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

SIMULATION WEB SERVICES WITH .NET TECHNOLOGIES

Richard A. Kilgore

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

ABSTRACT

The concept of w*eb services* represent the next generation of architectures for interoperability between software applications based on software industry standards. Presented here is an overview of web services, a discussion of the use of web services in the context of simulation and a demonstration of the use of web services for simulation as implemented in the Microsoft .Net software development and execution framework. The paper focuses on the vital role of industry *standards* in the definition and implementation of web services and relates this to the opportunities and challenges for similar standards and benefits for interoperability in simulation software.

1 INTRODUCTION

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 (Kilgore 2002). 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 first topic examines the role of standards in the establishment of platform-neutral and language-neutral design patterns for object-oriented simulation libraries. The challenge here is to balance competing desires for encapsulation of the structural *representation* of simulation objects and convenient, flexible and efficient *control* of the objects. The second topic examines the potential for interoperability through a common open-source simulation software specification possibly implemented in .Net languages. This third paper examines the role that *web services* 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.

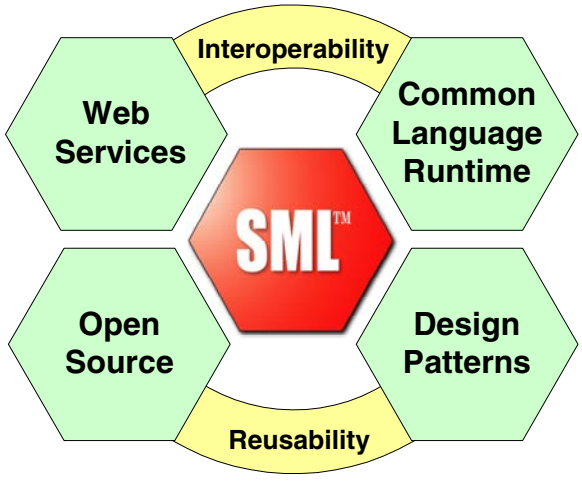

Figure 1: Simulation Interoperability and Reusability

What are web services? What is .Net? The long answer begins with statements like "web services can be defined as loosely coupled, reusable software components that semantically encapsulate discrete functionality and are distributed and programmatically accessible over standard Internet protocols" (Sleeper 2002). More simply, web services are a "stack" or "layers" of standards that enable one software application to interoperate with another application. The actual standard which enables the concept of web services is called the Simple Object Access Protocol (SOAP) (World Wide Web Consortium 2002) which is supported by most major software development organizations including Microsoft and Sun Microsystems which is significant since these companies rarely agree on anything.

While web services can be implemented with .Net, but web services are an industry standard and not proprietary to .Net or Microsoft. .Net is simply the designation for a product strategy by Microsoft to promote products around the concept of supporting web services. This is done throughout their language products (VB.Net, C#, C++.Net), development environment (VisualStudio.Net), server software (.Net Server), operating systems (Windows XP) and application software (Office, VBA.Net). The ex-

amples in this paper use .Net tools, but it is important to note that web services can be created and used regardless of hardware and software platforms.

What should make web services immediately of interest to the simulation community is the congruence between the objectives of web services and the objectives of *distributed simulation*. Distributed simulation is concerned with the execution of simulations on loosely coupled systems on geographically distributed computers interconnected via a wide area network such as the Internet (Fujimoto 2001). The benefits of distributed simulation include:

- Faster execution times of a simulation experiment through distribution of runs and alternatives to banks of available processors
- Geographic distribution of the simulation to allow more convenient collaboration
- Integration of simulations on different hardware devices and operating systems, particularly in training applications
- Integration of actual systems and simulated systems for test and evaluation or control

Since distributed simulations involve communication between the loosely coupled systems, some standards must be agreed to regarding the rules for communication. One such specification exists for Distributed Interactive Simulation (DIS) (IEEE 1995). Similarly, the High Level Architecture (HLA) initiative managed by the Defense Modeling and Simulation Office of the Department of Defense is a specification for obtaining reusability and interoperability in military simulation models (IEEE 2000). Web services is a promising enabling technology for implementation of the interface specifications sought in these standards initiatives in simulation software.

Section 2 continues with a general overview of web services and web services terminology. Section 3 discusses web services in simulation and Section 4 concludes with a discussion of non-technical issues in deployment and business relationships. Source code and updated simulation web services implementation information are posted at `http://www.sourceforge.net/opensml`.

2 WEB SERVICES

Web services are not one specific technology, but rather a group of established and emerging communication and interoperability protocols as shown in Figure 2. The *core* layers are properly stacked because each is implemented using standards specified in the underlying layer. But the *emerging* layers are misleading in many citations because they are not really essential elements of web service definition or implementation. The first lesson learned in the study of web services is that a great deal of effort is expended on keeping the core standard simple, but a great deal of effort is also spent by others on extending the standard beyond simple.

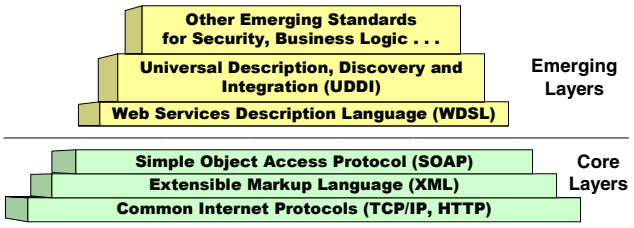

Figure 2: The Web Services Stack of Core and Emerging Layers of Standards

Despite intimidating acronyms, consortiums and complex specifications, the simple goal of web services is to enable a software application to embed references to other software applications. And the simple benefit is to extend the available libraries of functions and data to software on other computers in other locations on the intranet or internet. In existing software, it is a common task to reference an existing function, subroutine or class in a library located on a local hard drive. Web services extend this capability with protocols to allow the location of that function, subroutine or class to be anywhere on the network. Web services enable interoperability by developers to produce functions for reuse and to consume functions created by others anywhere on a network.

2.1 HTTP+XML+SOAP=WEB SERVICE

At a minimum, a web service entails a connection between two applications, a remote procedure call (RPC), in which requests and responses are exchanged in Extensible Markup Language (XML) over Hypertext Transport Protocol (HTTP). The fact that the underlying HTTP is restricted to text is critical to enable web services to pass through firewalls that would prevent binary exchanges. But instead of using HTTP to simply transport HTML content, a web service uses HTTP to transport XML-based requests and responses within an executing program. In a sense, this capability is available using CGI, CORBA and other technologies. But previous attempts at distributed computing (CORBA, Distributed Smalltalk, Java RMI) have yielded systems where the coupling between various components in a system is too tight to be effective. These approaches require too much agreement and shared context among systems. But standardized, XML-based web services remove much of the custom programming chores necessary in prior solutions and enable a much more powerful programming solution within traditional programming languages.

XML is used to encode all communications to and from a web service. For example, a client invokes a web service by sending an XML message, then waits for a corresponding XML response. Because all communication is

in XML, web services are not tied to any one operating system or programming language--Java can talk with C# and Windows applications can talk with Unix applications. If data is delivered as XML, web services can process that data in a variety of useful ways. For web services, XML's separation of content and presentation is ideal.

The Simple Object Access Protocol is the final layer of the core web service stack. SOAP uses XML messages to invoke remote methods. A web service could interact with remote machines through simpler HTTP methods (post and get), but SOAP is much more robust and flexible. In addition to being a protocol specification that defines a uniform way of passing XML-encoded data, SOAP also defines a way to perform remote procedure calls using HTTP as the underlying communication protocol. Most importantly, SOAP is a standard governed by the World Wide Web Consortium's XML Protocols Working Group so evolution and refinement of the standards will continue.

2.2 WSDL AND UDDI

There are three roles you can play in the development of web services

- *Service providers* who create, expose and maintain a registry that makes those services available.
- *Service brokers* who act as liaisons between service providers and service requestors.
- *Service requestors* who implement service brokers to discover web services, then invoke those services to create all or part of their applications.

The interaction between a web service provider and a web service requester is designed to be completely platform and language independent. Service requesters and providers concern themselves with the interfaces necessary to interact with each other. As a result, a service requester has no idea how a service provider implements its service, and a service provider has no idea how a service requester uses its service. Those details are encapsulated inside the requesters and providers.

That encapsulation is crucial for reducing complexity. In fact the client requesting a web service thinks it's calling methods on a local object. In fact, those method calls are translated to XML and sent over the network, processed on the other end, and the response message is carefully disguised as the return value from the method call.

But in such a distributed computing environment, it may be helpful to have a common standard for communication between providers, brokers and requestors. The first extended layer of the web service stack is a common XML format called the Web Service Description Language (WSDL). An example WSDL file appears in Figure 3 with "…" used to indicate some missing detail. Fortunately, the creation of the WSDL file is done by the software development environment like Microsoft's VisualStudio.Net (Microsoft 2002).

```
<binding name="EntityBinding"
         type="EntityType">
  <soap:binding style="rpc" transport=
  <operation name="getEntity">
    <soap:operation soapAction…/>
    <input>
      <soap:body type="int"
                 namespace="urn:"…"
                 encoding="…"/>
    </input>
    <output>
      <soap:body type="Entity"
                 encoding="…"/>
    </output>
  </operation>
</binding>
```

Figure 3: A WSDL File Describes The Web Service

Universal Discovery Description and Integration (UDDI) is an optional final component of the web service infrastructure. You can think of the UDDI registry as both a white pages business directory and a technical specifications library. UDDI represents a standard for exposing a WSDL. There are currently two public UDDI registries, hosted by IBM and Microsoft with another to be hosted by Hewlett-Packard in the future. These repositories synchronize their contents regularly so that information entered into one repository is quickly replicated to the others. To assist developers in understanding UDDI, each company also hosts a test repository that is intended for educational purposes (see http://demo.alphaworks.ibm.com/browser/). Remember that WSDL and UDDI registries are an optional layer of the web services stack and not required for web service development. They represent a potential for exposing software services the same way a business directory exposes general services.

2.3 WEB SERVICES DEVELOPMENT TOOLS

While standards enable the opportunity for web services deployment, the ultimate success of web services may be dependent on the ability of programming environments to encourage developers to conveniently and productively create, debug and consume web services. While the major software vendors offer competing development environments for web services support for different languages and platforms, the ultimate product of these development environments are still interoperable web services.

The IBM WebSphere SDK for Web Services is based on open specifications for Web services such as SOAP, WSDL, and UDDI and runs on both Linux and Windows operating systems. Collecting a number of technologies in a single package enables skilled early adopters to develop and test Java-based Web services. For development and deployment of Web services in a production environment, IBM of-

fers Web services-enabled environments such as IBM WebSphere Application Server and IBM WebSphere Studio as described at http://www-106.ibm.com/developerworks/webservices/wsdk/.

SunOpen Net Environment (Sun ONE) is Sun's standards-based software vision, architecture, platform, and expertise for building and deploying what Sun calls Services on Demand. It provides a highly scalable and robust foundation for traditional software applications as well as current web-based applications based on Java and Java-related technologies, while laying the foundation for the next-generation distributed computing models such as web services. The development environment is further described at http://www.sun.com/software/sunone.

The Mono Project not specifically related to web services at this point, but is a relevant product as it is an open development initiative that is working to develop a Unix version of the Microsoft .NET development platform. Its objective is to enable Unix developers to build and deploy cross-platform .NET Applications. The project will implement various technologies developed by Microsoft that have now been submitted to the ECMA for standardization. The current status is described at http://www.go-mono.net/index.html.

3 SIMULATION WEB SERVICES WITH .NET

The objective of this section is to demonstrate the creation of a web services in simulation model development using VS.Net. In the example, our simulation "model" will be a client application that desires to create and use information about an "entity" object. The vendor who supports the entity object is exposing a web services to allow us to create this object and obtain data about the objects properties (e.g., mean service time).

The first step in the process of providing a web service is to launch a new solution in VS.Net and add an ASP.Net Web Service project to the solution. The web service you will create in this section, EntityDefinition, exposes methods for creating an Entity and returning the service time of the Entity. At the top of the ".asmx" file that is created is the reference to the file containing the XML web service and the language for the service (C#, in this case).

```
@WebServiceLanguage="C#"Class="EntityDefinition"
```

Another file is created in the project called "EntityDefinition.asmx.cs". This file contains the actual class that encapsulates the functionality of the service. This class should be public and can optionally inherit from the WebService base class. Each web service class contains a [WebService] attribute with an optional description. Each method that will be exposed from the web service is flagged with a [WebMethod] attribute in front of it. Without this attribute, the method will not be exposed from the service. This is sometimes useful for hiding implementation details called by public web service methods.

```
using System;

 [WebSevice(
   Namespace =
    "http://localhost/SimulationWebService/",
   Description =
    "The EntityDefinition web service provides an
    example of how a web service could return an
    Entity reference or Entity information if
    given that reference")
 ]

public class EntityDefinition
{
 [WebMethod]
 public Entity getEntity()
 {
    return new Entity( );
 }

 [WebMethod]
 public double getMeanServiceTime(Entity ent)
 {
    return ent.getMeanServiceTime( );
 }
}
```

The second step is to build the Entity class referenced by the web service methods:

```
public class Entity
{
    public static double meanServiceTime = 10.0;

    public double getMeanServiceTime( )
    {
        return meanServiceTime;
    }
}
```

The result of these two simple steps is a web service. Behind the scenes, the necessary SOAP infrastructure will allow our second project, the client WebServiceModel class, that uses the web service. After creating the second project, we add a "Web Reference" by navigating to the page where the web service resides. There we are presented with the page you see in Figure 4 describing the

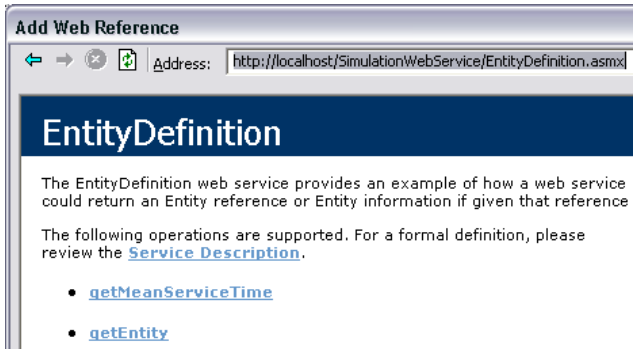

Figure 4: Adding a Web Service Reference

web services available. If needed, we can view the WSDL file to verify the input and output parameters of the methods made available to us. The Description field of the [WebService] attribute is presented as a short description of the service. After selecting this Web Reference, our client project code can use the EntityDefinition class and its methods as if the class were residing in the class path on the local machine.

```
public class WebServiceModel
{
  .
  .
  EntityDefinition myED=new EntityDefinition( );
  Entity myEnt = myED.getEntity( );
  .
  myEnt.delay(myED.getMeanServiceTime( myEnt ) );
  .
  .
  .
}
```

As shown in the VisualStudio.Net project window in Figure 5, the VS.Net project now contains local References to System class libraries and Web References to external web service classes supported elsewhere.

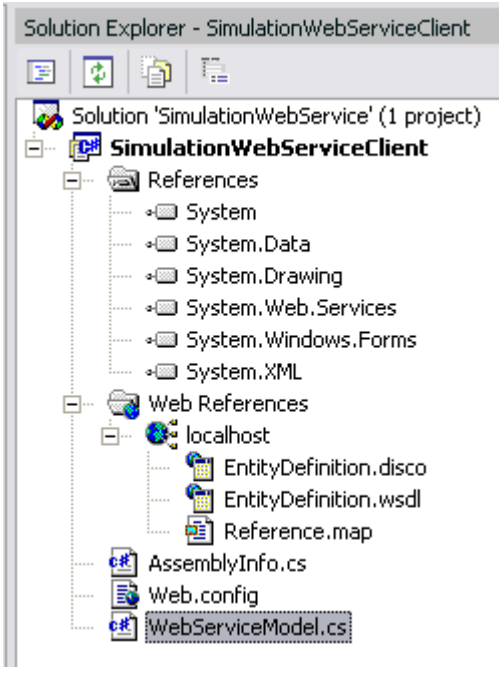

Figure 5: VS.Net Project with Web Reference

In this trivial example, a web service was created to access static information during the execution of the simulation. But that is just one of the many opportunities available to the simulation programmer using web services. In the HLA context mentioned in the introduction, the data exchange could be some dynamically changing property such as location, speed or time. In a distributed execution context, the exchange might be related to the output summary statistics from one node of the experimental design being distributed on available processors (an inexpensive approach to grid computing). Obviously, there are performance penalties, security considerations and availability risks that are not present if all of the code were locally compiled. But the benefits of the distributed design opportunities may outweigh these costs, particularly on an intranet architecture where data security is more easily and efficiently managed.

4 SUMMARY

Web services present a technical opportunity for integration of simulation software using software industry standards. But the largest risk to web service implementation is less technical in nature as it involves the establishment of a long term relationship between one or more software developers or vendors. The service provider must deliver and assist in the remote debugging process to help the service requestor insure that the posted interface is valid and available. Unless the service provider exposes source code, there is no certain validation of what is hidden and is the reason that open-source models may be the only way of establishing sufficient trust. Finally, a financial relationship must be established to pay for the use of the service. This business model of software as a service is not a familiar business relationship and fair pricing models may be difficult to negotiate.

REFERENCES

Fujimoto, R. M. 2001. Parallel and Distributed Simulation Systems. In *Proceedings of the 2001 Winter Simulation Conference*, ed B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer, eds., 147-157. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

IEEE. 2000. IEEE 1516-2000 IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Framework and Rules. Piscataway, NJ: The Institute of Electrical and Electronics Engineers, Inc.

IEEE. 1995. *IEEE Std 1278.1-1995 IEEE Standard for Distributed Interactive Simulation -- Application Protocols*. New York, NY, Institute of Electrical and Electronics Engineers, Inc.

Kilgore, R. A. 2002. Multi-Language, Open-source Modeling Using the Microsoft .Net Architecture. 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. 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., 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.

Microsoft 2002. Developer Tolls and .Net. Available online via <www.microsoft.com/net/products/tools.asp> [accessed April 16, 2002].

Sleeper, Brent. 2001. Defining Web Services. The Stencil Group. Available online via <www.stencilgroup.com/ideas_scope_200106wsdefined.html> [accessed May 23, 2002].

Tidwell, D. 2000. Web services -- the Web's next revolution. Available online via <www-105.ibm.com/developerworks/education.nsf> [accessed April 16, 2002].

Tidwell D., J.Snell and P. Kulchenko. 2001. Programming Web Services with SOAP. O'Reilly.

World Wide Web Consortium. 2002. Simple Object Access Protocol (SOAP) 1.1.. Available online via <www.w3.org/TR/SOAP/> [accessed July 12, 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>.