Exhibit ZTE-CC-SIMULATION

Matloff, Norm,

Introduction to Discrete-Event Simulation and the SimPy Language

Introduction to Discrete-Event Simulation and the SimPy Language

Norm Matloff

February 13, 2008

Contents

1 What Is Discrete-Event Simulation (DES)?

Consider simulation of some system which evolves through time. There is a huge variety of such applications. One can simulate a weather system, for instance. A key point, though, is that in that setting, the events being simulated would be **continuous**, meaning for example that if we were to graph temperature against time, the curve would be continuous, no breaks.

By contrast, suppose we simulate the operation of a warehouse. Purchase orders come in and are filled, reducing inventory, but inventory is replenished from time to time. Here a typical variable would be the inventory itself, i.e. the number of items currently in stock for a given product. If we were to graph that number against time, we would get what mathematicians call a **step function**, i.e. a set of flat line segments with breaks between them. The events here—decreases and increases in the inventory—are discrete variables, not continuous ones.

DES involves simulating such systems.

2 World Views in DES Programming

Simulation programming can often be difficult—difficult to write the code, and difficult to debug. The reason for this is that it really is a form of parallel programming, with many different activities in progress simultaneously, and parallel programming can be challenging.

For this reason, many people have tried to develop separate simulation **languages**, or at least simulation **paradigms** (i.e. programming styles) which enable to programmer to achieve clarity in simulation code. Special simulation languages have been invented in the past, notably SIMULA, which was invented in the 1960s and has significance today in that it was the language which invented the concept of object-oriented programmg that is so popular today. However, the trend today is to simply develop simulation *libraries* which can be called from ordinary languages such as C++, instead of inventing entire new languages.[1] So, the central focus today is on the programming paradigms, not on language. In this section we will present an overview of the three major discrete-event simulation paradigms.

Several **world views** have been developed for DES programming, as seen in the next few sections.

2.1 The Activity-Oriented Paradigm

Let us think of simulating a queuing system. Jobs arrive at random times, and the job server takes a random time for each service. The time between arrivals of jobs, and the time needed to serve a job, will be continuous random variables, possibly having exponential or other continuous distributions.

For concreteness, think of an example in which the server is an ATM cash machine and the jobs are customers waiting in line.

Under the **activity-oriented paradigm**, we would break time into tiny increments. If for instance the mean interarrival time were, say 20 seconds, we might break time into increments of size 0.001. At each time point, our code would look around at all the activities, e.g. currently-active job servicing, and check for the possible occurrence of events, e.g. completion of service. Our goal is to find the long-run average job wait

[1] These libraries are often called "languages" anyway, and I will do so too.

time.

Let SimTime represent current simulated time. Our simulation code in the queue example above would look something like this:

```
QueueLength = 0
NJobsServed = 0
SumResidenceTimes = 0
ServerBusy = false
generate NextArrivalTime  // random # generation
NIncrements = MaxSimTime / 0.001
for SimTime = 1*0.001 to NIncrements*0.001 do
   if SimTime = NextArrivalTime then
      add new jobobject to queue
      QueueLength++
      generate NextArrivalTime  // random # generation
      if not ServerBusy then
         ServerBusy = true
         jobobject.ArrivalTime = SimTime
         generate ServiceFinishedtime
         currentjob = jobobject
         delete head of queue and assign to currentjob
         QueueLength--
   else
      if SimTime = ServiceFinishedtime then
         NJobsServed++
         SumResidenceTimes += SimTime - currentjob.ArrivalTime
         if QueueLength > 0 then
            generate ServiceFinishedtime  // random # generation
            delete currentjob from queue
            QueueLength--
         else
            ServerBusy = false
print out SumResidenceTimes / NJobsServed
```

2.2 The Event-Oriented Paradigm

Clearly, an activity-oriented simulation program is going to be very slow to execute. Most time increments will produce no state change to the system at all, i.e. no new arrivals to the queue and no completions of service by the server. Thus the activity checks will be wasted processor time. This is a big issue, because in general simulation code often needs a very long time to run. (Electronic chip manufacturers use DES for chip simulation. A simulation can take days to run.)

Inspection of the above pseudocode, though, shows a way to dramatically increase simulation speed. Instead of having time "creep along" so slowly, why not take a "shortcut" to the next event? What we could do is something like the following:

Instead of having the simulated time advance via the code

```
for SimTime = 1*0.001 to NIncrements*0.001 do
```

we could advance simulated time directly to the time of the next event:

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | **sales@docketalarm.com** | **1-866-77-FASTCASE**

fastcase® Smarter legal research.