ELSEVIER

Fluid Phase Equilibria 194–197 (2002) 327–335

FLUID PHASE EQUILIBRIA

www.elsevier.com/locate/fluid

Web-based technologies for teaching and using molecular simulation

David A. Kofke*, Bryan C. Mihalick

Department of Chemical Engineering, State University of New York, Buffalo, NY 14260-4200, USA

Received 20 May 2001; accepted 2 August 2001

Abstract

Computing power and molecular simulation methodology have improved dramatically over the past decade, and for some systems molecular models are reaching a point where they can provide information that approaches or surpasses the quality of real experimental data. Molecular simulation can also be a potent tool for developing qualitative understanding, and this feature suggests widespread use of molecular simulation as a tool for teaching. Broader use of molecular simulation for these purposes requires tools that promote understanding and application of simulation software. We present an "object-oriented" view of molecular simulation that can help conceptualize and organize its elements, and discuss software development tools that we have constructed that enable non-experts in simulation or graphical programming to create instructive molecular simulations. We finish with an example of one such simulation applet, which we have used in the classroom. Tools described here are available on the web at www.ccr.buffalo.edu/etomica.

Keywords: Molecular simulation; Application programming interface; Java; Education; Software

1. Introduction

Molecular simulation is growing in use among researchers and practitioners of chemical engineering thermodynamics, and substantial research efforts are ongoing that aim to fuel this trend [1–3]. Great improvements in molecular simulation methodology have arrived over recent years, particularly for application to phenomena (such as phase equilibria) that are of great importance to engineering thermodynamics [4–6]. Additional progress is now being made in development of molecular models suitable for description of material behaviors at the extreme conditions often encountered in chemical processing. Advances in molecular simulation software can further aid in the widespread acceptance of molecular simulation as a tool for engineering practice. Codes developed by academic researchers represent the

* Corresponding author. Tel.: +1-716-645-2911; fax: +1-716-645-3822.
E-mail address: kofke@eng.buffalo.edu (D.A. Kofke).

0378-3812/02/$ – see front matter
PII: S0378-3812(01)00685-9

state-of-the-art, but are often difficult to use and are mostly unsupported. On the other hand, commercial packages fill some needs, but they are expensive and difficult to extend to new applications. On the drawing board are technologies that can harness a heterogeneous, geographically-distributed set of computational resources—both hardware and software—and bring the right set of tools to bear on a given modeling problem. Central to this development is the notion of a "computational grid", which promises to make computing power a utility much like electricity or gas, and which can be harnessed with the simplicity that we now access information via the world-wide web. Such developments in the molecular modeling domain require concerted efforts from engineers and computer scientists.

As molecular simulation becomes more useful to the practicing engineer, it is inevitable that it will be widely incorporated in undergraduate and graduate curricula. The only question now is whether this development will occur sooner or later. In the meantime, molecular simulation can be applied in the classroom as an aid to understanding thermodynamics and transport phenomena. Molecular-level events are responsible for a lot of the material behaviors exploited by chemical engineers. These behaviors cannot be understood from the standpoint of classical mechanics, yet this discipline forms the basis for physical intuition of young engineering undergraduates. Molecular simulation can be used to connect complex physical phenomena, such as phase transformations, to molecular behaviors treated classically (without quantum mechanics). Especially helpful to this end is the ability to visualize the simulation, and interact with it in real time, so that the student can manipulate the physical system and observe the response emerge from the collective interactions of the model molecules.

Development of interactive simulations requires knowledge of both molecular simulation and graphical programming. Few educators possess both sets of skills, or the time to acquire and apply them to develop simulation-based instructional tools. Improvements in software can help to remedy this situation. This manuscript outlines a development environment suitable for construction of interactive molecular simulations. The environment and the simulations produced from it are based on the Java programming language, and thus they can be used to develop applets that run over the world-wide web, or applications that can run on any common computer architecture. In this sense, they are web-based, and suitable for instruction, research, or engineering practice.

We use the name Etomica to refer to the collection of software tools we have developed for construction of a molecular simulation. Etomica consists of two separate parts. First is the application programming interface (API), which is a core set of Java classes (independent program units) that define the basic elements of a molecular simulation. Second is a graphically oriented development environment that permits assembly of a molecular simulation using the API, but without requiring any low-level Java programming. In this manner, a person lacking expertise in graphical programming, Java, and/or molecular simulation can assemble a new, nontrivial molecular simulation that conveys some concept of physical interest. In the following, we outline the basic features of these elements of the Etomica package, and finish with a brief description of a molecular simulation applet that can be used for instruction in thermodynamics.

2. Molecular simulation API

Java is an object-oriented programming (OOP) language, and accordingly the fundamental programming constructs are objects or classes. A class presents to the user an interface, which describes the types of data it can hold and actions it can perform on its data. The emphasis in OOP is on the interface, so

that one may use a particular class without concern for the details of how the interface is implemented, or programmed specifically. Moreover, different classes can present the same interface but perform different activities, and different overall behaviors can be obtained by selecting one or another instance of a class that conforms to a given interface. For example, a class for the intermolecular potential may provide as part of its interface a means to calculate the energy between two atoms. An instance of a Lennard–Jones potential when requested to do this will report a different result from that given by a square-well potential, all else being equal. More complex variations of this polymorphism arise in practice, but the basic idea is that a simulation can be constructed from standardized pieces, and different types of simulations can be formed by selecting different pieces for a given role.

An API is a collection of building-block classes that can be assembled to perform some complex activity. We describe here the basic structure of the Etomica molecular simulation API. For the simulation novice, examination of an API such as this one provides a way to conceptualize molecular simulation. The API classes taken together describe the fundamental features of a molecular simulation. Presently Etomica comprises more than 600 classes, some large and some very small, and a detailed description of all of them is beyond the scope of this article. A reader interested in more complete documentation of the available classes may consult material available on the Etomica web site (www.ccr.buffalo.edu/etomica).

2.1. Simulation

Each simulation is formed from a single Simulation class, which provides a central point of reference for all the other elements of the simulation. Among other things, the Simulation holds an instance of Mediator class, which has the job to tie together the various elements as they are created and added to the Simulation. The Mediator can be tailored to different purposes, but for the most part its activities are transparent to the user.

2.2. Space

The Space class defines all the features of the physical space in which the simulation is performed. Appropriate selection of the Space permits the simulation to be performed in one, two, or three dimensions, and will in future implementations enable simulations to be performed on a lattice. The Space class generates, among other things, appropriate Vector classes for the space it defines, and can produce different Boundary objects that define what molecules do when they reach the edge of the simulation volume.

2.3. Species

The Species class defines and creates the molecules used in the simulation. A molecule is defined by stating what type and how many atoms it contains. Basically, the molecule is just a data structure, and holds information such as where the atoms are and how fast they are moving. The way that the atoms interact with other atoms in the system is defined elsewhere (by the Potential), so there is, for example, no Lennard–Jones species. Instead, an instance of a Lennard–Jones Potential is assigned to an instance of a Species that defines its molecules as monatomic with spherical atoms. To simulate a mixture, one simply creates an instance of a Species for every component in the system.

2.4. Potential

The Potential class defines how the atoms interact with each other. There are two fundamental types of potential. Soft potentials (such as Lennard–Jones) can report an energy, force, virial, etc. for any configuration of the atoms, while hard potentials have impulsive forces, and can report the energy but not the force or virial. Instead, hard potentials can report the time to collision of any pair of atoms, and can provide a means to implement collision dynamics.

2.5. Phase

A Phase collects all the atoms that are interacting with each other, and provides Iterators that permit looping over these atoms or subsets of them. The Phase also holds an instance of a Boundary object (created by the Space class). Most simulations consist of a single phase, but there are many circumstances—such as Gibbs ensemble or parallel tempering simulations—in which more than one phase is appropriate.

2.6. Integrator

The Integrator is responsible for generating configurations of the molecules in each phase. Typically, an Integrator will do this by applying a molecular dynamics or Monte Carlo (MC) simulation algorithm. Different classes are defined that implement the Integrator interface but perform different simulation algorithms. So different types of simulations are conducted simply by selecting the corresponding integrator. The Monte Carlo integrator is actually a shell implementing the Monte Carlo algorithm. The MC integrator can be assigned to hold other objects of type MCMove, which define different types of Monte Carlo trials that it should perform. For example, an MC simulation in the isothermal—isobaric ensemble would be performed by an instance of the MC integrator with MCMoves added to it that perform atom-displacement and volume-change trials.

At regular intervals the Integrator will notify any interested object (or "listener") that is has made some degree of progress in the simulation. This interval can be adjusted, but typically would occur after each molecular-dynamics time step or MC sweep. Interested listeners might for example refresh the graphical display, or update averages that they are recording.

2.7. Controller

The Controller class runs the show, so to speak. It is responsible for setting the simulation in motion (by turning on the Integrator), and determining when it ends. In the simplest case, it does this in response to a button click by the user. It might instead be set to perform a series of simulations, changing some state variable along the way, so to permit evaluation of a thermodynamic integral.

2.8. Meter

The Meter class is responsible for collecting data from the simulation as it proceeds. Different versions of the Meter class measure the energy, pressure, radial distribution function, temperature, and so on. Meters also are responsible for performing simple analyses of their data, for example keeping track of the confidence limits. Each Meter can be instructed to record other types of information, such as the history

of values it measures, or a histogram of the data. Each Meter updates its data in response to notification of an interval event by the Integrator.

2.9. Display

The Display class is responsible for presenting information to the user. This often is done in graphical form, either via a table or a plot, but it might also involve writing results to file. At this point, we should note that all the foregoing simulation classes work with data in a fixed set of units, constructed from the base units: picosecond, Angstrom, Dalton. The Display class holds constructs that permit conversion of the data to more convenient units before presentation to the user. Each Display writes or refreshes its data display in response to notification of an interval event by the integrator.

2.10. Device

The Device class permits the user to interact with the simulation. It allows adjustment of state variables, placement of atoms, turning on and off the simulation, and other features by the user. Unit conversion can also be performed by these classes, if appropriate (so that temperatures can be set in Kelvin, for example).

3. Etomica development environment

Assuming all the required classes were developed, someone familiar with Java programming and the API itself could use it to write a new simulation in a text editor. However, the learning curve needed to accomplish this is steep, and we do not expect that many would wish to attack it. One benefit of the object-oriented approach is the ease with which it can be incorporated into a graphical development environment. Because all the simulation elements are designed to be modular, the job of the graphical environment is to enable selection of the pieces that one wishes to use to construct a new simulation, and to assemble them in a way that works. The Etomica development environment performs this task.

Fig. 1 presents a screen-shot of the development environment. The large window on the left is the simulation editor, which presents choices for different simulation elements. The elements are organized into different tab panes according to the API categories described above. When an element is added to the simulation, it shows up in the list on the right half of this window, and selection of it brings up the property sheet window, displayed in the figure on the right. Here, various properties of the element (such as the number of molecules of a species) can be edited by the user. Dimensioned quantities also provide a selection of units to use in inputting the values.

The Etomica environment provides other useful capabilities.

- A constructed simulation may be saved as a serialized object that can be read back into the environment in exactly the same state as when it was written (so if the simulation were run for some period, it would be in the same condition as when it was written). In principle, the simulation can be serialized in a form suitable for placement on an HTML page as an applet, but presently there is a known bug in Java that prevents this from working.
- A compiled simulation class, written perhaps in a text editor, can be read into the environment and modified or run as if it had been constructed there. This feature permits code developed outside the

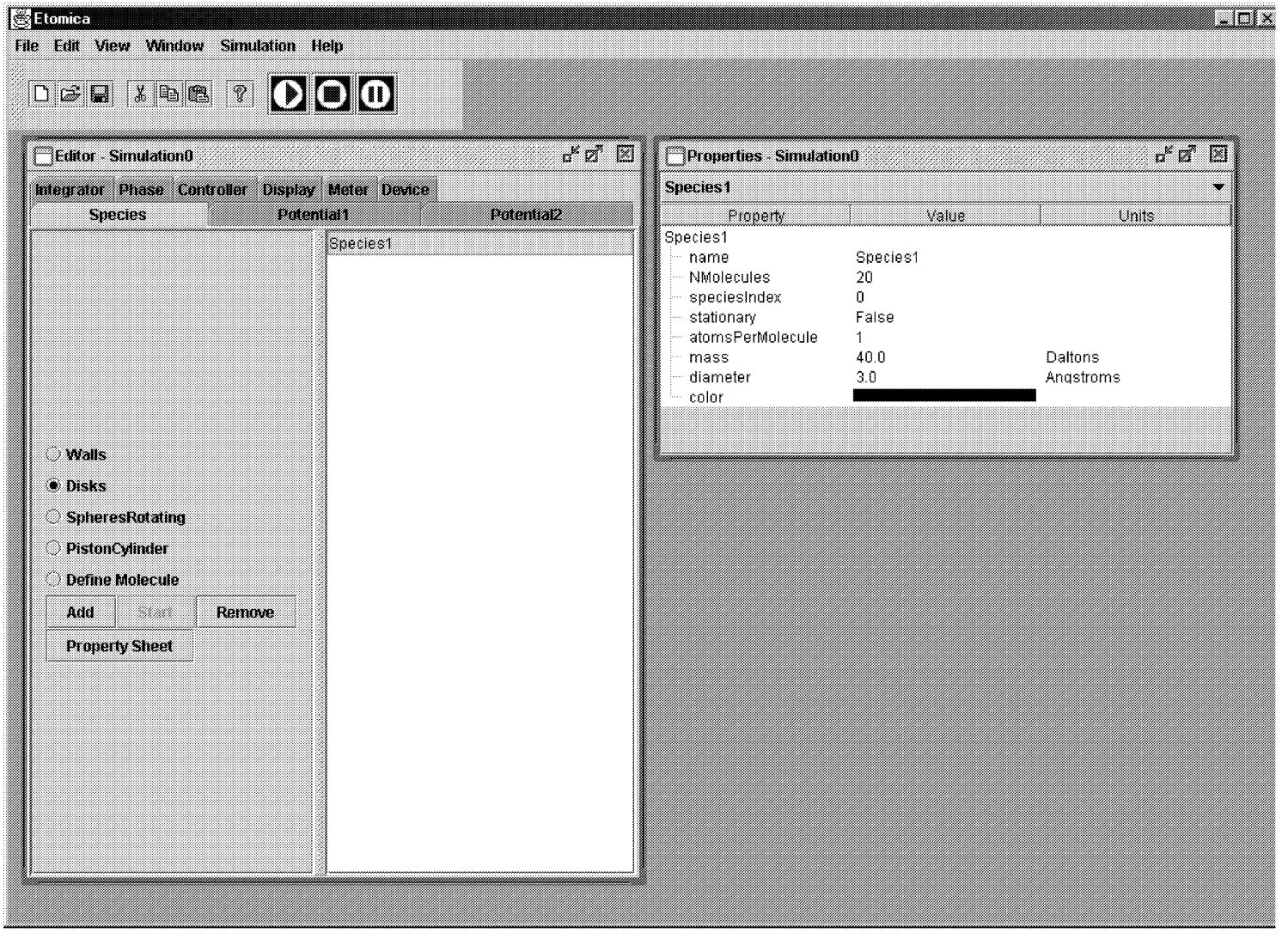

Fig. 1. Screen-shot of the Etomica development environment.

environment to be imported to take advantage of the interactive features it provides. Several standard simulations are available from a "library" menu.

- A file can be written containing the Java code needed to construct the simulation in its initial state (presently this feature is only partially developed). Coupled with the ability to read in a compiled simulation class, this feature in essence provides a scripting functionality. A simulation can be assembled and written as Java source, and then further edited more efficiently than possible in the graphical environment, and re-read back into the development environment.
- The environment is extensible, and uses Java's reflection capabilities to discover the editable features of any simulation element. This means that a new element (say, a meter that measures a new property) can be developed and compiled, and when placed into the Etomica classpath, it will be recognized by the Etomica environment. No modification of Etomica itself is needed to add it or have its properties available for editing.

Etomica is under very active development, and we expect new features to be added, and old ones debugged, on a continuing basis.

4. An instructional applet

We finish by describing a molecular simulation applet that was developed using the Etomica API. This applet models a piston–cylinder apparatus, and a screen-shot of it is presented in Fig. 2. The fluid in the apparatus is a simple system of hard spheres, and they bounce off each other according to the usual rules governing molecular dynamics. The spheres are contained by a set of hard walls, and the top wall is capable of moving up and down in response to forces acting on it. It has a constant downward force exerted due to an imposed external pressure, which can be adjusted with the slider seen in the figure. Opposing this are the impulsive collisions of the spheres, which give it an upward shot of momentum when they hit it. Because of the small system size, the cylinder fluctuates significantly about some value, but its average is measured by the simulation and presented in a data table along the bottom. In addition to adjusting the pressure, the student can set the apparatus to be adiabatic or isothermal, selecting from one of three temperatures. The fixed temperature is maintained via simple scaling of the velocities. Sometimes the piston moves off the scale of the display, and there are buttons that enable the image to be scaled up and down to fit everything on the screen, if desired.

Presently Etomica can visualize only two-dimensional simulations, so this piston–cylinder simulation is conducted this way. Nevertheless, the results and inputs are presented in terms of three-dimensional volume and pressure. It would present an unnecessary distraction to explain to the student the distinction between pressure in two and three dimensions, so we ascribe to the system a "depth" of 5 Å, which is used to "convert" to three-dimensional units. One can view this more as a modeling choice than an assumption.

We have used this applet as part of a sophomore thermodynamics course. We discussed some of the mechanics its operation in class, and assigned some homework problems that required the students to use it and record and analyze data from it. The problems had them measure and plot the equation of state, compute the second virial coefficient and the coefficient of thermal expansion, perform heat and work calculations and compute the heat capacity. They also manipulated the system to examine the nature of reversible and irreversible expansions, and saw the molecular-scale picture of how systems heat and cool upon compression and expansion.

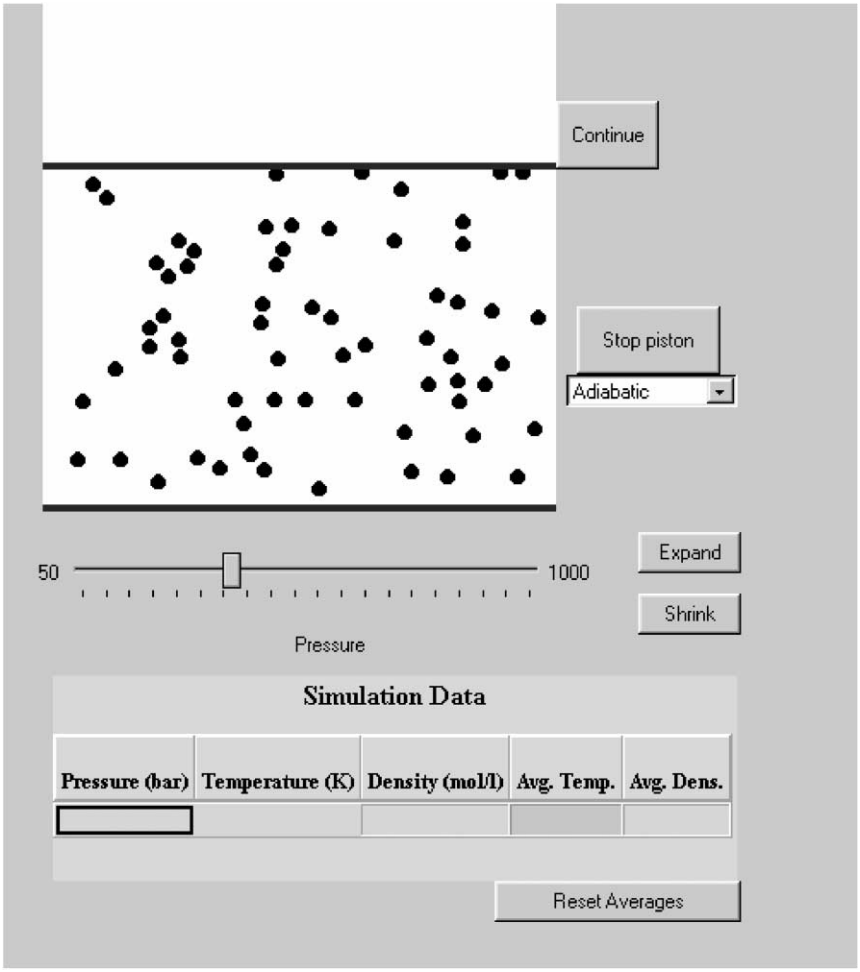

Fig. 2. Screen-shot of the piston–cylinder applet.

This particular applet is not highly refined, and it could benefit from some systematic study of how students use it and how well it succeeds in conveying different concepts. Supplemental instructional material is presently under development for this applet. Additional simulation-based instructional modules are under development, and examine other thermophysical phenomena, including heat conduction, diffusion, collision dynamics, reaction kinetics, and Joule–Thomson expansion.

5. Conclusion

Our aim in this development effort is to enable educators to construct molecular simulation tools for instruction. We have made substantial progress in this direction, and the Etomica development environment is nearly mature enough to accomplish this task. Some debugging and refinement of the interface

is needed, and this activity has the highest priority right now. Further development is needed to create new "building-block" classes that give a wider range of choices to the developer. Our experience to date is that each new simulation-development job requires introduction of a few new elements to provide some new functionality. Developing new classes requires a pretty deep understanding of the API, and we wish to avoid requiring such knowledge of the developer. On the other hand, with some guidance it is possible for a novice to write new components. In the summers of 1999 and 2000, we held workshops at the UB Center for computational research, in which about 10 students from local high schools participated in activities to acquaint them with molecular modeling and computation. Working in pairs, their capstone activity was the development of a new molecular simulation applet. All succeeded in developing their applets, and some of the working results can be viewed from links at the Etomica web site (www.ccr.buffalo.edu/etomica). This site also includes links to download the development environment and the source code for the Etomica API classes, as well as documentation for both.

Acknowledgements

This work has been supported by the National Science Foundation through a grant to the CAChE Corporation. Additional support has been provided by the University at Buffalo. This work has benefited from conversations with other participants in the NSF-supported module-development project, notably Dan Lacks, Peter Cummings, Ed Maginn, Phil Westmoreland, Randy Snurr, Richard Rowley and David Ford. We are very grateful to Jim Ely and the Chemical Engineering Department of the Colorado School of Mines for facilitating some of these interactions.

References

[1] M.P. Allen, D.J. Tildesley, Computer Simulation of Liquids, Clarendon Press, Oxford, 1987.
[2] D. Frenkel, B. Smit, Understanding Molecular Simulation: From Algorithms to Applications, Academic Press, New York, 1996.
[3] M.W. Deem, AIChE J. 44 (1998) 2569–2596.
[4] A.Z. Panagiotopoulos, Mol. Phys. 61 (1987) 813–826.
[5] A.Z. Panagiotopoulos, Mol. Sim. 9 (1992) 1–23.
[6] D.A. Kofke, in: D.M. Ferguson, J.I. Siepmann, D.G. Truhlar (Eds.), Monte Carlo Methods in Chemistry, Vol. 105, Wiley, New York, 1998.