Proceedings of the 2002 Winter Simulation Conference
E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, eds.

MULTI-LANGUAGE, OPEN-SOURCE MODELING USING THE MICROSOFT .NET ARCHITECTURE

Richard A. Kilgore

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

ABSTRACT

This presentation reports on the opportunities and limitations Microsoft .Net architecture for supporting the development of a common, open-source, multi-language platform for simulation software support. While the paper supporting the presentation focuses on the underlying foundation within the .Net architecture, the conference presentation represents an important milestone in the OpenSML project corresponding to the first release of a common library supporting the C#, VB.Net and Java/J# languages.

1 INTRODUCTION

The software industry has gone through a number of rapid transformations in the last 10 years, primarily led by the development of the internet. While it was once sufficient to develop a good business software application, now it is necessary to develop good internet-capable business software applications. To integrate support for internet capabilities, a number of changes are taking place in basic programming languages and software development tools. These changes offer the opportunity for yet another rewrite, yet another redesign and yet another rethinking about the future of simulation software.

The basic premise of the rethinking represented in this paper is that this redesign and rewrite may be an opportunity for a common, open-source, multi-language approach to simulation software development. The fundamental trends which create this opportunity include:

- the convenience of internet-based collaboration
- the success of the open-source/consortium model
- the acceptance of object-oriented programming
- the movement of performance responsibility (execution speed) from software to hardware
- the continuing lack of reusability, interoperability and discipline in simulation software development

While considering this last trend, also consider the enormous amount of academic, commercial and military simulation software created over the last twenty years and wonder at the number of times the simulation software wheel has been reinvented and re-implemented. And while each implementation builds on lessons learned from previous developments, the resulting solutions continue to be proprietary solutions with rarely more than one-time application.

This is one of three papers presented at this conference that discuss related aspects of emerging general software standards that further the reusability and interoperability of commercial software. The unifying theme in these presentations is the emergence of various standards in software development and the opportunity these standards have for similar standards for simulation software development. The other papers look at topics of design patterns and web services in the search for reusability and interoperability (Kilgore 2002). This paper examines the role that *open source* and *common language runtimes* could play in the evolution of standards for interoperable software applications and the opportunities this creates for standards for interoperable simulation applications and components. As shown in Figure 1, the OpenSML initiative is intended as a proving ground where these topics turn from discussions to implementation.

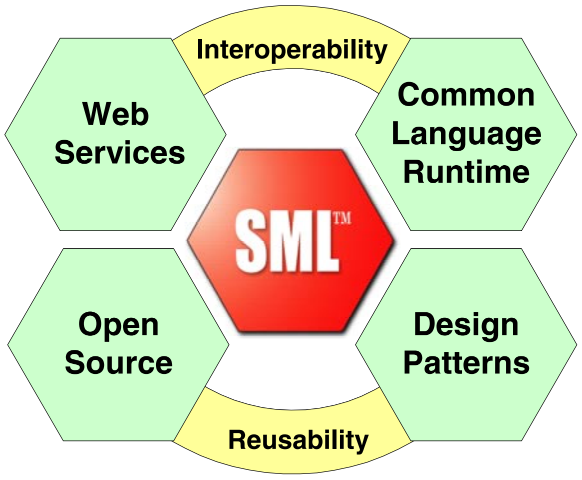


Figure 1: Simulation Interoperability and Reusability

Section 2 of this paper is an introduction to the Microsoft .Net common language runtime and multi-language interoperability between C#, VB.Net and Java in OpenSML and Section 3 is a review of the OpenSML open source development model. Section 4 is an overview of some OpenSML design goals. One source of the current status of the OpenSML software is available at http://www.sourcforge.net/opensml.

2 .NET COMMON LANGUAGE RUNTIME

The Microsoft .Net (Microsoft 2002) architecture consists of many related initiatives, but the three basic elements are:

- *.Net Framework* as a unifying replacement to the Windows API on the desktop
- .Net Enterprise Servers as a unifying replacement to DNA and other dissimilar server-side software
- .Net integrated Web Services and ASP.Net development support tools

The core of the *.Net Framework* is:

- the .Net Framework classes
- the .Net virtual machine known as the Common Language Runtime (CLR)
- the .Net languages including C#, VB.Net, C++.Net and J#

The feature of the Common Language Runtime most relevant to the OpenSML project is the cross-language integration that can be achieved within this architecture. All .Net languages compile to a common Microsoft Intermediate Language (MSIL or IL), which is then compiled and executed by the CLR. For example, the C# .Net class for "Hello World":

```
using System;

public class Module{
    public static void Main (String[] args) {
        Console.WriteLine("Hello World");
    }
}
```

is actually implemented in the following IL code:

```
.method public static void  Main(string[]args)
{
 .entrypoint
 .maxstack  8
 ldstr "Hello World"
 call void System.Console::WriteLine(string)
 ret
}
```

which is very similar (but not completely identical) to the corresponding IL file that would be produced by the following VB.Net code:

```
Imports System

Public Class HelloWorldVB
    Shared Sub Main()
        Console.WriteLine("Hello World")
    End Sub
End Class
```

In addition to the underling IL instructions, a .Net program unit, called an *assembly*, contains the full and complete metadata that fully describes each and every type (class, structure, etc.) in this assembly and every externally referenced assembly. Whereas previous COM components allowed cross-language compatibility in binary code at runtime, IL allows cross-language references at compile time.

At execution time, the IL instructions are compiled into meaningful CPU instructions on the fly. As the IL instructions are compiled into machine code, .Net will cache the results in memory. Subsequent calls made to methods will use the cached instructions eliminating the perform-

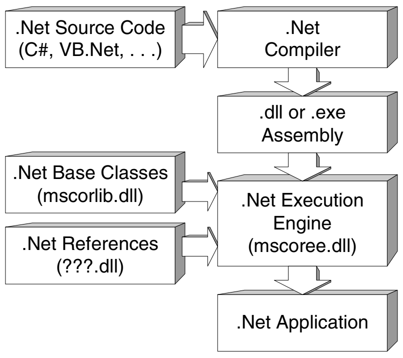

.Net Source Code (C#, VB.Net, . . .)
.Net Compiler
.dll or .exe Assembly
.Net Base Classes (mscorlib.dll)
.Net References (???.dll)
.Net Execution Engine (mscoree.dll)
.Net Application

Figure 2: .Net Compilation and Execution Overview

ance penalty normally associated with interpreted software. As shown in Figure 2, the actual core execution engine within .Net framework is the mscoreee.dll library and the base class library is mscorlib.dll.

In order to qualify as a .Net CLR language, a language must adhere to certain standards known as the Common Language Specification (CLS). Compatibility with the CLS standards insures that your language will be completely interoperable with others programming to the CLS no matter what language is being used. As expected, most CLS rules relate to definitions and parameters for public classes and methods and not to the internal implementation of a .Net type. Ideally, the CLS specification has the potential for cross platform execution beyond Windows. The Mono project (Ximian 2002) is an open source, Linux-based version of the NET development platform incorporating key .NET compliant components, including a C#

compiler, a Common Language Runtime compiler and class libraries.

3 OPEN SOURCE AND OpenSML

Just as the CLS specification enables interoperability between programming languages, the OpenSML (Open Simulation Modeling Language) is a working title for a project with the objective of enabling interoperability between simulation libraries (Figure 3). As an open-source project, the details of how it does that exactly is up to those who participate in the design and implementation. The initial scope of the project is to define an "extendible standard" for common implementation in compatible .Net object-oriented programming languages.

Open source development means that the OpenSML community or consortium will ultimately define what OpenSML will be, not the author of this paper or the initial authors of the software. Based on the experiences of the initial year, it is evident that a more complete starting point is necessary to encourage participation. But ideally, OpenSML will evolve based on the skills, passion, requirements and resources of the participants and their clients. The current specification is simply one small step of a multi-year, multi-language, multi-application journey that open source initiatives like OpenSML experience. 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.

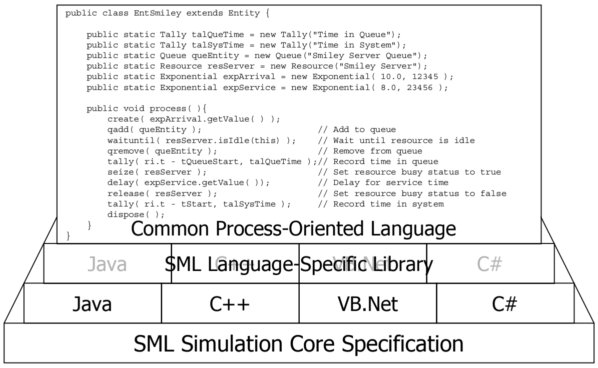
The mission of SML is to produce reusable simulation software at both the simulation source code and modeling source code levels. Reusability requires at a minimum that the code be **readable**, **modular, interoperable** and **extendible**. Note that performance is not directly addressed in the mission statement implying that OpenSML will sacrifice performance to achieve reusability.

Readability means that the target audience for the code is 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 OpenSML 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 OpenSML module without having to understand or modify large amounts of OpenSML source code.

Interoperability is related specifically to the degree to which standards can be used to allow independent executing simulations to communicate through web services or other custom programmed interaction.

Extendibility means that OpenSML is designed to be easily modified and repackaged for specific applications.



```
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( );
    }
}
```

Common Process-Oriented Language
Java
C++
VB.Net
C#
SML Language-Specific Library
Java
C++
VB.Net
C#
SML Simulation Core Specification

Figure 3: OpenSML Architecture

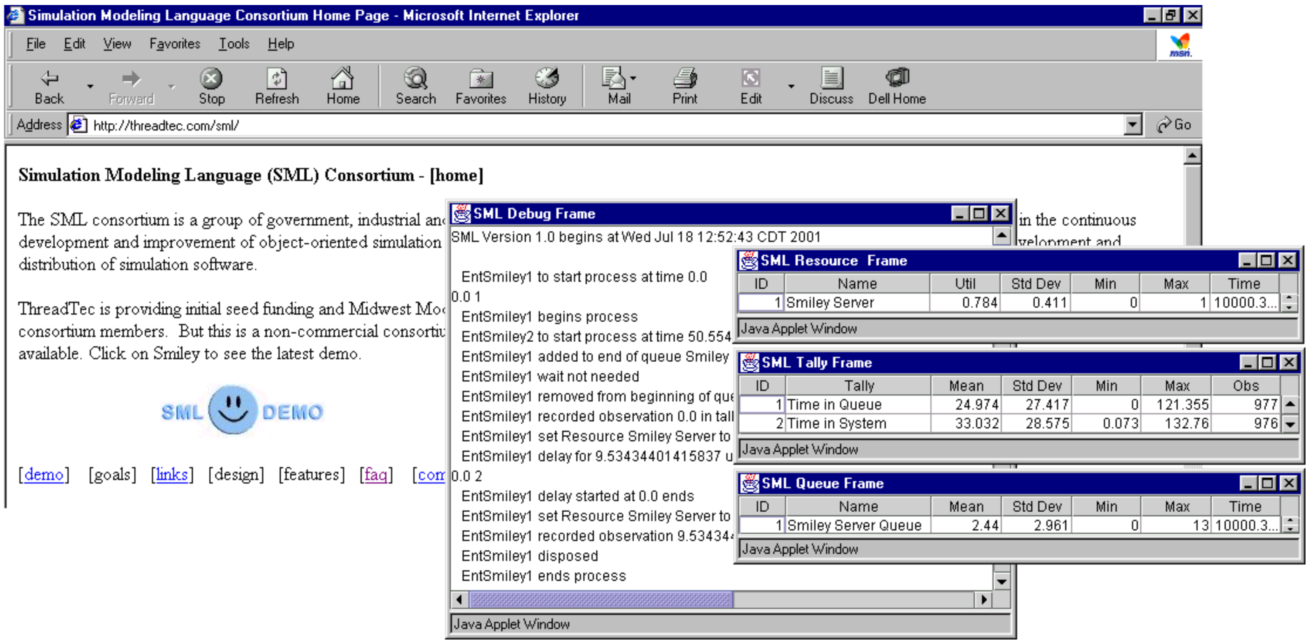
As mentioned previously, simulation languages are usually biased towards a particular target application based on the experiences and anticipated needs of the modeler or developer. But if properly designed, OpenSML methods for manufacturing system simulation and communication systems simulation may appear unique at the modeling level, but extend identical methods at the core simulation library level.

The original plan for OpenSML is to distribute simulation language source code under a modified Lesser/Library General Public License (Free Software Foundation, 2001) that ends where the OpenSML simulation language ends and the OpenSML-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 OpenSML code includes a linked list queue object that holds an indexed list of OpenSML entity objects. The OpenSML class includes a qadd( ) method that adds an entity to the end of a queue. If an OpenSML 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 OpenSML 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. Proper OpenSML 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.

4 OpenSML DESIGN GOALS

The overall design goal of OpenSML is to improve the *validity* and the *effectiveness* of the use of simulation models in the support of decision-making. *Validity* in this sense is a long term improvement in that the starting point of simula-


Simulation Modeling Language Consortium Home Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit Discuss Dell Home
Address http://threadtec.com/sml/
Go
SML DEMO

Simulation Modeling Language (SML) Consortium - [home]

The SML consortium is a group of government, industrial an
development and improvement of object-oriented simulation
distribution of simulation software.

in the continuous
velopment and

ThreadTec is providing initial seed funding and Midwest Mo
consortium members. But this is a non-commercial consorti
available. Click on Smiley to see the latest demo.

[demo] [goals] [links] [design] [features] [faq] [com

SML Debug Frame

```
SML Version 1.0 begins at Wed Jul 18 12:52:43 CDT 2001

 EntSmiley1 to start process at time 0.0
0.0 1
 EntSmiley1 begins process
 EntSmiley2 to start process at time 50.554
 EntSmiley1 added to end of queue Smiley
 EntSmiley1 wait not needed
 EntSmiley1 removed from beginning of que
 EntSmiley1 recorded observation 0.0 in tall
 EntSmiley1 set Resource Smiley Server to
 EntSmiley1 delay for 9.53434401415837 u
0.0 2
 EntSmiley1 delay started at 0.0 ends
 EntSmiley1 set Resource Smiley Server to
 EntSmiley1 recorded observation 9.53434
 EntSmiley1 disposed
 EntSmiley1 ends process
```

Java Applet Window

SML Resource Frame

ID	Name	Util	Std Dev	Min	Max	Time
1	Smiley Server	0.784	0.411	0	1	10000.3...

Java Applet Window

SML Tally Frame

ID	Tally	Mean	Std Dev	Min	Max	Obs
1	Time in Queue	24.974	27.417	0	121.355	977
2	Time in System	33.032	28.575	0.073	132.76	976

Java Applet Window

SML Queue Frame

ID	Name	Mean	Std Dev	Min	Max	Time
1	Smiley Server Queue	2.44	2.961	0	13	10000.3...

Java Applet Window

Figure 4: OpenSML Demonstration

tion models is at a more advanced position because of the greater potential of reuse of previously created simulation code. OpenSML standards should cover not only rules for the creation of simulation models, but also methods for archiving and reassembling models components. *Effectiveness* is improved through flexibility and performance of simulation related functions (data input generation, scenario specification, debugging, execution, output analysis). Even though the validity of the simulation model sets the upper bound on the potential value of a simulation study, the value of simulation software is often measured on the effectiveness of the software for the performance of these functions.

Usability refers to characteristics of simulation software that support the ability to express model behavior consistent with system descriptions. This is a obviously a subjective goal in that the choice of expression is a personal preference usually based on the users experience. But if there is one design for readability that transcends syntax, grammar and style, it is that each statement in OpenSML simulation code should model one unit of system behavior. Even when the underlying programming code is used to augment the simulation code, the ability to wrap the detail in a readable method is an essential design goal.

Because OpenSML is multi-lingual, only the common denominator of core statements and keywords available in all languages should be used. For example, the .Net architecture support a Java variant syntax called J#. In order to maintain maximum interoperability within the .Net, the OpenSML design only aims for implementation in J#.

5 SUMMARY

Probably the greatest challenge to the success of OpenSML is cultural. The simulation industry is not large relative to other industries where open source initiatives have been successful. There are a limited number of individuals with full time responsibilities writing simulation code and the payoff from involvement in OpenSML is much more intangible and uncertain that the payoff from continuing the status quo. Of those simulation professionals with sufficient programming background to consider software design issues, few have ever worked in collaboration over long periods. Nevertheless, as shown in Figure 4, demonstration models and OpenSML source code continues to steadily develop toward a generic, multi-language standard.

Nevertheless, the OpenSML open source simulation project continues to evolve and continues to leverage emerging technologies and standards and apply them to simulation software development model. The goal of cooperating in simulation software, but competing in modeling software should retain the economic incentive necessary to support commercial simulation companies. 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.

REFERENCES

Free Software Foundation, 2001. Available online via <www.opensource.org>. [accessed July 1, 2002].

Kilgore, R. A. 2002. Object-Oriented Simulation with Java, Silk and OpenSML .Net languages. In *Proceedings of the 2002 Winter Simulation Conference,* ed., E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes. Piscataway, NJ: Institute of Electrical and Electronics Engineers.

Kilgore, R. A. 2002. Simulation Web Services with .Net Technologies. In *Proceedings of the 2002 Winter Simulation Conference,* ed., E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes. Piscataway, NJ: Institute of Electrical and Electronics Engineers.

Microsoft 2002. The .Net Platform for XML Web Services, Official Site. Available online via <www.microsoft.com/net/> [accessed April 16, 2002].

Ximian Corp. 2002. Mono: Open Source implementation of the .NET Development Framework. Available online via <www.go-mono.net/index.html> [accessed July 25, 2002].

AUTHOR BIOGRAPHIES

RICHARD A. KILGORE is a consultant in the development of industrial simulation and scheduling solutions and President of ThreadTec, Inc., the distributor of the Java-based Silk simulation language. Dr. Kilgore 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>.