simulation files. The new *Etomica* document wizard in turn, required writing a discovery class. One of the design requirements was that all modifications and additions to *Etomica*—to both the

```
import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.FillLayout;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Group;
import org.eclipse.swt.widgets.Shell;

import osg.OrientedObject;
import osg.R enderWindow;

public class Simple {
        public Simple() {
                super();
        }
        public static void main(String[] args) {
                Display display = new Display();
                Shell shell = new Shell(display);
                FillLayout shell_layout = new FillLayout();
                shell.setLayout(shell_layo   ut);
                shell.setSize(800, 800);
                Group group = new Group(shell, SWT.NONE);

                shell.open();

                int handle = group.handle;
                RenderWindow r1 = new RenderWindow(handle);

                OrientedObject.appendToSearchPath("C:    \\OpenSceneGraph -Data");
                OrientedObject o1 =   OrientedObje ct.createFromFile( "molecule   .osg" );
                r1.addObject(o1);

                float[]  xaxis = new float[]{1, 0, 0};
                while (!shell.isDisposed()) {
                        display.readAndDispatch();
                        r1.render();
                        float delta = 0.2f;
                        o1.rotate(delta, xaxis);
                }
                r1.dispose( );
                display.dispose();
        }
}
```

Figure 2. Simple program in Java/SWT to display and rotate a 3D model.

main jar file and to the project's subfolder—should be automatically apprehended by the system. Therefore the user had the ability to choose among stock types or create its own sets of simulations, specialized potentials, data sources, devices, meters and so on.

An approach using XML files and extension points as implemented for Eclipse's plugins was considered but discarded afterwards for being hard to maintain for non-developers, and we chose instead to implement a brute force scheme.

Eclipse already provides a notification framework based on the IResourceChangeNotification interface and that was used to catch modifications made after the plugin's initialization. However, for the initial discovery process upon start we created a specialized resource crawler to discover and query all classes derived from our set of nine basic abstract classes. This task cannot be accomplished by using Java class loaders directly because they do not provide a method to list all classes they represent - they may be located remotely, for example.

The solution was to crawl over two well defined sets of data:

1. All entries in the plugin's bundle whose URL resolve to a file;

2. All entries in the java.class.path system property.

Jar files were further traversed for single .class files. When found, restrictions were placed so to allow only classes belonging to the "etomica" package—or a subpackage. Classes (instances of java.lang.Class) were then queried for assignment to one of those base abstract classes cited before. If a match is found, the class object is added to the list of available objects. This list is used mainly to populate combo boxes to offer the user choices when configuring a simulation. The NewEtomicaDocumentWizard, for example, queries this registry and presents the user with a selection of ready-to-use simulation templates or, if a custom simulation is desired, with a selection of an available combination of Space and master Potential. This framework allowed for an added flexibility on a ongoing development effort as *Etomica*.

4. SUMMARY

Etomica is an API and an Eclipse-based development environment for building, conducting, and analyzing molecular simulations. It is still a work in progress. With it we aim to enable a broader range of scientists, engineers, and educators to make use of the powerful capabilities of molecular simulation.

5. ACKNOWLEDGMENTS

Development of *Etomica* is supported by the National Science Foundation, and by an Eclipse Innovation Grant from IBM. HB is supported in part by the University at Buffalo Center for Computational Research. Thanks to Ruben Lopez for releasing part of osgedit [4] code, essential for the integration OSG/Gtk.

6. REFERENCES

[1] Frenkel, D. and B. Smit, *Understanding Molecular Simulation: From Algorithms to Applications*. 2nd ed. 2002, San Diego: Academic Press.

[2] Erich Gamma *et al. Design Patterns: Elements of Reusable Object-Oriented Software* (Addison-Wesley, 1995).

[3] Eclipse User Interface Guidelines, Nick Edgar, Kevin Haaland, Jin Li, and Kimberley Peter; Last updated: February 2004. http://www.eclipse.org/articles/Article-UI-Guidelines/Contents.html

[4] OSGEdit - an open editor for an open scene graph. http://osgedit.sourceforge.net.

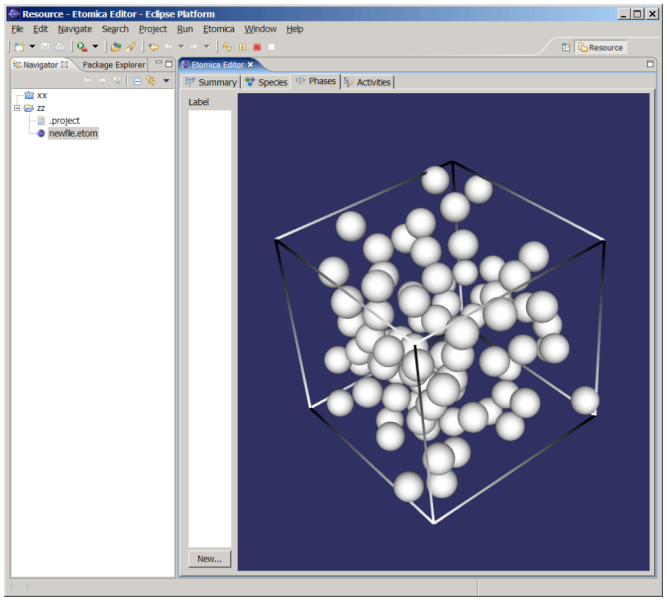

Figure 3. Screen shot of the Etomica-Eclipse IDE, showing a configuration of atoms