The Architecture of GenDevs: looking under the hood of DEVSJAVA 3.0

Bernard P. Zeigler
ACIMS
Updated January 2004
Scalability, Flexibility and Inter-operability Through Interface Standardization

- Single processor
- Distributed Simulator
- Real-Time Simulator
- Non DEVS
- C++
- Java
- Other Representation
- DEVS Simulation Protocol
Scalable Distributed/Networked Enterprise

**scalability:** as system expands or performance demands increase, can reimplement same functionality (interfaces) with more instances or more capable classes (threaded, distributed, event channel)

- **Scalability at software level, e.g., XML**
- **Scalability at middleware level, e.g., CORBA real-time event channel**
- **Scalability at hardware level, e.g., TINI (Java executing Tiny TCP/IP Interface)**

**DEVS**
- Modeling Interfaces
- Simulator Interfaces
- Ensemble Collection Interfaces

- **Model Definition/Manipulation structures**
- **Simulation/exeuction structures**
- **Distributed Object computing structures**
GenDevs Packages

GenDevs

GenCol

java.util, java.lang

DEVS Models Simulators and Interfaces

- Bag, Relation, Function
- ensemble Collection classes and interfaces

- Collection classes and interfaces
  - Threads
  - Sockets
  - reflect
# Collections, Maps, Relations

<table>
<thead>
<tr>
<th>Collection</th>
<th>Defining Property</th>
<th>Useful For</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collection</td>
<td>- Indefinite size</td>
<td>- Variable sized collections</td>
</tr>
<tr>
<td>List</td>
<td>- Indexed elements</td>
<td>- sequencing</td>
</tr>
<tr>
<td></td>
<td>- insert/remove anywhere</td>
<td>- basis for queues/stacks</td>
</tr>
<tr>
<td>Set</td>
<td>- admission based on equality (no duplicates)</td>
<td>- unique tracking of object occurrences</td>
</tr>
<tr>
<td>Bag</td>
<td>• no admission criteria multiplicities are counted</td>
<td>- word frequency counts</td>
</tr>
<tr>
<td>Map</td>
<td>- one-one correspondence (keys to values)</td>
<td>- dictionary (one meaning per word)</td>
</tr>
<tr>
<td>Relation</td>
<td>- many-many correspondence</td>
<td>- dictionary (multiple meanings per word)</td>
</tr>
</tbody>
</table>
HashSet s = new HashSet();
s.add("a");
Iterator it = s.iterator();
while (it.hasNext()) {
    Object o = it.next();
}

public Object anyOne(Set s){
    Iterator i = new Iterator(s);
    if (i.hasNext())
        return i.next();
    else return null;
}
Java Collection Interface Hierarchy

**Collection**
- +add(Object)
- +remove(Object)
- +contains(Object)
- +size()
- +iterator()

**Set**
- +add(Object)
- +remove(Object)

Set contains no duplicate elements, i.e., contain no pair of elements e1, e2 s.t. e1.equals(e2) and at most one null element.

**List**
- +add(int, Object)
- +get(int): Object
- +remove(int)
- +remove(Object)
- +listIterator(int)

Allows insertion anywhere in the list

+add(int, Object) inserts object only if it is not equal to any in the set
Java Collection Class Hierarchy

- AbstractCollection
  - Collection
    - Set
      - AbstractSet
        - HashSet
      - List
        - AbstractList
          - Vector
          - LinkedList

Uses Hashtable as implementation so uses hash and equals for key equality ((see next Java Map slide))
Java Map Class Hierarchy

- Map
  - AbstractMap
    - HashMap
    - TreeMap
  - SortedMap
    - Dictionary
    - Hashtable

Methods:
- +put(key: Object, value: Object)
- +get(key: Object)

Key to value mapping (one-to-one)

Hashxx uses hash code and key equals method to determine associate value

```
if ((e.hash == hash) && key.equals(e.key)) return e.value;
```
Ensemble Methods

Ensemble or bulk methods act on all the elements in a collection uniformly.
Ensemble Methods (cont’d)

- **tell-all command args** - send the `command(args)` message to all objects in the container,
- **ask-all query? args** - send the `query?(args)` message to all objects in the container and collect the results in a returned container (see ).
- **which? query? args** - send the `query?(args)` message to all objects in the container and collect objects returning TRUE in a new container.
- **which-one? query? args** - return the one entity in `which? query? args` provided there is exactly one; otherwise return an unspecified entity in `which? query? args`.
- **reduce query? token args** - pass the token from object to object in the container in an unspecified order. Each successive object replaces the token with the results of `query?(token,args)`. After all replacements are done, the token is the final result returned.
interface ensembleBasic {
    public void tellAll(String MethodNm, Class[] classes, Object[] args);
    public void askAll(ensembleCollection result, String MethodNm, Class[] classes, Object[] args);
    public void which(ensembleCollection result, String MethodNm, Class[] classes, Object[] args);
    public Object whichOne(String MethodNm, Class[] classes, Object[] args);
}

interface ensembleCollection extends ensembleBasic, Collection {
    public void print();
    public void wrapAll(ensembleCollection Result, Class cl);
    public ensembleCollection copy(ensembleCollection ce);
}
ensembleBag b = new ensembleBag();
b.add(e);
b.add(e);
b.add(f);
HashSet c = new HashSet();
Class [] classes = {java.util.HashSet};
Object [] args = {"e"};
b.which(c,"equalName",classes,args);
return c.size() == 1;
Providing ensemble capability to any Collection

- ensembleWrap
  - $make(Collection)
    :ensemble

  - tellAll
  - AskAll
  - which
  - whichOne

- Collection:c

- $ensemble
  - ensemble()
  + tellAll
  + AskAll
  + which
  + whichOne

Thread

Coord     Timer     Holder
Sequence Diagram: TellAll

tellAll(command,args)

:ensemble

:coord

:timer

:collection

:Object

:Object

start

start

Iterator()

new holder(command,args)|start()

command(args)

decrement()

decrement()

interrupt()

while(coord.alive())

waitForNt()
public interface EntityInterface{
    public String getName();
    public Object equalName(String name);
    public ExternalRepresentation getExtRep();
}

public interface ExternalRepresentation{
    class ByteArray implements ExternalRepresentation{}
}

//overrides pointer equality of Object
public boolean equals(Object o){
    if (!(o instanceof entity))return false;
    else return eq(((entity)o).getName());
}

public String toString(){
    return getName();
}
Port, Content and Message

- ContentInterface
- MessageInterface
- PortInterface
- EntityInterface
- ensembleBag
- ensembleCollection

Diagram:
- ContentInterface to content (0-n)
- PortInterface to port (1)
- value (1)
- message (0-n)
- MessageInterface to message (dashed)
- ensembleBag (dashed)
- Collection (dashed)
public interface MessageInterface extends Collection{
  public boolean onPort(PortInterface port, ContentInterface c);
  public Object getValOnPort(PortInterface port,ContentInterface c);
  public void print();
  /* examples of using ensembleBag approach */
  //public ensembleBag getPortNames();
  //public ensembleBag valuesOnPort(String portName);
}

ensembleBag b = x.getPortNames();
if (b.size()>= 2) //both stop and start arrive
  holdIn("active",10);
else if (b.contains("stop"){
  if (phaseIs("active")
    passivate();
  }
else if (b.contains("start"){
  if (phaseIs("passive")
    holdIn("active",100);
  })
}
public void deltext(double e,message x){
  Continue(e);
  for (int i=0; i< x.getLength();i++)
    if (messageOnPort(x,"in",i))
      entity ent = x.getValOnPort("in",i);
      passivate();
}
interface IODevs {
    public void addInport(String portName);
    public void addOutport(String portName);
    public ContentInterface makeContent(PortInterface port, EntityInterface value);
    public boolean messageOnPort(MessageInterface x, PortInterface port, ContentInterface c);
}

interface basicDevs {
    public void deltext(double e, MessageInterface x);
    public void deltcon(double e, MessageInterface x);
    public void deltint();
    public MessageInterface Out();
    public double ta();
    public void initialize();
    public void showState();
}

interface coupledDevs {
    public void add(basicDevs d);
    public void addCoupling(basicDevs src, String p1, basicDevs dest, String p2);
    public basicDevs withName(String nm);
}

interface atomicDevs {
    public void Continue(double e);
    public void passivate();
    public void passivatesn(String phase);
    public void holdn(String phase); // double time argument
    public void holdn(String phase, double time, Activity a);
    public boolean phasesn(String phase);
}

interface IOBasicDevs {
}

interface Coupled {
}

interface AtomicInterface {
}

interface DevsInterface {
}
DEVS-Canonical Implementation

Message Handler

coupledDevs

IODevs

BasicDevs

IOBasicDevs

atomicDevs

atomic

digraph

devs

EntityInterface

entity

Coupled

Atomic
Simulator Interfaces

Non-DEVS components satisfying this interface can inter-operate with DEVS

CoreSimulator Interface

AtomicSimulator Interface

CoupledSimulator Interface

Coordinator Interface

public interface coreSimulatorInterface {
    public void initialize();
    public Double nextTNDouble();
    public void computeInputOutput(Double d);
    public void DeltFunc(Double d);
    public MessageInterface getOutput();
    public void simulate(int numIter);
}

public interface AtomicSimulatorInterface {
    public void wrapDeltfunc(double t, MessageInterface x);
    public void showModelState();
    public void computeInputOutput(double t);
    public void showOutput();
}

public interface CoupledSimulatorInterface {
    public void putMessages(ContentInterface c);
    public void sendMessages();
    public void setModToSim(Function mts);
    public void addPair(Pair cs, Pair cd); //coupling pair
    public void showCoupling();
    public void startActivity(ActivityInterface a);
    public void returnResultFromActivity(EntityInterface result);
}

public interface CoordinatorInterface {
    public void addSimulator(IOBasicDevs comp);
    public void setSimulators();
    public void informCoupling();
}
RTSimulator Interfaces

interface RTSimulatorInterface {
    public long timeInSecs();
    public long timeInMillis();
    public void setTN();
    public double getTN();
    public void stopSimulate();
}

RT interfaces add in Runnable SimulatorInterface and interpret time as real wall clock time
Simulator Classes (Non-RT)

IODevs Simulator

coreCoordinator

coupledCoordinator

1:n

1:n

Atomic Simulator

Atomic Simulator Interface

1:n

coreSimulator Interface

Coordinator Interface

Note: should be opposite
Real Time Simulator Classes

- Atomic Simulator
- RTAtomic Simulator
- coupledRT Simulator
- RTcoordinator
- coordinator
- AtomicSimulator Interface
- RTSimulator Interface
- 1:n coupling relationships
Distributed Simulator Classes

coordServer

1:n

RTCoordinator Interface

simulatorProxy

coupledSimulator

1:1

CoupledSimulator Interface

clientSimulator
Migrating activities between logical time and real time

public interface ActivityInterface extends Runnable{
    public void setSimulator(CoupledSimulatorInterface sim);
    public double getTimeToDeadline();
    public String getName();
    public void kill();
    public void start();
    public EntityInterface computeResult();
}
Hierarchical Construction

Coordinator Interface

CouplingProtocol Interface

HierParent

Activity Protocol Interface

coupledSimulator Interface

coupledCoordinator Interface

coupledCoordinator

coupledSimulator

coupledCoordinator

coordinator

parent

myCoupled

myModel
public coupledCoordinator(Coupled c) {
    super(c);
}

public void setParent(CoupledCoordinatorInterface p) {
    myParent = p;
}

public coordinator(coupledDevs c) {
    simulators = new ensembleSet();
    public setSimulators() {
        while (cit.hasNext()) {
            IOBasicDevs iod = cit.nextComponent();
            if (iod instanceof atomic) {
                addSimulator(iod);
            } else if (iod instanceof digraph) {
                addCoordinator((Coupled) iod);
            }
        }
    }
    public void addCoordinator(Coupled comp) {
        coupledCoordinator s = new coupledCoordinator(comp);
        simulators.add(s);
        modelToSim.put(comp.getName(), s);
    }
}
• computeInputOutput
  • tell all computeInputOutput
    • output: use external output coupling
    • input: use internal coupling
  • tell all send messages
    • send output to others using downloaded cplng
      (myself, outport)(other,inport)
• deltFunc
  • apply external input coupling to incoming and add to input
    (myself, inport)(component,inport)
  • apply wrapDeltFunc to augmented input
Connecting to the Real World via DEVS on a Chip

SUPPLIER → PUSH() → RT-EVENT CHANNEL → PULL() → CONSUMER

SUPPLIER → PUSH() → RT-EVENT CHANNEL → PULL() → CONSUMER

DEVS

RT-DEVS Execution Engine

Activities

CORBA Real-Time Event Channel

Ethernet Network

DEVS on a chip

DEVS

RT-DEVS Execution Engine

Activities

TINI Java TCP/IP Interface

Sensors
Middleware

- Sometimes used to denote custom-programmed “glue” that allows a collection of existing applications to federate into a subsuming integrated application

- As defined by [1], *middleware is reusable, expandable set of services and functions that benefit many applications in a networked environment*

- Middleware represents an expansion of the infrastructure to
  - subsume functions needed by many applications
  - improve certain characteristics of the applications
  - enhance interoperability among applications
  - reduce the complexity encountered by application developers and end users
  - improve the usability to end users.

- Middleware typically includes a set of components (such as resources and services) that can be utilized by applications either individually or in various subsets.

NSF CISE Advisory Committee
Subcommittee on the Middleware Infrastructure
Middleware (cont’d)

Middleware lies above the transport layer (e.g., TCP), but below the application environment
• may be embedded within operating systems, or may be separate,
• the boundary may change with time.
DEVS as Middleware

- *DEVS middleware is reusable, expandable set of services and functions that benefit distributed simulation in a networked environment*
- DEVS Middleware represents an expansion of the infrastructure to
  - subsume functions needed to easily construct distributed simulations
  - improves simulations due to the beneficial formal properties of DEVS
  - enhance interoperability among components adhering to DEVS protocol
  - reduce the programming complexity by hiding lower level middleware details and providing right level of abstraction for modeling and simulation
  - improve the usability to end users -- supports distributed programming by modeling
- DEVS Middleware includes components for model construction and mapping into simulators or real-time executors.
DEVS Middleware (cont’d)

- Applications (domain model collections)
- Model construction services
- Simulation services
- Connection middleware services (e.g. CORBA)
- Network, processing, and storage infrastructure