The Parsimony Project:
A Distributed Simulation Testbed in Java

Bruno R. Preiss
Ka Wing Carey Wan
Electrical and Computer Engineering
University of Waterloo

Outline of the Talk

- requirements for distributed discrete-event simulation
- how Java supports distributed discrete-event simulation
- modeling and simulation in Parsimony
- Parsimony simulators
- an example
- conclusions
Requirements for Distributed Discrete-Event Simulation

- modeling support
- dynamic loading
- support for multiple execution threads
- transparent and extensible networking support
How Java Supports Distributed Discrete-Event Simulation

- models as classes, events as runnable objects
- logical processes as threads
- the Java Virtual Machine as simulation engine
- object serialization
- remote method invocation (RMI)
Coupling Event Objects and Model Instances

class Model
{
    State state = new State();
    class Event implements java.lang.Runnable
    {
        public void run()
        {
            modify(state);
            schedule(new Event());
        }
    }
}
Modeling and Simulation in Parsimony

- physical processes $\rightarrow$ logical processes

- simulation vs. simulator

- entity models and the system model

- events as run-once runnable objects

- message $+$ handler $=$ event
The Entity Model and System Model Classes

- entity models are derived from AbstractModel class
- events are derived from AbstractEvent class
- system model is derived from AbstractSimulation class
- message handlers implement the MessageHandler interface
AbstractSimulation extends AbstractModel

system model

extends

AbstractEvent extends AbstractModel

extends

MessageHandler implements

instantiates

AbstractSimulation

MessageHandler

extends

MessageHandlers

extends

event handlers

entity models

Classes defined in the Parsimony Package

User-defined Classes
Achieving the Separation of Concerns

- separate user-defined, application-specific simulation code from domain of the simulator

- allow completely transparent support for multiple simulators
calls method
createProcess

simulator
instance

is bound to

system model
instance

creates

calls methods
send(Message m)
schedule(Event e)

system process
instance

is bound to

entity model
instance

logical process

← provided by Parsimony × user-defined →
Simulators

- SequentialSimulator
- MultiListSimulator
- ThreadedMLSimulator
- ThreadedCMBSimulator
- ThreadedTWSimulator
- RealTimeSimulator
Distributed Simulators

- DistributedMLSimulator
- DistributedCMBSimulator
- DistributedTWSimulator
- DistributedRTSimulator
An Example—A Single-Server Queueing Network

Source \rightarrow \text{QueueAndServer} \rightarrow \text{Sink}

customers

chan0 \rightarrow \text{QueueAndServer} \rightarrow \text{chan1}
class Source extends AbstractModel
{
    RandomVariable interDepartureTime;

    public Source (long mean)
    {
        super(0, 1);
        interDepartureTime = new ExponentialRV(mean);
    }

    public void initialize (long time)
    {
        schedule(new Departure(time));
    }

    class Departure extends AbstractEvent
    {
        Departure(long time) { super(time); }

        public void run ()
        {
            send(new VoidMessage(getTime()));
            schedule(new Departure(Math.round(getTime()) +
                                interDepartureTime.nextDouble()));
        }
    }
}
Sink Model

class Sink extends AbstractModel
{
    Sink()
    {
        super(1, 0);
        setMessageHandler(new ArrivalHandler());
    }

class ArrivalHandler
    implements MessageHandler
    {
        public void run(Message message) {}  
    }
}
Queue-and-Server Model

class QueueAndServer extends AbstractModel {
    RandomVariable serviceTime;
    int numberInQueue = 0;
    boolean serverBusy = false;

    QueueAndServer (long mean) {
        super(1, 1);
        serviceTime = new ExponentialRV(mean);
        setMessageHandler(new ArrivalHandler());
    }

    class ArrivalHandler implements MessageHandler {
        public void run (Message message) {
            if (serverBusy) ++numberInQueue;
            else {
                serverBusy = true;
                schedule(new Departure(Math.round(getTime()) +
                    serviceTime.nextDouble()));
            }
        }
    }
}
class Departure extends AbstractEvent
{
    Departure (long time) { super(time); }

    public void run ()
    {
        send(new VoidMessage(getTime()));
        if (numberInQueue == 0)
            serverBusy = false;
        else
        {
            --numberInQueue;
            schedule(new Departure(Math.round(getTime()) +
                                      serviceTime.nextDouble()));
        }
    }
}
class Queueing extends AbstractSimulation {
    public void run () {
        Channel chan0 = createChannel();
        Channel chan1 = createChannel();
        createProcess(new Source(1000),
                new ChannelHead[] {}, new ChannelTail[] { chan0 });
        createProcess(new QueueAndServer(1000),
                new ChannelHead[] { chan0 }, new ChannelTail[] { chan1 });
        createProcess(new Sink(),
                new ChannelHead[] { chan1 }, new ChannelTail [] {});
        super.run();
    }
}
Summary and Conclusions

- Parsimony as vehicle for research in distributed discrete-event simulation

- project goals and status

- contributions of paper:
  - identification of requirements of discrete-event simulation with respect to the underlying implementation technology
  - show how Java language and JVM directly support these requirements