Proceedings of the 1986 Winter Simulation Conference
J. Wilson, J. Henriksen, S. Roberts (eds.)

THE SMALLTALK SIMULATION ENVIRONMENT

Verna Knapp
Computer Research Laboratory
Tektronix Laboratories
Beaverton, OR 97077

ABSTRACT

The Smalltalk language provides easy to use support for discrete event simulation. Smalltalk is an object oriented language which is descended from Simula. There is a set of Smalltalk classes which directly support simulation. A simulation of a multiprocessor computer architecture with snooper caches and shared global virtual memory has been implemented in Smalltalk. This simulation is discussed here to illustrate the techniques involved.

1. INTRODUCTION

The purpose of this paper is to show how the Smalltalk language supports discrete event simulation. The Smalltalk classes which implement simulation support will be discussed, and an example using these classes will be presented. This example was written as part of an investigation of multiprocessor computer architectures with caching and shared global memory. It runs on a Tektronix 4406 workstation.

2. CLASSES AND OBJECTS IN SMALLTALK

Everything in Smalltalk is an "object". Every object is an instance of a "class". A class contains:

- Class variables
- A template for the instance variables of instances of objects of that class
- Methods (procedures) for processing messages sent to objects of that class

Classes can have subclasses. A subclass inherits the variables and methods of its parent class. Classes are arranged in a tree structure with each class a subclass of some other class. The root class of the class structure is "Object".

In Smalltalk execution proceeds through objects sending messages to other objects and waiting until the other objects reply.

3. SIMULATION SUPPORT IN SMALLTALK

Smalltalk provides classes for discrete event simulation. The user will either use these classes directly or extend them through the subclass mechanism to control the simulation and to act as objects in the simulation. These classes include Simulation, SimulationObject, DelayedEvent, WaitingSimulationObject, Resource, ResourceProvider, and ResourceCoordinator. There are also classes to provide the necessary probability distributions.

Smalltalk also provides an easily used graphics capability, which can be used to display simulation results either after the fact as a static display, or as an animated display of guages which show the current state of the simulation as it runs.

3.1 Class Simulation

```
Object subclass: #Simulation
        instanceVariableNames: 'resources currentTime eventQueue
        processCount stoppedFlag '
        classVariableNames: 'RunningSimulation '
        category: 'Simulation-support'
```

An instance of class Simulation controls a single running simulation. It initializes the simulation, controls it, and does any necessary termination processing. The class Simulation itself has a single global variable, RunningSimulation, which points to the running instance of Simulation in the system. Any object may request a pointer to this active instance of Simulation from the class Simulation. This allows objects which are instances of the other classes to send messages to the simulation control. There can be only one simulation active at a time.

An instance of Simulation creates resources and provides access to them. It responds to messages asking that it coordinate a resource, produce a resource, provide a resource to the requestor, or state whether a resource would be available if requested. Class Resource and its subclasses describe resources further.

During initialization a Simulation object schedules the creation of instances of SimulationObject according to a probability distribution of interarrival times or at a specified simulation time. After initialization it responds to messages requesting that it schedule the creation or arrival of a SimulationObject either according to a probability distribution of interarrival times, or after a delay interval or at a specified simulation time.

The Simulation manages an event queue. When a process sends the Simulation a message asking that it be rescheduled after some delay or at a specified simulation time, the Simulation creates an instance of DelayedEvent and puts it on the event queue. When the DelayedEvent is scheduled, a reply to this message will be sent to the original process and it will resume execution. The Simulation proceeds by scheduling all instances of DelayedEvent which are to start at the same time as the first event on the event queue, and then suspending itself until all of these processes have either terminated or delayed themselves to a later simulated time. The Simulation continues until either the event queue is empty, or some process sends the Simulation a message requesting that it stop.

The Simulation maintains the simulated time clock, and responds to messages requesting the current simulated time.

3.2 Class SimulationObject

```
Object subclass: #SimulationObject
        instanceVariableNames: ''
        classVariableNames: ''
        category: 'Simulation-support'
```

The class SimulationObject provides a skeleton which the user will flesh out to describe the objects which are being simulated. A SimulationObject has a series of tasks to do. These tasks describe the simulated activites. A SimulationObject can request access to resources or the creation of resources from the Simulation. It can request the Simulation to schedule the creation of any SimulationObject, or to schedule the arrival of any existing SimulationObject which it knows about. It can request that the Simulation delay it until some future simulated time. It can also stop the Simulation. Statistics are often collected by SimulationObjects.

3.3 Class DelayedEvent

```
Object subclass: #DelayedEvent
        instanceVariableNames: 'resumptionSemaphore resumptionCondition '
        classVariableNames: ''
        category: 'Simulation-support'

DelayedEvent subclass: #WaitingSimulationObject
        instanceVariableNames: 'amount resource '
        classVariableNames: ''
        category: 'Simulation-resources'
```

An object of class DelayedEvent or its subclass WaitingSimulationObject is created when it is necessary to suspend a process until some simulated time or until some resource becomes available. It contains a semaphore which will be signaled to restart the process at the appropriate time. A DelayedEvent will be placed in the event queue of the Simulation, and will be scheduled at the requested simulated time. A WaitingSimulationObject will be placed in the pending queue of a resource, and will be scheduled when the resource becomes available.

3.4 Class Resource

```
Object subclass: #Resource
        instanceVariableNames: 'pending resourceName '
        classVariableNames: ''
        category: 'Simulation-resources'

Resource subclass: #ResourceCoordinator
        instanceVariableNames: 'whoIsWaiting '
        classVariableNames: ''
        category: 'Simulation-resources'

Resource subclass: #ResourceProvider
        instanceVariableNames: 'amountAvailable '
        classVariableNames: ''
        category: 'Simulation-resources'
```

There are two subclasses of Resource, ResourceProvider and ResourceCoordinator. ResourceProvider describes resources which are created and consumed or which are requested from a pool of resources of a given type and then returned to the pool. Examples of ResourceProvider resources are a car dealer's inventory of cars, or an auto rental agency's pool of cars. ResourceCoordinator describes a server/customer relationship. A ResourceCoordinator will coordinate one or more servers with one or more customers. At any given time its pending queue will consist either entirely of servers or entirely of customers.

4. MULTIPROCESSOR ARCHITECTURE SIMULATION CLASSES

The example we will discuss is the simulation of a multiprocessor with snooper caches, virtual memory, and shared global memory. This architecture is discussed extensively in the author's PhD dissertation. Each processor has a set associative cache which snoops the bus for activity which affects the contents of the cache. The cached data is tagged with the system virtual address of the data. In each cache there are identical banks of tags so that the snooper control can read one bank while the processor cache control is reading the other bank. Writes to the tag memory require access to both banks at the same time by whichever controller is updating the memory. Thus each tag memory has two resources, 'bank1n' and 'bank2n', where n is the processor number. There is a single bus, and an associated 'bus' resource. The memory management units are associated with the banks of global memory rather than with the processors. The memory management units contain a translation lookaside buffer, and fetch missing entries from the page tables. Actual page faults are not simulated, since the purpose of this simulation is to determine the appropriate size for the caches, the effectiveness of cache management algorithms, the appropriate sizes of pages, and the desirable number of memory management units for the system under simulation.

The parameters of the system include the number of processors, the number of memory management units, the page size, the number of cache entries, the cache set size, and the cache line size.

The results of the simulation are displayed as guages drawn on the bitmap display. This provides an animated display of the simulation as it progresses. The display includes such information as the actual MIPS rate of the processors, the cache hit ratio, bus contention, and bus utilization for the most recent simulated interval. This display is quite useful in developing and understanding of the dynamic behavior of the model. Simulation results are also written to a disk file for offline analysis.

4.1 Class SimMP

```
Simulation subclass: #SimMP
    instanceVariableNames: 'myProcessors myBus statGrabber
    mmuCount processorCount index page set '
    classVariableNames: ''
    category: 'MP-Cache-Simulation'
```

SimMP is a subclass of Simulation. It initializes the system by creating processors, a bus, memory management units, and a statistics recording and display process. The processors are scheduled for simulation time 0.0, and the statistics recording and display process is scheduled to occur at regular intervals after an initial startup time has elapsed. SimMP also creates the bank and bus resources. At the termination of the simulation SimMP closes the statistics recording file.

4.2 Class SimProcessor

```
SimulationObject subclass: #SimProcessor
    instanceVariableNames: 'myLabel myCache reads readTime
    writes writeTime '
    classVariableNames: ''
    category: 'MP-Cache-Simulation'
```

SimProcessor is a subclass of SimulationObject. At initialization time a processor creates its own cache controller and its own snooper controller. It gives the bus a pointer to the snooper. During the simulation it reads an address and a read or write command from a trace of an actual processor, and presents these to its cache controller. When the cache controller replies to this message, it records statistics about the elapsed simulated time of the reads and writes. Then it reschedules itself immediately.

A SimProcessor responds to messages to record and display statistics from the statistics gathering process. It also passes a message to its cache and its snooper to record and display their statistics.

4.3 Class SimCache

```
SimulationObject subclass: #SimCache
    instanceVariableNames: 'tagMemory statistics bus bank1
    bank2 '
    classVariableNames: ''
    category: 'MP-Cache-Simulation'
```

SimCache is a subclass of SimulationObject. A SimCache creates its own tag memory when it is initialized. A cache processes requests to read and write data which are sent to it by the processor. To read the tag memory, it must first obtain the 'bank1n' resource. This resource is also used by the snooper when updating tag memory. It requests this resource from the active instance of SimMP. This request is queued as a WaitingSimulationObject in the 'bank1n' resource's pending queue until the resource is available. When the cache controller has this resource, it reads the tag memory to determine whether the data is in the cache, and what state it is in. Then it releases the 'bank1n' resource.

The same data can exist with read permission in many caches, or with write permission in a single cache at a time. There is a cache coherency protocol to maintain this state. The protocol requires cache controllers and snooper controllers to communicate with each other by means of the bus.

Data in the cache can exist in one of four states: invalid, read exclusive, read shared, and dirty. If this is a read request and the data is read exclusive, read shared, or dirty, the cache delays for one cache read time, updates the statistics counters, and returns a reply to the SimProcessor which made the request. If this is a write request and the data is dirty or read exclusive, the cache delays for one cache write time, updates the statistics counters, and returns a reply to the SimProcessor which made the request. All other requests involve a bus access.

To access the bus, the SimCache must first obtain possession of the 'bus' resource. It requests this resource from the active instance of SimMP. This request is queued as a WaitingSimulationObject in the 'bus' resource's pending queue until the resource is available. When the cache controller has this resource, it sends the appropriate message to the SimBus. When it receives the reply to this message it releases the 'bus' resource. The possible messages include 'invalidate', 'readForRead', 'readForWrite', and 'writeBack'. When the SimCache regains control, it delays as necessary to simulate updating the cache, and then it updates the appropriate statistics counters and returns control to the SimProcessor.

During the simulation SimCache gathers statistics about its own operation, and responds to requests to record and display them.

4.4 SimBus

```
SimulationObject subclass: #SimBus
        instanceVariableNames: 'snoopers statistics mmus '
        classVariableNames: ''
        category: 'MP-Cache-Simulation'
```

SimBus is a subclass of SimulationObject. The SimBus receives 'invalidate', 'readForRead', 'readForWrite', and 'writeBack' messages. These messages cause it to pass requests to the SimSnoopers and SimMMUs, collect statistics on its own activity, and delay as necessary to simulate the elapsed time. SimBus also responds to requests to record and display its statistics.

4.5 SimSnooper

```
SimulationObject subclass: #SimSnooper
        instanceVariableNames: 'processorLabel bank1 bank2 tag-
        Memory statistics '
        classVariableNames: ''
        category: 'MP-Cache-Simulation'
```

SimSnooper is a subclass of SimulationObject. In response to messages from the SimBus the SimSnoopers read and write their associated tag memories to support the cache coherency protocol. To read its tag memory, a SimSnooper must obtain the 'bank2n' resource from the active instance of SimMP. To write its tag memory, it must obtain both 'bank1n' and 'bank2n'. The SimSnoopers record and display statistics concerning their accesses to these resources.

4.6 SimMMU

```
SimulationObject subclass: #SimMMU
        instanceVariableNames: 'pages statistics id lastAdd size '
        classVariableNames: ''
        category: 'MP-Cache-Simulation'
```

SimMMU is a subclass of SimulationObject. A SimMMU maintains entries for the 64 most recently accessed pages. If it receives a message requesting access to one of these pages it delays only for the time required for a memory access. If it receives a message requesting access to some other page, it delays for the time required to look up the page translation plus the time for a memory access. It also records and displays statistics on its own activity.

5. COMMENTS ON THE MULTIPROCESSOR ARCHITECTURE SIMULATION

The multiprocessor architecture simulation contains 14 classes and 140 methods. It was written and debugged by one person in 2 days. This is typical of the ease of programming in a Smalltalk environment. An example simulation using a random number generator to generate the address stream required 6 minutes of elapsed time to simulate a 4 processor system running 20,000 processor cycles per processor. This simulation was run on a Tektronix 4406 workstation. This workstation has a Motorola MC68020 cpu, no cache, and 5 megabytes of real memory. An address trace of over a million memory accesses is currently being collected to drive the simulated processors. The simulation will be run repeatedly, varying the parameters to determine expected performance of various possible implementations of this architecture. Because Smalltalk programs are very easily modified, a number of other possible architectures can be investigated.

In conclusion, Smalltalk provides easily used support for discrete event simulation. Performance is good enough for detailed simulation of a complex computer architecture, even while running an animated display of the current state of the simulation. The programming environment provided by Smalltalk facilitates creating simulations, and the animated graphics display capability aids understanding of the simulation results. Smalltalk is a highly useful simulation language.

REFERENCES

Goldberg, A., and Robson, D. (1983) *Smalltalk-80 The Language and its Implementation* Addison-Wesley, Menlo Park, Ca

Knapp, V., Virtually Addressed Caches for Multiprogramming and Multiprocessing Environments, PhD Dissertation, Dept of CSci, TR-85-06-02, University of Washington, 1985

AUTHOR'S BIOGRAPHY

Verna E. Knapp is a Senior Engineer with the Advanced Systems Architectures Group of the Computer Research Laboratory in Tektronix Laboratories. She received the PhD degree in computer science from the University of Washington in 1985. Her research interests include multiprocessor memory architectures and interconnect networks.

Verna E. Knapp
Computer Research Laboratory
Tektronix Labs, MS 50-662
Tektronix, Inc.
P.O. Box 500
Beaverton, OR 97077