Proceedings of the 2001 Winter Simulation Conference
B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer, eds.

OPEN SOURCE SIMULATION MODELING LANGUAGE (SML)

Richard A. Kilgore

SML Consortium and ThreadTec, Inc.
P. O. Box 7
Chesterfield, MO 63006, U.S.A.

ABSTRACT

The Simulation Modeling Language (SML™) is an open source, web-based, multi-language simulation development project guided by a consortium of industrial, academic and government simulation consultants, practitioners and developers. The vision of an open source simulation software initiative is to leverage the unique communication and distribution opportunities created by the internet to open the development of simulation software to a worldwide community of talented software developers, researchers and modelers. For the simulation community, the open source movement represents an opportunity to improve the quality of common core simulation functions, improve the potential for creating reusable modeling components from those core functions, and improve the ability to merge those components using XML, HLA and other simulation community standards. This paper describes the SML software, the goals of the SML organization and relates the origins, philosophy and procedures of the open source movement to the objectives and needs of the simulation community.

1 INTRODUCTION

Is there a need for an open source simulation language? There is only one way to find out.

On June 1, 2001, the Simulation Modeling Language (SML) initiative began with the goal of answering this question. The SML software is available at no cost and should be available for download in Java, C++, C# and VB.Net versions by the final publication of this paper. The SML software is licensed for unrestricted use for teaching, research or commercial modeling. If you don't like something, you can extend or change SML any way you like as long as you agree to upload and share your extension or change with the SML community. The final SML site was not determined at the time this paper was submitted so please check threadtec.com, sml.info or sml.sourceforge.net for current SML versions and download information.

While that information might be a sufficient introduction for many people, those that continue to read this paper are encouraged to visit the SML web site or attend the conference session for the most current status of the SML project before forming a first impression based on this paper. Unlike proprietary simulation software, even if that first impression is not positive, then you have every opportunity to voice your opinion or even roll up your programming sleeves to get involved and show the world how to do it better.

The open source movement is a revolutionary perspective on how software should be created (Pavlicek 2000). While the movement is most often associated with the development of the Linux operating system and related software projects (Linux 2001), there are open-source initiatives throughout the software industry and new projects are constantly emerging. Some projects are immediately successful while others start slowly. One popular repository of open source software (Open Source Development Network 2001), lists over 23,000 open-

Astronomy/Physics	16
Software Engineering/Testing	14
Computer-Aided Design/Graphics	12
Games/Entertainment/Training	12
Chemistry/Molecular	9
Math/Continuous/Equations	**6**
Network/Computer Systems Simulation	**6**
Biology/Geneology	5
Electrical engineering	5
Discrete-Event/Process Simulation	**4**
Petri Nets/Agent-based	**2**
Robotics	2
Air traffic/Vehicle control	2
Financial	2
Simulation Experimentation	**1**
Random Numbers /Monte Carlo	**1**
Database	1

Figure 1: Categories of Open Source Simulation Software Projects at freshmeat.net (July, 2001)

source projects. As shown in Figure 1, there are presently 100 open source projects with a simulation theme including four discrete-event simulation projects (Nutaro 2001; Realiant Systems 2001; Fisher 2001; Varga 2001) and one supporting distributed batch mode simulations (Belding 2001). So what will distinguish SML? In the world of open source, truth is cited as the most valuable of all virtues, so the truthful answer is that SML or any similar open source project will only be as good as the people that get involved in its design and implementation (PITAC 2000).

As stated earlier, this paper and the corresponding conference session serve primarily as an announcement of the SML project and an invitation to participate in the project. Section 2 is an introduction to the open-source software development process as it relates to simulation software. Section 3 describes open source licensing and a proposed modification to distinguish open source simulation and proprietary modeling. Section 4 briefly defines the initial SML objectives and presents results from the first phase of the SML/Java demonstration prototype.

2 OPEN MIND, OPEN SOURCE

Every revolution seems to have a manifesto and the declaration of programming independence of the open source movement seems to be *The Cathedral and the Bazaar* (Raymond 1999). The paper contrasts the traditional software development *cathedral* in which a small group of cloistered developers design, code and deliver a finished product with the *bazaar* in which there is concurrent and chaotic interaction between users and developers. In the open source *bazaar*, the user community becomes the de facto owner of the software and all are invited to participate. Although there may be user feedback in the *cathedral* design process, only those inside the developing organization or group actually get to participate in the coding process. The organizational structure in the *bazaar* of open source development is more democratic in the sense that the only qualifications for participation are an understanding of the problem and the skill and perseverance to pursue a solution through testing and implementation. The Internet facilitates open source development by providing the mechanism for economic and timely communication and distribution of new source code and documentation.

Let's begin by discussing SML and simulation as it relates to a few modifications of Raymond's principles for open source development summarized below in Figure 2.

*1. Every good work of **simulation** software starts by scratching a **modeler's** personal itch.*

Everyone who builds simulation models defines what simulation software *should* do based on what that modeler *has* done. The more models you have done and the more simulation software you have used, the greater the personal itch becomes to improve simulation software to solve those modeling challenges you have experienced. SML is designed to make it easier for every modeler to scratch their own "personal" itch as it develops during their career by providing a modular design, source code

1. Every good work of software starts by scratching a developer's personal itch.
2. Good programmers know what to write. Great ones know what to rewrite (and reuse).
3. Plan to throw one away; you will, anyhow.
4. If you have the right attitude, interesting problems will find you.
5. When you lose interest in a program, your last duty to it is to hand it off to a competent successor.
6. Treating your users as co-developers is your least-hassle route to rapid code improvement and effective debugging.
7. Release early. Release often. And listen to your customers.
8. Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone.
9. Smart data structures and dumb code works a lot better than the other way around.
10. If you treat your beta-testers as if they're your most valuable resource, they will respond by becoming your most valuable resource.
11. The next best thing to having good ideas is recognizing good ideas from your users. Sometimes the latter is better.
12. Often, the most striking and innovative solutions come from realizing that your concept of the problem was wrong.
13. Perfection in design is achieved not when there is nothing more to add, but rather when there is nothing more to take away.
14. Any tool should be useful in the expected way, but a truly great tool lends itself to uses you never expected.
15. When writing gateway software of any kind, take pains to disturb the data stream as little as possible -- and *never* throw away information unless the recipient forces you to!
16. When your language is nowhere near Turing-complete, syntactic sugar can be your friend.
17. A security system is only as secure as its secret. Beware of pseudo-secrets.
18. One can test, debug and improve in bazaar style, but it would be very hard to *originate* a project in bazaar mode.

Figure 2: Open Source Guidelines from *The Cathedral and the Bazaar*

access and licensing permission to change the tool as needed when needed.

*2. Good **simulation** programmers know what to write. Great ones know what to rewrite (and **make it easier for modelers** to reuse).*

Scratching the personal itch may be sufficient motivation for an individual modeler to get involved in community-based, open source development. But scratching the simulation industry-wide itch for reusability is the global benefit that SML and open source simulation can provide. Consider all of the simulation code thrown away during the last 30 years. Consider all of the simulation code that is never seen by anyone other than the modeler. An open source initiative likes SML provides an opportunity for modelers to learn from the past and learn from each other and invest in reusable code.

*7. Release **models** early. Release **models** often. And listen to your **modeling** customers.*

Although we all make errors, it is particularly difficult for programmers and modelers to expose their errors in public. But it may be a much more productive experience for the modeler and much more educational experience for the modeling community to see the process that other modelers use to develop simulation code and not simply the final results. As mentioned earlier, good code and good models require rewrite and exposing the errors and ideas earlier on may make the rewriting experience more productive than hiding these problems for fear of public ridicule. Simulation programmers all have different perspectives, different experiences and different customers, so SML will be successful only if it creates an environment where open discussion is encouraged. The best way to encourage discussion is to encourage listening and consideration of all incoming suggestions regardless of the qualifications of the source of the suggestions or the quality of the partially completed solution.

*18. One can test, debug and improve a **simulation language** in bazaar style, but it would be very hard to originate a **simulation language** project in bazaar mode.*

It is much easier to encourage discussion if you have a running prototype to discuss so the SML project site was developed with working code as shown in Figure 3. The initial SML prototype was a Java-based, discrete-event, process-oriented class library based on an entity-thread data structure. But ideally, the same SML specification will support C++, C#, VB.Net and any other implementations where a supporting community of developers emerges. All SML decisions cannot be made in committee, and inevitable disagreements on what is "best" are to be expected. But with a sound fundamental core

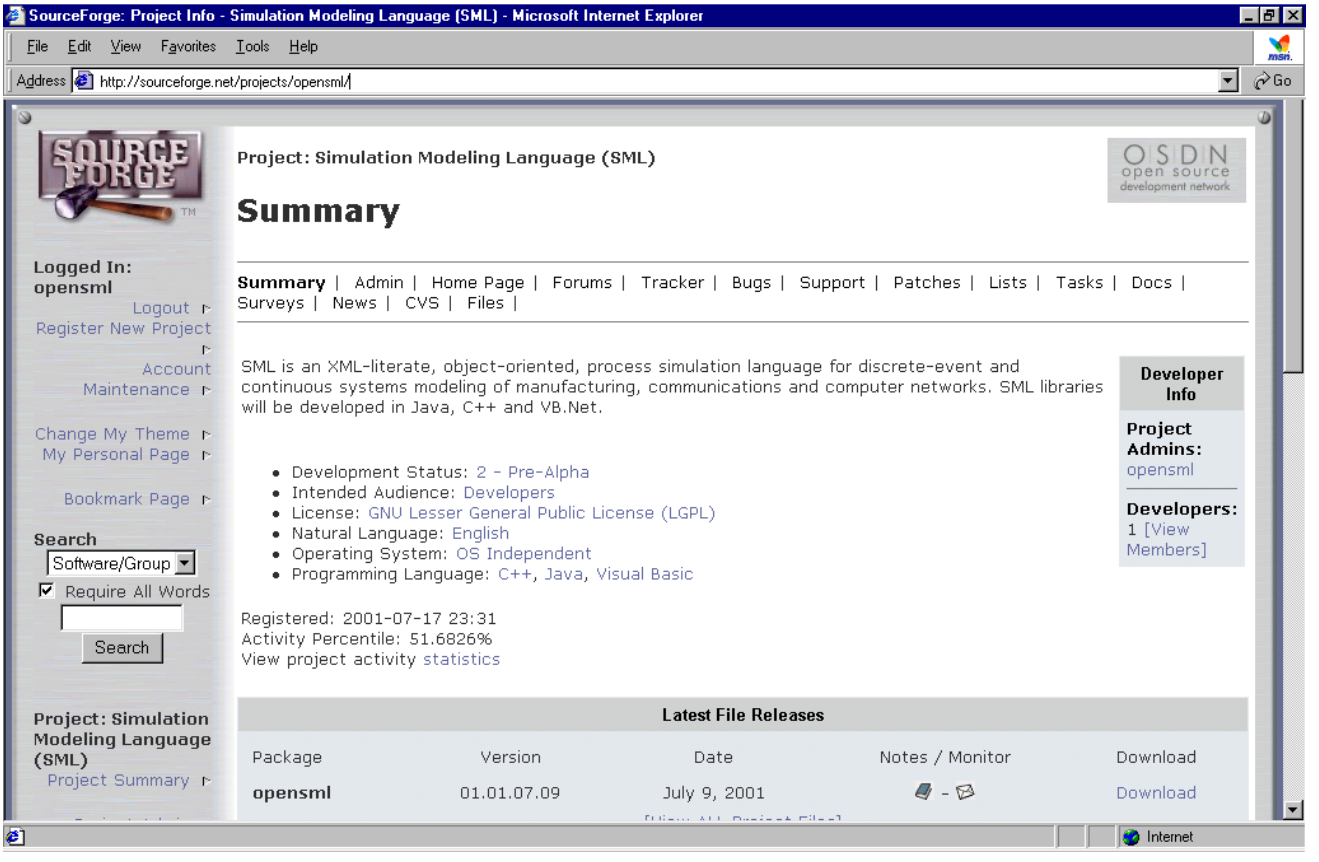

Figure 3: SML/Java Open Source Repository at `http://sourceforge.net/projects/opensml/`

library of extendible functions, SML participants will be unlimited in their ability to build alternative implementations. For example, if you don't like the queue object, you have access to the source code to develop your own variation of the queue object or the ability to extend the queue object to add additional properties and methods. All that is required from a licensing perspective is that the modification or extension is published for all other SML participants to share.

3 SML LICENSING AND "COOPETITION"

The issue of what to share and how to share it is an important issue for distinguishing between free SML simulation code and proprietary modeling code. The present plan for SML is to distribute simulation language source code under a modified Lesser/Library General Public License (Free Software Foundation, 2001) that ends where the SML simulation language ends and the SML-based simulation model starts. Normally, all extensions and modifications of LGPL licensed software must be distributed under the same LGPL license under which the software was acquired. Obviously, this restriction cannot be applied to software that uses the SML code to create a specific model.

For example, the current SML code includes a linked list queue object that holds an indexed list of SML entity objects. The SML class includes a qadd( ) method that adds an entity to the end of a queue. If an SML users needs a function that ranks and re-sorts the list based on one or more properties, the user is allowed under LGPL to add the additional capability to the language. But the LGPL required that the user share that improvement by returning the revised code to the SML repository. Some might take the position that the improvement is a "modeling" function that cannot be shared because the names and types of properties and ranking rules used for the re-sort are proprietary to the modeling application. Pproper SML sharing principles would require that the user comply by depositing a generic or example version of the method that does not contain proprietary property names or ranking rules. Can you write a license that defines such behavior? Can you enforce such a license? Open source opponents and their business managers say no. Open source proponents say maybe not, but the benefits of SML will still outweigh the risks and costs from those who cheat or otherwise abuse the mutual trust and honor that the open source system requires.

*The economic vision behind SML will be to allow users to **compete** in a marketplace of models and modeling components based on SML, but to **cooperate** in the development of compatible, extendible SML objects and methods which support those models. The resulting "**coopetition**" among simulation companies and professionals will be better for the long term viability and profitability of the simulation industry than the current system of incompatible products and lack of standards.*

This business model is neither pure communism nor pure capitalism. It is similar to the industrial cooperation that allows automotive companies to standardize on lug nuts but compete on car models and television manufacturers to cooperate on broadcast format but compete on receiver hardware. It is an imperfect and delicate alliance amongst opponents that is viewed in advance as wishful folly and viewed in retrospect as insightful wisdom. Sometimes *coopetition* happens because governments decree that change should occur, but more often *coopetition* happens because influential and powerful users decree that the change should occur. Sometimes it happens long after it is initially proposed. Sometimes it happens immediately. The final movement toward simulation language *coopetition* may not be SML-based and it may not be now. But it will eventually happen because someone will someday expose the value of wasted simulation software created to date and publish that figure to allow influential users realize they have the economic justification to demand something different.

4 SML – READABLE, MODULAR, EXTENDIBLE

Open source development means that the SML community or consortium will ultimately define what SML will be, not the author of this paper or the initial authors of the software. Ideally, SML will evolve based on the skills, passion, requirements and resources of the participants and their clients. So the following specification is simply one small step down one trail of the multi-year, multi-language, multi-application journey that open source initiatives like SML propose. Consequently, a positive outcome of this section will be a bold and passionate critique of everything written from this point on by people who are also bold and passionate enough to put their improvement out there for additional critique.

The mission of SML is to produce reusable simulation software at both the simulation source code and modeling source code levels. Reusability requires at a minimum that the code be **readable**, **modular** and **extendible**. Contrary to most simulation products, SML will sacrifice performance to achieve reusability.

Readability means that the target audience for the code is the closer to the first-time reader with limited programming background than to the experienced hacker. Most simulation practitioners are not computer science graduates and are capable, but not expert programmers. The goal of SML readability is to encourage participation by part time programmers interested in quickly finding and modifying without extensive debugging and testing.

Modularity is related to readability in that a part time developer can make a change to the source code or replace an entire SML module without having to understand or modify large amounts of SML source code.

Extendibility means that SML is designed to be easily modified and repackaged for specific applications. As mentioned previously, simulation languages are usually biased towards a particular target application based on the experiences and anticipated needs of the modeler or developer. But if properly designed, SML methods for manufacturing system simulation and communication systems simulation may appear unique at the modeling level, but extend identical methods at the core simulation library level.

The best method of achieving the combination of readable, modular and extendible that SML desires is through object-oriented development. Even though the original SML prototype is Java-based, any object-oriented language is a potential candidate for an SML community. As shown in Figure 4, it would seem possible that a common SML core specification will emerge for utility functions (e.g. calendar, distribution generators, statistical functions.) so that implementation of these functions can be as similar as possible in all of the supported languages. And, ideally, a common process-oriented language (e.g. qadd, waituntil, qremove, seize, delay, release) will emerge for each SML entity class definition. While it may not be possible for identical implementations in each of the SML language communities, the common object-oriented, process-oriented modeling design patterns may be sufficiently similar so that models and methods developed in one base language can be easily migrated to the other supported languages.

The entity-object-thread design pattern used in the Java-based Silk simulation language is implemented in the

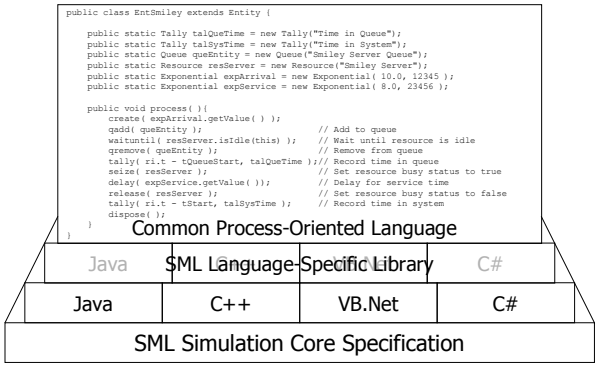

Figure 4: SML Architecture

initial SML/Java prototype (Kilgore et al. 1998). In this pattern, each simulation object defines its process-oriented behavior description in a common *process( )* method that is started when the entity-thread is started. The entity-thread is suspended during a time-based or status-based delay and resumed when the delay is complete or a system status delay allows the entity to proceed.

The SML/Java prototype demonstration code for a single-server queuing system is shown in Figure 5. Color is used in electronic versions of this paper to distinguish Java keywords (blue), SML keywords (red), comments (green) and user-defined identifiers (black). The example code follows the SML readability objective and should be fairly self-explanatory to those familiar with object-oriented syntax of Java or C++. It is explained at the SML web site and elsewhere in more detail (Kilgore 2001). One unusual variable in the example code is the *ri.t* variable representing that value of simulation time *t* for each instance of run *ri*. Since SML/Java was written to

```
public class EntSmiley extends Entity {

    public static Tally talQueTime = new Tally("Time in Queue");
    public static Tally talSysTime = new Tally("Time in System");
    public static Queue queEntity = new Queue("Smiley Server Queue");
    public static Resource resServer = new Resource("Smiley Server");
    public static Exponential expArrival = new Exponential( 10.0, 12345 );
    public static Exponential expService = new Exponential( 8.0, 23456 );

    public void process( ){
        create( expArrival.getValue( ) );
        qadd( queEntity );                          // Add to queue
        waituntil( resServer.isIdle(this) );        // Wait until resource is idle
        qremove( queEntity );                       // Remove from queue
        tally( ri.t - tQueueStart, talQueTime );// Record time in queue
        seize( resServer );                         // Set resource busy status to true
        delay( expService.getValue( ));             // Delay for service time
        release( resServer );                       // Set resource busy status to false
        tally( ri.t - tStart, talSysTime );         // Record time in system
        dispose( );
    }
}
```

Figure 5: SML/Java Single Server Model

facilitate distributed execution, the variable *ri.t* represents simulation time for an instance of the run control class which allows a separate simulation clock for each replication on each processor.

Because SML is an open architecture and an open-source development process, other design patterns may emerge which are superior in general or superior for a particular class of applications. In the Linux community the rule that decides what is superior is "let the best code win". In the simulation community, this rule may need to be modified to "let the best code *for this application* win". The hope is than even application-specific code be developed in a style modular enough to extend rather than replace an alternative implementation. Most important for the simulation community is that there will be a common forum and laboratory where these issues can be expressed, debated and tested rather than a continuation of proprietary simulation camps that don't speak the same language.

The output of a single replication of the SML single server model using the SML demonstration applet is shown in Figure 5. A debug window echoes the line by line execution of the model. Other windows show the statistical summaries for entity, queue and resource statistics. In open-source projects, it is common practice to always maintain various executable versions of the software for new users evaluating the project and a development version containing untested or un-debugged improvements. All of the SML/Java builds are archived so that developers with alternative ideas can return to a build where their ideas can be most easily implemented.

5 SUMMARY

Simulation is not a large industry compared to other industries where open source initiatives have been successful. There are only a handful of individuals with full time responsibilities writing simulation code. The initial development of the simulation software industry was supported primarily through academic projects that were spun off into small commercial companies with a 5-15 year life span. Proprietary simulation code has created a history of wasteful rewrites and limited reusability of simulation models.

The SML open source simulation project will evaluate a potential evolution in the simulation software development model. The goal of cooperating in simulation and competing in modeling software will retain the economic incentive necessary to support commercial simulation companies. But the existence of common, powerful and inexpensive object-oriented languages and an instantaneous, internet-based worldwide communication means that simulation software development need not be a product of proprietary, closed-source, vendor-based licensing. Should SML be as successful as other open source initiatives, readers may find that the state of the SML organization and SML products may be quite different by the time they read this paper and the SML web-sites will contain the current status of the project. More important than the near-term popularity of the language will be the steady, long-term progress toward libraries of easily extendible and easily reusable simulation code.

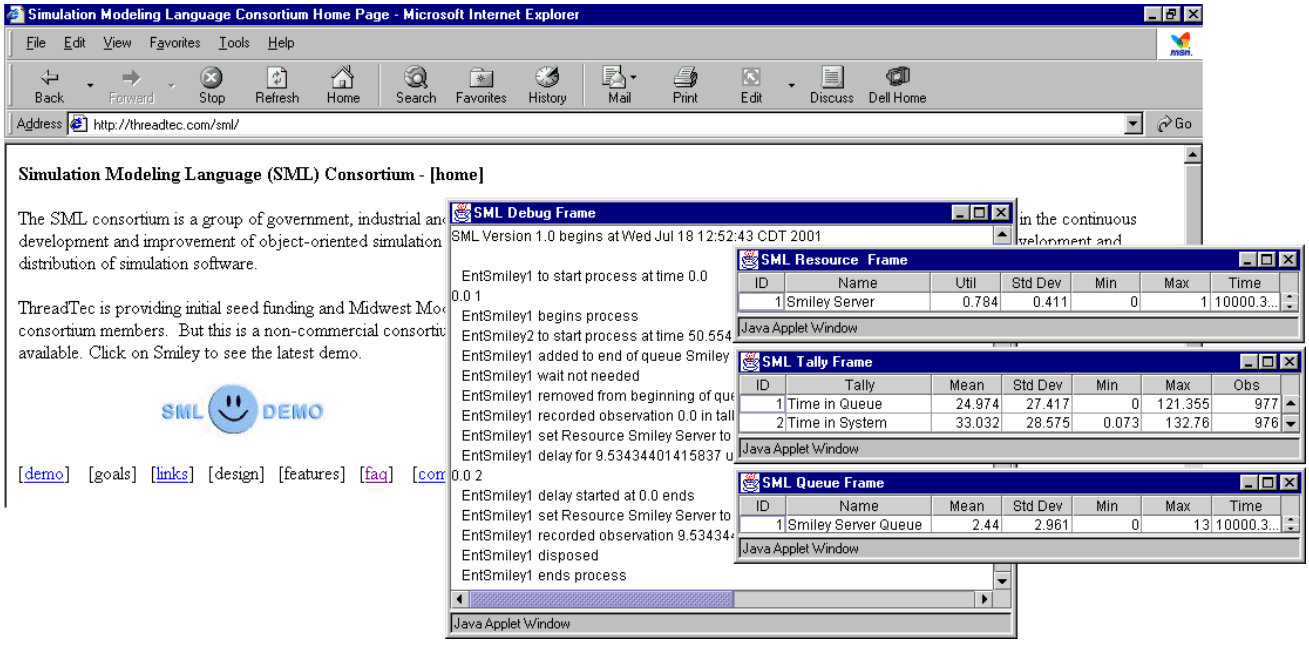

Figure 6: SML/Java Demonstration Applet

REFERENCES

Belding, Theodore C. 2001. *Drone*. University of Michigan Center for the Study of Complex Systems. drone.sourceforge.net.

Collier, A. 2001. *rgen*. rgen.sourceforge.net.

CPN Group, University of Aarhus, Denmark. 2001. *Design/CPN*, www.daimi,au.dk/designCPN

Fisher, J., K. Ahrens and D. Witaszek. 2001. *ODEM*. odem.sourceforge.net.

Free Software Foundation, 2001. www.opensource.org.

Kilgore, R. A. 2001. Open-Source SML and Silk for Java-Based, Object-Oriented Simulation. In *Proceedings of the 2001 Winter Simulation Conference,* ed., B. Peters, J. Smith. Piscataway, NJ: Institute of Electrical and Electronics Engineers.

Kilgore, R. A., Healy, K. J. and Kleindorfer, G. B. 1998. The future of Java-based simulation. *Proceedings of the 1998 Winter Simulation Conference Proceedings*, ed. D. J. Medeiros, E. F. Watson, J. S. Carson, M. S. Manivannan. Piscataway, NJ: Institute of Electrical and Electronics Engineers.

Linux, 2001. www.linux.org.

Nutaro, Jim. 2001 *adevs*, www.ece.Arizona.edu/~nutaro.

Open Source Development Network, 2001. www.osdn.org.

Pavlicek, Russell C. 2000. *Embracing Insanity: Open Source Software Development*. Sams Publishing, Indianapolis, IN.

Realiant Systems. 2001. *FROGS*. www.carbonkernel.org.

Raymond, Eric 1999. *The Cathedral & The Bazaar*. O'Reilly, Cambridge, MA.

PITAC, The Presidents Information Technology Advisory Committee: Panel on Open Source Software for High End Computing, 2000. *Developing Open Source Software to Advance High End Computing*. National Coordination Office for Computing, Information and Communications, Arlington, VA.

Varga, Andras. 2001. *OMNet++*. www.hit.bme.hu/phd/vargaa/omnet.

AUTHOR BIOGRAPHY

RICHARD A. KILGORE is a co-author of SML and the Silk language and a consultant in the development of industrial simulation and scheduling solutions. Dr. Kilgore is a founding member of the open-source Simulation Modeling Language (SML) Consortium. He has over 20 years of experience as a modeling consultant to Fortune 500 firms in a variety of industries with a variety of simulation and scheduling tools. He received his B.B.A. and M.B.A degrees from Ohio University and Ph.D. in Management Science from the Pennsylvania State University. Formerly, he was a capacity-planning analyst with Ford Motor Co. and Vice-President of Products for Systems Modeling Corp. His e-mail address is <kilgore@threadtec.com>.