SimPlus Extension

Mark Randles
randlem@bgsu.edu

Dan Sinclair
dsincla@bgsu.edu

Academic Supervisor
Hassan Rajaei, Ph.D

Department of Computer Science
Bowling Green State University, Ohio
28 April 2005

Abstract

This projects goal was to modify and extend the functionality of a already developed simulation engine, SimPlus. This engine was written using C++ and object-orientated design practices. We were to extend and add functionality that as not already present or needed to be added before it was possible for SimPlus to become a useful engine for teaching the principals of distributed event simulation.

Contents

Chapter 1

Introduction

1.1 Purpose

This project was intended to make necessary modifications to SimPlus, a discrete simulation engine, in order to make it available for classroom use in the future. A good, robust, and easy to understand simulation engine would be an excellent complement to a class on discrete simulation, and may also be used as a starting point for a parallel simulation engine that could be used in a parallel simulation course.

1.2 Goals and Objectives

The aim of our project was to improve the current implementation of SimPlus as well as add new features that could be usable and expandable for the future. We felt that the code base of SimPlus was rushed by its original authors and needed to be further improved before any external modifications could be made. Our first goal was to improve the stability of SimPlus, and make it more platform independent. Our second goal was to improve the existing code by including more distributions and random number generators. Our third goal was to make a meaningful improvement to SimPlus that could be extended by future developers to make SimPlus into a full-fledged simulation package. We felt that goal could be best met by adding a logging system to SimPlus. We felt that in order for SimPlus to be a usable engine for the future, we should concentrate our project efforts on further internal development of SimPlus instead of simply adding another level of complexity on top of an existing engine that contained problems.

1.3 Backgrounds

SimPlus was a simulation engine developed from a basic design implemented in the simlib simulation engine. SimPlus was created as a project for a course on simulation during the course of the semester. It is a C++ engine that uses classes and an object oriented programming methodology. The SimPlus engine was developed by Scott Harper, Tom Mancine, and Ryan Scot who were former BGSU CS graduate students. It was developed as a way to improve simlib's user interface. The addition of classes and OOP make SimPlus a much easier to understand engine that would be an excellent engine for study of a simulation course.

Chapter 2

Project Phases

Our project phases were bounded by the few short weeks that we had available to us to spent on the project. Our first phase was to assess the state of SimPlus and evaluate our modifications to the existing code. We found the code to be rushed by the original authors, and we had to make many modifications to bring the code up to a state where we could begin to add to SimPlus. Our original version of SimPlus would not even execute its own demo with out segmentation faults. Our second phase was to develop some internal improvements to SimPlus that would allow future students to develop more robust and interesting simulations with this package. This included the implementation of new distributions as well as new ways to produce random numbers with the package. Our third phase was to develop additions to SimPlus that would help future students to extend SimPlus in to a more robust simulation package. This was implemented by developing a new design for the distribution/RNG system, as well as implementing a XML logging system.

Our project phases were not always easy to define. From the beginning our project was in a constant state of flux. We were continually updating and changing our project requirements based not only on our analysis of the original code, but also ideas put forth by Dr. Rajaei. At times it was difficult to discern what direction our project should go in order to best fulfill user requirements. In the end we came up with something that we felt was the most usable package for future expansion of the package.

Chapter 3

Project Details

The first thing was needed to be performed was an evaluation of the original SimPlus code and find the deficiencies and strong-points therein. From this survey we created a time table with a number of hopeful goals that could be completed in about a weeks time. The initial project goals were to fix encountered bugs, modify the makefile so it was easier to work with it on multiple platforms, add more random number distributions, try to fix the NetRNG generator, add at least one more random generator type, improve the stats functionality, create a calendar queue, add more examples from the book, and plan out a future visualization engine.

However, it was nearly impossible to keep with these goals after work had begin. A variety of things complicated our planning, not the least of which was general sloth on the group members part. However, some of the goals were accomplished, namely creating more distributions and adding a few more code demos, ”fixing” the NetRNG was also tried, but in the end it was decided to abandon any work on it. There were some intrinsic issues with it’s operation that encumbered it’s use. Also, new things were added such as an asynchronous text output class, called LogFile. The stats functionality was also improved, with it now logging the numbers to allow for further analysis, such as standard deviation and variance calculation, along with the possibility to output the numbers into a CSV file.

New distributions were added to the project including the Bernoulli, binomial, Poisson, log-logistic, triangle, and Weibull. We had attempted to add other distributions, but were unable to find the correct formula for the variate function. It was interesting to discover how variates are generated in a DES. Some are much more difficult than others and require numeric sampling methods to get a correct value. Many other do not have a closed form and require approximation to make them work correctly.

A new type of random generator was added to the program. This RNG was called FileRNG, and uses a file of randomness to generate numbers distributed between

(0,1), and can be used as a source of randomness for all the different distributions. It is unclear what this RNG could be useful for, as it's period and randomness is heavily reliant upon the structure in the source file. Also, since it uses the LocalRNG to generate randomness to gather the random bytes from the actual file, it would be difficult to structure the data in such a way to gather tailor-made random data from it. However, it is a novel generator, and could prove to be useful. It should be noted that this RNG is reliant on the source data being in a specific folder and containing a certain title. The directory is called `rand` and the file should be named with a two digit number between 0 and 99 with the extension of `.bin`.

A few new demos were added, and at the time of this writing, the added demos were a M/M/1 queue and a bank simulation with queue jumping for entities in the simulation. These demos further show what is possible to do with the simulation engine, and provide a better basis for learning how to use the engine. Both of the demos are working, and produce correct simulation results.

Also finished was a XML schema document to describe what could be a general-purpose XML trace file, that would be easy to parse and use as fodder for a simulation visualization engine. Included in the schema is the ability for the simulation to describe and define various functions and entity details. However, the integration of the trace file into the engine was well past the scope of this projects limited time. Right now, however, all of the various pieces of support code, such as an asynchronous output class and the XML schema re done, so work could immediately begin on integrating the visualization trace functionality into the engine.

Some of the larger bugs that were fixed with the simulation engine were related to variable initialization and functions that should have been available but were not. The original engine used 0 as a default pointer value, instead of the constant NULL. Though the constant NULL is the same as 0, they are no the same semantically, and proper design practice is to set pointers to NULL. All instances where this would be a problem have been fixed. Also a function was added to the base class for event lists that would return the number of elements in the list. This was an important feature to have been left out and was corrected.

Chapter 4

Results and Analysis

Our results were such that we did finish some objectives, and created some others. Overall the state of SimPlus is better then it was before, and a good foundation for future work has been laid. This we believe was one of the important goals of our project and feel that this goal was met. However, this project was not without it's disappointments and failings, which should be enumerated for the record.

One of the largest disappointments was the failure to delivered what was initially promised in the original time table. Though we fulfilled some of these goals, they all were not finished, and any completed were usually done way past the completion date. This was due to a variety of things, some the groups fault and others due to circumstances out of their control.

The largest contributer to this problem was the lack of good time management. This is a critically important thing to do well on an ambitious project such as this, and it was mishandled. Neither group member was well skilled at time management, and this led to the quick slippage of scheduled events, which snowballed as more and more complications arose and eventually required abandoning the original time table all together for a more limited version that was more more free-form the then already basis timetable.

However, even if the timetable had been kept, there still would have been issues with implementing some of the more advanced ideas though of for this project, namely the visualization trace file. Any type of visualization requires a tight coupling of the simulation engine to the visualization tool. To combat this we were going to use a trace file which would take much of the platform dependence and tight coupling of the visualization tool out of the simulation engine and put it in another tool that could be created independently. Also, this would allow for storage of simulation runs for later study and various other features that may be more complicated with the visualization tool is tightly coupled.

Even still, without a tight coupling of the visualization tool, the trace output functionality must be tightly coupled to the engine. This would require substantial work, since there would be many places that would need to be reworked so that proper output of elements would occur. At the time of this writing the details for the various code changes haven't been worked out, so there are in all likely unforeseen circumstances that may further complicated the matter.

Chapter 5

Proposal for Future Work

5.1 Visualization

The largest and possibly most complex of any future work would be to integrate the trace file and add a visualization tool to the library. This would require extensive modification to the existing library, and plans would need to be drawn up for a cross-platform version of the visualization engine. Right now the library will run on most platforms, and has been tested on Mac OSX, Gentoo Linux, and will compile on BGUNIX (Sun Solaris) without the Image class needed should the statistic lib need to output graphics. It should be possible to get a visualization tool to run on a majority of these platforms, though it may be difficult to do with BGUNIX.

During this project, we created an XML schema that should work as a good, general trace file. The actual schema is attached as an Appendix, and is stored in the `xml` project directory. It consists of a head section that stores all the information about entities in the system such as both visual and simulation intrinsics of the source nodes, server nodes, and other system object. The second section a trace block, contains the pertinent details about each of the events triggered by the system.

For the most part, this is a simple XML schema, and should be rather flexible. The trace file needs to really only record the time of an event, the event type, and what entity handled it. The visualization engine, with information from the header section, should be able to processes these event objects and build a good visualization from it.

Once concern is that the trace file will require great amounts of disk storage, or more storage then needed. If such was the case, an alternate format could be proposed to alleviate this concern, or the trace file broken down into smaller chunks which intern are compressed. File size is one of the drawbacks to using XML, but we think

it acceptable given the scope of the project, the limited time in development, and the size of simulation that this library was designed for.

Time to complete the integration of the trace file into the engine: 5-10 hours. Time to complete the visualization engine: 3-4 weeks for a simple version.

5.2 Calendar Queue

We attempted to get a working implementation of the calendar queue data structure to store the event list in a more efficient manner. Currently the event list is stored in a heap structure, which is O(log N). The calendar queue, however, approximate a O(1) algorithm, which could drastically improve the speed of program execution. However, the calendar queue is a bit tricky to implement, and to keep the schedule from slipping, we abandon work and move on to something that could be completed which would put us back on track. Time to complete: 5-7 hours.

5.3 Log Facilities and Error Handling

Right now, the logging and error handling facilities in SimPlus are limited. Most error handling is achieved through return codes on function calls and other C-like structures. However, since the library is written in C++ it would be better to implement a try-throw-catch exception handling. This would facilitate better error reporting and would allow for giving the simulation the ability to recover from non-fatal errors. However, this will require a bit of work on the programmers part, as not only will a robust exception class need written, but the critical points will need to be found in the code and have the exception handling added. Time to complete: 10-15 hours.

5.4 Expand the Statistics Collection

Since the class now stores the time and data point in a STL map, it is possible to do a number of things with the data. Right now, only calculation of the standard deviation (and by relationship the variance) and mean are calculated. However, it would be possible to add ANOVA, curve fitting, and other statistical functions on the stored data. In addition to this it would be possible to output a graphic that would chart the data vs. time. This could simplify the secondary packages that would need to be available to perform analysis on the data. Another possible graph would be a box plot of the data. Time to complete: 7-10 hours.

5.5 Unit Tests

As SimPlus becomes more complicated, testing and validation of the parts will become more and more important to ensure the consistency and validity of each part of the engine. To do this, we propose implementing a unit testing structure. Hopefully this would speed up development and maintenance of the library, though a bit of work would be required for each new component implemented. Time to complete: 4-5 hours per test.

5.6 Factory Class and OOP Factories

SimPlus is heavily invested in the Factory Design method to create a variety of objects for use in the simulation, not the least of which are the distributions and the random number generators. Right now, these do not follow the factory design pattern per-say, but it would not be complicated to implement a full factory design pattern.

To simplify this we propose the development of a robust factory base class that could provide a concrete and general base for all future factory classes. The base class should allow objects to be registered with it, so the user would be able to add his or her own entities to be created by the base class. We do not think this would difficult to develop, but would require some work and research into the subject. Time to complete: 5-10 hours for the base, and 1-5 hours per derived factory.

5.7 Automake and Autoconf

Since SimPlus is becoming more diverse and complicated, the project should leverage the versatility of the GNU Automake and Autoconf tools. By doing this the project would not only be easier to build across multiple UNIX-like platforms, but it also would be able to add or remove features that any user may or may not want to compile into the program. Also, adding new demos and additional class (functionality) to the library would become far easier then the strange process it is now.

However, this would not be without it's caveats, which would include having to write all of the various files that would be needed to make Automake/Autoconf work. Also, further research would need to be done to adapt the Automake/Autoconf process to the current directory structure. In an initial attempt was made to do such a thing, but was abandon after it was realized that it would take more time then was possible at the current stage in the project. Time to complete: 3-5 hours.

Chapter 6

Conclusion

In conclusion, lots of future work has been laid out that would improve both the quality and functionality of the SimPlus library. Also, during the course of this project, lots of work has been done to further improve the very basics of the SimPlus library, everything from simple code reading issues like setting pointers equal to NULL, up to writing full-scale simulations to show the power of the library. This project did show marked improvement over the old code, and hopefully has provided a solid base and road-map to future work.

Appendix A

XML Trace Schema

File: `xls/trace.xsd`

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="simulation">
<xs:complexType>
<xs:sequence>
    <xs:element name="head">
        <xs:complexType>
        <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="data" type="xs:string"/>
        </xs:sequence>
        <xs:element name="object" maxOccurs="unbounded">
            <xs:complexType>
                <xs:element name="name" type="xs:string"/>
                <xs:element name="var" type="xs:string" maxOccurs="unbounded"/>
            </xs:complexType>
        </xs:element>
        <xs:element name="eventType" maxOccurs="unbounded">
            <xs:complexType>
                <xs:element name="name" type="xs:string"/>
                <xs:element name="action" type="xs:string" maxOccurs="unbounded"/>
            </xs:complexType>
        </xs:element>
        </xs:complexType>
    </xs:element>
```

```
    <xs:element name="trace">
    <xs:complexType>
        <xs:element name="event" maxOccurs="unbounded">
            <xs:complexType>
            <xs:sequence>
                <xs:element name="time" type="xs:decimal"/>
                <xs:element name="target" type="xs:string"/>
            </xs:sequence>
            </xs:complexType>
        </xs:element>
    </xs:complexType>
    </xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

Appendix B

Extension of the Stats Class

B.1 Introduction

This is posted as a followup, as the work was done after the semester concluded, but before grades were posted. The code is documented, though is more then likly not fully debugged. It will **NOT** compile on BGUNIX like the previous library would. This is due to a missing library, libpng, which the image class uses to output the image.

The new methods implemented will do three things. The first and simplest function will dump the data contained in the map to a text file in the format of a standard CSV file. The function takes a single parameter which is a file name. The resulting file should enable the data to be imported into a spreadsheet application for further analysis. The data fields are encased in m̈arks, and the lines are terminated with a ASCII carrage return.

The second method implemented will draw a histogram of the data that is contained in the map. In addition to a file name, it takes the width and height of the image desired and the numerical width of a single box on the histogram. The numerical width or range, is the part of the number-line that each count will encompass. The number of boxes will be defined by the total range of the data divided by the entered range. Each box is drawn in blue and outlined in black. The hashmarks on the x-axis will delinate each box, and the hash marks on the y-axis will delinate the counts. Sadly, due to limitations n the `Image` class, it's impossible to draw words on the image, so there are no data labels.

The third method implemented will draw a line plot of the data as entered, with the time value being along the x-axis and the observation along the y-axis. The hash-marks are calculated from the data, but like the histogram cannot have numerical

labels. All the data points observed will be graphed, so there may be unexpected results if there are lots, and lots of observations.

B.2 Conclusion and Future Work

These have been tested by using random data from a few different RNGs provided in the SimPlus engine. The CSV function should be bug-free. Any issues would more likly be a problem with bad spreadsheet software, then with the acutal writing function.

The histogram function was tested with a variety of random data, and should work as expected. An issue may arise with discrete data and a passed range that is smaller then the smallest range between discrete values. An example of this would be using data from a Binominal generator and generating a histogram with a range of 0.5 which is smaller then the actual range between the points which is 1.0. It is doubtful that the function will crash, but the results are undefined.

The line plot function was tested with random data from a Uniform generator. The graph produced appeared to correspond with the data, so it is assumed to be working. However, like all software, bugs may still exist.

There shouldn't be much future work on these three functions. These also fulfill some of the future work described in the may body of the paper in the Future Work chapter (Chapter 5). However, this doesn't fulfill all of the requirements of that section, so there is still work to be done. The `StatST` class would also benefit from the extension of the Image class to include support for outputting text on images that it renders.