Developing Aspects for a Discrete Event Simulation system

M. Uğur Aksu

Faruk Belet

Bahadır Özdemir

OUTLINE

- Introduction
- Simulations & DES Modeling with EGs
- Simkit
- Identifying Concerns & AOSD Solutions
- Conclusion

Introduction

What is the Subject?

- Simkit is a simulation modeling tool that implements a number of concerns.
- These concerns crosscut over multiple modules in the system.
- This increases the complexity and reduces the maintainability and ease of use of this tool.
- Identify the crosscutting concerns in Simkit and try to refactor it with Aspect Oriented Software Design.

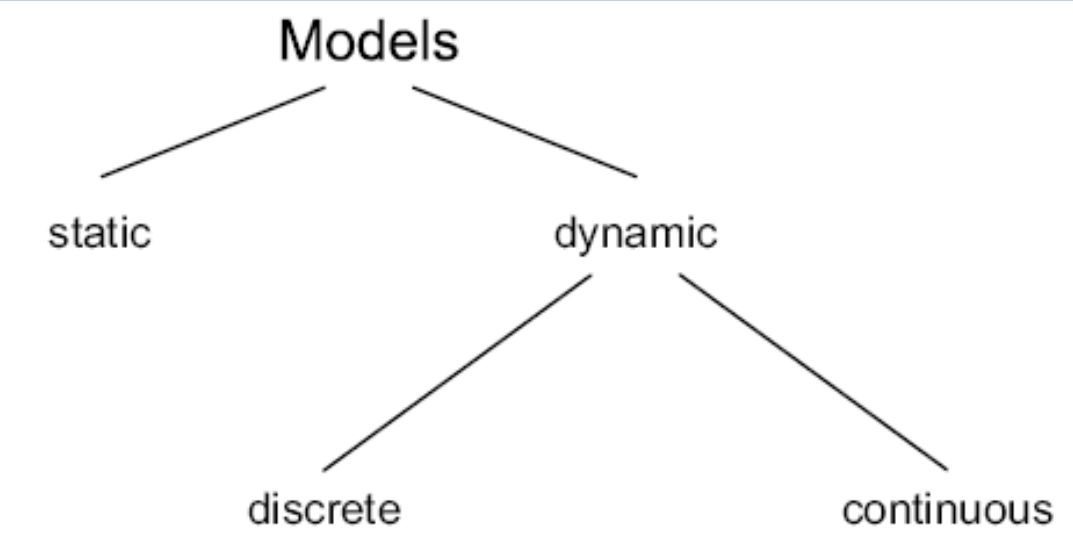
Simulations

What is Simulation?

- Description of a real system by modeling,
  - using abstraction and idealization
- Used for experimentation
- Helpful for understanding the behavior of systems
- Helpful for exploring alternative strategies for its operation

Simulations - II

How to Categorize Simulation Models?



In Discrete Event Simulations (DES), all system state changes are mapped to discrete events, assuming nothing relevant happens in between, while in continuous simulations states change steadily over time.

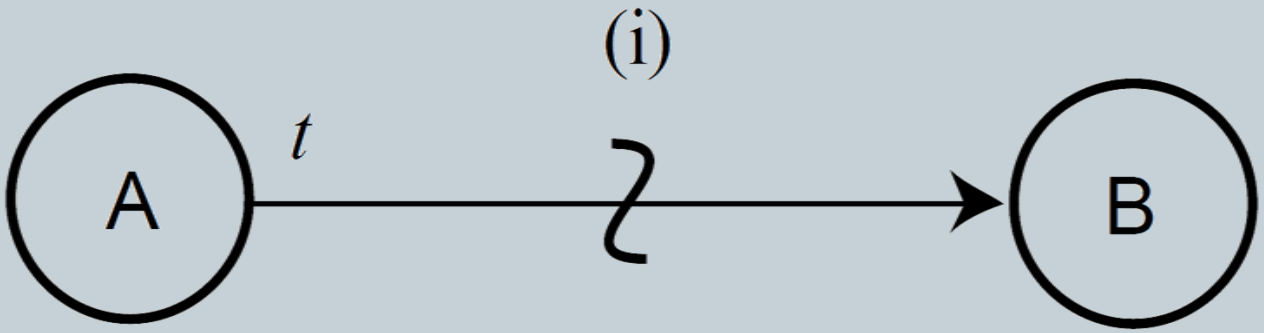
Discrete Event Simulations (DES)

Components of Discrete-Event Simulation (DES)

- Simulation time
- A set of state variables
- A set of events
  - In DES, events occur at the points in time at which state variables change values and simulation time is updated to the current event's time. Events are kept in an Event List which is simply a to-do list.

Event Graphs

Event Graphs are graphical notation for DES



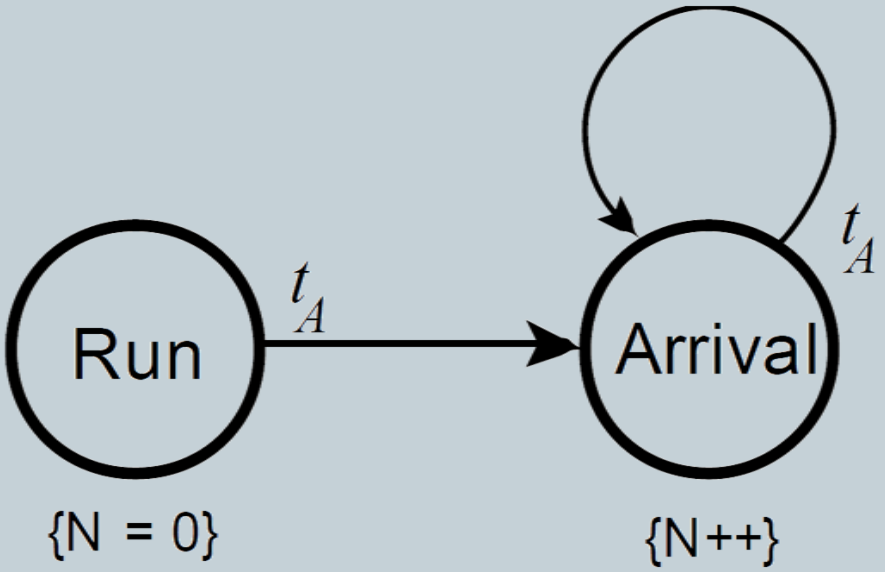
Simkit

What is SimKit?

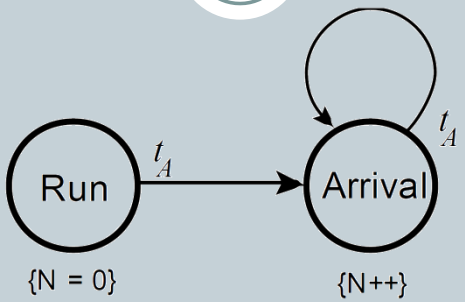
- SimKit is an Event Graph based DES modeling tool with an implementation of general purpose high level programming language.
- Every element in an Event Graph model has a corresponding element in SimKit.

Event Graphs & Simkit

Example: Event Graph for Arrival Process



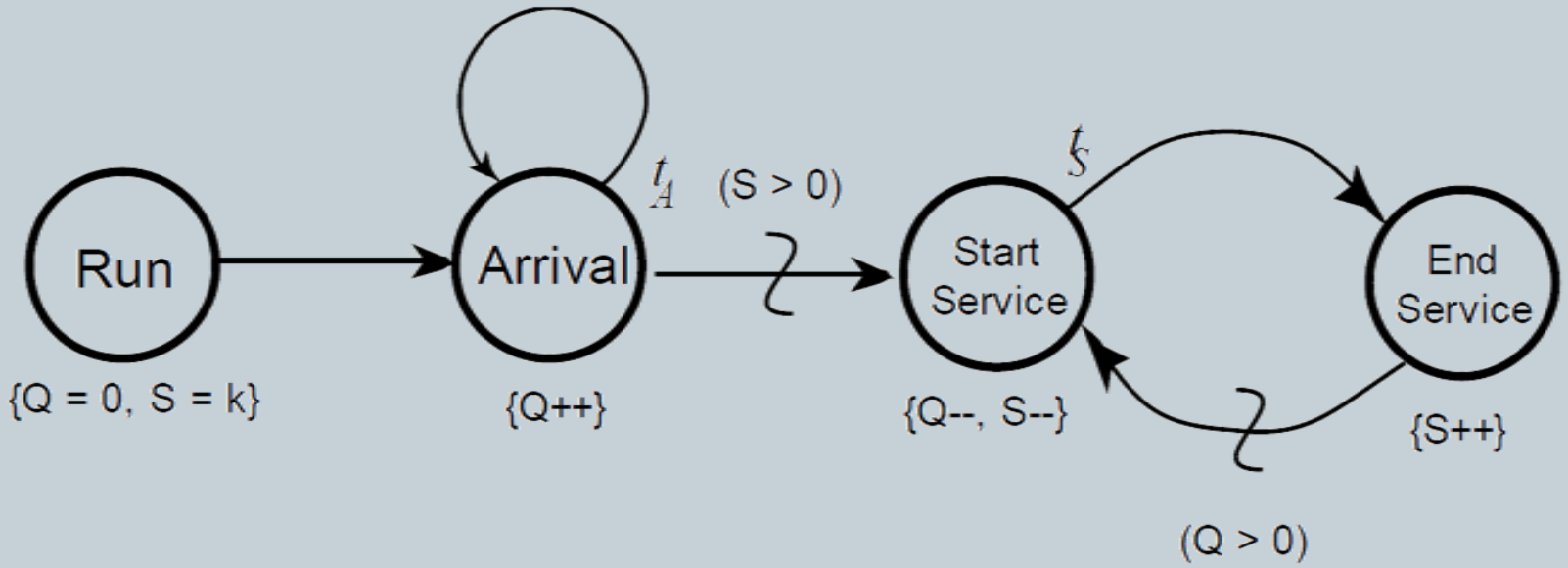
Event Graphs & Simkit - II




```
private RandomVariate interarrival;
protected int numberArrivals;
public void reset() {
    numberArrivals = 0;
}
public void doRun() {
    waitDelay("Arrival", interarrival.generate());
}
public void doArrival() {
    firePropertyChange("numberArrivals
        ",numberArrivals, ++numberArrivals);
    waitDelay("Arrival", interarrival.generate());
}
```

Event Graphs & Simkit - III

Example: Event Graph for Multiple Server Queue



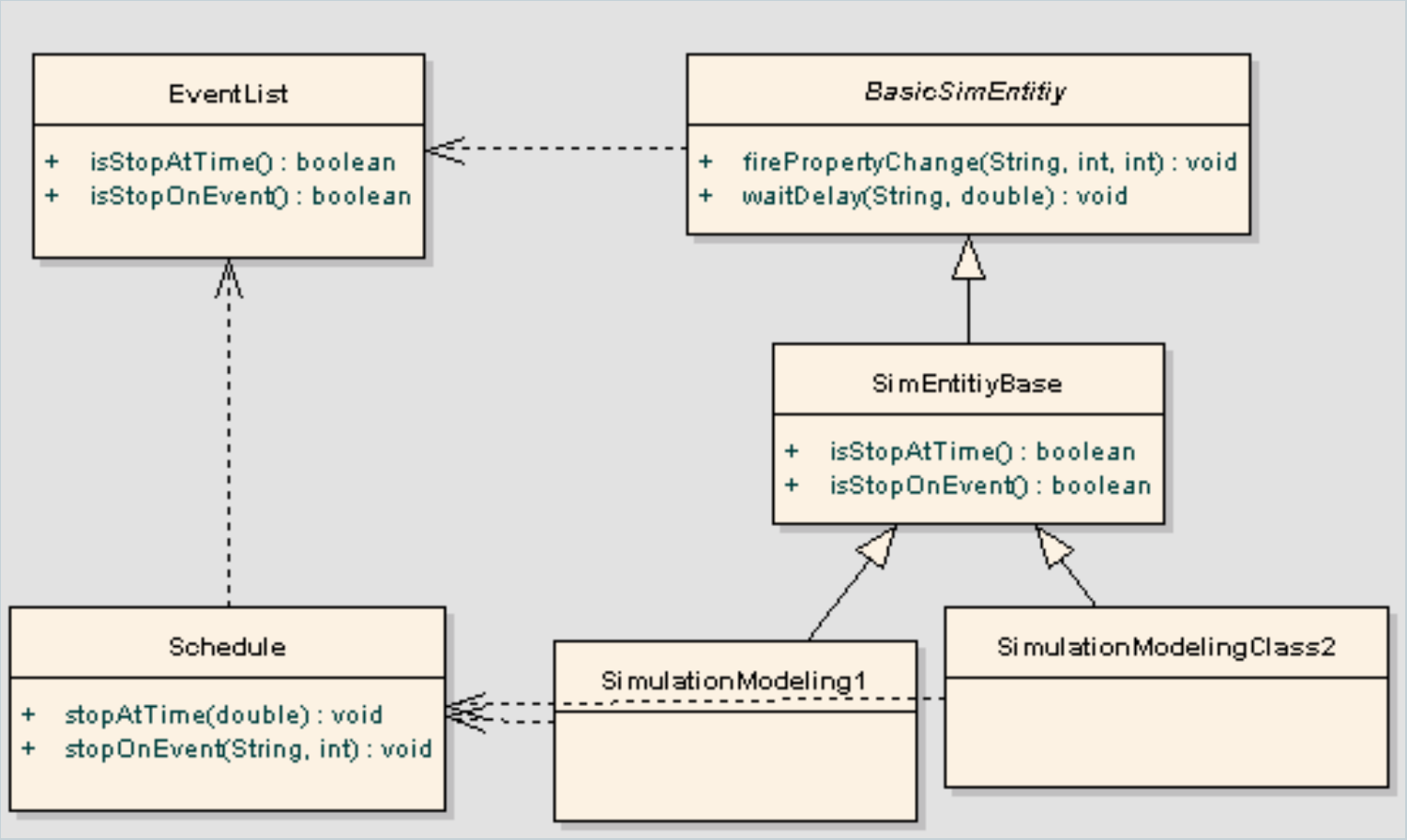
OO Design of Simkit

Fundamental packages of SimKit

- simkit: the core simulation related logic
- simkit.random: a random number generation factory
- simkit.stats: utility package to collect simulation related statistics.

OO Design of Simkit - II

A High Level Class Diagram



Design Patterns

- The SimKit library includes several design patterns such as observer, factory method, adapter etc.
- A new Façade pattern is added to the library
  - Combines several components in it.
  - Makes the library easier to use and understand.
  - Makes code that uses the library more readable.

AOSD Solutions to Crosscutting Concerns

Production Concerns

- Simulation Termination Rules
- Persistence: Restoring a Simulation Run
- Resource Pooling Concern
- Observer Pattern Concern

Simulation Termination Rules

- There are a number of ways of terminating a DES
  - Specifying a fixed length model time
  - Defining a simulation termination event
  - Defining a fixed length sample size
  - Terminating the simulation in terms of the model's state.
- The concern with regard to terminating simulations must be implemented cohesively and separately

Simulation Termination Rules – Code


```
privileged public aspect SimStopRulesAspect {

    public pointcut somePointInSimRunLoop(simkit.EventList eventList) :
          execution(public boolean simkit.EventList.isSingleStep()) && this(eventList);

    before(simkit.EventList eventList) : somePointInSimRunLoop(eventList) {
        eventList.checkStopState();
    }

    private int simkit.EventList.stopValue = 0;
    private simkit.SimEntityBase simkit.EventList.simObject = null;
    private String simkit.EventList.propertyName = "";
    private boolean simkit.EventList.isStopAtStateIfGreater = false;
    private boolean simkit.EventList.isStopAtState = false;

    public static void simkit.Schedule.stopAtTime(double atTime) {
        defaultEventList.stopAtTime(atTime);
    }

    public static void simkit.Schedule.stopOnEvent(int numberEvents,
            String eventName, Class... eventSignature) {
        defaultEventList.stopOnEvent(numberEvents, eventName, eventSignature);
    }

    public void simkit.EventList.stopAtTime(double time) {
        // Related code..
    }

    public void simkit.EventList.stopOnEvent(int numberEvents,
            String eventName, Class... signature) {
        // Related Code..
    }
```

Simulation Termination Rules – Code II


```
    public static void simkit.Schedule.stopOnEvent(int numberEvents,
            String eventName, Class... eventSignature) {
        defaultEventList.stopOnEvent(numberEvents, eventName, eventSignature);
    }

    public void simkit.EventList.stopAtTime(double time) {
        // Related code..
    }

    public void simkit.EventList.stopOnEvent(int numberEvents,
            String eventName, Class... signature) {
        // Related Code..
    }

    public static void simkit.Schedule.stopAtState(int stopValue,
            boolean isStopAtStateIfGreater, simkit.SimEntityBase simObject,
            String propertyName) {
        defaultEventList.stopAtState(stopValue, isStopAtStateIfGreater,
                simObject, propertyName);
    }

    public void simkit.EventList.stopAtState(int stopValue,
            boolean isStopAtStateIfGreater, simkit.SimEntityBase simObject,
            String propertyName) {
        this.stopValue = stopValue;
        this.simObject = simObject;
        this.propertyName = propertyName;
        this.isStopAtStateIfGreater = isStopAtStateIfGreater;
        this.isStopAtState = true;
    }

    public void simkit.EventList.checkStopState() {
        // Related code..
    }
}
```

Persistence: Restoring a Simulation Run

- Persistence is a generic production concern that has a crosscutting and tangling nature.
  - Simulation runs are costly in terms of time.
  - Simulations might terminate unexpectedly due to a number of reasons.
  - Simulations can be terminated properly by users.

Persistence – Code


```
public privileged aspect StoreEventListStateAspect {

    private static Formatter fileOutput;

    public void simkit.EventList.storeEventListState() {
        StoreEventListStateAspect.storeEventListState(this);
    }

    public static void storeEventListState(simkit.EventList eventList) {
        // Related code: stores event list state
    }

    public pointcut storePoint(simkit.EventList eventList) :
          call(boolean simkit.EventList.isStopOnEvent()) && this(eventList)
          && (withincode(public void simkit.EventList.startSimulation()));

    after(simkit.EventList eventList) : storePoint(eventList) {
        eventList.storeEventListState();
    }
}
```

Persistence – Code II

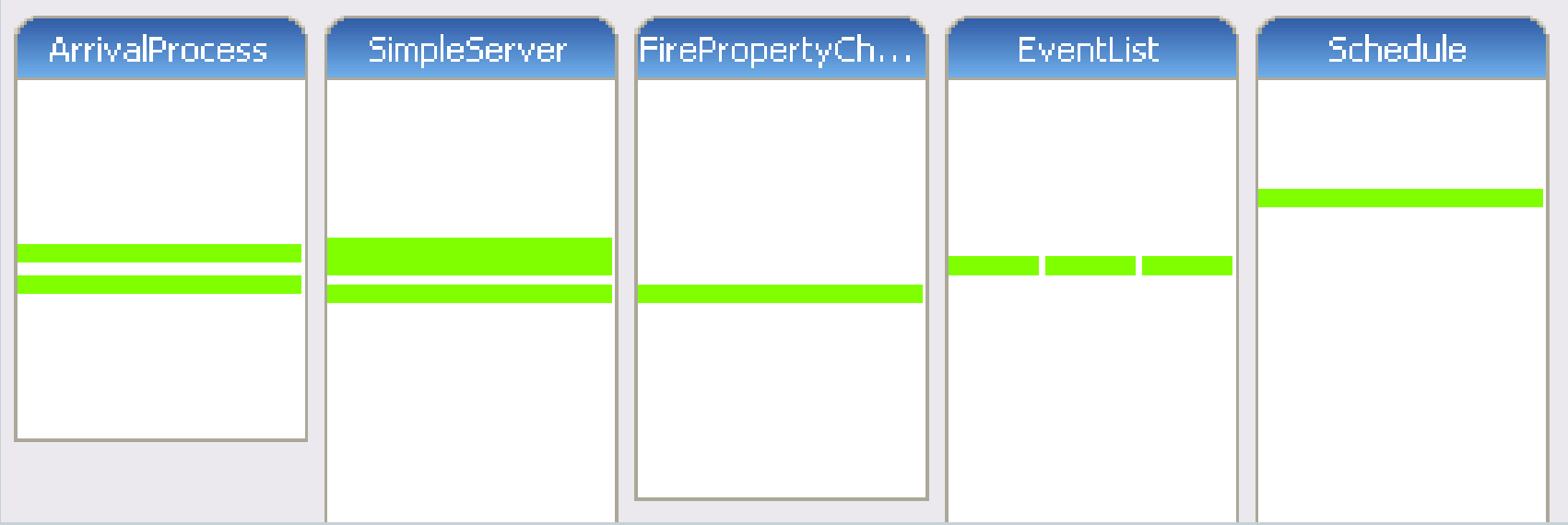

```
public aspect StoreSimVariablesStatesAspect {

    private static Map stateVariableTable = new TreeMap();
    private static Formatter fileOutput;

    pointcut propertyChangeMethods(String propertyName, int newPropertyValue):
        call(* simkit.BasicSimEntity.firePropertyChange(String, int))
        && ! within(simkit..*) && args(propertyName, newPropertyValue);

    before(String propertyName, int newPropertyValue) :
        propertyChangeMethods(propertyName, newPropertyValue) {
        stateVariableTable.put(propertyName, newPropertyValue);
    }

    public void simkit.EventList.storeStateVariablesStates() {
        StoreSimVariablesStatesAspect.storeStateVariablesStates();
    }

    public static void storeStateVariablesStates() {
        // Related code: stores state variables' states
    }

    public pointcut storePoint(simkit.EventList eventList) :
          call(boolean simkit.EventList.isStopOnEvent()) && this(eventList)
          && (withincode(public void simkit.EventList.startSimulation()));

    after(simkit.EventList eventList) : storePoint(eventList) {
        eventList.storeStateVariablesStates();
    }
}
```

Persistence – Code III


```
privileged public aspect PersistenceAspect {

    public static boolean isResumeSimulation = false;

    public static void simkit.Schedule.resumeSimulation(boolean isResume) {
        PersistenceAspect.isResumeSimulation = isResume;
    }

    public pointcut resetProperty() :
        ! within(simkit..*) && set(protected int simkit.SimEntityBase+.*)
        && withincode(public void reset());

    after() : resetProperty() {
        // Related code: initiliaze using the last state variables' states
    }


    pointcut locateDoRun(simkit.SimEntityBase object):
        execution(* simkit.SimEntityBase+.doRun()) && ! within(simkit..*)
        && target(object);

    void around(simkit.SimEntityBase object): locateDoRun(object) {
        // Related code: set the event list to the last stable state recorded
    }

    public static String readEventListState() {
        // Related code: read event list from a file
    }

    public static String readSimVariablesStates() {
        // Related code: read state variables from a file
    }
}
```

Persistence: Crosscutting View

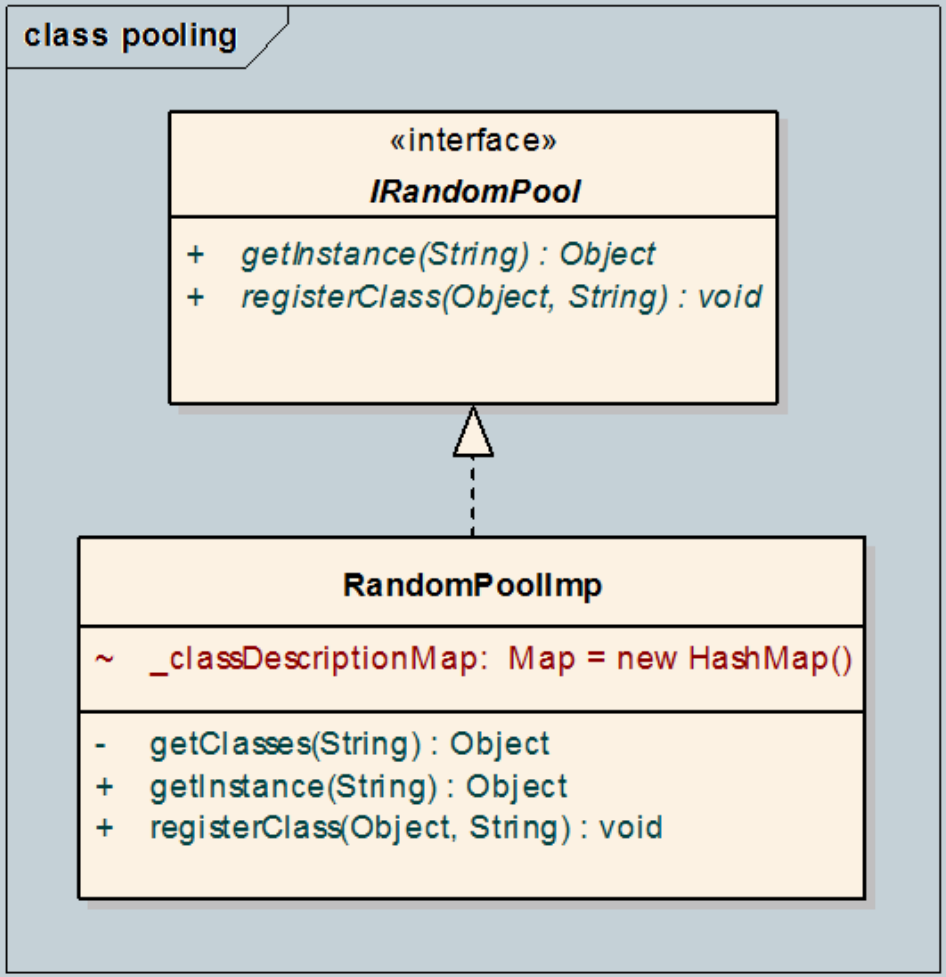
Crosscutting Nature of the Persistence Concern



Resource Pooling Concern

- Resource pooling is a system-wide and generally a scattered concern.
  - It keeps the resources created earlier, so they can be reused later instead of recreating new ones.
  - This feature is especially important in terms of time-efficiency.
  - Enable resource pooling only if the desire for improved speed outweighs the cost of extra memory.

Resource Pooling Concern – Class Diagram



Resource Pooling Concern – Code


```
public aspect RandomPoolingAspect {

    /**
     * Holds the registered resources in the resource pooling
     */
    IRandomPool _randomPool = new RandomPoolImp();

    pointcut instanceCreation(String className):
        (call(* simkit.random..*Factory+.findFullyQualifiedNameFor(String))
        || call(* simkit.random..*Factory+.getClassFor(String)))
        && args(className);

    /**
     * around advise that returns objects from pool if it exist in the resource pool
     * using RandomPoolImp class.
     *
     */
    Object around(String className) : instanceCreation(className) {
        Object myClass = _randomPool.getInstance(className);
        //Related Code..
        //Return or register object in resource pool
    }
}
```

Observer Pattern Concern

- Observer patterns is a proven solution
  - Elegant way for solving distributed event handling systems.
  - However, observer pattern concern is scattered over the design.
- The concern of observer pattern must be implemented cohesively and separately

Observer Pattern Concern – Code


```
public aspect FirePropertyChangeAspect {

    public pointcut propertyChange(Object value) :
        ! within(simkit..*) && set(protected int simkit.SimEntityBase+.*)
        && withincode(public void do*()) && ! withincode(public void doRun())
        && args(value);

    void around(Object value) : propertyChange(value) {
        SimEntityBase target = (SimEntityBase) thisJoinPoint.getTarget();
        String field = thisJoinPoint.getSignature().getName();
        Object old = target.getProperty(field);

        proceed(value);
        // Related call: call corresponding firePropertyChange function

    }
```

Development Concerns

- Checking Main Method's Parameters
  - Checking main method's parameters separately from the core logic offers a more cohesive main method that prevents tangling.

Check Main Method's Paramaters - Code


```
public pointcut captureMain(String[] args) :
      execution(void *.main(String[])) && args(args);


void around(String[] args) : captureMain(args) {
    if (args.length < 2 || args.length > 3) {
        args = new String[] { "someString", "anotherString" };
        System.err.println(CORRECTUSAGE);
    }
    proceed(args);
}
```

Enforcement Concerns

- Enforcement Concerns

  Asking simulation modelers to comply with a set of constraints is a very noticeable tip that tells us to employ enforcement aspects. Using enforcement aspects, we not only get a more robust design but also get rid of the tangling concerns we would have otherwise.

  - Constraints for State Variables
  - Constraints for “do” Method

Constraints for State Variables

- Restricting Set Accesses to State Variables
  - Each state variable in Simkit models is implemented by an instance variable with protected access and should have public getter methods but setter methods, so that state variables can only be changed by calls to firePropertyChange(propertyName, oldValue, newValue) methods.

```
pointcut illegalSetAccesses():
    ! within(somePackage) && set(protected * someClass.someVariable)
    && !(withincode(public void aMethod()) ||
         withincode(public void anotherMethod(..))) ;

declare error : illegalSetAccesses() : "ERROR MESSAGE ";
```

Constraints for State Variables - II

- Checking State Variable Declarations
  - For the first arguments of each property change method, there must be a matching state variable declaration.

Checking State Variable Declarations


```
public aspect PolicyEnforcementForStateVariables2 {

    List<String> declaredStateVariables = new ArrayList<String>();

    pointcut locateASimulationClass():
        execution(simkit.SimEntityBase+.new(..)) && !within(simkit..*);

    before():locateASimulationClass(){
        // Related code:
        // collect context for state variables by Java Reflection
    }

    pointcut propertyChangeMethods(String propertyName):
        call(* simkit.BasicSimEntity.firePropertyChange(String, ..))
        && ! within(simkit..*) && args(propertyName, ..);

    before(String propertyName) :
        propertyChangeMethods(propertyName) {
        if (!declaredStateVariables.contains(propertyName)) {
            System.err.println("ERROR MESSAGE ");
            simkit.Schedule.stopSimulation();
            System.exit(1);
        }
    }
}
```

Constraints for “do” Methods

- Constraints for “do” Methods
  - Each event in a Simkit model is implemented as user-defined “do” methods.
  - And events are scheduled by calls to waitDelay(string, delay) functions.
  - Simkit uses Java’s reflection to determine the corresponding method for scheduled events.
  - First of all, when there do not exist corresponding instance methods, for example, such as doRun() and doArrival() for events such as Run and Arrival respectively, the simulation will run incorrectly.
  - Secondly, there is no restriction which prevents calling the “do” methods by the programmer.

Constraints for “do” Methods - Code


```
public aspect PolicyEnforcementForEventHandling perthis(locateASimulationClass()){

    List<String> declaredDoMethods = new ArrayList<String>();

    pointcut locateASimulationClass():
        execution(simkit.SimEntityBase+.new(..)) && !within(simkit..*);

    before():locateASimulationClass(){
        // Related code:
        // collect context for declared event handling methodsby Java Reflection
    }

    pointcut schedulingEventMethods(String scheduledEventName):
        call(public simkit.SimEvent simkit.SimEntityBase+.waitDelay(String, ..))
        && ! within(simkit..*) && args(scheduledEventName, ..);

    before(String schedulingEventName) :
        schedulingEventMethods(schedulingEventName) {
        if (!declaredDoMethods.contains(schedulingEventName)) {
            System.err.println("ERROR MESSAGE ");
            simkit.Schedule.stopSimulation();
            System.exit(1);
        }
    }
}
```

Constraints for “do” Methods – Code II


```
pointcut illegalCallToFirePropertyChange():
    ! within(simkit..*) && within(simkit.SimEntityBase+)
    && call(public void firePropertyChange(..));


declare error : illegalCallToFirePropertyChange()
    : "ERROR MESSAGE";
```

Conclusion

- First of all, we have proved one more time that using only object oriented paradigm is not sufficient to manage all the concerns, i.e. the crosscutting concerns.

- Secondly, we have shown that, concerns such as simulation termination rules, restoring a simulation run (persistence) and creating random number generators are inherently crosscutting for simulation systems.

Conclusion - II

- Thirdly, we have shown that, using a high level programming language that does not offer a graphical user interface, imposes many constraints on simulation tools users. Such concerns can be overcome by enforcement aspects smoothly.
- Finally, we demonstrated that when working on legacy code for refactoring purposes, using Java Reflection and aspects together, might prove to be very convenient and less costly. Thus it can be adopted as an idiom.

Thank You!

ANY QUESTIONS?