Etomica: An API for Molecular Simulation

David A. Kofke

Department of Chemical and Biological Engineering
University at Buffalo, the State University of New York
Object-Oriented Programming

• Programming accomplished through the actions and interactions of objects
  – everything is an object

• Forces abstract thinking about the structure and activities of a program

• Promotes re-use of code and extension to new applications

• Good design is difficult to develop
  – requires thorough understanding of application
  – conversely, its use facilitates a better understanding of application
    • presents a good vehicle for teaching

• It’s fun!
What is an Object?

• A fancy variable
  – stores data
  – can perform operations using the data

• Every object has a type, or “class”
  – analogous to real, integer, etc.
  – you define types (classes) as needed to solve your problems
  – types differ in the data they hold and the actions they can perform on it
  – every object is an “instance of a class”

• A class has an interface
  – what the object presents to enable its manipulation
  – implementation (how it accomplishes its operations) can be hidden
  – object is viewed in terms of its “actions” and not its “thoughts”

• Inheritance
  – different classes can inherit the same interface, but implement it differently to produce different behaviors
Makeup of an Object

• **Fields**
  – primitive types (integer, float, double, boolean, etc.)
  – handles to other objects
    • complex objects are composed from simpler objects (composition)
  – Fields are usually not part of the interface
    • “private”

• **Methods**
  – “subroutines and functions”
  – may take arguments and return values
  – have complete access to all fields of object
  – methods are defined to set and get field values
Detailed Look: Molecule and Atom

• Atom methods
  – Vector getPosition()
    • Returns an object that represents the atom’s coordinate
  – AtomType getType()
    • Returns an object that specifies important parametric features of the atoms, such as its size, shape, mass, and how it is drawn
  – int getIndex()
    • Returns an integer used to store the Atom instance in an array

• Molecule methods
  – AtomList getChildList()
  – Species getType()
  – int getIndex()

• Click here for the complete API specification
Design Considerations

• Goals
  – Extensible, broadly applicable
  – Computational efficiency
  – Suitable to run interactively or in batch

• Guidelines
  – Highly granular pieces with convenience classes that assemble them
  – Separate components as much as possible
    • Graphics separate from other parts
    • Used objects don’t know about user
  – Try to re-use themes that guide design of data and other constructs
    • Agent model
    • Event model
Agent Model

2. new Agent

1. Request new Agent instance

Construct at initialization

Client object; implements AgentSource

AgentManager

1. Request Agent for object

n

2. Locate Agent

n

3. Return Agent handle to client

n

3. Tag to Object

Agents

Objects

1

1

2

2

n

n

n

n
Event Model

Register as listener when simulation is initialized

Event Source

Event Listener

Notify all listeners when event occurs
Simulation

- Simulation
  - Organizes other elements
  - Common point of reference
  - Independent entity—no simulation knows about or interacts with another Simulation instance
  - No graphical elements
  - Develop new simulations by extending Simulation
    - Assemble simulation in constructor
    - Most fields publicly accessible
    - Reusable in different contexts
- SimulationContainer gives simulation an interface
  - Graphical elements
  - Remote access as a future consideration
- Space is assigned to Simulation at construction
Space

• Factory for objects that depend on or define the physical space
  – Vector, Tensor, Orientation, Boundary

• All object methods are implemented in a spatially-independent manner
  – Vector methods defined for vector addition, scalar multiplication, dot product, simple compound operations, etc.

• Easy to convert from simulation in one dimension to another
Vector

• Defines Cartesian vector and operations performed on it

• Some methods
  – double squared()
  – double dot(Vector v)
  – void E(Vector v)
  – void PE(Vector v)
  – void Ea1Tv1(double a, Vector v)
  – Vector Mv1Squared(Vector y)
  – void normalize()
  – Etc.

• Different implementations done for different dimensions
Data Structures: Atom

• Atom
  – Represents physical atom being simulated

• Some Important fields
  – position
    • class that holds and manipulates position vectors
  – type
    • class that specifies important parametric features of the atoms, such as its size, shape, mass, and how it is drawn
  – index
    • an integer used to store the Atom instance in an array
AtomFactory

- Builds a molecule according to a specification
- “Atom” is defined generally
  - “Leaf” atom corresponds to a physical atom
  - Group of atoms, even molecules, are represented by instances of Atom
  - Molecule is represented by a tree structure, using AtomTreeNode

AtomFactoryMono, AtomFactoryHomo, AtomFactoryHetero

- Hierarchical: Large molecules built from factories that comprise other factories that build the molecule subunits

Each factory attaches a unique AtomType to all the Atoms it builds

Factory has a Conformation that arranges atoms
Data Structures: Box

- **Box**
  - Collects all atoms that interact with each other

- **A single Simulation may employ multiple Box instances**
  - Parallel tempering, Gibbs ensemble
  - No atoms in one Box interact with atoms in another Box

- **Box holds a Boundary instance**
  - Constructed by Space
  - Implements (or not) periodic boundary conditions

- Manages addition/removal of molecules

- Additional information associated with Box via BoxAgentManager
Data Structures: Species

• Species classes collect information needed to construct and manage molecules
• Subclasses defined for specific molecules
• Serves as a “molecule type” for doing potential calculations
Data Structures: AtomsetIterator

• AtomSet
  – Interface for a set of atoms
    • Atom, AtomPair most often used

• Many types of atom-set iterators
  – Iterate atoms or atom pairs at a particular level in hierarchy
  – Iterate pairs formed with a particular atom
  – Iterate in one or both directions from a given atom
  – Many interfaces defined
    • AtomsetIteratorPhaseDependent
    • AtomsetIteratorBasisDependent
    • AtomsetIteratorDirectable
    • AtomsetIteratorTargetable
    • AtomsetIteratorListDependent
    • etc.
Models: Potential

- **Potential**
  - Defines manner of interaction of atoms
  - `public void energy(AtomSet atoms)`

- **Subclasses specific to 1-body, 2-body, etc. forms**

- **Interfaces for hard and soft potentials**
  - **PotentialSoft**
    - `energy, virial, hypervirial, gradient`
  - **PotentialHard**
    - `energy, collisionTime, bump`

- **PotentialMaster class** collect potentials and manages iterators
Models: PotentialGroup

• PotentialGroup
  – Collects several potentials that all interact on a single AtomSet

• 1-body PotentialGroup
  – acts on a single Atom (which typically is a group of atoms)
  – collects intramolecular interactions

• 2-body PotentialGroup
  – acts between two Atom instance
  – collects intermolecular interactions
Flow Control: Action and Activity

• Action
  – interface for abstract, elementary action that does something
  – public void actionPerformed()
  – can be grouped for series implementation
  – for example
    • AtomActionRandomizeVelocity
    • AtomActionTranslateBy
    • IntegratorReset
    • PhaseInflate

• Activity
  – more complex, time-consuming extension of Action
  – can be started, stopped, paused, resumed
  – can be grouped for series or parallel implementation
  – for example
    • ActivityIntegrate
    • EquilibrationProduction
Flow Control: Controller

- Two ways to conduct simulation
  - interactively
  - batch
  - (or hybrid of both)

- Specification of actions must be mutable
  - even while simulation proceeds

- Controller
  - schedules actions to be performed
  - single instance constructed for each Simulation
  - actions/activities can be added to queue
  - urgentAction can be requested for immediate implementation
    - all GUI-driven changes follow this path
  - carefully synchronized
Flow Control: Integrator

- **Integrator**
  - repeatedly changes configuration to follow a sampling algorithm
  - public void doStep()
  - deploys subclass-specific agent to each atom
  - only one integrator acts on a given box
  - some integrators act on multiple boxes
    - IntegratorGEMC (Gibbs ensemble Monte Carlo)
    - IntegratorPT (Parallel tempering)

- **IntegratorMD**
  - IntegratorVelocityVerlet
  - IntegratorHard
    - discontinuous molecular dynamics

- **IntegratorMC**
Flow Control: IntegratorMC

- **IntegratorMC**
  - Monte Carlo sampling
  - Selects trial move, performs trial, decides acceptance, notifies move and other listeners

- **MCMove**
  - Performs Monte Carlo trial
  - Reports information needed to determine acceptance
    - \( \ln(p_{\text{new}}/p_{\text{old}}), \ln(t_{ij}/t_{ji}) \)
    - Holds fields needed for evaluation
  - Does appropriate update for acceptance or rejection
  - For example
    - MCMoveAtom
    - MCMoveInsertDelete
    - MCMoveRotateMolecule
    - MCMoveVolume
  - Sampled ensemble is determined by set of MCMoves added to integrator
Flow Control: IntegratorEvent

- **IntegratorEvent**
  - integrator fires event to registered listeners to notify of progress with simulation

- **IntegratorListener**
  - **IntegratorIntervalListener**
    - receives repeated events reporting progress
  - **IntegratorNonintervalListener**
    - receives only events indicating initialization, start, end, etc.
  - For example
    - objects pushing data measurement and processing
    - cell- and neighborlist-updating
Data Processing: DataSource, DataSink

• DataSource
  – interface for class that can provide data
  – data is generally represented by array of double
  – public double[] getData();
  – Meter is a DataSource that acts on a Box
  – for example
    • MeterDensity, MeterEnergy, MeterRDF, MeterTemperature
    • DataSourceCountCollisions, DataSourceCountTime

• DataSink
  – interface for class that can receive data
  – public void putData(double[] data);
  – for example
    • DisplayBox, DataSinkConsole, DataBin
    • DataPipe
Data Processing: Pipelines

- Data is pushed from a source to a sink
  - It may pass through other elements along the way
  - Each pushes data on to the next element

- DataPipe
  - Abstract, implements DataSink
  - Takes data given to it, does something to it, and pushes new data
  - DataAccumulator
    - Collects statistics on data it receives, and pushes it on at intervals
    - e.g. AccumulatorAverage, AccumulatorHistory, AccumulatorHistogram
  - DataTransformer
    - Modifies data and immediately pushes it downstream
Data Processing: DataPump

- **DataPump**
  - Extends DataProcessor
  - Holds a DataSource, and moves data from it to the sinks
  - Provides the impetus for moving the data from a source into a pipe
  - Implements Action
    - Typically activated via Integrator IntervalEvent, or GUI action
Data Flows in Etomica

- **DataSource** sends data to **DataFork**
- **DataFork** sends data downstream to **DataProcessor** and **DataProcessor**
- **DataProcessor** processes data: tally average, std dev, histogram
- **DataSink** writes data to file, displays as table, and displays as graph
- **Integrator Listener** and **GUI button** interact with the system
- **DataPump** prompts actions: Get data from source, send downstream
I/O and Graphics: Display

- **Display**
  - Object to present data in graphical interface

- **Boxes, plots, tables, etc.**

- **All are treated as implementing DataSink**

- **Logging capabilities still not well developed**

- **Units**
  - Internally, all data are represented in a common unit system
    - picosecond, Angstrom, Dalton
  - Unit classes are defined to handle conversions
  - All I/O and graphics classes hold a Unit instance
  - Classes can declare Dimension for fields so that appropriate units are offered
I/O and Graphics: Device

- **Device**
  - Widget that allows user to interact with simulation

- **Examples**
  - **DeviceButton**
    - Connects to an action, performs action when button is pressed
  - **DeviceSlider**
    - Changes value of some quantity with movement of a slider
  - **DeviceThermoController**
    - ComboBox that permits selection from several temperatures
  - **DeviceCheckBox**
    - Toggles a boolean value using a checkbox
  - **DeviceControllerButton**
    - Start/stop/pause/resume simulation

- **Acts via Controller**
  - Invokes urgentAction
  - Controller handles Action request ASAP
    - Pauses current Activity, or finishes current Action
    - then attends to requested Action
  - Prevents collision between user and integrator threads
Utilities

• Utility classes developed as needed
  – versatile lattice capabilities
  – Polytope for defining shapes
  – very small set of math classes
    • linear algebra
    • special functions
    • permutations/combinations
Supporting Tools

• CVS
• JUnit
  – facility for developing unit tests
• javadoc
  – facility to generate hyperlinked documentation from comments
• bugzilla
  – bug tracking
• tinderbox
  – performance tracking
### Supporting Tools: Tinderbox

<table>
<thead>
<tr>
<th>Build Time</th>
<th>Guilty</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>05/24 19:35</strong></td>
<td></td>
</tr>
<tr>
<td><strong>05/24 17:53</strong></td>
<td></td>
</tr>
<tr>
<td><strong>05/24 16:11</strong></td>
<td></td>
</tr>
</tbody>
</table>

- **tinderbox: etomica**
- **SHMD3D times**
  - 190.39, 219.22, 270.05
  - 193.73, 237.41, 273.69
  - 193.36, 219.59, 276.06

- **LJMC3D times**
  - 0.22, 293.39
  - 90.22, 289.39
  - 90.22, 289.39

- **SWChain times**
  - 406.63, 460.97
  - 408.53
  - 408.67, 466.06
  - 408.79, 464.91

- **SWChain wall times**
  - 6210K
  - 6210K
  - 6209K

- **SWChain mem**
  - 41557K
  - 41559K
  - 41560K

- **HSMD3D wall times**
  - 193 240 277
  - 193 240 277

- **HSMD3D mem**
  - 4181K
  - 4181K
  - 4185K

- **LJMC3D wall times**
  - 95 288
  - 95 288

- **LJMC3D mem**
  - 1505K
  - 1505K
  - 1530K

- **rusty Linux**
Supporting Tools: Tinderbox

HSMD3D_32000
(rusty)

Y-axis (zoom[0% 100%], Days[all days], Style[lines|steps], Points[on|off], Average[on|off])

rusty HSMD3D 32000

Generated: 24-May/05 20:36

• Other rusty tests: (startup, rollbackopen, pageload, show all tests)
• Show the raw data for this plot