See published version:

Mohammad Sabah and Osman Balci (2005), "Web-based Random Variate Generation for Stochastic Simulations," International Journal of Simulation and Process Modelling 1, 1-2, 16-25.

Web-based Random Variate Generation for Stochastic Simulations

MOHAMMAD SABAH
OSMAN BALCI

Department of Computer Science
660 McBryde Hall, MC 0106
Virginia Tech
Blacksburg, Virginia 24061, U.S.A.

Contact Author: Osman Balci
http://manta.cs.vt.edu/balci/

ABSTRACT

This paper presents a Random Variate Generation (RVG) World Wide Web (Web) service and an RVG Web-based software application, both of which are developed in compliance with the Java 2 Enterprise Edition (J2EE) architecture and run under the IBM WebSphere Application Server. The RVG Web service can be invoked programmatically over the Web by using Simple Object Access Protocol (SOAP) through the HyperText Transfer Protocol (HTTP) using the eXtensible Markup Language (XML). The RVG Web service and software both generate random variates from 27 probability distributions, and provide general statistics, scatter plot, and histogram of the requested random variates. The plots and histograms are created in the Scalable Vector Graphics (SVG) format. The RVG Web service can be used for stochastic input data modeling of a Web-based simulation. The RVG software can be used as a Web-based application for a variety of purposes including teaching random variate generation, probabilistic modeling, and random experimentation.

Keywords: Random variate generation, simulation experiment design, simulation input data modeling, Web service, Web-based simulation.

Biographical Notes:

MOHAMMAD SABAH received his B.S. degree in Computer Science and Engineering from Regional Engineering College (Allahabad, India) in 2000 and M.S. degree in Computer Science from Virginia Polytechnic Institute and State University (Virginia Tech) in 2003. He worked as a Software Engineer at River Run Software Group between 2000 and 2001, and served as a Graduate Teaching Assistant in the Computer Science Department at Virginia Tech between 2001 and 2003. Presently he is working as a J2EE consultant with a major telecom company.

OSMAN BALCI is Professor of Computer Science at Virginia Polytechnic Institute and State University (Virginia Tech). He received B.S. and M.S. degrees from Boğaziçi University (Istanbul) in 1975 and 1977, and M.S. and Ph.D. degrees from Syracuse University (New York) in 1978 and 1981. Dr. Balci served as Editor-in-Chief of two international journals: *Annals of Software Engineering*, 1993-2002 and *World Wide Web*, 1996-2000. He currently serves as the Verification, Validation and Accreditation (VV&A) Area Editor of *ACM Transactions on Modeling and Computer Simulation*; Modeling and Simulation (M&S) Category Editor of *ACM Computing Reviews*, and Area Editor of *Simulation: Transactions of the Society for M&S International* (SCS). He serves as a member of the Winter Simulation Conference Board of Directors representing SCS. His current areas of expertise center on software engineering; e-systems engineering (e.g., e-business, e-solutions); and M&S. His e-mail and Web addresses are balci@vt.edu and http://manta.cs.vt.edu/balci.

1. INTRODUCTION

Random phenomena are often modeled by using random variables such as inter-arrival time of objects, activity duration time, travel time, processing time, or system failure time. Such a random variable is commonly characterized by a probability distribution such as exponential, normal, or weibull. A stochastic simulation model mimics the behavior of a random phenomenon by way of generating random values (variates) for a random variable in such a manner that the generated variates form the probability distribution of the random variable. Generation of such random values is called *random variate generation* (RVG), which is an integral part of any stochastic discrete-event simulation model. The RVG topic is covered by simulation textbooks such as [1] and [2].

During the last five years, we have witnessed a paradigm shift in the way software is developed and used. Traditionally, software is developed as a shrink-wrapped product, sold and purchased as a product, and installed on computers as a product. Under the new paradigm, software is treated as a service, runs on a server computer, and the users subscribe to use it over the Internet instead of buying it as a product and installing it on their own computer. This paradigm shift has many advantages for the vendors and users [3]. Two platforms have emerged recently for developing software as a service: Java 2 Enterprise Edition (J2EE) industry-standard architecture and the Microsoft .Net Framework. We used J2EE for developing a World Wide Web (Web) service and a Web-based software application for RVG, which can be currently accessed at http://sunfish.cs.vt.edu/RVGWebService/.

As a result of the paradigm shift to treating software as a service, we foresee more and more simulation applications being developed and used on the Web. Wiedemann presents application service providing under the new paradigm for simulation purposes [4]. Kilgore presents an overview of Web services, discusses the use of Web services in the context of simulation, and provides a demonstration of the use of Web services for simulation as implemented under the Microsoft .Net framework [5]. Chandrasekaran et al. examines the synergy between Web service technologies and simulation [6]. Fishwick emphasizes the importance and benefits of using eXtensible Markup Language (XML) for simulation modeling [7].

One major advantage of the new paradigm is that software reuse and interoperability are greatly facilitated [8]. A Web service, such as ours, is provided in a platform- and implementation language-independent manner. A Web-based simulation application, which is built in any programming language and running on any hardware platform, can use our RVG Web service. The reuse and interoperability are enabled by the de facto standards such as XML, Simple Object Access Protocol (SOAP), Web Service Description Language (WSDL), and Universal Description, Discovery and Integration (UDDI) [9].

The purpose of this paper is to describe an RVG Web service and an RVG Web-based software application. The RVG Web service is presented in section 2. Section 3 provides an overview of the RVG Web-based software architecture. Activities for the verification and validation of the RVG service and software are described in section 4. Conclusions are given in section 5.

2. RVG WEB SERVICE

The RVG Web service is developed using open technology standards under the layered architecture depicted in Figure 1 [8, 9]. The Internet layer consists of the interconnected computers that communicate with each other using the Transmission Control Protocol / Internet Protocol (TCP/IP). The Web layer runs on top of the Internet and uses the HyperText Transfer Protocol (HTTP), which is a request/response-oriented protocol.

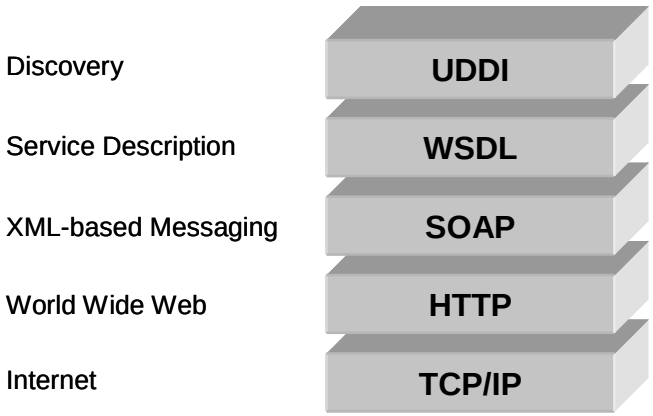

Figure 1. RVG Web service architecture

XML is used to structure the request/reply message and SOAP is used to transfer the message between the client simulation application and the RVG Web service. WSDL is used to describe how to access the RVG Web service. The details of the RVG Web service can optionally be published in one of the publicly available UDDI registries so that clients can search and discover the RVG Web service.

Figure 2 illustrates the use of the RVG Web service by a simulation application. Using the WSDL description on how to access the RVG Web service, a simulation application software with Internet connection sends a request message to the RVG Web service provider computer. The request message is formatted according to the RVG request schema in XML. SOAP is used to transfer the message to the RVG Web service provider computer over the HTTP layer using TCP/IP on top of the Internet backbone. The message is received by the RVG Web service provider computer through the HTTP server. The RVG Web service processes the request, generates the service results, and creates an XML file containing the service results formatted according to the RVG reply schema. For data persistency among subsequent requests from the same simulation application, Entity Enterprise Java Beans (EJBs) are used to represent the persistence data stored in the database. SOAP is used to transfer the message in XML format to the client simulation application over the HTTP layer using TCP/IP on top of the Internet backbone.

A simulation application requiring random variates is expected to request the variates in batches. The batch size (e.g., 5000, 25000) is determined depending on the needs of the simulation application and performance considerations. Typically, the double-buffering approach is used to improve performance. Upon receiving a batch of random variates, the request for the next batch is placed. While the request is being processed by the RVG Web service provider, the simulation application uses the batch that is already received.

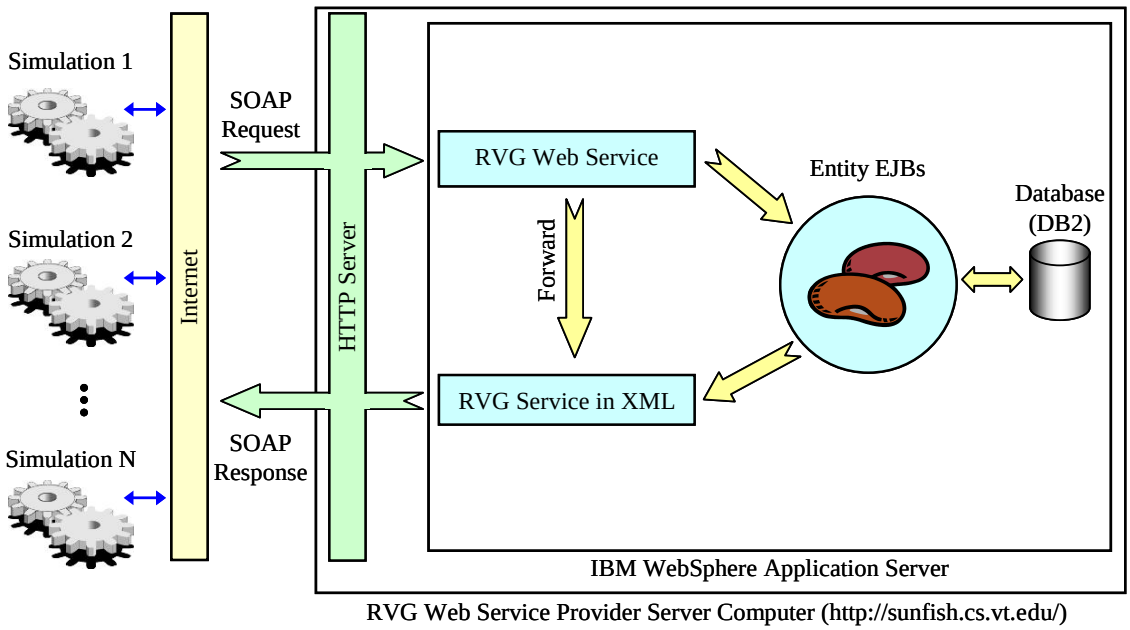

Figure 2. RVG Web service usage by a simulation application

A stochastic simulation model may represent many sources of randomness. Each source of randomness requires a different sequence of random variates. The sequence is commonly called a *stream* and starts with its own *seed* value. Thus, a simulation application may need k streams of random variates with k different seed values as depicted in Table 1. The random variates are requested in batches from the RVG Web service. The batch size may differ from one stream to another. In Table 1, $X_{i,j,bi}$ represents a random variate from stream i in batch j with a batch size of bi for the ith stream.

Table 1. Streams of random variates in batches with seeds

StreamID	$Stream_1$	$Stream_2$	$Stream_3$	…	$Stream_k$
Seed	$Seed_1$	$Seed_2$	$Seed_3$	…	$Seed_k$
$Batch_1$	$X_{1,1,1}$ $X_{1,1,2}$ $X_{1,1,3}$: $X_{1,1,b1}$	$X_{2,1,1}$ $X_{2,1,2}$ $X_{2,1,3}$: $X_{2,1,b2}$	$X_{3,1,1}$ $X_{3,1,2}$ $X_{3,1,3}$: $X_{3,1,b3}$	…	$X_{k,1,1}$ $X_{k,1,2}$ $X_{k,1,3}$: $X_{k,1,bk}$
$Batch_2$	$X_{1,2,1}$ $X_{1,2,2}$ $X_{1,2,3}$: $X_{1,2,b1}$	$X_{2,2,1}$ $X_{2,2,2}$ $X_{2,2,3}$: $X_{2,2,b2}$	$X_{3,2,1}$ $X_{3,2,2}$ $X_{3,2,3}$: $X_{3,2,b3}$	…	$X_{k,2,1}$ $X_{k,2,2}$ $X_{k,2,3}$: $X_{k,2,bk}$
:	:	:	:	:	:
$Batch_m$	$X_{1,m,1}$ $X_{1,m,2}$ $X_{1,m,3}$: $X_{1,m,b1}$	$X_{2,m,1}$ $X_{2,m,2}$ $X_{2,m,3}$: $X_{2,m,b2}$	$X_{3,m,1}$ $X_{3,m,2}$ $X_{3,m,3}$: $X_{3,m,b3}$	…	$X_{k,m,1}$ $X_{k,m,2}$ $X_{k,m,3}$: $X_{k,m,bk}$

The stream identification numbers, seed values, and batch sizes are all stored in the database and held by the Entity EJBs for accomplishing data persistency with respect to each simulation application.

2.1 RVG Web Service Functions

A client Web-based simulation application can send a message to the RVG Web service and request one or more of the following functions:

1. Generate *n* number of random variates from one of the 27 probability distributions listed in Table 2, given the values of the probability distribution's parameters (e.g., location, scale, shape), stream identifier, and seed.
2. Create a histogram of the generated random variates, given the number of intervals.
3. Create a scatter plot of the generated random variates.
4. Compute the following statistics for the generated random variates: minimum observation, maximum observation, mean, median, variance, coefficient of variation, and skewness.

The stream identifier enables the generation of random variates from the same stream in subsequent calls to the RVG Web service. The seed identifies the starting value of a stream of random variates. The same sequence of random variates in a stream can be generated by specifying the same seed. This capability is useful for replicating identical experimental conditions when comparing alternative operating policies.

Table 2. Probability distributions for which RVG is provided

Category	*Probability Distribution*
Non-negative Continuous	Exponential Gamma Inverse Gaussian Inverted Weibull Log-Laplace Log-Logistic Lognormal Pareto Pearson Type V Pearson Type VI Random Walk Weibull
Bounded Continuous	Beta Johnson SB Triangular Uniform
Unbounded Continuous	Extreme Value Type A Extreme Value Type B Johnson SU Laplace Logistic Normal
Discrete	Binomial Discrete Uniform Geometric Negative Binomial Poisson

The histogram graphically characterizes the probability distribution formed by the generated random variates. The RVG Web service creates the histogram in the Scalable Vector Graphics (SVG) format. An example histogram is shown in Figure 3.

The scatter plot enables visual inspection for the randomness of the generated random variates. The RVG Web service creates the scatter plot in the SVG format. An example scatter plot is shown in Figure 4.

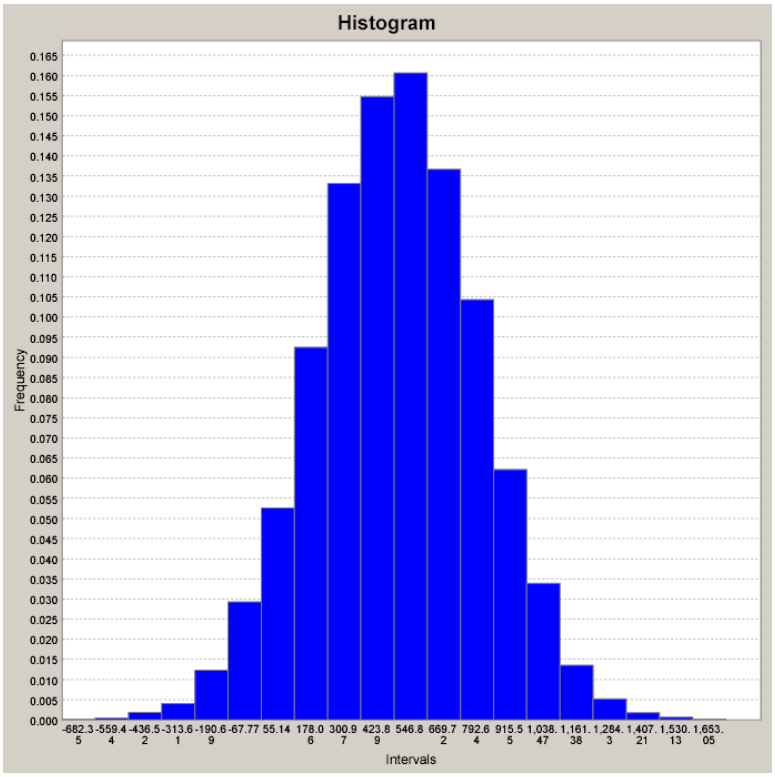

Figure 3. Histogram created by the RVG Web service in the SVG format

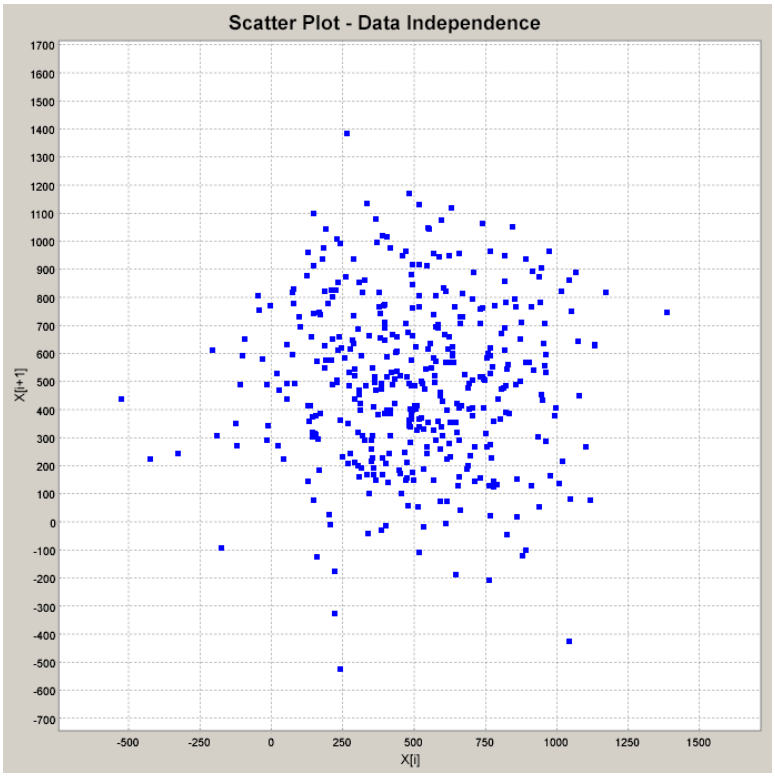

Figure 4. Scatter plot created by the RVG Web service in the SVG format

2.2 RVG Web Service Description

Description of how to access the RVG Web service is given in a WSDL document, which consists of the following two sections:

- *Abstract Definitions* – These define the "what" of the functionality that is offered. Abstract definitions define the SOAP messages that flow between the RVG Web service requestor and the RVG Web service in a language- and platform-independent manner.
- *Concrete Definitions* – These define the "how" and "where" of the functionality that is offered. Concrete descriptions specify the message formats and network protocols for the RVG Web service and the endpoint where the RVG Web service is accessible.

Figure 5 shows the abstract definitions of the RVG Web service. The salient points to note are:

- The request and reply schemas for the document/literal Web service are imported into the WSDL file. These schemas lay down the format of the request and reply messages in the Web service invocation.
- There are two messages, in their own separate "message" elements that correspond to the request and reply messages in the Web service invocation. The bodies of the respective SOAP messages have to follow the imported schema specification.
- The "portType" element declares the one operation exposed in the RVG Web service, `getVariateFromXML`, along with the input and output messages that flow.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="VariateGenerator"
   targetNamespace="http://Session.RVG.sunfish.vt.edu.wsdl/VariateGenerator/"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:tns="http://Session.RVG.sunfish.vt.edu.wsdl/VariateGenerator/"
         xmlns:xsd=http://www.w3.org/2001/XMLSchema
         xmlns:xsd1="http://Session.RVG.sunfish.vt.edu/">
   <import location="VariateRequestSchema.xsd" namespace="http://Session.RVG.sunfish.vt.edu/"/>
   <import location="VariateReplySchema.xsd" namespace="http://Session.RVG.sunfish.vt.edu/"/>
   <message name="getVariateFromXMLRequest">
      <part name="inputXML" type="xsd1:variateRequest"/>
   </message>
   <message name="getVariateFromXMLResponse">
      <part name="result" type="xsd1:variateReply"/>
   </message>
   <portType name="VariateGenerator">
      <operation name="getVariateFromXML" parameterOrder="inputXML">
         <input message="tns:getVariateFromXMLRequest" name="getVariateFromXMLRequest"/>
               <output message="tns:getVariateFromXMLResponse"  name="getVariateFromXMLResponse"/>
      </operation>
   </portType>
</definitions>
```

Figure 5. RVG Web service WSDL abstract definitions

Figure 6 shows the bindings of the RVG Web service. The salient points to note are:

- It imports the WSDL file fragment that contains abstract definitions.
- The "binding" element declares the message formats and serialization/de-serialization for the SOAP messages that flow between the RVG Web service requestor and the RVG Web service. The binding is document/literal.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="VariateGeneratorBinding"
   targetNamespace="http://Session.RVG.sunfish.vt.edu.wsdl/VariateGeneratorBinding/"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:interface="http://Session.RVG.sunfish.vt.edu.wsdl/VariateGenerator/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://Session.RVG.sunfish.vt.edu.wsdl/VariateGeneratorBinding/">
<import location="VariateGenerator.wsdl"
          namespace="http://Session.RVG.sunfish.vt.edu.wsdl/VariateGenerator/"/>
<binding name="VariateGeneratorBinding" type="interface:VariateGenerator">
      <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="getVariateFromXML">
         <soap:operation soapAction="" style="document"/>
         <input name="getVariateFromXMLRequest">
            <soap:body
               encodingStyle="http://xml.apache.org/xml-soap/literalxml"
               namespace="http://tempuri.org/edu.vt.sunfish.RVG.Session.VariateGenerator"
               parts="inputXML" use="literal"/>
         </input>
         <output name="getVariateFromXMLResponse">
            <soap:body
               encodingStyle="http://xml.apache.org/xml-soap/literalxml"
               namespace="http://tempuri.org/edu.vt.sunfish.RVG.Session.VariateGenerator" use="literal"/>
         </output>
      </operation>
</binding>
</definitions>
```

Figure 6. RVG Web service WSDL concrete definitions – bindings

Figure 7 shows the service definition of the RVG Web service. The salient points to note are:

- It imports the WSDL file fragment that contains the bindings described in Figure 6.
- The RVG Web service is available at http://sunfish.cs.vt.edu/RVGWebService/servlet/rpcrouter, which is specified in the "service" element. The name of the Web service is "VariateGeneratorService".

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="VariateGeneratorService"
   targetNamespace="http://Session.RVG.sunfish.vt.edu.wsdl/VariateGeneratorService/"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:binding="http://Session.RVG.sunfish.vt.edu.wsdl/VariateGeneratorBinding/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://Session.RVG.sunfish.vt.edu.wsdl/VariateGeneratorService/">
   <import location="VariateGeneratorBinding.wsdl"
namespace="http://Session.RVG.sunfish.vt.edu.wsdl/VariateGeneratorBinding/"/>
  <service name="VariateGeneratorService">
      <port binding="binding:VariateGeneratorBinding" name="VariateGeneratorPort">
         <soap:address location="http://sunfish.cs.vt.edu/RVGWebService/servlet/rpcrouter"/>
      </port>
   </service>
</definitions>
```

Figure 7. RVG Web service WSDL concrete definitions – service

WSDL contains a switch that changes the mode from Remote Procedure Call (RPC)-style Web service to document-style Web service. If the style is RPC, the message within the SOAP envelope is enclosed within a wrapper element in the body, with the name of the element corresponding to the name of the exposed method. On the other hand, if the style is document, this wrapper element within the body of SOAP envelope is absent, and the message is inserted as-is or encoded. In the RVG Web service, since we make use of request and reply schemas, SOAP encoding is avoided, and the SOAP binding is "literal".

The major reasons for selecting the document-style Web service as opposed to the RPC-style Web service in our work are given below [10]:

- *Full use of XML:* Using an XML schema for request allows the Web service client to validate the document before sending the request. Furthermore, the reply sent back by the Web service can be validated by the reply schema by the client application. On the other hand, encapsulating the XML request and reply documents as SOAP-encoded string parameters does not allow enforcement of "high-level business rules" [10] using XML schemas, and buries the validations inside methods.
- *Flexibility:* Using the document-style makes the RVG Web service and its clients loosely coupled, in that they are not tied to the interface, as would have been in a remote procedure call. Changing a method signature of the RVG Web service has no effect on the Web service clients in the document style.
- *Asynchronous processing:* The RVG Web service provides for synchronous processing as of now. Since the request and reply documents are self-contained, the RVG Web service can very easily be used in asynchronous modes using message queues.
- *Publish RVG Web service for outside clients:* With the use of request and reply schemas, the RVG Web service can be published very easily to clients outside Virginia Tech with no assumptions about the target system or encodings used.

2.3 Using the RVG Web Service

The first step for using the RVG Web service is to create an XML document containing the service request that is structured based on the request schema. An example service request is presented in Figure 8.

```
<?xml version="1.0" encoding="UTF-8"?>
<rv:variateRequest xmlns:rv=" http://Session.RVG.sunfish.vt.edu/"
                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <distribution>
            <sequenceID>1</sequenceID>
                <genParameters xsi:type="rv:uniformParameterType">
                        <nameDist>Uniform</nameDist>
                        <streamID>1</streamID>
                        <numberOfVariates>5000</numberOfVariates>
                <seed>1313.0</seed>
                <lowerValue>100.0</lowerValue>
                <upperValue>200.0</upperValue>
                </genParameters>
                <genStatistics>false</genStatistics>
                <genHistogram>
                        <genHist>true</genHist>
                        <numberIntervals>10</numberIntervals>
                </genHistogram>
                <genScatterPlot>false</genScatterPlot>
   </distribution>
    <distribution>
            <sequenceID>2</sequenceID>
                <genParameters xsi:type="rv:uniformParameterType">
                        <nameDist>Uniform</nameDist>
                        <streamID>23</streamID>
                        <numberOfVariates>5000</numberOfVariates>
                <seed>15793</seed>
                <lowerValue>1928</lowerValue>
                <upperValue>2349</upperValue>
                </genParameters>
                <genStatistics>true</genStatistics>
                <genHistogram>
                        <genHist>true</genHist>
                        <numberIntervals>20</numberIntervals>
                </genHistogram>
                <genScatterPlot>true</genScatterPlot>
   </distribution>
</rv:variateRequest>
```

Figure 8. A sample request for the RVG Web service

In the next step, the client Web-based simulation application sends the service request to the RVG Web service by using the client proxy given in Figure 9.

```
package proxy.soap;
import java.net.*;
import java.util.*;
import org.w3c.dom.*;
import org.apache.soap.*;
import org.apache.soap.encoding.*;
import org.apache.soap.encoding.soapenc.*;
```

```
import org.apache.soap.rpc.*;
import org.apache.soap.util.xml.*;
import org.apache.soap.messaging.*;
import org.apache.soap.transport.http.*;
public class VariateGeneratorProxy
{
  private Call call;
  private URL url = null;
  private String stringURL="http://sunfish.cs.vt.edu/RVGWebServices/servlet/rpcrouter";
  private java.lang.reflect.Method setTcpNoDelayMethod;

  public VariateGeneratorProxy()
  {
    try
    {
                    setTcpNoDelayMethod = SOAPHTTPConnection.class.getMethod ("setTcpNoDelay",
                                                                    new Class[]{Boolean.class});
     }
     catch (Exception e)
     {
     }
     call = createCall();
   }
  public synchronized void setEndPoint(URL url) {    this.url = url;  }
  public synchronized URL getEndPoint() throws MalformedURLException  {  return getURL();  }
  private URL getURL() throws MalformedURLException
  {
    if (url == null && stringURL != null && stringURL.length() > 0)
    {      url = new URL(stringURL);    }
    return url;
  }
public synchronized org.w3c.dom.Element getVariateFromXML(org.w3c.dom.Element inputXML)
                                                                    throws Exception
  {
     String targetObjectURI = " http://tempuri.org/edu.vt.sunfish.RVG.Session.VariateGenerator ";
                                          //name of web service
     String SOAPActionURI = "";
     String stringURL = "http://sunfish.cs.vt.edu/RVGWebService/servlet/rpcrouter ";
     call.setMethodName("getVariateFromXML");
     call.setEncodingStyleURI(Constants.NS_URI_LITERAL_XML); //SOAP encoding style
     call.setTargetObjectURI(targetObjectURI);
     Vector params = new Vector();
     Parameter inputXMLParam = new Parameter("inputXML", org.w3c.dom.Element.class,
                                                  inputXML, Constants.NS_URI_LITERAL_XML);
     params.addElement(inputXMLParam);
     call.setParams(params);
     Response resp = call.invoke(stringURL, SOAPActionURI);

     //Check the response.
     if (resp.generatedFault())
     {
          Fault fault = resp.getFault();
                call.setFullTargetObjectURI(targetObjectURI);
                throw new SOAPException(fault.getFaultCode(), fault.getFaultString());
    }
```

```
   else
   {
         Parameter refValue = resp.getReturnValue();
         return ((org.w3c.dom.Element)refValue.getValue());
   }
  }
}    //end of class
```

Figure 9. RVG Web service client proxy

The client proxy exposes a method called "getVariateFromXML" that has the same signature as that of the RVG Web service. It accepts an XML document as parameter (based on the published request schema), and returns another XML document (based on the published reply schema).

The proxy hides the communication details with the RVG Web service from the calling application by enabling the client Web-based simulation application to merely call the "getVariateFromXML" method of the client proxy passing input parameter `inputXML` obtained after parsing the RVG request XML file, an example of which is given in Figure 8. The proxy then takes responsibility for invoking the remote method of RVG Web service.

The "call" encapsulates the request and contains information about the invocation. The encoding style of the SOAP body of the request packet is set by

```
call.setEncodingStyleURI(Constants.NS_URI_LITERAL_XML);
```

The name of the RVG Web service is set by

```
call.setTargetObjectURI(targetObjectURI);
```

The following four lines set the parameters of the call, where inputXML corresponds to the parsed XML request document.

```
Vector params = new Vector();
Parameter inputXMLParam = new Parameter("inputXML",
          org.w3c.dom.Element.class, inputXML,
          Constants.NS_URI_LITERAL_XML);
params.addElement(inputXMLParam);
call.setParams(params);
```

The actual invocation of the RVG Web service is done by the statement shown below, where stringURL corresponds to the URL of the RVG Web service.

```
Response resp = call.invoke(stringURL, SOAPActionURI);
```

If the invocation of the RVG Web service is successful, the proxy method returns the `org.w3c.dom.Element` obtained after parsing the XML document returned from the RVG Web service, as indicated by the following statements.

```
Parameter refValue = resp.getReturnValue();
return ((org.w3c.dom.Element)refValue.getValue());
```

All client Web-based simulation applications follow the code structure shown in Figure 9. The client applications can be implemented in any programming language and running on any platform.

3. RVG WEB-BASED SOFTWARE

The RVG Web-based software (http://sunfish.cs.vt.edu/RVGWebService/) is structured based on the Java 2 Enterprise Edition (J2EE) industry-standard architecture as depicted in Figure 10.

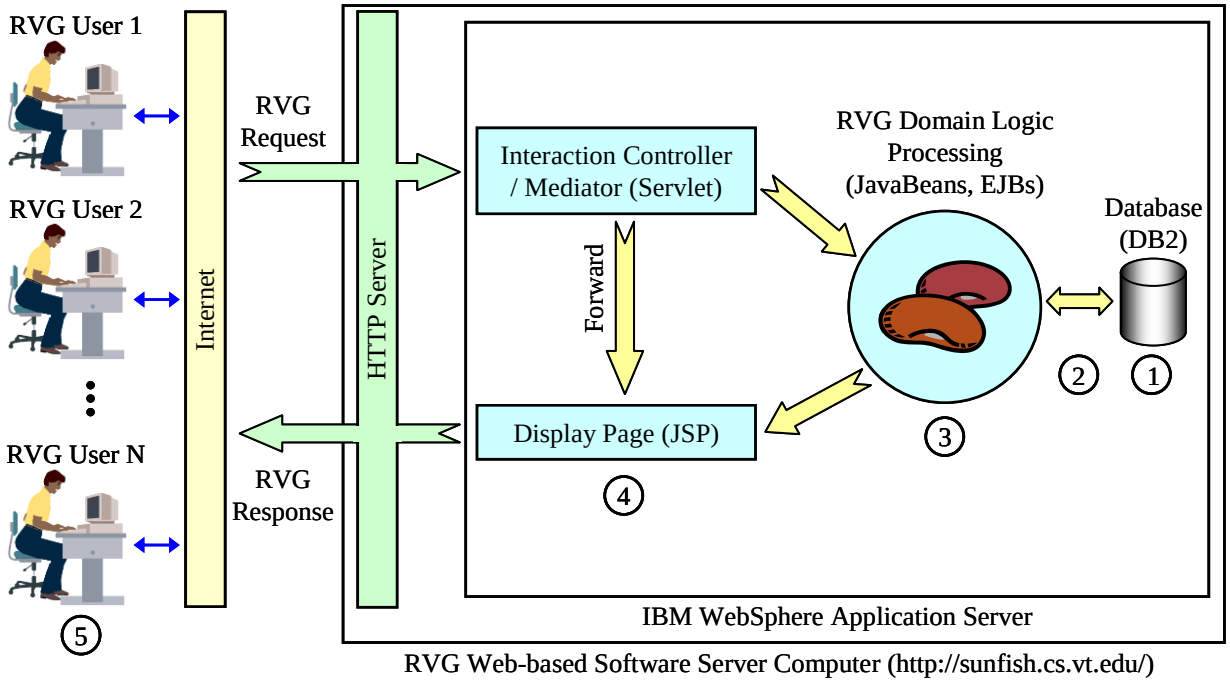

Figure 10. RVG Web-based software architecture

The architecture layers are shown by the circled numbers in Figure 10 and described below.

- Layer 5: *Client-Side Presentation* (HTML, JavaScript)
- Layer 4: *Server-Side Presentation* (Java Server Pages (JSPs))
  Controller/Mediator (Servlets)
- Layer 3: *Domain Logic* (Enterprise JavaBeans (EJBs), JavaBeans)
- Layer 2: *Data Mapping* (Java DataBase Connectivity (JDBC), Entity EJBs)
- Layer 1: *Data Source* (DB2 relational database)

The RVG Web-based software is developed as a two-tier client/server system using the IBM WebSphere Studio Application Developer. IBM WebSphere Application Server is used to run the RVG software on a server computer together with the IBM DB2 relational database management system.

The RVG Web-based software provides the same functions described in section 2.1. It can be used by a Web browser (e.g., Internet Explorer, Netscape) for a variety of purposes including (a) generating random variates and using them in another application by copy-and-paste or by saving them into a file, (b) teaching simulation input data modeling, probability distributions, and RVG, and (c) creating a histogram and a scatter plot of the generated random variates in the SVG format and using them in other applications.

4. VERIFICATION AND VALIDATION

Verification and validation (V&V) deal with the assessment of accuracy of the RVG Web service and RVG Web-based software. *Verification* deals with the assessment of transformational accuracy and *validation* deals with the assessment of representational or behavioral accuracy [11]. We performed V&V hand in hand with the development life cycle.

The RVG algorithms are created based on the specifications given in the textbook by Law and Kelton [1]. The textbook specifications of the RVG algorithms are programmed into an executable code by using the Java programming language. We used desk checking, structured walkthroughs, and inspections [11] to assess the accuracy of the transformations (i.e., verification) of the RVG algorithms from their textbook specifications into their executable forms in Java.

Validation of the RVG service and software was performed by using the *Comparison Testing* technique [11]. We used the ExpertFit commercial software product [12] to perform the comparison testing of the RVG algorithms, histograms, scatter plots, and general statistics. ExpertFit is used to identify which probability distribution best represents a data set. It supports all of the 27 probability distributions from which random variates can be generated by the RVG Web service and software. The comparison testing was conducted by using the procedure presented in Figure 11.

Perform for j = 1, 2, 3, …, 27

Step 1: Using the RVG Web-based software

Step 1.1: Generate 7000 random variates from probability distribution j

Step 1.2: Create a histogram of the 7000 random variates

Step 1.3: Create a scatter plot of the 7000 random variates

Step 1.4: Create general statistics for the 7000 random variates

Step 2: Feed the 7000 random variates into ExpertFit

Step 2.1: Identify the best probability distribution representing the variates.
If ExpertFit does not identify the probability distribution j as the best fit **then**

{The RVG software contains a bug. Fix the bug and redo the testing.}

Step 2.2: Create a histogram of the 7000 random variates.
If the histogram is not the same as the one created by the RVG software **then**

{The RVG software contains a bug. Fix the bug and redo the testing.}

Step 2.3: Create a scatter plot of the 7000 random variates.
If the scatter plot is not the same as the one created by the RVG software **then**

{The RVG software contains a bug. Fix the bug and redo the testing.}

Step 2.4: Create general statistics for the 7000 random variates.
If the general statistics are not the same as the ones created by the RVG software **then**

{The RVG software contains a bug. Fix the bug and redo the testing.}

endPerform

Figure 11. Comparison testing procedure

We performed the testing procedure in Figure 11 numerous times and found no bugs in the RVG Web service and RVG Web-based software.

5. CONCLUSIONS

An RVG Web service and an RVG Web-based software are developed to generate random variates from 27 probability distributions and create histograms and scatter plots with general statistics for the generated random variates. A Web-based stochastic simulation application can invoke the RVG Web service by sending a request message formatted in XML according to a published request schema. In response, the RVG Web service sends the service results, formatted in XML according to a published reply schema, to the requesting simulation application.

The RVG Web service and the RVG Web-based software are developed using open technology standards such as J2EE, XML, SOAP, WSDL, HTML, and SVG and open design frameworks such as struts.

The RVG Web service provides reusability and enables interoperability for Web-based simulation applications. The RVG Web-based software can be used with a Web browser for a variety of purposes including the use of the generated random variates and their histograms, scatter plots, and general statistics in other applications as well as for teaching purposes.

REFERENCES

1. Law, A.M. and Kelton, W.D. (2000) *Simulation Modeling and Analysis*, Third Edition, McGraw-Hill, New York, NY.

2. Banks, J., Carson, J.S., Nelson, B.L. and Nicol, D.M. (2001) *Discrete-Event Simulation*, Prentice Hall, Upper Saddle River, NJ.

3. Tao, L. (2001) 'Shifting paradigms with the application service provider model' *IEEE Computer*, Vol. 34, No. 10 (Oct.), pp. 32-39.

4. Wiedemann, T. (2001) 'Simulation application service providing (SIM-ASP)', *Proceedings of the 2001 Winter Simulation Conference*, Arlington, VA, pp. 623-628.

5. Kilgore, R.A. (2002) 'Simulation Web services with .Net technologies', *Proceedings of the 2002 Winter Simulation Conference*, San Diego, CA, pp. 841-846.

6. Chandrasekaran, S., Silver, G., Miller, J.A., Cardoso, J. and Sheth, A.P. (2002) 'Web service technologies and their synergy with simulation', *Proceedings of the 2002 Winter Simulation Conference*, San Diego, CA, pp. 606-615.

7. Fishwick, P.A. (2002) 'Using XML for simulation modeling', *Proceedings of the 2002 Winter Simulation Conference*, San Diego, CA, pp. 616-622.

8. Chung, J.-Y., Lin, K.-J. and Mathieu, R.G. (2003) 'Guest editors' introduction: Web services computing: advancing software interoperability' *IEEE Computer* , Vol. 36, No. 10 (Oct.), pp. 35-37.

9. Turner, M., Budgen, D. and Brereton, P. (2003) 'Turning software into a service' *IEEE Computer*, Vol. 36, No. 10 (Oct.), pp. 38-44.

10. McCarthy, J. (2002) 'Reap the benefits of document style Web services', IBM, June. http://www-106.ibm.com/developerworks/webservices/library/ws-docstyle.html

11. Balci, O. (1998) 'Verification, validation, and testing', in J. Banks (ed.), *The Handbook of Simulation*, John Wiley & Sons, New York, NY, Aug., pp. 335-393.

12. Law, A.M. and Vincent, S. (1995) *ExpertFit User's Guide*, Averill M. Law & Associates, Tucson, AZ.