Random Variate Generation Web Service

Mohammad Sabah

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science

August 29, 2003
Blacksburg, Virginia

Keywords: Random variate generation, model view controller, design patterns, Web service, J2EE, WSDL, SOAP, SVG

Random Variate Generation Web Service

Mohammad Sabah

Abstract

Simulation and statistical applications often mimic the behavior of a random phenomenon by way of generating random observations that form a known or empirical probability distribution with estimated parameter values. Generation of such random observations is called *Random Variate Generation* (RVG). The number of simulation and statistical applications provided on the World Wide Web (Web) is on the rise. To facilitate the development of simulation and statistical applications on the Web by way of reuse, there is a need for providing RVG as a Web service. This research involves the development of such a Web service for RVG, which can be invoked programmatically over the Web by using SOAP over the HyperText Transfer Protocol (HTTP) running on top of the Internet. To provide the RVG Web service, an RVG Web application is developed based on the Java 2 Enterprise Edition (J2EE) architecture. The RVG Web application is engineered by using the *IBM WebSphere Studio Application Developer* and runs on the *IBM WebSphere Application Server*. A client simulation and statistical application may call the RVG Web service and request the generation of random variates from 27 probability distributions. In addition, the RVG Web service also provides general statistics, scatter plot, and histogram of the requested random variates. The plots and histograms are created in Scalable Vector Graphics (SVG). The RVG Web service: (a) accepts requests in the Extensible Markup Language (XML) format, which is specified according to a request schema, and (b) sends the results to the client application also in the XML format specified according to a reply schema. The interface specification and access information needed to invoke the RVG Web service are provided in the Web Service Description Language (WSDL) document. Any Web-based simulation or statistical application that needs generation of random variates, their scatter plots and histograms, can invoke the RVG Web service programmatically at http://sunfish.cs.vt.edu/RVGWebService .

Acknowledgments

First of all, I would like to thank my parents for their love and support all through my life. Second, I would like to thank Dr. Osman Balci for being my advisor in this challenging and interesting work, and for ideas and guidance at all stages. Third, I would like to thank my friends who have been with me through this endeavor, and helped make my stay at Virginia Tech a mixed balance of work and fun.

Table of Contents

List of Figures

List of Tables

List of Code Listings

List of Acronyms

CORBA	Component Object Request Broker Architecture
EJB	Enterprise JavaBean
EIS	Enterprise Information Services
ERP	Enterprise Resource Planning
FTP	File Transfer Protocol
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
J2EE	Java 2 Enterprise Edition
J2ME	Java 2 Mobile Edition
J2SE	Java 2 Standard Edition
JDBC	Java Database Connectivity
JFC	Java Foundation Classes
JMS	Java Messaging Service
JNDI	Java Naming and Directory Interface
JSP	Java Server Pages
JVM	Java Virtual Machine
MIDP	Mobile Information Device Profile
RDBMS	Relational Database Management System
RMI	Remote Method Invocation
RMI/IIOP	Remote Method Invocation/Internet Inter-Orb Protocol
RPC	Remote Procedure Call
RVG	Random Variate Generation
SMTP	Simple Mail Transfer Protocol
SVG	Scalable Vector Graphics
UDDI	Universal Description, Discovery and Integration
URL	Uniform Resource Locator
URI	Uniform Resource Identifier
WAS	WebSphere Application Server
WSAD	WebSphere Studio Application Developer
WSDL	Web Service Description Language
XML	Extensible Markup Language

Chapter 1. Introduction

Random Variate Generation (RVG) is an integral part of the domain of discrete event system simulation. It comes into play whenever there is a need to simulate the uncertainty in behavior of an entity in the system. If the pattern of randomness of the entity is known, the entity is said to follow a particular probability distribution; otherwise, the randomness is simulated by empirical methods.

1.1. Random Variate Generator

A random phenomenon is characterized by:

- Collection of data on the random phenomenon,
- Fitting the collected data to a probability distribution, and
- Estimation of parameters of the probability distribution.

Computer simulation of the random phenomenon begins once the random phenomenon has been characterized. For example, if the random phenomenon is arrival of jobs to a computer system, the inter-arrival times of jobs to the system can be characterized to follow an exponential distribution with a known mean. Simulating this random phenomenon requires random values for inter-arrival times of jobs in such a way that they form an exponential distribution with the same mean.

Formally speaking, a *random variable* is a real-valued function that maps a sample space into the real line. For example, inter-arrival time of jobs to a computer system can be a random variable in a system. *Random variate* refers to a particular value of a random variable. RVG deals with generation of random variates for a given random variable, in such a way that the generated values form the probability distribution of the random variable. The algorithm that is responsible for carrying out random variate generation is called a *Random Variate Generator*. For example, a random variate generator for the exponential distribution generates random variates that satisfy exponential probability distribution. Random variate generators are of two kinds: univariate and multi-variate. A univariate RVG involves the generation of a single variate at a time; a multi-variate involves generation of a vector of variates at a time, that do not demonstrate mutual independence.

This thesis is concerned with implementation of univariate random variate generators.

1.2. Enterprise Application Development

In its most generic form, the term *enterprise* refers to an organization that has set out to accomplish certain goals. The organization may be small, medium or large-scale corporation, an educational institution or any other non-profit organization. *Enterprise applications* can be thought of as “business software” in layman terms that provide a host of functionality like user interface, communication services, and application and Web enabling.

Enterprise applications typically are different from other software in a lot of respects [Fowler 2002]:

- *Persistent Data*: Data needs to be preserved for multiple runs of the program, often for a number of years.
- *Lots of Data*: All enterprise applications use some of kind of data store to archive the large amount of data that needs to be preserved.
- *Data concurrency*: A small-scale enterprise application may have hundreds of concurrent accesses, while for a Web-based system, this number goes up by many orders of magnitude. Handling concurrency of data access and transaction processing are prime concerns of all enterprise applications.
- *Lots of User Interface Screens*: A large number of screens are typically needed in enterprise applications, to handle presentation of the enormous amount of data. Furthermore the screens have to be tailored for users with different technical expertise.
- *Integration with other enterprise applications*: On many occasions, enterprise applications need to be integrated with other applications built using different technologies and at different times.
- *Conceptual Dissonance*: Even within the enterprise, an entity may be viewed differently by different divisions. This creates a non-trivial problem when there are hundreds of records with the fields having subtly different interpretations.
- *Complex business logic*: The rules of the application are driven by the business needs, and are convoluted and complicated.
- *Distributed Application*: The components of the application very often span different nodes. Hence issues related to distributed processing and computing need to be considered.
- *Scalability*: An enterprise application should have a robust architecture so that it scales well to thousands of users as in the case of Web applications.
- *Maintaining Security*: Many applications also require that the users be authenticated, and authorized before carrying out a subset of the operations.

This thesis is concerned with development of Web service and a Web application using the Java 2 Enterprise Edition (J2EE) architecture for generating random variates.

1.3. Web Services

Web Service is to application-to-application interaction as Web was to program-to-user interaction. Essentially, Web services are a distributed computing paradigm and allow applications written in diverse languages, and running on multiple platforms to interoperate and integrate more easily and less expensively than other traditional methods. Extensible Markup Language (XML) forms the backbone of the Web Service technology. All message interactions occur at a level much higher than the network stack, depending more on the service level semantics between the applications. The Web service protocols are built on top of application-level network protocols like Hyper Text Transfer Protocol (HTTP), Simple Mail Transfer Protocol (SMTP), and File Transfer Protocol (FTP).

SOAP and Web Services Description Language (WSDL) are the component protocols used in Web services. Use of Universal Description, Discovery and Integration (UDDI), which provides SOAP-based APIs to publish and find Web services at a central location or registry, is optional and provides one way for the Web service provider to make the Web service available to potential requestors.

This thesis is concerned with the development of RVG Web service, which can be invoked by Web-based simulation and statistical applications, using SOAP over HTTP.

1.4. Our Work

Our work involves development of the RVG Web service that supports 27 random variate generators, and can be invoked by Web-based simulation and statistical applications. Underlying the Web service is the RVG Web application, a J2EE-based enterprise application deployed on WebSphere Application Server (WAS). The RVG Web application provides a Graphical User Interface (GUI), allowing the user to enter values for input parameters of the random variate generators, and simulating the programmatic interactions that occur between the RVG Web service client and the RVG Web service. In addition to generating random variates, the RVG Web service gives the calling application the option to request for:

- generation of histogram (frequency-plot of the random variates along different intervals),
- generation of scatter plot (plot that identifies correlation between the generated random variates), and
- generation of general statistics on the random variates.

The RVG Web service accepts an XML file (SOAP-literal-encoded) in a well-defined format from the requesting application, and returns to it another XML file (SOAP-literal-encoded) in another well-defined format. Both the input and output formats, along with the Uniform Resource Locator (URL) of the Web service, are included as part of the RVG Web Service Description Language (WSDL) document that the requesting application must download to invoke RVG Web service. Through a sample RVG Web service client, the thesis demonstrates the RVG Web service invocation and usage.

Overall Architecture

Figure 1 shows the overall architecture of the RVG Web application and Web service. The deployment machine, which is depicted as “RVG Web service provider” in Figure 1 runs IBM WebSphere Application Server, IBM HTTP Server and IBM DB2. The underlying business and domain logic are encapsulated in the session and entity enterprise beans, and are exposed to client invocations as RVG Web service. All communication between the requesting application and the RVG Web service takes place using SOAP. The RVG WSDL document contains all the information that the client applications need to invoke the RVG Web service.

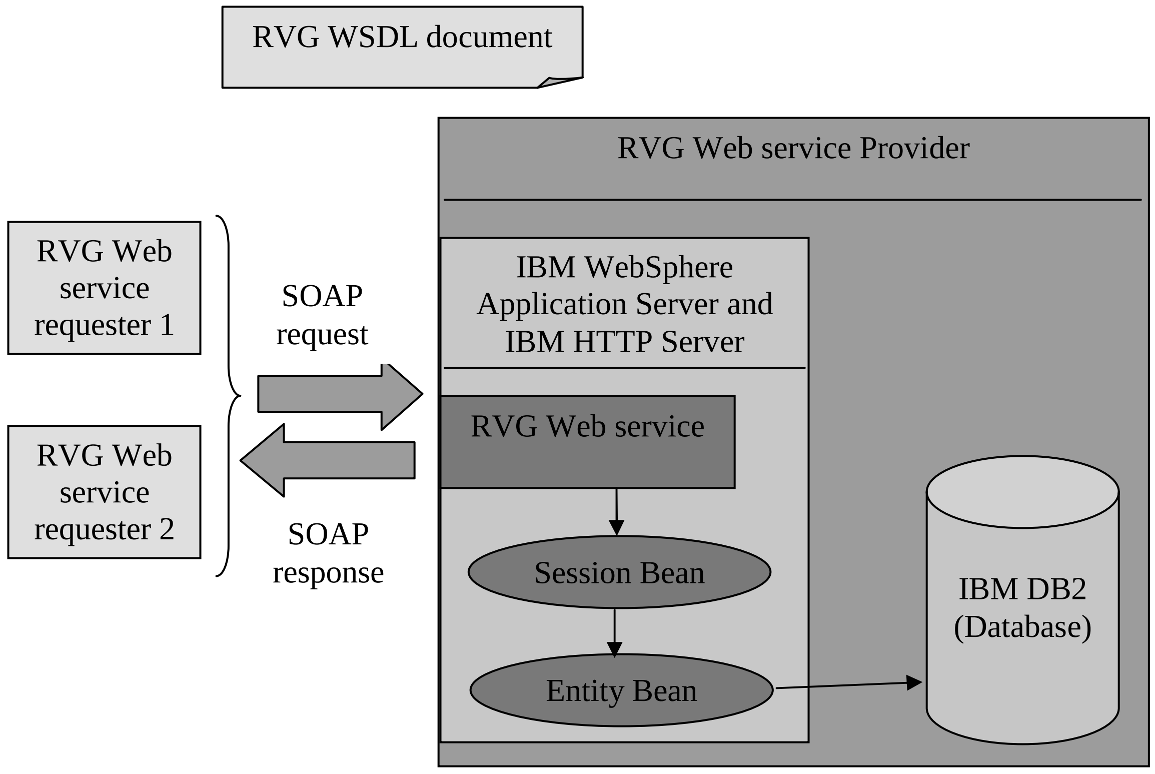


Figure 1. RVG Web service: Overall architecture

1.5. Organization

The thesis is organized as follows:

- Chapter 2 provides the background for the work and covers topics of Random Variate Generation, J2EE, Web Services, and SVG.
- Chapter 3 provides a high-level architectural view of RVG Web application, and discusses the design patterns that have been combined to implement each tier of the application.
- Chapter 4 provides a detailed design of RVG Web application, and shows how the classes used in the application implement the architecture presented in Chapter 3.
- Chapter 5 provides information about RVG Web service which is tested through a sample Web service client. This chapter also explains the structure of WSDL file for the Web service.
- Chapter 6 provides information about the activities carried out for the verification and validation of RVG Web application and RVG Web service.
- Chapter 7 provides a summary of the work done as part of this research, along with suggestions for improvement.

Chapter 2. Background

This chapter provides background information on RVG, J2EE, Web services and Scalable Vector Graphics (SVG).

2.1. Random Variate Generation

In stochastic simulation experiments as well as in statistical applications, there is often a need to generate random observations from known probability distributions. These random observations of a random variable that form a desired probability distribution are called *random variates*, and the algorithm that generates random variates for a particular probability distribution is called a *Random Variate Generator*. Examples of random observations include inter-arrival times of objects, processing times of a server, and failure times of a machine.

Statistical and simulation applications need the inherent random phenomena to be characterized before running the model. A random phenomenon is characterized by gathering observations, fitting an appropriate probability distribution to the observations, and estimation of the parameters of the fitted probability distributions. Once the random phenomenon has been characterized, the model invokes the appropriate random variate generator with the estimated values of parameters. The generated random variates are then used to drive the model.

There are well-known algorithms that exist for random variate generation for common probability distributions; some probability distributions have alternative algorithms. The next section discusses general approaches to the design of random variate generators.

2.1.1. General Approaches to Random Variate Generation

There are many techniques for random variate generation for different probability distributions; however, based on their theoretical bases, nearly all the techniques can be classified into one of the following general approaches.

2.1.1.1. Inverse Transformation Method

Let X be a continuously distributed random variable. Then its distribution function is given by $F(x)$, called the cumulative density function, having the property:

$$\Pr\{a \le X \le b\} = \int_a^b f(x)dx$$

The distribution function can be evaluated from the density function as

$$F(x) = \int_{-\infty}^{x} f(z)dz$$

The distribution function being monotonically increasing and continuous, the equation $F(x) = u$ (for any $0 < u < 1$) can be solved to give a unique x. Or written alternatively, $X = F^{-1}(u)$. This essentially is the inverse transformation method for generating random variates.

Algorithm:

1) Generate $U \approx U(0,1)$.
2) Return $X = F^{-1}(u)$.

The inverse transformation method for discrete random variates is similar. Suppose that X can take only the value $x_1, x_2, ..., x_n$ with probabilities $p_i = \Pr(X = x_i)$. The distribution function is given by

$$F(x) = \Pr(X \le x) = \sum_{x_i \le x} p(x_i)$$

The inverse is defined by $F^{-1}(u) = \min\{x \mid u \le F(x)\}$.

Algorithm:

1) Generate $U \approx U(0,1)$.
2) Determine the smallest positive integer i such that $U \le F(x_i)$ and return $X = x_i$.

2.1.1.2. Convolution Method

This method is used when the random variate X can be expressed as a sum of other random variates. Thus this can be written as: $X = Y_1 + Y_2 + ... + Y_m$. The assumption here is that $Y_1 + Y_2 + ... + Y_m$ has the same distribution as X.

Algorithm:

1) Generate $Y_1, Y_2, ..., Y_m$ each with distribution function G.
2) Return $X = Y_1 + Y_2 + ... + Y_m$.

This method is straightforward provided that generation of each Y_j is easy. However it is not the most efficient algorithm. Convolution can be considered a special case of the more general method of transforming some intermediate random variates into the final variate required.

2.1.1.3. Composition Method

This method may look similar to the “Convolution” technique discussed above, but is fundamentally different. This is used when the distribution function F can be expressed as a weighted sum of r other densities:

$$F(x) = \sum_{j=1}^{\infty} \mathrm{p}_j \mathrm{F}_j(\mathrm{x}) \qquad \text{where} \quad p_j \geq 0, \sum_{j=1}^{\infty} p_j = 1$$

In the above, each F_j is a distribution function and the density F is called compound or mixture density. An example of a compound density is a queuing simulation where the inter-arrival times of customers are dependent upon the properties of the different customers.

Algorithm

1) Generate a positive random integer J such that $P(J = j) = p_j$, for j = 1, 2, …
2) Return X with distribution function F_J.

2.1.1.4. Acceptance-Rejection Method

This approach is used when the above three methods cannot be used for the generation of random variates for a particular probability distribution. It is less direct than the other approaches. There is one assumption that needs to be made regarding the density function $f(x)$: There is another function $e(x)$ that dominates $f(x)$, in the sense that $e(x) \geq f(x)$. The function e is said to majorize f.

It is possible to generate points uniformly scattered under the graph $e(x)$. Let us denote the coordinate of a typical point by (X,Y). If the graph of $f(x)$ is drawn on the same diagram, the point (X,Y) will be above or below it according as $Y > f(x)$ or $Y \leq f(x)$ [Banks 1998].

The method works by generating points (X,Y) and returning the X-coordinate only if the point lies under $f(x)$. Let us denote by $r(x)$ the function: $r(x) = e(x)/c$, such that $c < \infty$.

Algorithm:

1) Generate Y having density r.
2) Generate $U \approx U(0,1)$, independent of Y.
3) If $Y \leq f(Y)/e(Y)$, return $X = Y$. Otherwise go to step 1.

Requirements for constructing e(x):

- The area between f and e should be small to keep the proportion of rejected points small.
- It should be easy to generate uniformly distributed points under $e(x)$.

The average number of points (X, Y) needed to produce one acceptable X is called the *trials ratio*. The closer the trials ratio is to 1, the more efficient the generator.

2.1.1.5. Special Properties

Some techniques of random variate generation do not fall into any of the above four methods. The needed random variate is generated by some transformation of another random variate. Convolution can be considered a special case of the use of "special properties" to generate random variates. For example, a lognormal variate is simply an exponentiated normal variate. There exists a broad class of such transformations called *location-scale* models, which is explained below:

Let X be a continuously distributed random variate with density $f(x)$. Then we can rescale and reposition the distribution by the following linear transformation: $Y = aX + b$, where a is a given scale constant, and b is a given location constant.

2.1.2. Supported Probability Distributions

A random variate can either take on any value in some interval of the real line (called a *continuous* random variate), or take on some subset of non-negative integers (called a *discrete* random variate). Based on this differentiation, the 27 probability distributions for which random variate generators have been implemented in this work fall into one of the following categories:

Non-negative Continuous Distribution

This probability distribution restricts the random variate to be strictly larger than a specific lower bound value.

Bounded Continuous Distribution

This probability distribution restricts the random variate to take on values strictly within a range viz. strictly larger than a specific lower bound value, and strictly lower than a specific upper bound value.

Unbounded Continuous Distribution

This probability distribution places no restriction on the values that the random variate can assume.

Discrete Distribution

This probability distribution restricts the random variate to take on values from a subset of the set of non-negative integers.

Table 1 presents the 27 probability distributions for which random variate generators are supported in the RVG Web application and RVG Web service.

Table 1. Implemented Probability Distributions and their Types

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 Continous	Extreme Value Type A Extreme Value Type B Johnson SU Laplace Logistic Normal
Discrete	Binomial Discrete Uniform Geometric Negative Binomial Poisson

2.2. J2EE Overview

The Java 2 Enterprise Edition (J2EE) platform represents a standard for building and deploying enterprise applications. The J2EE platform has been developed as an open-source process through contributions from experts in the field of enterprise applications. The platform provides client-side and server-side services to build multi-tiered and distributed applications which are designed with the following tiers:

- Client tier – provides the user interface
- Middle tier – provides business logic and client services
- Back-end – enterprise information systems that provide data management.

2.2.1. Main Features

This section highlights some of the main features of the J2EE platform.

Multitier Model

As illustrated in Figure 2, the J2EE platform provides a multitier application model – the client tier, the middle tier, and the back-end tier. The various components of the application may run on different devices. The client tier supports clients, both outside and inside the firewall. The middle tier supports client services and business logic through a *Web Container* (Web Tier) and an *EJB Container* (EJB Tier).

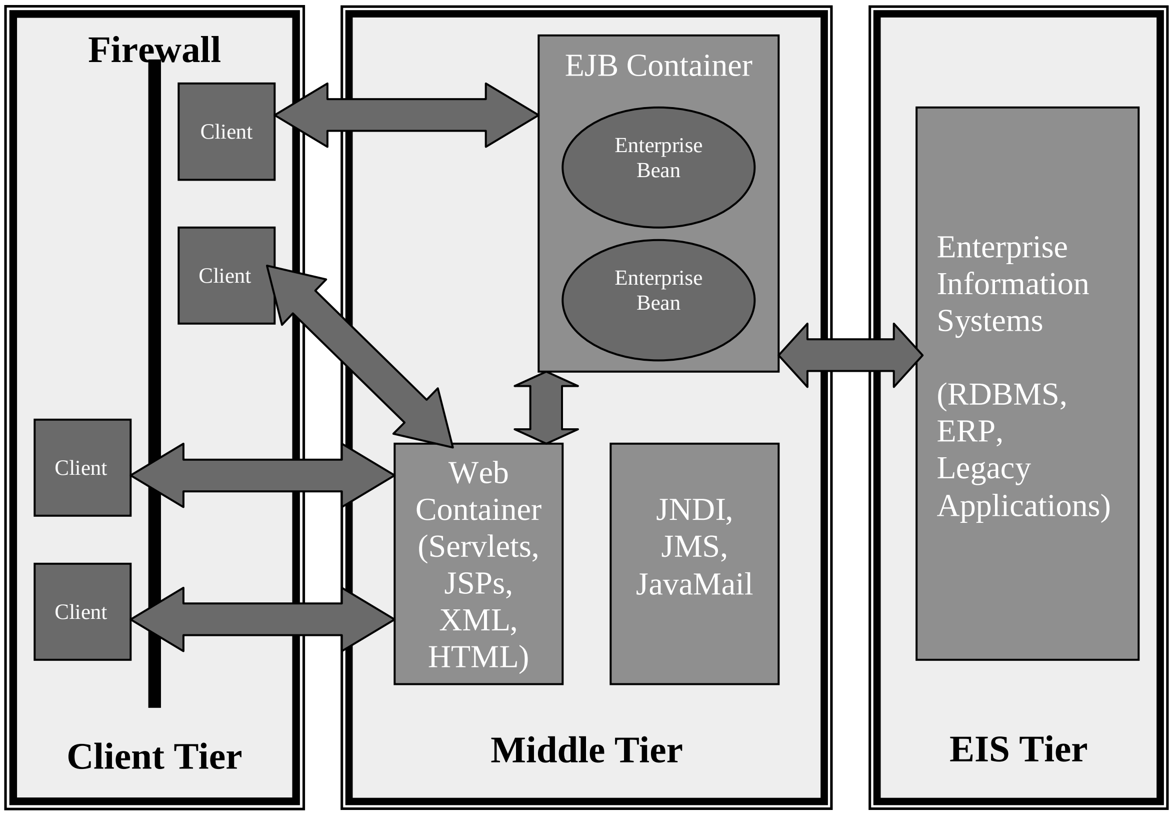


Figure 2. J2EE Architecture [Singh et al. 2002]

Container-based Component Management

The concept of *containers* is central to the J2EE platform. Containers are standardized environments that provide specified services, and are available on any J2EE-compliant vendor product. Examples of standard services include naming and directory services, life-cycle management, session management, and transaction management. Containers also provide a mechanism for selecting application behavior at assembly or deployment time through the use of *deployment descriptors*, which are XML files that hold configuration information pertaining to the deployed components and the vendor-specific container. Thus an application written in one J2EE-compliant server can easily be ported to another J2EE-compliant server by merely changing the deployment descriptors, without touching the component code. Examples of properties that can be configured through deployment descriptors include security and transaction management among others.

Support for Client Components

J2EE supports a variety of client types. Clients can be offered services through static HyperText Markup Language(HTML) pages, HTML generated through Java Server Pages (JSPs) or servlets, Java applets or as standalone applications.

Support for Business Logic Components

Business logic is implemented in the middle tier as Enterprise Java Beans (EJBs, or enterprise beans). The component provider concentrates on writing the business logic in the enterprise beans, leaving the complexities of transaction management, security management and scalability issues to the container. The EJB layer forms the backbone of industrial strength J2EE applications.

2.2.2. Component Technologies

The J2EE component supports the following kinds of components, in addition to Java Beans components that are a part of J2SE (Java 2 Standard Edition):

- Application Clients
- Applets
- Enterprise JavaBeans components
- Web components
- Resource Adapter Components

J2EE Client Types

The J2EE platform allows different kinds of clients to interact with server-side components:

- Applets: Applications that execute within a Web browser, and have access to all the features of Java.
- Application Clients: Applications that execute in their own *client containers*. It is the responsibility of the client container to provide J2EE services and handle Remote Method Invocation / Internet Inter-Orb Protocol (RMI/IIOP) such as Java Naming & Directory Information (JNDI) services, transaction services, asynchronous messaging services, Java Database Connectivity (JDBC) and the like.
- Java Web Start-enabled rich clients: Applications that have Java Foundation Classes (JFC)/Swing APIs and are enabled for J2EE platform through the Java Web Start technology.

- Wireless clients: Applications that are based on Mobile Information Device Profile (MIDP) for the Java 2 Mobile Edition (J2ME) platform.

Web Component

A Web component is an entity that takes in a request, and processes the response to be sent back. The J2EE platform supports two kinds of Web components, both of which use the services provided by the Web Container:

- Servlets: These Web components, executing within the Web Container, extend the capabilities of a Web server in an efficient way. The Web server contains mappings of Uniform Resource Locators (URLs) to the deployed servlet classes, so that whenever a request comes in for a particular URL, the corresponding servlet action gets invoked. The servlet processes the request by accessing the business tier components e.g. enterprise beans, or by directly accessing the database through JDBC APIs. Servlets also maintain session information on behalf of the user as well as enforce security in the Web Tier.
- Java Server Pages (JSP): These Web components work like servlets in producing dynamic content to requests. The rationale for creating JSP is to separate HTML formatting code from Java code doing business processing. A JSP contains template data (which is HTML or XML), and JSP elements and scriptlets (that contain embedded Java code). Like servlets, the JSPs also have access to the deployed enterprise beans in the business tier and directly to the database. Internally a JSP is compiled into a servlet by the JSP engine, and then executed.

Enterprise JavaBeans Component

Enterprise JavaBeans also known as enterprise beans are server-side components that contain business logic of the enterprise application. They are of three kinds: session beans, entity beans and message-driven beans. Session and entity beans have two kinds of interfaces: a home interface and a component interface. A home interface contains life-cycle management methods like creation of an EJB, finding an EJB (in the case of entity beans), removing an EJB and the like. The component interface contains the business methods exposed to their clients. The home and component interfaces can be either remote or local. Remote interfaces are Remote Method Invocation (RMI) interfaces providing location transparency i.e. the client can be running in the same Java Virtual Machine (JVM) as the enterprise bean or in a different JVM. Parameters and results are passed by value between the client and the remote enterprise bean which results in a serialization overhead. Local interfaces, on the other hand, run in the same JVM as their clients. Parameters and return values are passed by reference, and hence the serialization overhead is removed.

- Session enterprise bean: A session bean is created on behalf of a client, and exists for the duration of the client session. A session bean can be stateful or stateless. A stateful session bean maintains conversational state across methods and transactions, while a stateless session bean does not. However session beans are not persistent objects, unlike entity beans.

- Entity enterprise bean: An entity bean essentially represents data maintained in a data store, and is identified by a primary key. Persistence of entity beans are of two kinds – container managed, and bean managed. In container-managed persistence, the developer of the bean merely specifies the fields of the bean class that need to be made persistent, and lets the EJB container handle it without having to write any persistence logic in the bean source code. On the other hand, in the case of bean-managed persistence, the developer has to write persistence logic in the bean source.
- Message-driven enterprise bean: A message-driven bean allows asynchronous clients to access the business logic of the application. Message-driven beans are activated when a message arrives in the Java Messaging Service (JMS) queue on which they are listening. Unlike session and entity beans, message-driven beans do not have home and remote interfaces.

2.3. Web Services

Web Services are an integration model for applications that are not browser-based. In other words, Web services or XML Web services provide a foundation for building distributed applications, using component applications written in different programming languages and running on diverse devices and operating systems, all developed and deployed independently.

Web Services complement technologies for building distributed and component-based applications like J2EE and Component Object Request Broker Architecture (CORBA), in that, they act as a technology for *deploying* and *providing access* to applications implemented using J2EE, CORBA and other standards.

2.3.1. Definition of Web Services

Many definitions of Web services are found in the literature. One definition could be – "A Web Service is an interface that describes a collection of operations that are network-accessible through standardized XML messaging" [Heather 2001]. A more complete definition – "A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards" [Haas 2003]. The description of the service using XML notation is called *service description*, and includes all information necessary to communicate with the service like message formats, transport protocols and location. The service description hides the implementation details of the service, making Web-service-based applications loosely-coupled. The service description needs to be *published* at an accessible place where the interested parties can easily discover it.

2.3.2. Web Services Model

The Web Services model is based upon the interactions between three roles that the model mandates: service provider, service registry and service requestor.

- **Service Provider**: From a business perspective, this is the owner or the developer of the Web service; from an architectural viewpoint, this represents the platform that hosts the implementation of the Web service.
- **Service Registry**: This is the searchable registry of Web service descriptions where service providers publish their service descriptions, and service requestors find existing services.
- **Service Requestor**: From a business perspective, this is the business that requires certain operations to be completed. From an architectural viewpoint, this is an application that is looking to invoke the operations of a published Web service. The service requestor could be another Web service.

Figure 3 shows the different roles and their interactions in a Web Service scenario. A typical Web service invocation involves three operations:

- **Publish**: This operation refers to the Web service requestor storing information about the Web service at the Web service registry, where potential clients can find it.
- **Find**: This operation refers to the Web service requester querying the Web service registry for a particular service or type of Web service, or retrieving a Web service description directly. The *find* operation can be done either at design time (static) or at runtime (dynamic).
- **Bind**: This operation refers to the actual invocation of the Web service by the Web service requestor, using the details in the Web service description document.

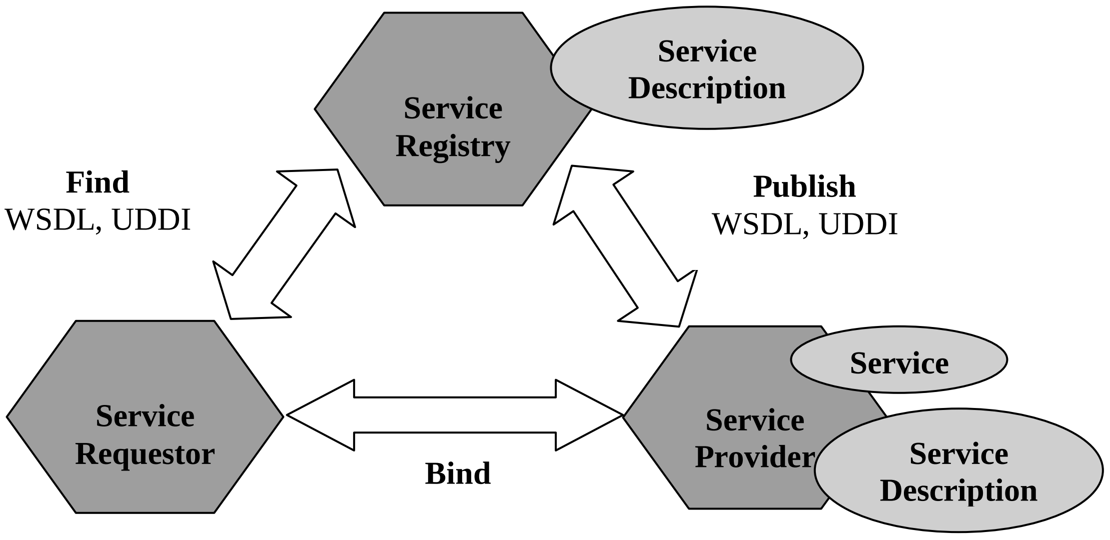


Figure 3. Web Services Architecture: Roles and Operations [Kreger 2001]

2.3.3. Web Services Protocol Stack

Figure 4 shows a conceptual Web Services stack. The upper layers build on the capabilities of the lower layers. The vertical towers represent requirements that must be satisfied at each layer of the stack.

- **Network**: The foundation of the Web services stack is the network. HTTP is the de-facto protocol used for Web services. Other protocols like File Transfer Protocol (FTP) and Simple Mail Transfer Protocol (SMTP) can also be supported.
- **XML-Based Messaging**: This layer represents the use of XML as the basis for a messaging protocol. SOAP is most often used as it is standardized and simple. Furthermore, SOAP messages support the *publish*, *find* and *bind* operations of Web services.
- **Service Description**: This layer represents the description documents, with Web Services Description Language (WSDL) being the de-facto standard. WSDL defines the interface and mechanics of Web service interaction.

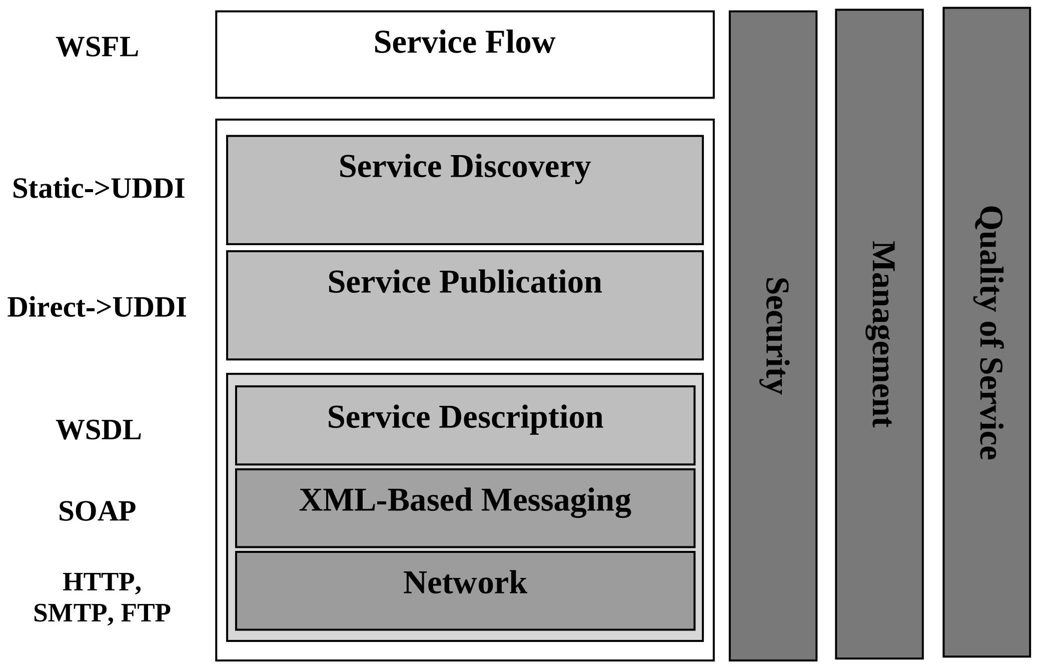


Figure 4. Web Services Conceptual Stack [Kreger 2001]

The above mentioned layers are required to provide or use any Web service, and forms the interoperable base stack that all inter-enterprise, or public, Web services must support. The present work uses HTTP for the network layer, SOAP for the XML-Based Messaging layer and WSDL for the service description layer.

The next two layers – service publication and service discovery – can be implemented in various ways.

- **Service Publication**: Any action that makes a WSDL document available to a Web service requestor can be classified as s*ervice publication*. The simplest way is *Direct Publication* viz. the Web service provider sending a WSDL document to a Web service requestor through email, for example. Alternatively, the Web service provider can publish the WSDL document in a private UDDI registry or UDDI operator node.
- **Service Discovery**: Any mechanism that allows a Web service requestor to gain access to the Web service description, and make it available at application runtime can be classified as *service discovery*. The simplest example of discovery is when the Web service requestor retrieves the WSDL document from a local file. Alternatively, the Web service can be discovered at design time or runtime of the application using a private UDDI registry or UDDI operator node.

2.4. Scalable Vector Graphics

The Scalable Vector Graphics (SVG) format is a new XML grammar for defining scalable vector-based 2D graphics for the Web and other applications, and usable as an XML namespace [SVG 2003]. Simply put, it is a text-only collection of XML commands for creating 2D graphics which can be edited in simple text editors.

Two of the most commonly used formats on the Web are GIF and JPEG, both of which are pixel-based, which lose quality when zooming. Furthermore, rendering a pixel-based image requires a lot of work as each pixel must hold information required to display an image. SVG, on the other hand, is based on an XML language to describe 2D graphics via vector graphics. Since it is based on vector graphics, an SVG image looks the same on zooming, and on all devices, whether it is displayed on a laptop or a handheld or even in printed form. Since SVG files are pure text, they are smaller than JPEG and GIF files, and hence can optimize browser's performance.

SVG also has a host of nice features like animation, filtering, masking, scripting and linking. It works well with Java technology, and complements Java's high-end graphics engine – the Java 2D API. Any parser that can read XML can also read SVG.

The current work has made use of the Batik SVG Toolkit and JFreeChart class libraries to generate and manipulate SVG images through Java.

2.4.1. Batik SVG Toolkit

"Batik is a Java technology based toolkit for applications or applets that want to use images in SVG format for various purposes, such as viewing, generation or manipulation" [Batik 2003]. It is an open-source framework that allows free access to the source code along with the binaries.

2.4.2. JFreeChart Class Libraries

JFreeChart [JFreeChart 2003] is another open-source initiative that provides a free Java class library for generating charts in Java like pie charts, scatter plot, and bar charts. The class library also supports exporting the charts to different formats like SVG.

Chapter 3. High-Level Design – RVG Web Application

This chapter presents the high-level architecture of the RVG Web application. Chapter 4 shows how this architecture is mapped to the classes in the RVG Web application.

The J2EE comes with a bunch of recommended best practices to be followed in designing an enterprise application to exploit the capabilities of the platform to the maximum. These best practices are presented through sample fully-developed applications, the most popular of which is the Java PetStore.

Before going into the high-level design and architecture of the application, it is worthwhile to lay down the requirements of the RVG Web application.

3.1. Requirements Specification for the RVG Web Application

The application essentially allows a user to request random variate generation for specified probability distributions. The 27 probability distributions that are supported are:

- Beta
- Binomial
- Discrete Uniform
- Exponential
- Gamma
- Geometric
- Extreme Value Type A
- Extreme Value Type B
- Inverse Gaussian
- Inverted Weibull
- Johnson SB
- Johnson SU
- Laplace
- Logistic
- Log-Laplace
- Log-Logistic
- Lognormal
- Negative Binomial
- Normal
- Pareto
- Pearson Type V
- Pearson Type VI
- Poisson
- Random Walk

- Triangular
- Uniform
- Weibull

RVG Web application supports four operations:

- Generation of random variates,
- Statistical data on the generated random variates,
- Histogram of the generated random variates, and
- Scatter Plot of the generated random variates.

Use case analysis was used to gather the requirements for RVG Web application.

3.1.1. Use Cases

Figure 5 shows the functional specification for the user interface of RVG Web application.

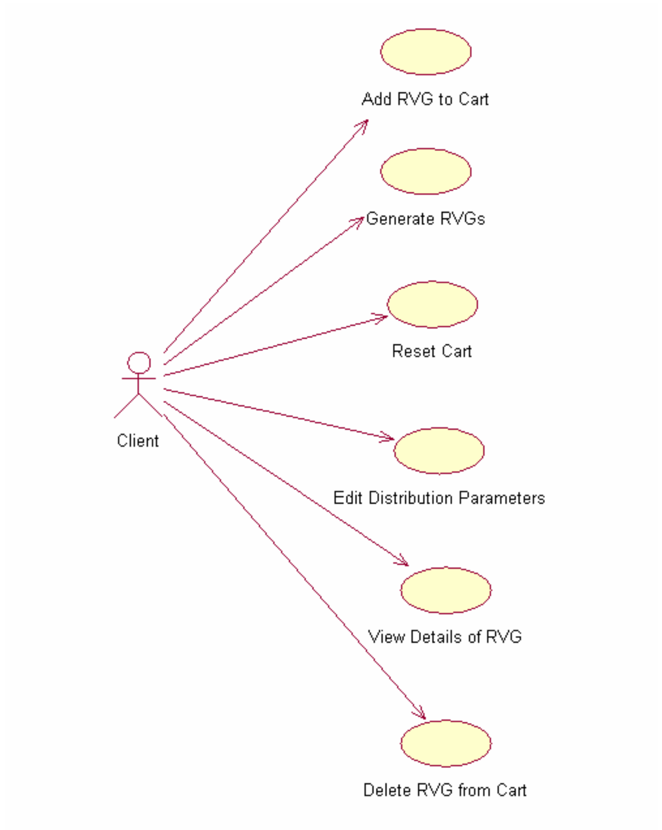


Figure 5. Use Case Diagram for RVG Web Application

Just as Web applications having customer transactions provide the feature of the ubiquitous “Shopping Cart”, RVG Web application provides the feature of a “Distribution Cart”. A distribution cart contains a summary of the probability distributions which the user has selected for random variate generation, or for whom the generation of random variates has been successful.

The user interface for RVG Web application has the following salient points:

- A navigation bar that allows the user to add an RVG to the distribution cart, generate RVGs from the distribution cart, reset the distribution cart, view general information about the project, and go to the home page of the application.
- An information bar that lists the contents of the distribution cart at all times. If the list displays requested RVGs, the user can edit the parameters of a specific RVG, or remove it from the distribution cart by clicking on “Edit Distribution” or “Delete Distribution”. On the other hand, if the list displays generated RVGs, the user can view the details of a generated RVG by clicking on “View Details”.
- The detailed view of a generated RVG comprises of General Statistics (if requested), Histogram (if requested), Scatter Plot (if requested), and a listing of the generated random variates.
- A header that displays the name of the RVG Web application.

3.1.2. User Interface Screens

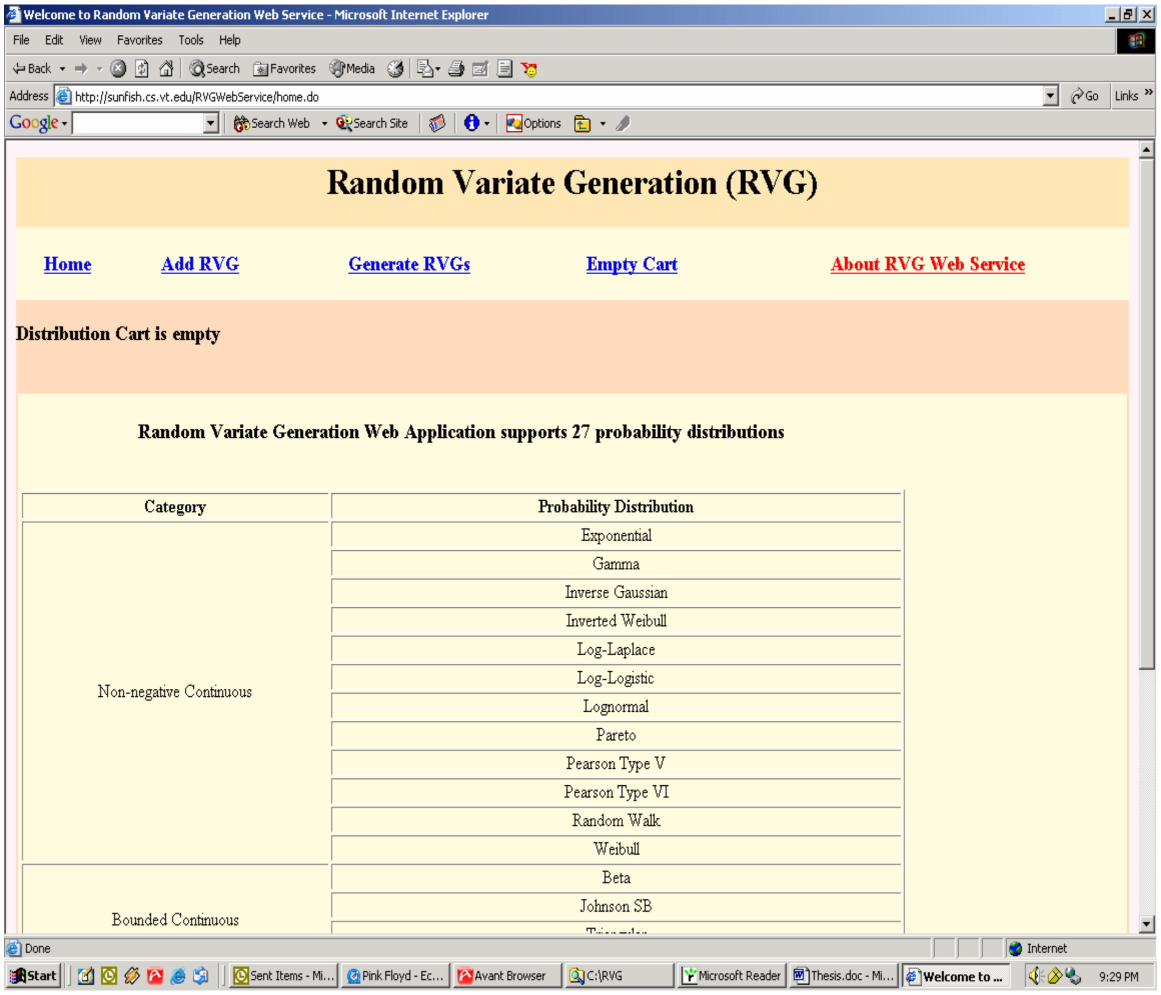
This section provides screenshots of some of the important screens of the RVG Web application Website.

Welcome Screen

The homepage of the application is located at http://sunfish.cs.vt.edu/RVGWebService . Figure 6 shows the home page of the application.

The home page has hyperlinks for:

- Home: Clicking this brings the user to the homepage of the application.
- Add RVG: Clicking this allows the user to add a probability distribution to the list of requested RVGs in the distribution cart.
- Generate RVGs: Clicking this generates random variates for the distributions in the distribution cart.
- Empty Cart: Clicking this empties the contents of the distribution cart.
- About RVG Web Service: Clicking this displays information about this research work.


Welcome to Random Variate Generation Web Service - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://sunfish.cs.vt.edu/RVGWebService/home.do
Go Links
Search Web Search Site Options
Done
Internet
Start Sent Items - Mi... Pink Floyd - Ec... Avant Browser C:\RVG Microsoft Reader Thesis.doc - Mi... Welcome to ...
9:29 PM

Random Variate Generation (RVG)

Home Add RVG Generate RVGs Empty Cart About RVG Web Service

Distribution Cart is empty

Random Variate Generation Web Application supports 27 probability distributions

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

Figure 6. RVG Web Application: Welcome Screen

Distribution Selection Screen

Clicking "Add RVG" brings the user to the Probability Distribution Selection Screen, where the user can select the distribution to generate random variates for. The selection screen is shown in Figure 7.

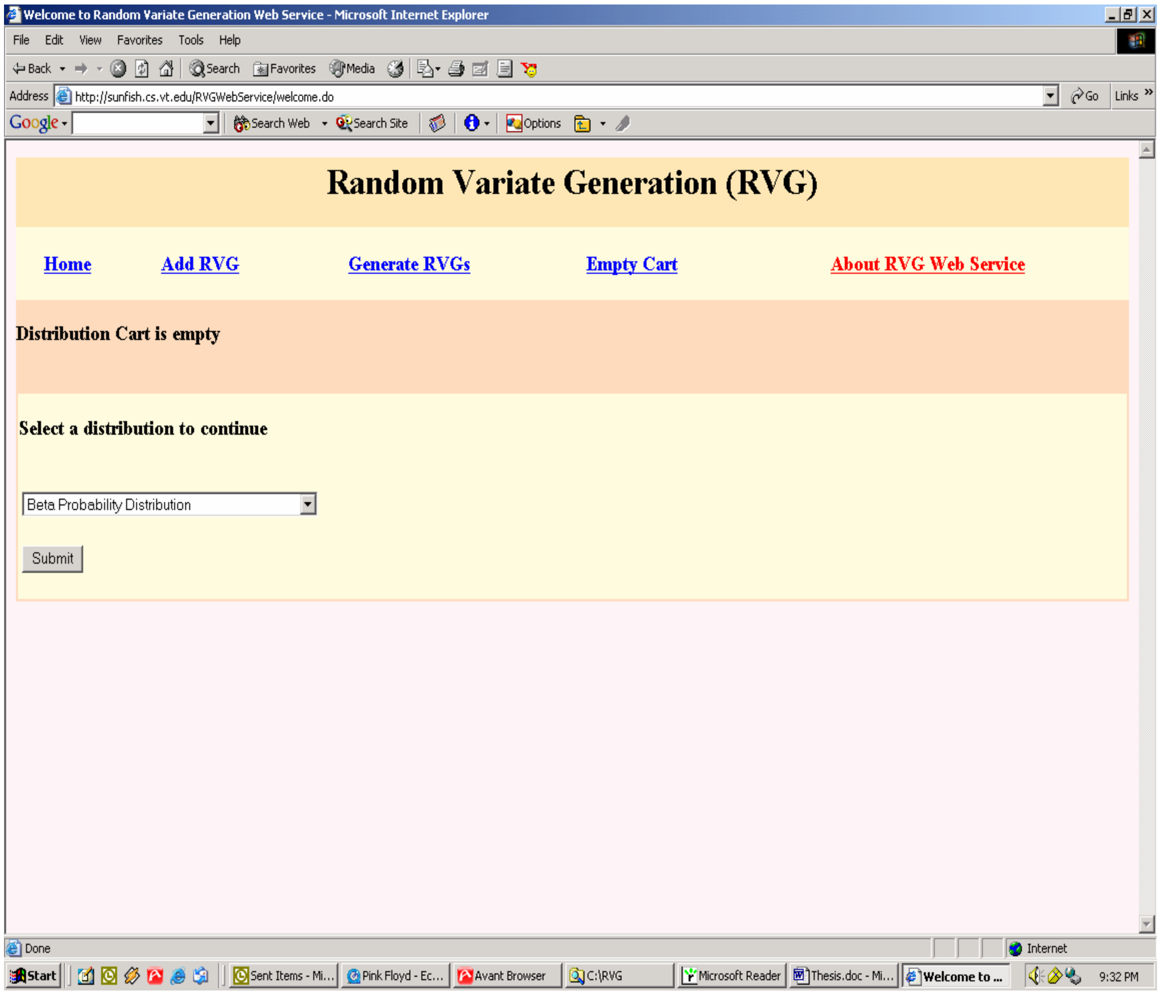

Welcome to Random Variate Generation Web Service - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://sunfish.cs.vt.edu/RVGWebService/welcome.do
Random Variate Generation (RVG)
Home
Add RVG
Generate RVGs
Empty Cart
About RVG Web Service
Distribution Cart is empty
Select a distribution to continue
Beta Probability Distribution
Submit
Done
Internet
9:32 PM

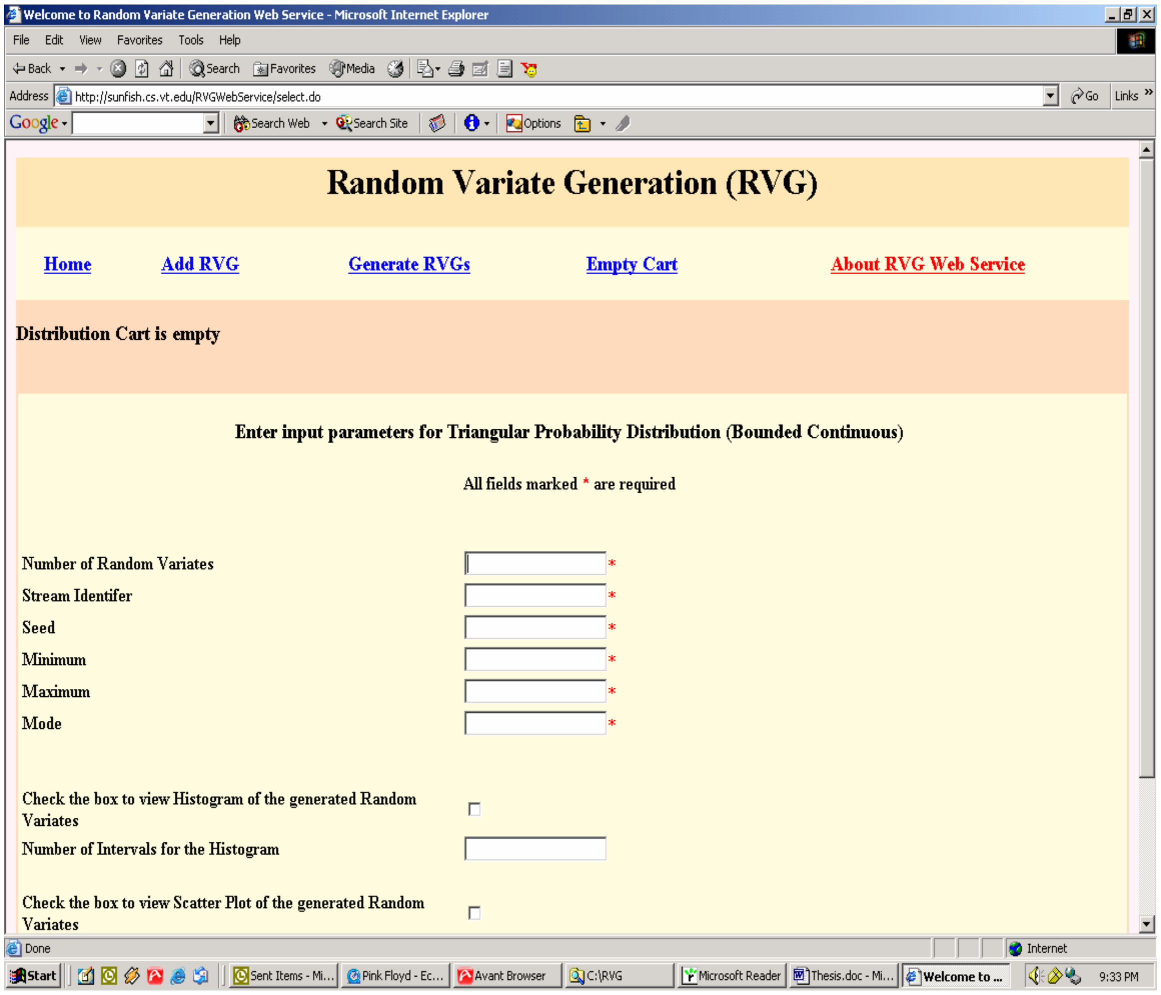
Figure 7. RVG Web Application: Probability Distribution Selection Screen

Distribution Input Screen

Selecting a particular distribution from Figure 7, and clicking "Submit" brings the user to the Input Screen for the selected distribution. Figure 8 shows the input screen for Triangular Probability Distribution.

The screen has three buttons:

- Add to Distribution Cart: Clicking this button validates the input parameters entered by user, and adds the distribution to the distribution cart on successful validation. Otherwise the user is prompted to correct the erroneous parameter values. An example of erroneous parameters is shown in Figure 9.
- Reset Input: Clicking this button clears the values input by the user, and refreshes the page.
- Cancel: Clicking this button brings the user to the home page, shown in Figure 6.


Welcome to Random Variate Generation Web Service - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://sunfish.cs.vt.edu/RVGWebService/select.do

Random Variate Generation (RVG)

Home | Add RVG | Generate RVGs | Empty Cart | About RVG Web Service

Distribution Cart is empty

Enter input parameters for Triangular Probability Distribution (Bounded Continuous)

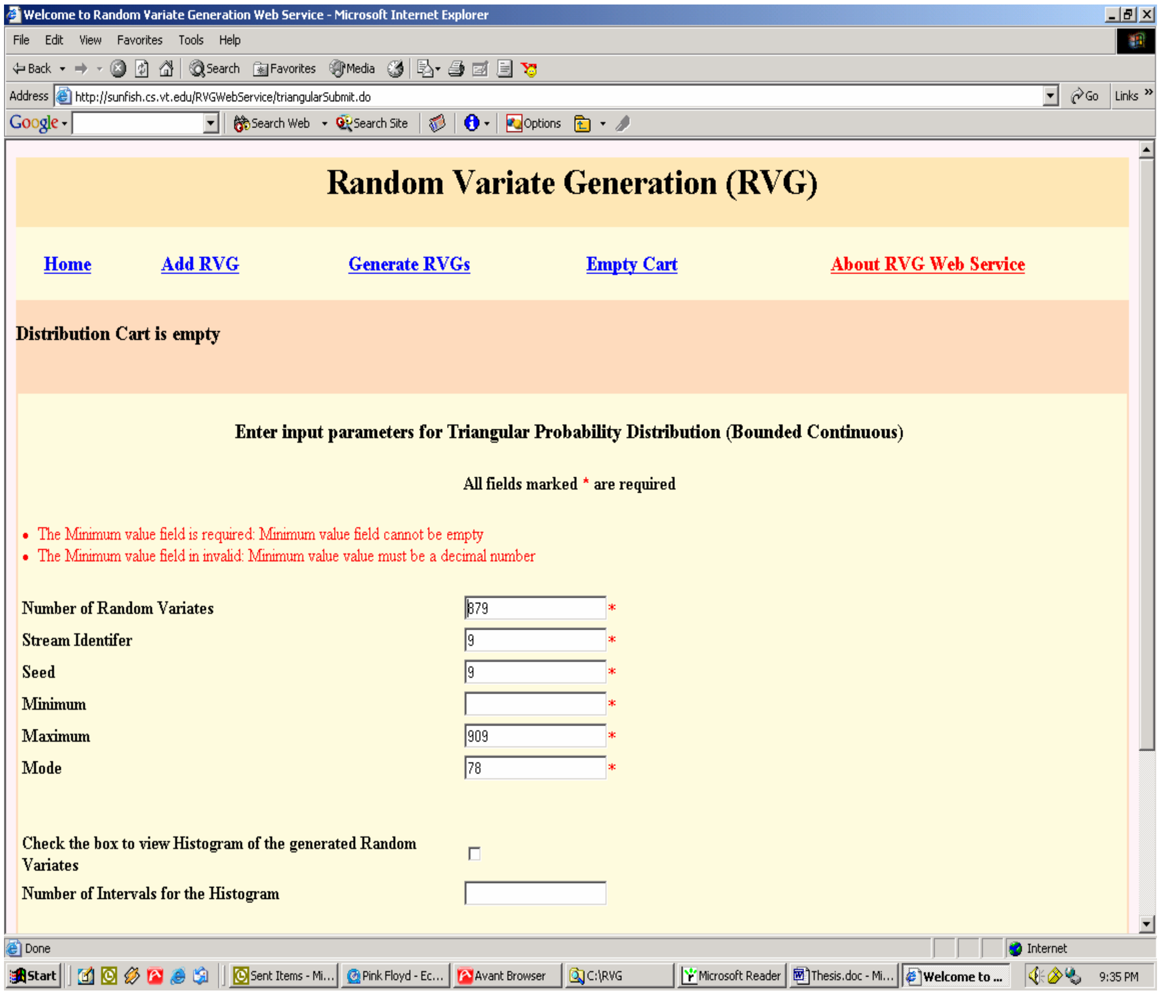
All fields marked * are required

Number of Random Variates	*
Stream Identifer	*
Seed	*
Minimum	*
Maximum	*
Mode	*
Check the box to view Histogram of the generated Random Variates	☐
Number of Intervals for the Histogram	
Check the box to view Scatter Plot of the generated Random Variates	☐

Figure 8. RVG Web Application: Distribution Input Screen

Distribution Cart – Requested Distributions

On successful validation of the input parameters, the distribution gets added to the distribution cart. The contents of the distribution cart show a summary of the requested RVGs, as shown in Figure 10.


Welcome to Random Variate Generation Web Service - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://sunfish.cs.vt.edu/RVGWebService/triangularSubmit.do

Random Variate Generation (RVG)

Home
Add RVG
Generate RVGs
Empty Cart
About RVG Web Service

Distribution Cart is empty

Enter input parameters for Triangular Probability Distribution (Bounded Continuous)

All fields marked * are required

- The Minimum value field is required: Minimum value field cannot be empty
- The Minimum value field in invalid: Minimum value value must be a decimal number

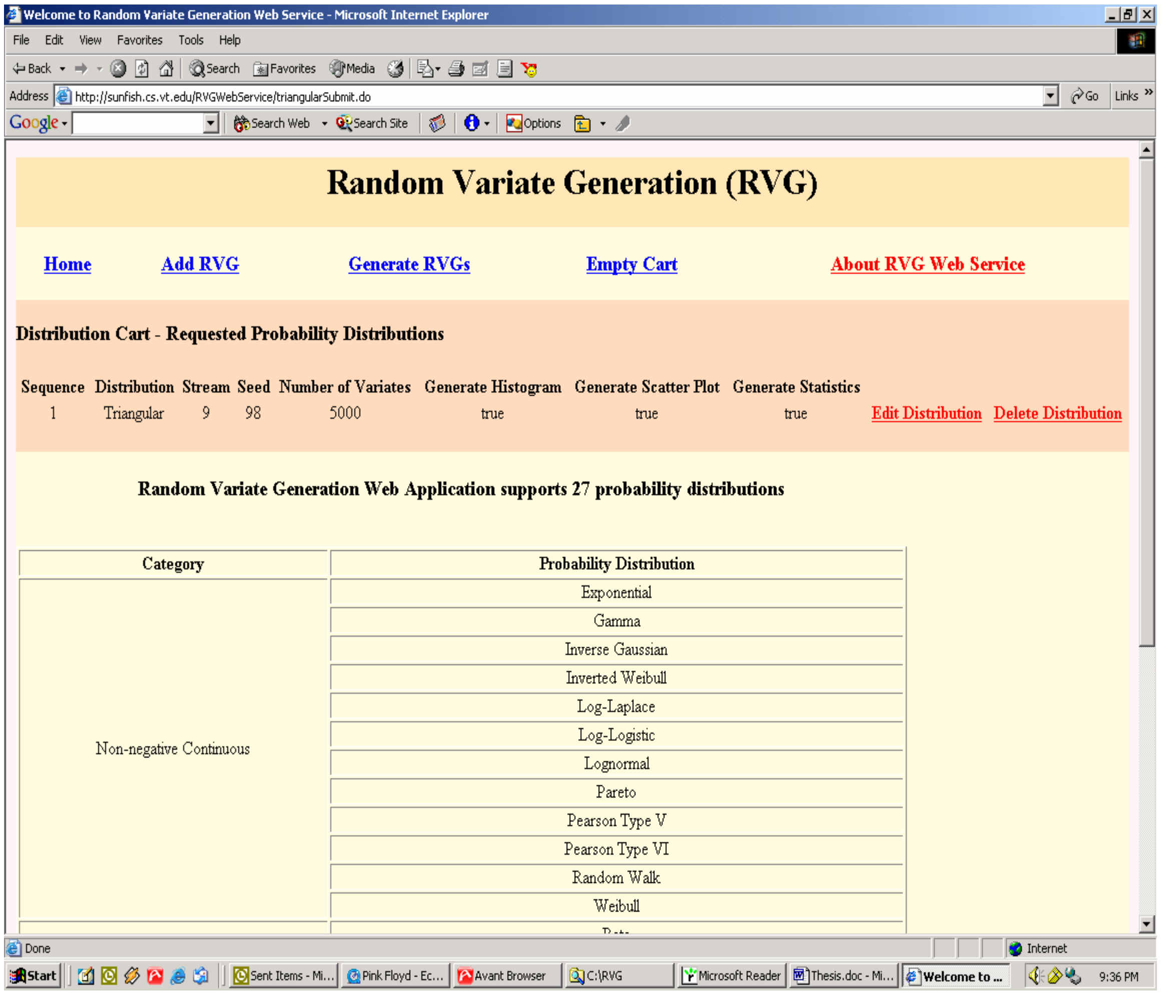
Number of Random Variates	879 *
Stream Identifer	9 *
Seed	9 *
Minimum	*
Maximum	909 *
Mode	78 *
Check the box to view Histogram of the generated Random Variates	☐
Number of Intervals for the Histogram	

Figure 9. RVG Web Application: Error Correction Screen

Edit Distribution Screen

As shown in Figure 10, the distribution cart has two hyperlinks:

- Edit Distribution: Clicking this allows the user to modify the parameters of the input distribution. Figure 11 shows an example of the Edit Distribution Screen. The Edit Distribution Screen contains three buttons:
    - Update Distribution Cart: Clicking this validates the user-specified input, and updates the cart if validation succeeds.
    - Reset Input: Clicking this clears the contents of the screen, and refreshes the page.
    - Cancel: Clicking this brings the user to the home page, without modifying the input parameters.
- Delete Distribution: Clicking this allows the user to remove the particular distribution from the distribution cart.



Welcome to Random Variate Generation Web Service - Microsoft Internet Explorer

Address http://sunfish.cs.vt.edu/RVGWebService/triangularSubmit.do

Random Variate Generation (RVG)

Home Add RVG Generate RVGs Empty Cart About RVG Web Service

Distribution Cart - Requested Probability Distributions

Sequence	Distribution	Stream	Seed	Number of Variates	Generate Histogram	Generate Scatter Plot	Generate Statistics		
1	Triangular	9	98	5000	true	true	true	Edit Distribution	Delete Distribution

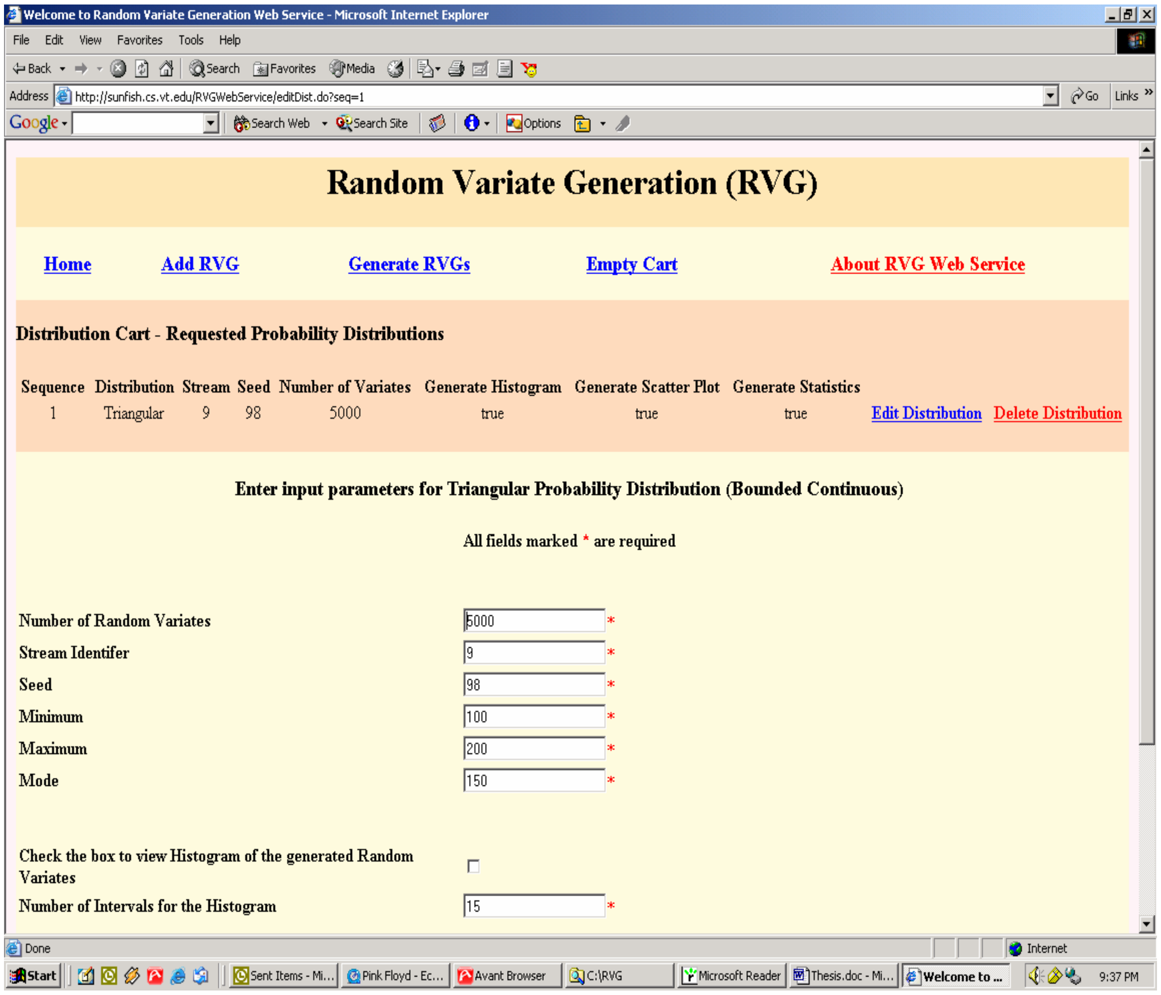
Random Variate Generation Web Application supports 27 probability distributions

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

Figure 10. RVG Web Application: Distribution Cart

Distribution Cart – Generated Distributions

Clicking on “Generate RVGs” generates Random Variates for the distributions in the distribution cart, along with histogram, scatter plot and statistics, if specified. Figure 12 shows the contents of the distribution cart on successful generation of random variates. The distribution cart lists a summary of the generated distribution along with a hyperlink that says “View Details”.



Welcome to Random Variate Generation Web Service - Microsoft Internet Explorer

Address http://sunfish.cs.vt.edu/RVGWebService/editDist.do?seq=1

Random Variate Generation (RVG)

Home | Add RVG | Generate RVGs | Empty Cart | About RVG Web Service

Distribution Cart - Requested Probability Distributions

Sequence	Distribution	Stream	Seed	Number of Variates	Generate Histogram	Generate Scatter Plot	Generate Statistics		
1	Triangular	9	98	5000	true	true	true	Edit Distribution	Delete Distribution

Enter input parameters for Triangular Probability Distribution (Bounded Continuous)

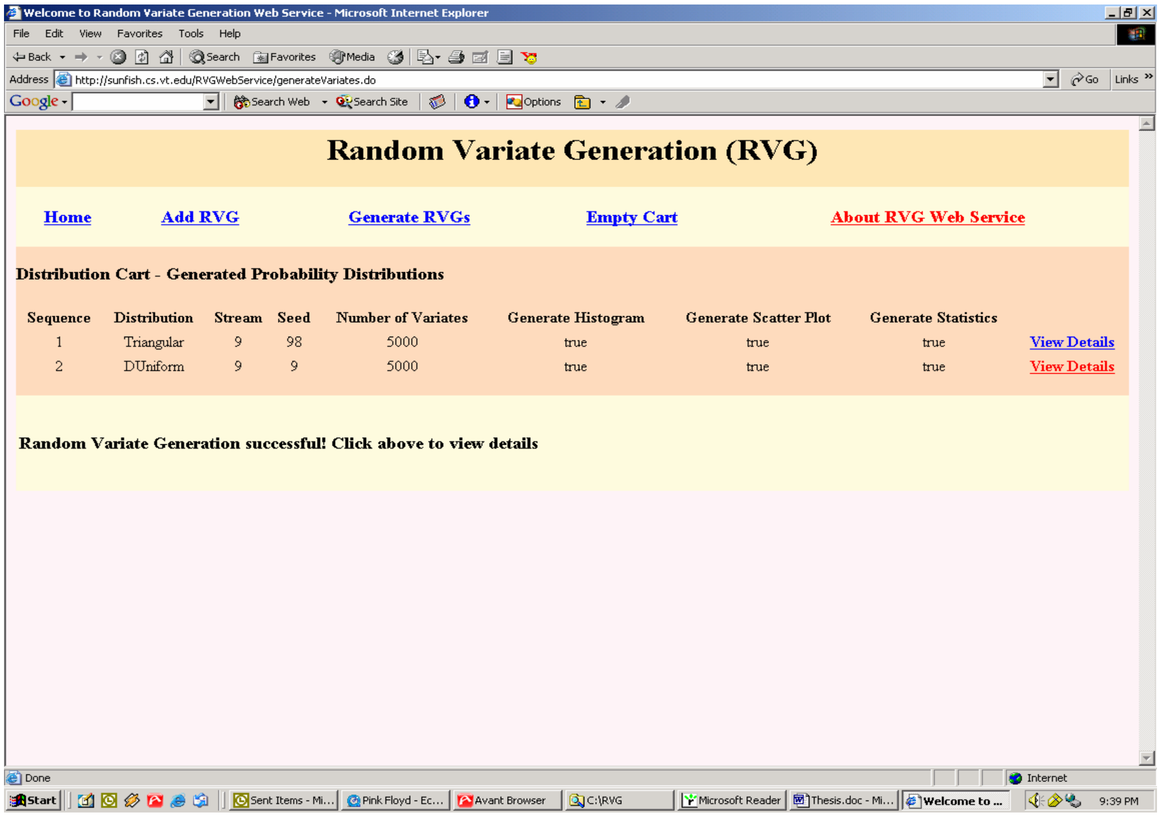
All fields marked * are required

Number of Random Variates	5000 *
Stream Identifer	9 *
Seed	98 *
Minimum	100 *
Maximum	200 *
Mode	150 *
Check the box to view Histogram of the generated Random Variates	☐
Number of Intervals for the Histogram	15 *

Figure 11. RVG Web Application: Edit Distribution Screen

Output Screen

Clicking on “View Details” for any distribution in the generated distribution cart, as shown in Figure 12, allows the user to view the generated Random Variates, along with optional histogram, scatter plot and statistics, for that distribution. Figure 13 through Figure 16 show different areas of the output screen.


Welcome to Random Variate Generation Web Service - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://sunfish.cs.vt.edu/RVGWebService/generateVariates.do

Random Variate Generation (RVG)

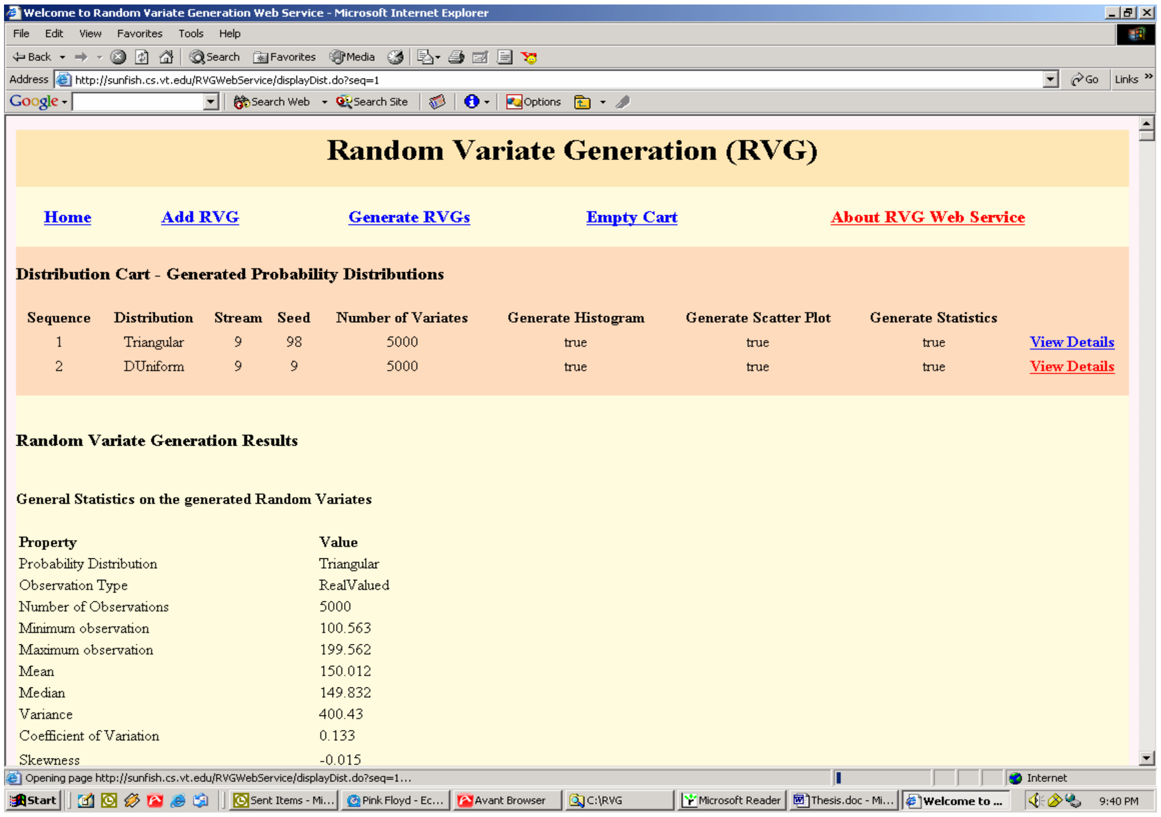
Home
Add RVG
Generate RVGs
Empty Cart
About RVG Web Service

Distribution Cart - Generated Probability Distributions

Sequence	Distribution	Stream	Seed	Number of Variates	Generate Histogram	Generate Scatter Plot	Generate Statistics	
1	Triangular	9	98	5000	true	true	true	View Details
2	DUniform	9	9	5000	true	true	true	View Details

Random Variate Generation successful! Click above to view details

Figure 12. RVG Web Application: Distribution Cart - Generated Distributions


Welcome to Random Variate Generation Web Service - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://sunfish.cs.vt.edu/RVGWebService/displayDist.do?seq=1

Random Variate Generation (RVG)

Home
Add RVG
Generate RVGs
Empty Cart
About RVG Web Service

Distribution Cart - Generated Probability Distributions

Sequence	Distribution	Stream	Seed	Number of Variates	Generate Histogram	Generate Scatter Plot	Generate Statistics	
1	Triangular	9	98	5000	true	true	true	View Details
2	DUniform	9	9	5000	true	true	true	View Details

Random Variate Generation Results

General Statistics on the generated Random Variates

Property	Value
Probability Distribution	Triangular
Observation Type	RealValued
Number of Observations	5000
Minimum observation	100.563
Maximum observation	199.562
Mean	150.012
Median	149.832
Variance	400.43
Coefficient of Variation	0.133
Skewness	-0.015

Opening page http://sunfish.cs.vt.edu/RVGWebService/displayDist.do?seq=1...

Figure 13. RVG Web Application: Output Screen - General Statistics

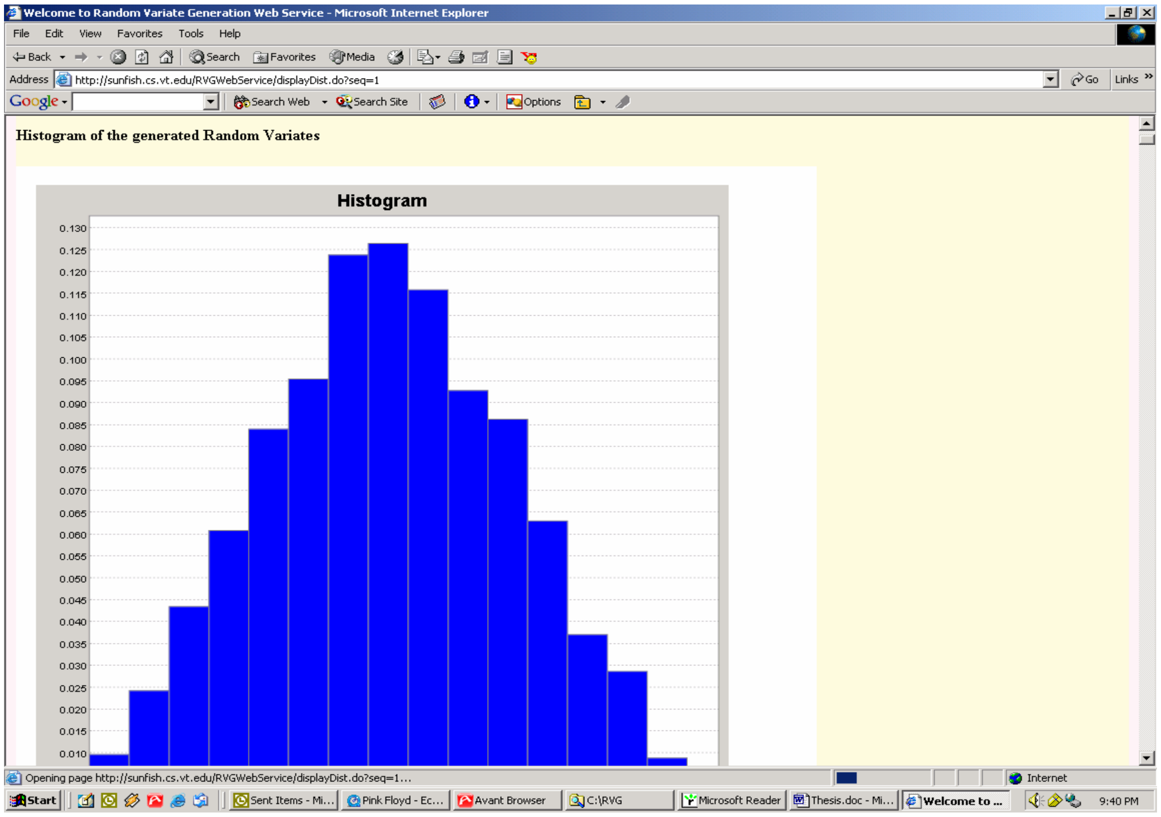

Welcome to Random Variate Generation Web Service - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://sunfish.cs.vt.edu/RVGWebService/displayDist.do?seq=1
Histogram of the generated Random Variates
Histogram
0.130
0.125
0.120
0.115
0.110
0.105
0.100
0.095
0.090
0.085
0.080
0.075
0.070
0.065
0.060
0.055
0.050
0.045
0.040
0.035
0.030
0.025
0.020
0.015
0.010
Opening page http://sunfish.cs.vt.edu/RVGWebService/displayDist.do?seq=1...
Internet
9:40 PM

Figure 14. RVG Web Application: Output Screen – Histogram (SVG)

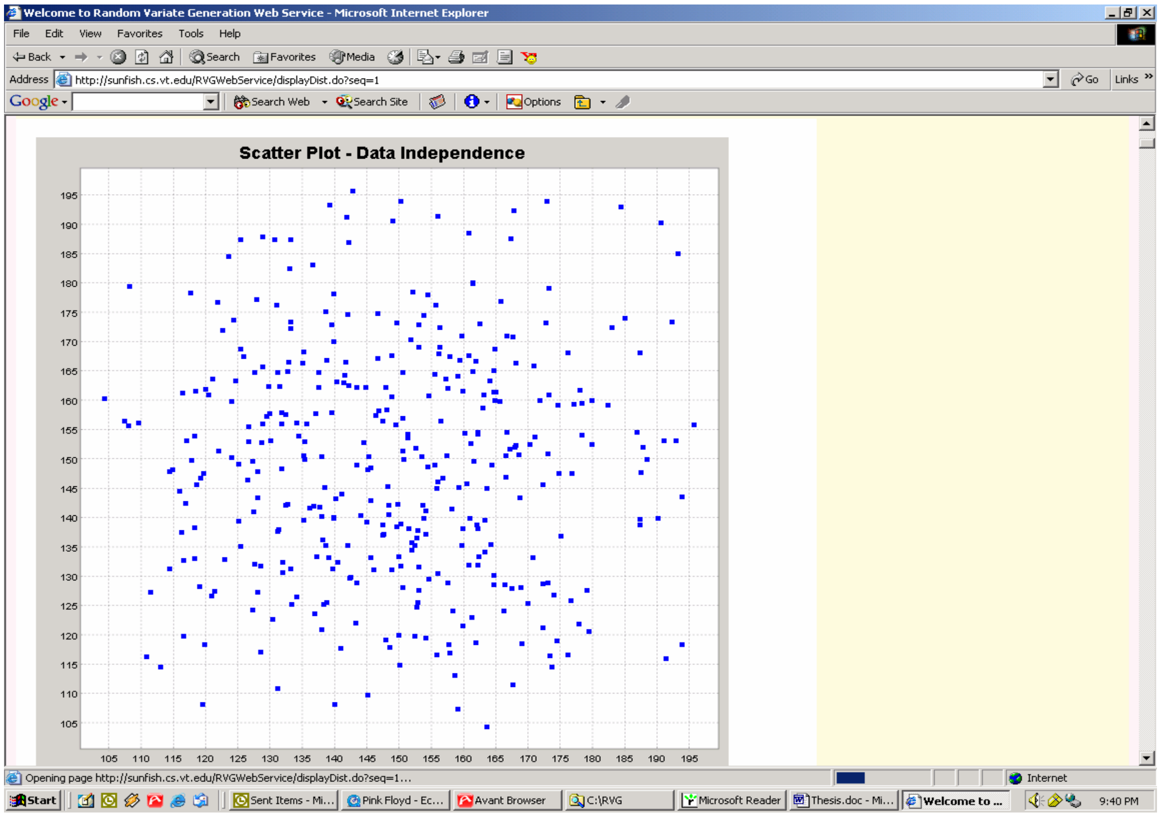
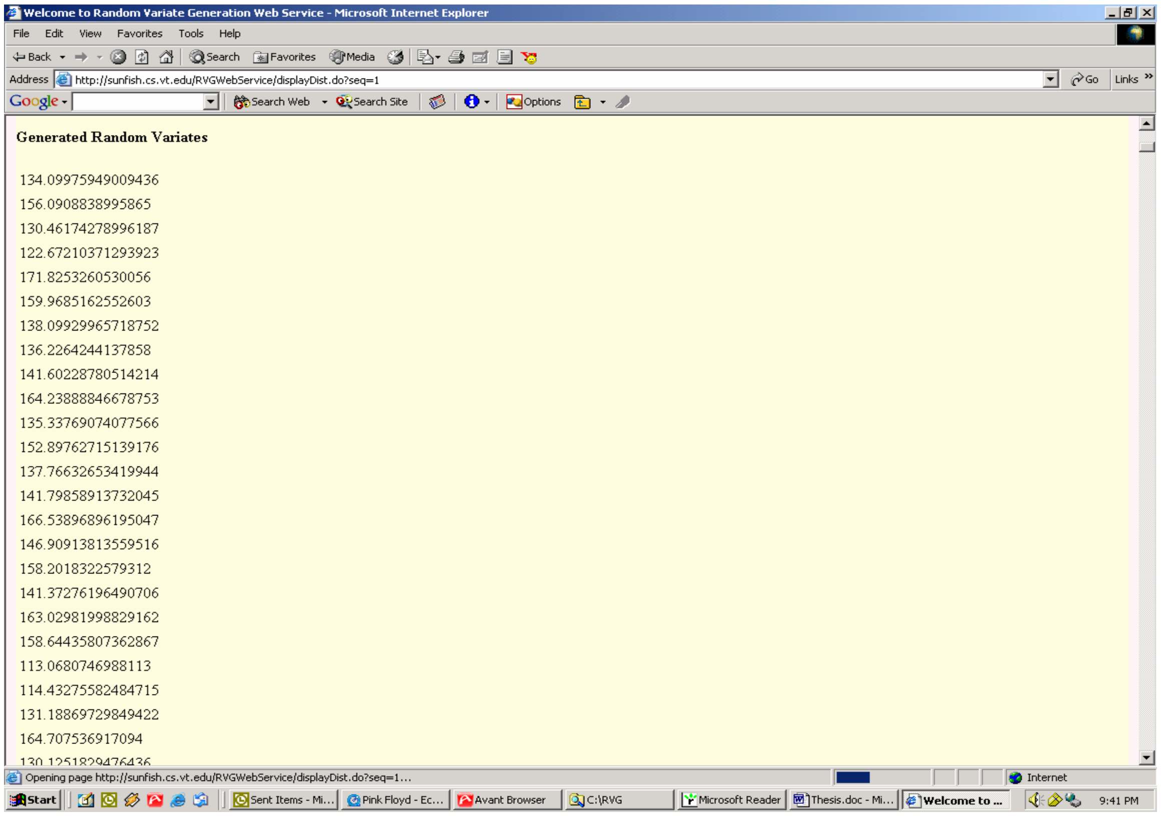

Welcome to Random Variate Generation Web Service - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://sunfish.cs.vt.edu/RVGWebService/displayDist.do?seq=1
Scatter Plot - Data Independence
195
190
185
180
175
170
165
160
155
150
145
140
135
130
125
120
115
110
105
105 110 115 120 125 130 135 140 145 150 155 160 165 170 175 180 185 190 195
Opening page http://sunfish.cs.vt.edu/RVGWebService/displayDist.do?seq=1...
Internet
9:40 PM

Figure 15. RVG Web Application: Output Screen - Scatter Plot (SVG)


Welcome to Random Variate Generation Web Service - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://sunfish.cs.vt.edu/RVGWebService/displayDist.do?seq=1
Go Links
Google Search Web Search Site Options
Opening page http://sunfish.cs.vt.edu/RVGWebService/displayDist.do?seq=1...
Internet
Start Sent Items - Mi... Pink Floyd - Ec... Avant Browser C:\RVG Microsoft Reader Thesis.doc - Mi... Welcome to ...
9:41 PM

Generated Random Variates

```
134.09975949009436
156.0908838995865
130.46174278996187
122.67210371293923
171.8253260530056
159.9685162552603
138.09929965718752
136.2264244137858
141.60228780514214
164.23888846678753
135.33769074077566
152.89762715139176
137.76632653419944
141.79858913732045
166.53896896195047
146.90913813559516
158.2018322579312
141.37276196490706
163.02981998829162
158.64435807362867
113.0680746988113
114.43275582484715
131.18869729849422
164.707536917094
130.1251829476436
```

Figure 16. RVG Web Application: Output Screen - Random Variates

After having covered the requirements of the application, the high-level design of the RVG Web application is presented below.

Section 3.2 gives an overview of J2EE patterns and best practices. Sections 3.3 through 3.6 describe the design patterns that have been used in the architecture of the RVG Web application.

3.2. Best Practices

This section presents an overview of what design patterns are, and what relevance they have with respect to architecting an enterprise application using J2EE. Special attention is paid to Model-View-Controller (MVC) design pattern because of its ubiquity in Web applications. This is followed by an overview of Struts framework.

3.2.1. Design Patterns

Patterns are about communicating *problems* and *solutions*. Simply put, patterns capture a common, recurring problem in a domain (in this case, in the software domain), and their solution in a particular context. They represent the collective experience of software architects in a domain, gained from successfully executing numerous projects. A simple definition of a pattern is: “A pattern is an idea that has been useful in one practical context and will probably be useful in others” [Fowler 2002]. The following are some of the salient points about design patterns:

- Patterns are gathered through experience.
- Patterns prevent reinventing the wheel.
- Patterns exist at different levels of abstraction.
- Patterns are reusable and present solutions to *recurring* problems. They need to be refined and tailored according to the needs of the new problem.
- Patterns complement each other and can be used in combination to together solve a problem.
- Patterns undergo continuous improvement from their application to different applications.
- Patterns represent *best practices* and *designs***.**

An excellent introduction to design patterns can be found in [Gamma et al. 1994].

3.2.2. J2EE Design Patterns

J2EE Design patterns refer to a collection of J2EE-based solutions to common problems [Alur et al. 2001]. The patterns describe typical problems encountered by enterprise application developers in building their systems, and provide solution frameworks and best practices for the same. In other words, J2EE design patterns refer to the collective wisdom gained from architecting successful enterprise applications on the J2EE platform.

Being a multi-tiered platform, the architecture of any enterprise application built on J2EE is viewed in terms of *tiers*. A tier represents a logical partitioning of the separation of concerns or functionalities in a system. Each tier can be viewed as a logical whole, loosely coupled to the other tiers and having distinct responsibilities in the system.

Figure 17 shows the Tiered approached adopted in architecting J2EE enterprise applications.

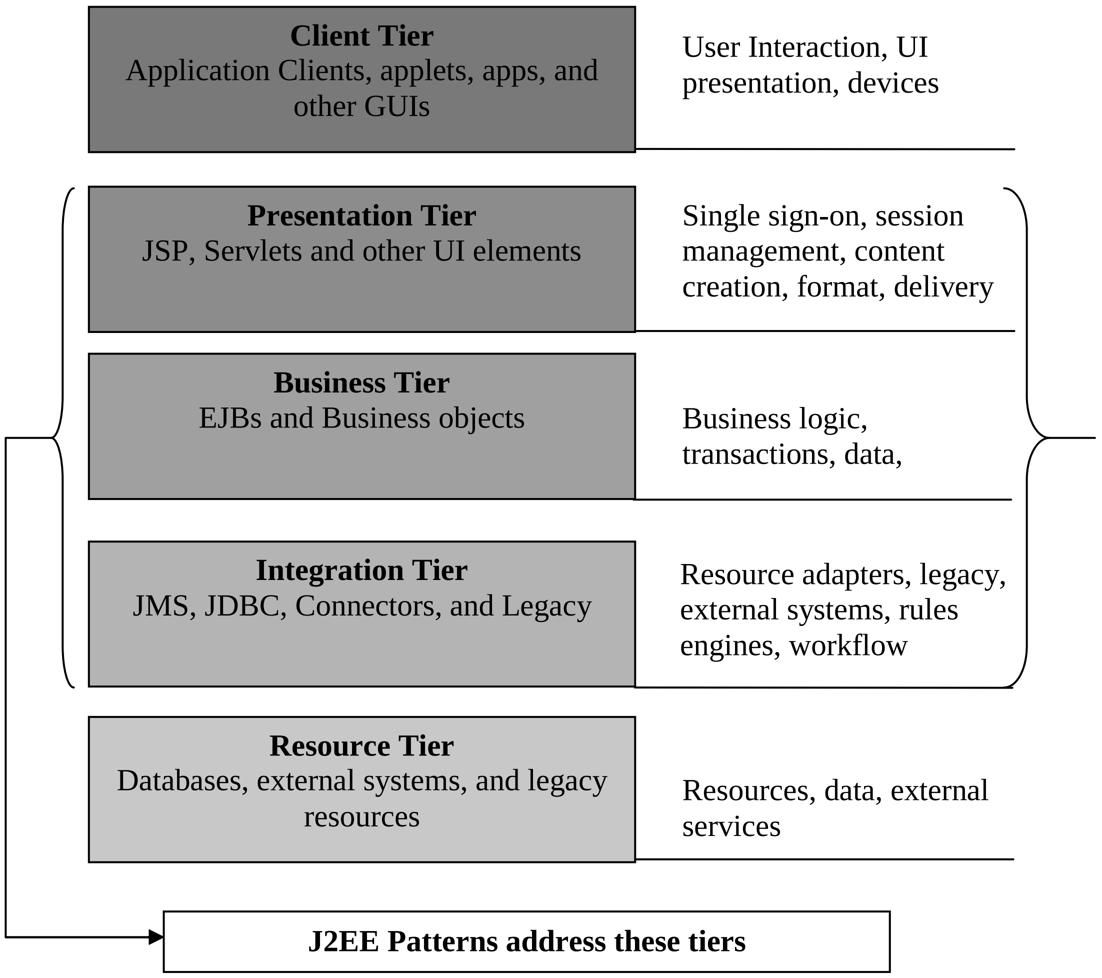

Client Tier
Application Clients, applets, apps, and other GUIs
User Interaction, UI presentation, devices
Presentation Tier
JSP, Servlets and other UI elements
Single sign-on, session management, content creation, format, delivery
Business Tier
EJBs and Business objects
Business logic, transactions, data,
Integration Tier
JMS, JDBC, Connectors, and Legacy
Resource adapters, legacy, external systems, rules engines, workflow
Resource Tier
Databases, external systems, and legacy resources
Resources, data, external services
J2EE Patterns address these tiers

Figure 17. Tiered Approach [Alur et al. 2001]

- *Client Tier*: represents devices and clients accessing the application.
- *Presentation Tier*: represents presentation logic required to service the clients accessing the application.
- *Business Tier*: represents the business services and functionality required by the clients.
- *Integration Tier*: represents all communication with external resources and systems such as data stores and legacy systems.
- *Resource Tier*: represents business data and external resources like data stores, legacy systems and integration systems.

3.2.3. MVC

MVC stands for Model View Controller, and is the most common pattern found in Web applications. Simply put, the MVC pattern splits user interface interaction into three distinct roles.

- *Model*: This represents an object or objects that contain information about the domain, typically maintained in back-end or remote store.
- *View*: This represents the displaying of the information (in the model) to the end-user.
- *Controller*: This takes user input from the View, and manipulates the Model. In other words, it maintains conditional logic which decides which screens to present to the user, and how to handle error conditions.

Separating Model from View makes the model reusable across different presentations. New presentations can be added to the application without requiring any change in the model. Furthermore, the code becomes maintainable.

Figure 18 shows the MVC Architecture for a Web application. The numbered arrows indicate the following:

- **1**: The client browser issues an HTTP request to the application.
- **2**: The Controller component receives the HTTP request, and decides how to process it based on the business logic contained within it.
- **3**: The Model performs the actual service, interacting with persistent data stores to fulfill the service.
- **4**: Based on the results returned from the model, and its state, the Controller determines the View that should be used to render the HTTP response.
- **5**: The chosen View component renders the HTTP response.

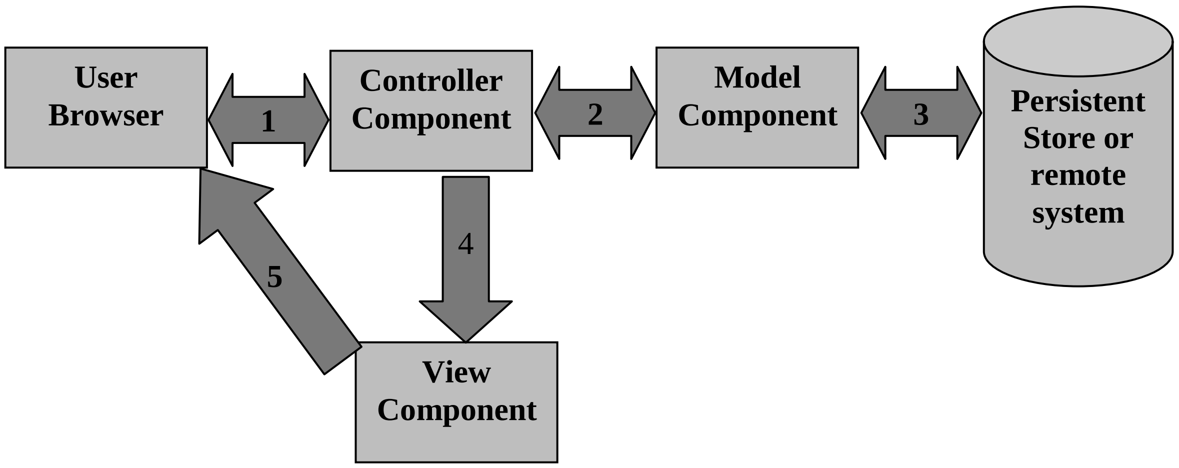


Figure 18. MVC Architecture for a Web Application [Turner and Bedell 2002]

The following section presents a framework, used in this work, that provides an implementation of the MVC design pattern.

3.2.4. Struts

In the J2EE Web application model, there are two models that are pertinent to the presentation tier, called *Model 1* and *Model 2* access models.

- *Model 1*: Here, the initial HTTP request from the client goes to a JSP page, which does all the business processing (along with beans that it interacts with), and contains conditional logic to display the next page to the user.
- *Model 2*: Here, a servlet receives the initial HTTP request from the client and processes the request (along with beans). The servlet then stores the result in a bean, and passes it to a JSP page which renders the HTTP response. The conditional logic to display the next page to the user rests in the servlet.

Model 1 architecture has been deprecated as it gives rise to highly intertwined and unmaintainable code, and ugly-looking JSP pages which do more than they should be doing – displaying pages. The open source framework – Struts – is an implementation of the highly recommended Model 2 architecture.

Struts Implementation of MVC

The Struts framework models MVC using a combination of JSPs, custom JSP tags, and Java servlets.

Figure 19 elaborates on how the Struts framework implements the MVC pattern.

- **1**: An HTTP request is made from View 1.
- **2**: The request is intercepted by ActionServlet (acting as the Controller), which looks up the requested URI in an XML file (The Struts configuration file). From the file the ActionServlet determines the Action class that will handle the business logic.
- **3**: The Action class performs business logic using Model components.
- **4**: The Action class, on completion of its action, returns control to the ActionServlet along with a key that provides the result of its processing. The ActionServlet uses the key to look up the next destination in the XML configuration file.
- **5**: The ActionServlet forwards the request to View 2 that was linked to the returned key from the Action class.

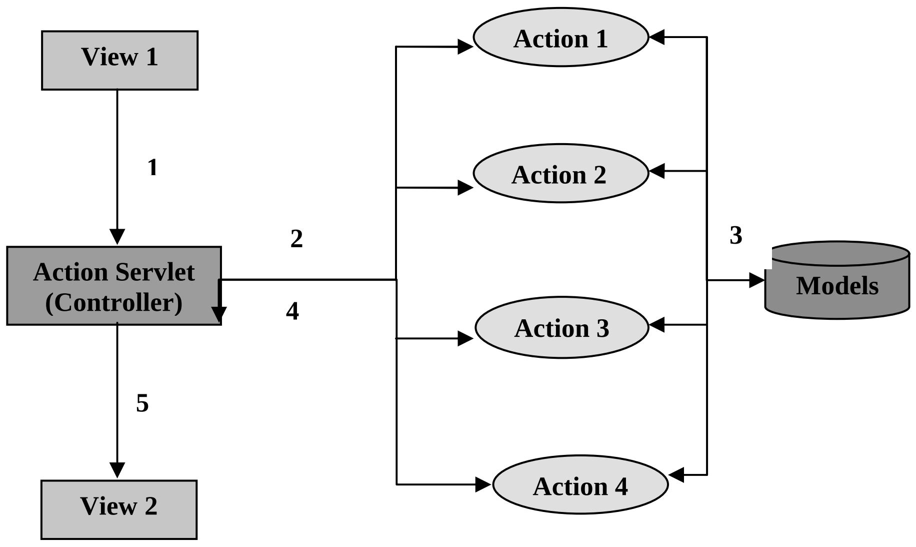


Figure 19. Struts implementation of MVC [Goodwill 2002]

3.3. RVG Web Application Architecture

This section explains the architecture of the RVG Web application at a high level, based on design choices made and design patterns implemented at each tier.

3.3.1. Design Choices

The architecture of the RVG Web application was guided by the following design choices.

Layering

RVG Web application involves generation of random variates, and their manipulation to yield different graphs. The very nature of the application makes it a client-server application, with a thin browser-based client that displays only HTML and SVG. All the logic – presentation, business and data access – rests at the server side. This is straightforward as it makes the system independent of any issues or discrepancies at the client side.

- *Presentation:* This represents the interaction between the user and the application. Using Browser scripting and downloadable applets to do some validation at the client side does avoid round-trips to the server, improving the responsiveness of the application. But it reduces the application's browser compatibility. Furthermore, with the presentation logic being split between the client and the server, maintaining the code base becomes difficult. Hence, in the case of the RVG Web application all the presentation logic is at the server side.

- *Domain/Business:* Domain or business corresponds to the work that the application needs to do. In the case of RVG Web application, the domain is random variate generation and SVG graph generation. The need to support random variate generation and graph generation for 27 probability distributions gives rise to complex business logic. Hence it makes sense to have a separate layer for domain or business.

- *Data Source/Integration:* Data source or integration corresponds to the logic for communicating with other systems carrying out tasks on behalf of the application. In the case of RVG Web application, this external system is a database that stores the stream number and seed number of each random variate generation request.

Local versus Distributed Architecture

The advantage of using distributed objects is that it provides transparency to invoking objects – the remote objects invoked can be either in the same process or in a different process on the same machine, or on another machine altogether. However the penalty to be paid is serious – performance gets affected. The design of RVG Web application has been done using local objects as much as possible – using distributed objects only when required. A few examples where distributed objects made sense are:

- The traditional clients (thin browser-based) that call RVG Web application, and the server functionality run as different processes, and on possibly different machines.
- The server-based J2EE application server (IBM WASin this case) and the database (IBM DB2) run in separate processes, but on the same machine.
- The IBM HTTP server and the IBM WAS run in different processes on the same machine.

Apart from the above instances, distributed object architecture has been avoided at all places.

Web-centric versus EJB-centric design

In a Web centric architecture, Web tier components are responsible for most of the application's functionality. They handle dynamic content generation, content presentation, and business logic and even access the backend enterprise information system. In an EJB-centric design, enterprise beans running on EJB servers encapsulate the enterprise information system and the core business logic, with the Web tier handling only user requests, and dynamic content generation.

The architecture of RVG Web application is EJB-centric for the following reasons:

- *Enterprise-level services*: The EJB container provides services for transaction handling and persistence. With these issues taken care of, the code for the application is more focused and simpler, concentrating on the core domain logic.
- *Component-based development*: The EJB architecture supports a programming paradigm that promotes use of components. Thus going for an EJB-centric design makes the application more manageable, as opposed to a Web-centric design.
- *Performance Scalability*: EJB-centric design provides much better code and performance scalability when it comes to building a large application like the RVG Web application, as compared to a Web-centric design.

MVC framework (Struts)

The use of Struts as an implementation of the MVC design pattern was justified by the complex logic involved in the navigation of screens. Furthermore, the use of Struts makes the code maintainable; changing the presentation layer has no effect on the backend business logic.

Sections 3.4 through 3.6 discuss the J2EE patterns that have been combined to come up with the architecture of the RVG Web application.

3.4. Presentation Tier Patterns

This section discusses the J2EE design patterns that have been used at the presentation tier of the RVG Web application.

3.4.1. Front Controller

The Front Controller provides the initial point of contact for handling requests for a Web site. The controller centralizes request processing, view selection, content and error handling, and provides a consistent and coordinated processing of each request. RVG Web application uses only one Front Controller for the system.

How it Works

A Front Controller is partitioned into two parts:

- *Web handler*: A class that decides what action to initiate based on the request and URL passed. The Web handler chooses which command to execute for the request.
- *Commands*: Classes that show behavior particular to a specific request. The command classes carry out the actions once the handler has delegated the request to them.

Justification for use in RVG Web Application

- *No code duplication*: The common functionality of the View is encapsulated in the Front Controller; otherwise, the logic would be duplicated in the individual views.
- *Cleaner code*: Centralizing the decisions points in Front Controller reduces the amount of Java code in JSPs (scriptlets), thereby yielding a cleaner code base.
- *Ease of Maintenance*: With the control being concentrated at one place, and not being distributed in multiple server pages, maintenance and incorporation of changes to the screen flow are greatly eased.
- *Separates View Content from View Navigation*: Without a Front Controller, the JSPs used as views would hold logic to decide on the next page to be displayed, thus mixing view content and view navigation. Use of a Front Controller removes this commingling.

3.4.2. View Helper

A View Helper, as the name indicates, is responsible for helping the View or Controller complete it’s processing. It encapsulates business logic in a class instead of spreading it throughout the View, which can be reused by multiple Views and Controllers to present the model state.

How it Works

A view is typically implemented through Java-Beans and JSP custom tags.

Justification for use in RVG Web Application

- *Modular design*: Using view helpers to separate business logic from the individual Views yields a modular design. The same view helper can be reused by multiple Views.
- *Ease of Maintenance*: The application becomes more flexible and easy to maintain as the business data access logic is not mixed with presentation formatting logic. Hence changes in the business logic needs to be made in only one place (in the view helper), rather than being spread across multiple server pages.

3.4.3. Composite View

Composite View provides a mechanism to provide a consistent layout for all the pages of the application. It does this by having Web pages composed of atomic pages, with each atomic page having formatting code within it. Thus the layout of the page as a whole is independent of the content.

How it Works

The Tiles framework that RVG Web application uses implements this pattern to build the pages of the application.

Justification for use in RVG Web Application

- *Consistent look-and-feel*: Use of this design pattern yields a consistent look-and-feel across all pages of the application.
- *Ease of Maintenance*: It is easier to maintain the application when the layout changes or individual Views change.

3.5. Business Tier Patterns

This section discusses the J2EE design patterns that have been used at the Business tier of the RVG Web application.

3.5.1. Session Façade

A Session Façade provides a simple interface to clients of the system, hiding the numerous business objects and their complex interactions. It is implemented as a stateless session bean.

How it Works

Session facade provides a coarse-grained service layer, that separates the business implementation from the business abstraction.

Justification for use in RVG Web Application

- *Reduces coupling*: By having a session façade between the client and the business logic implementation, coupling between the client and the business objects is greatly reduced.
- *Network Performance*: Session façade reduces the number of remote method invocations between client and server, thereby improving performance.
- *Simpler Interface*: Session façade provides a simple interface to the client. Instead of exposing the numerous entity beans in the RVG application, the client is exposed to only one session bean.
- *Hides Interaction among business objects*: A Session façade hides the interactions that take place between the business objects to complete the business operation.

3.5.2. Value Object/Data Transfer Object

A Value Object (also called Data Transfer Object) is used to carry data across tiers, in order to reduce network calls. This object needs to be serializable to across the connection.

How it Works

A value object essentially is a bunch of fields, with setters and getters to set and get the values of those fields. When the enterprise bean receives a request for the business data, the enterprise bean constructs a value object, sets its attributes, and passes it to the client.

Justification for use in RVG Web Application

- *Reduces Network Overhead*: Using a value object to return multiple items of data in a single method is greatly more efficient than using multiple remote method calls to get the individual attributes of the value object separately.

3.5.3. Server Session State

Essentially this pattern is about storing session state in an object in the application server.

How it Works

The session state is in an active process in memory, which can be serialized in case memory resources are needed.

Justification for use in RVG Web Application

- The RVG Web application has needs for storing the state of a session. For example, the distribution cart needs to be kept updated at all times.

3.6. Integration Tier Patterns

This section discusses the J2EE design patterns that have been used at the Integration tier of the RVG Web application.

Domain Model

This pattern involves building an object model of the domain, to encapsulate business data and behavior.

How it Works

Building the domain model in the J2EE world involves coming up with enterprise beans – session and entity – and plain java objects, to model the domain. In the case of RVG Web application, the domain model is made up of entity beans that use Container Managed Persistence (CMP) and encapsulate the random variate generators for the supported probability distributions, and a session bean that acts as a wrapper over these entity beans.

Justification for use in RVG Web Application

- *Change in Behavior*: Since the business logic of the RVG Web application is subject to change, it calls for a domain model to encapsulate the modifiable content. For example, the model may have to support more random variate generators, or store additional fields in the database.

Chapter 4. Detailed Design – RVG Web Application

This chapter discusses the detailed design of the RVG Web application in terms of classes and their interactions. The overall design of RVG Web applications follows the MVC design pattern, while the internal design of its components follows the J2EE patterns, documented in Chapter 3

4.1. RVG Web Application Components

The Web application is made up of several types of components:

- Enterprise beans
- JSPs
- Servlets
- XML files
- Utility & Exception classes
- Action classes
- ActionForm classes
- Message resources

Appendix B lists the components classes.

4.2. Class Interactions

This section shows class interactions that take place in two sample scenarios.

4.2.1. Generate Random Variates

The sequence of method calls that take place when the user clicks "Generate RVGs" is shown in Figure 20. Let us consider a situation where the distribution cart contains Beta and Uniform distributions as the requested RVGs. The sequence of method calls are as follows:

- The user clicks "Generate RVGs", thereby invoking doPost () (i.e. service ()) method of ActionServlet.
- ActionServlet invokes execute () method of GenerateVariatesAction class, a subclass of Action class.
- GenerateVariatesAction class uses the instance of DistCart EJB that is stored in session context, or creates a new instance if none exists, to invoke generateVariates ().
- DistCartEJB, which is a stateful session enterprise bean, has the list of requested distributions stored as a member variable, and passes it to the

Session Façade – VariateGenerator EJB in invoking generateVariates () method of the latter.

- VariateGenerator EJB calls Beta EJB, and Uniform EJB, one after the other, and passes the results to DistCart EJB, which in turn passes a value object to GenerateVariatesAction (See Sections 3.5.2 and 4.4.2)
- GenerateVariatesAction passes control back to ActionServlet, placing a newly populated value object in session scope (the list of generated distributions), and informing the ActionServlet of the JSP to display.
- ActionServlet uses the forward page from GenerateVariatesAction to invoke displayList.jsp, which in turn uses the session-scoped value object to display the list of generated distributions.

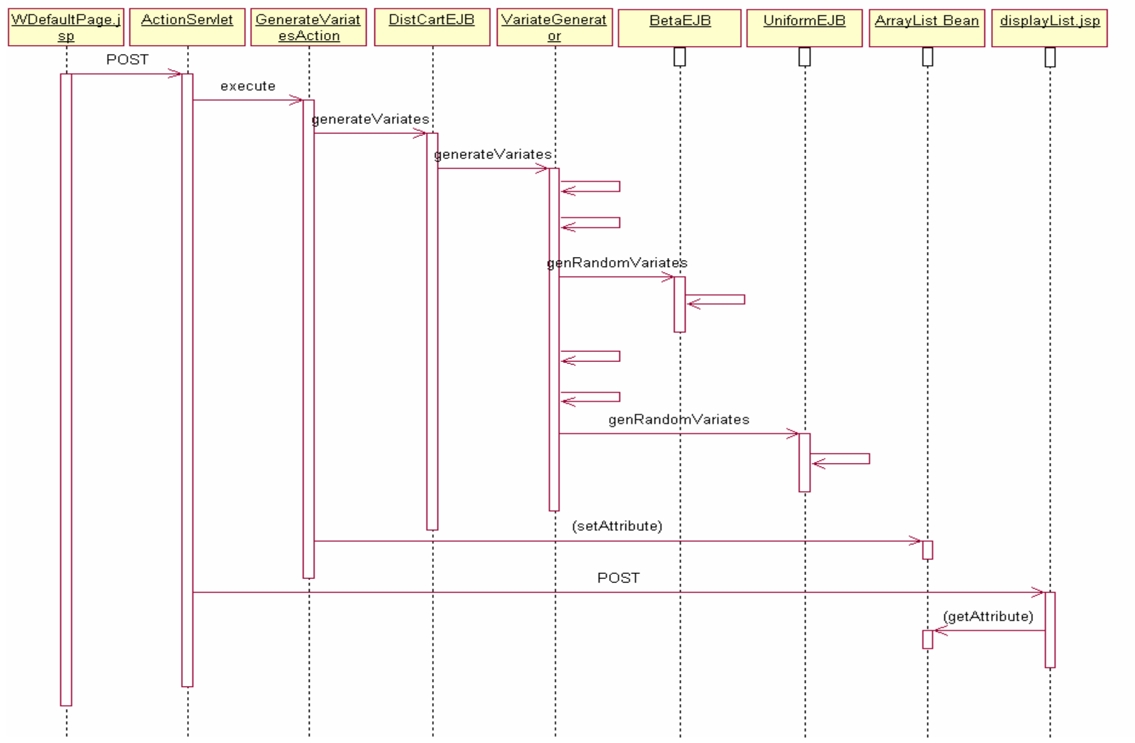


Figure 20. Sequence Diagram for generating RVGs

4.2.2. View Details of a Generated Distribution

The sequence of method calls that take place when the user clicks a particular generated distribution to view its details is shown in Figure 21. The sequence of method calls are as follows:

- The user clicks a hyperlink in the list of generated RVGs, thereby invoking doPost () of ActionServlet.
- ActionServlet invokes execute () of DisplayDistAction, a subclass of Action.

- DisplayDistAction invokes displayDist () method of DistCart EJB, which returns the details of the specific distribution in a value object to DisplayDistAction.
- DisplayDistAction passes control back to ActionServlet, passing the JSP to display, and populating a value object placed that the forward page will use to display the details on the page.
- ActionServlet forwards to displayDetail.jsp, as decided by DisplayDistAction.
- displayDetail.jsp uses the session-scoped value object to display the details. It also uses the services of two other servlets – RVGDisplayHistogram and RVGDisplayScatterPlot – to embed SVG images in the output page.

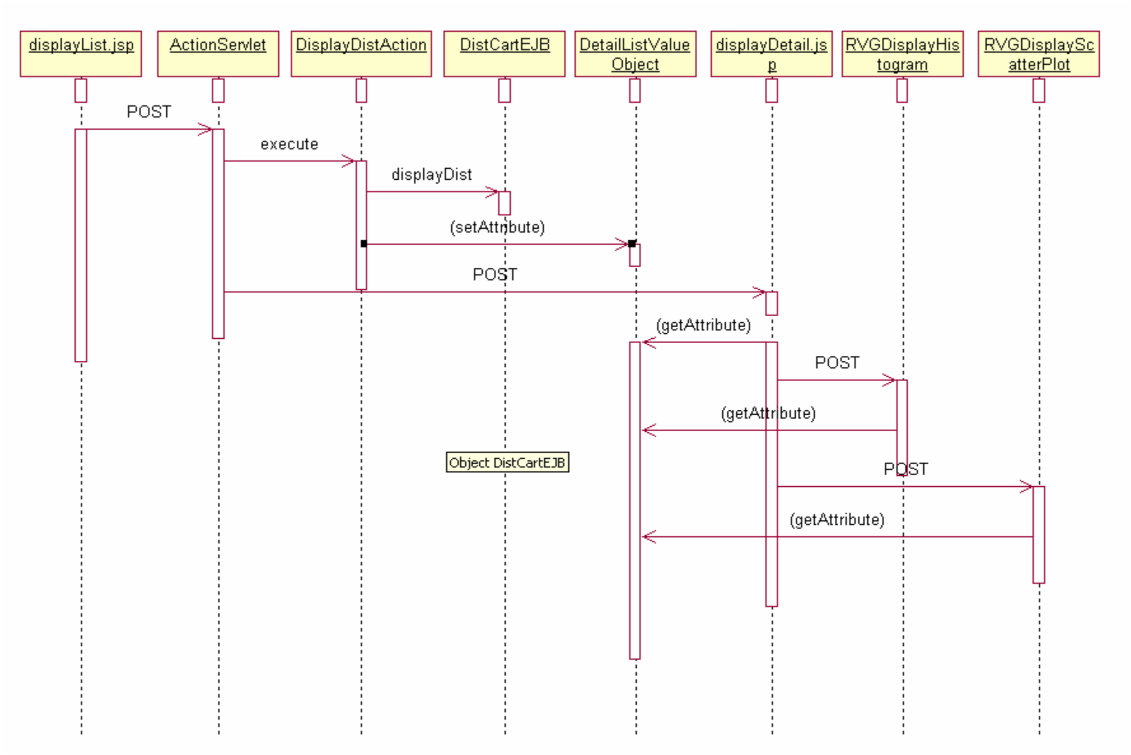


Figure 21. Sequence Diagram for viewing details of a generated RVG

4.3. Presentation Tier

This section discusses how the RVG Web application implements design patterns of the Presentation Tier, mentioned in Section 3.4.

4.3.1. Front Controller

The Struts framework provides a servlet called ActionServlet that implements the Front Controller design pattern. ActionServlet acts as the main point of entry for Web requests for RVG Web application. All requests that end in *.do are mapped to go through the ActionServlet for processing, by configuring the Web.xml (Web Deployment Descriptor). Struts also provides for ActionForms (which handle user input on input screens) and Action classes (that contain the business logic for the particular incoming request). The job of ActionServlet is to forward the input request to the correct Action class, which does the business logic with the help of enterprise beans, and then returns to the ActionServlet the forward page. The Action class optionally also populates a value object that will be used by the target page to display data. ActionServlet then delegates to the forward page to display the next screen.

Figure 22 illustrates the implementation of Front Controller pattern in RVG Web application.

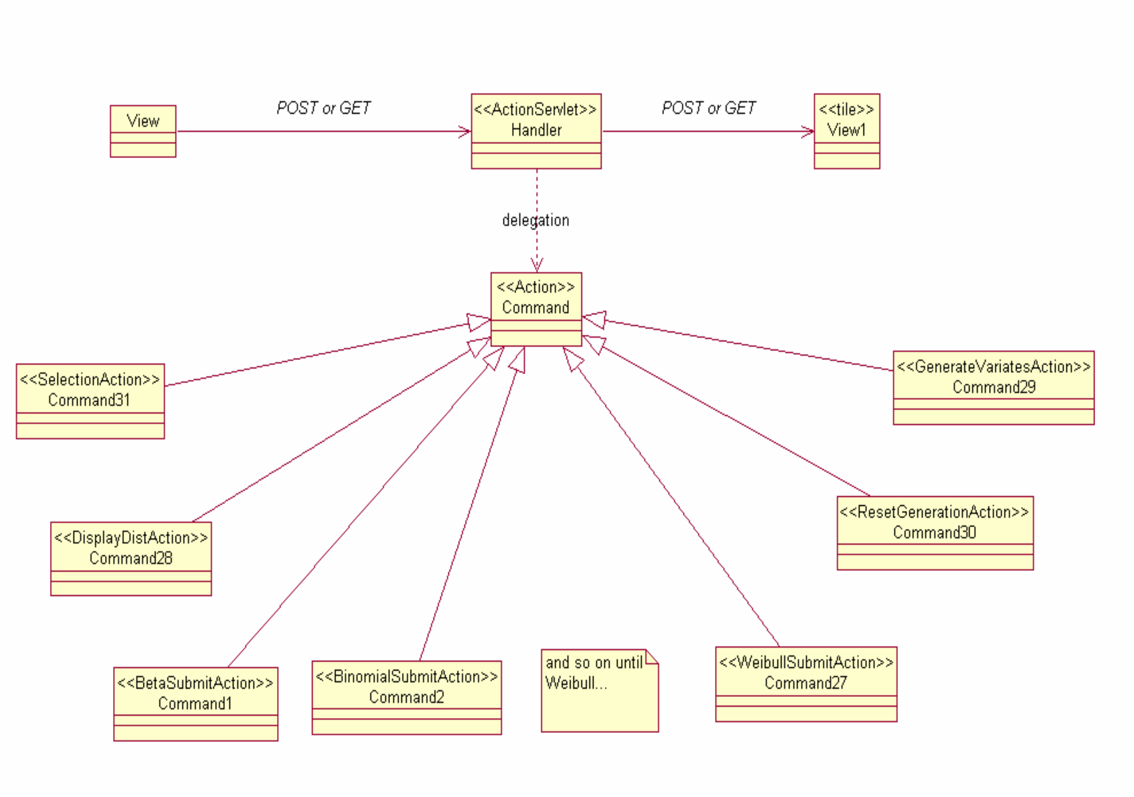

View
POST or GET
<<ActionServlet>> Handler
POST or GET
<<tile>> View1
delegation
<<Action>> Command
<<SelectionAction>> Command31
<<GenerateVariatesAction>> Command29
<<DisplayDistAction>> Command28
<<ResetGenerationAction>> Command30
<<BetaSubmitAction>> Command1
<<BinomialSubmitAction>> Command2
and so on until Weibull...
<<WeibullSubmitAction>> Command27

Figure 22. Class Diagram illustrating the Front Controller pattern

4.3.2. View Helper

Figure 23 illustrates the classes involved in implementing the View Helper pattern.

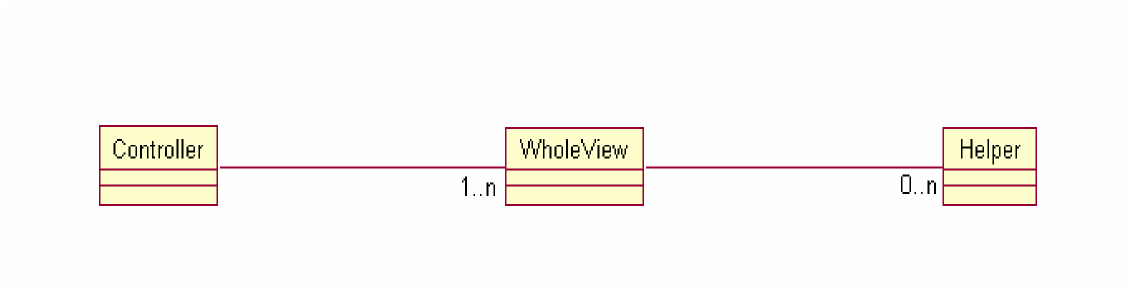


Figure 23. Class Diagram illustrating the View Helper pattern [Alur et al. 2001]

Both JavaBean helper and custom tag helper strategies have been used to implement View Helper in RVG Web application. Infact, Struts provides a number of view helpers, in the form of JSP tag libraries which are shown in Table 2.

Table 2. Struts-provided View Helpers

Tag Library Descriptor	Purpose
struts-html.tld	JSP tag extensions for HTML forms
struts-bean.tld	JSP tag extensions for handling JavaBeans
Struts-logic.tld	JSP tag extensions for testing the values of properties

An example of the use of View helpers in RVG Web application is shown in Code Listing 1, which is a code sample from displayList.jsp.

```
<logic:iterate id="distParams" name="<%= RVGConstants.LIST_DIST_KEY%>">
 <tr align=left>
  <logic:notPresent name="<%= RVGConstants.DIST_GENERATED_KEY%>">
    <td align="center"><bean:write name="distParams" property="sequence"/></td>
    <td align="center"><bean:write name="distParams" property="exptType"/></td>
    <td align="center"><bean:write name="distParams" property="stream"/></td>
    <td align="center"><bean:write name="distParams" property="seed"/></td>
    <td align="center"><bean:write name="distParams" property="number"/></td>
    <td align="center"><bean:write name="distParams" property="genHist"/></td>
    <td align="center"><bean:write name="distParams" property="genScat"/></td>
    <td align="center"><bean:write name="distParams" property="genStat"/></td>
  </logic:notPresent>
 </tr>
```

Code Listing 1. Use of JSP tag libraries

Figure 24 illustrates the View Helper pattern through the sequence of method calls that take place when the user is shown the list of requested RVGs in distribution cart, after Beta distribution has been successfully added to the list of requested RVGs. The role of DistParamValueObject is of relevance here, as it acts as a View Helper. BetaSubmitAction () sets this JavaBean in session scope, which is used by displayList.jsp in displaying the list of requested RVGs.

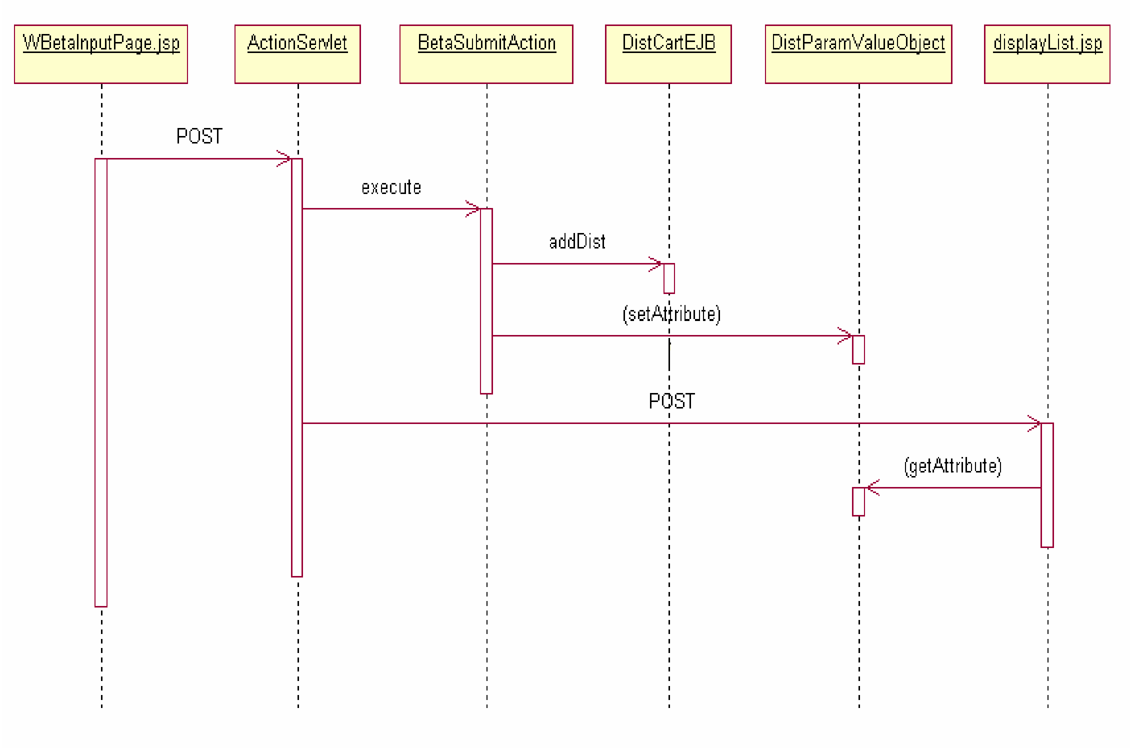


Figure 24. Sequence Diagram illustrating the View Helper pattern

4.3.3. Composite View

Composite View pattern is implemented in RVG Web application through the use of Tiles framework. Essentially, "Tiles builds on the *include* feature provided by the Java Server Pages specification to provide a full-featured, robust framework for assembling presentation components from component parts" [Tiles 2003].

Figure 25 illustrates the Composite View pattern through the class diagram. Essentially, a CompositeView is made up of an aggregate of BasicView's, which in turn is specialized by a simple view – View1 – or CompositeView.

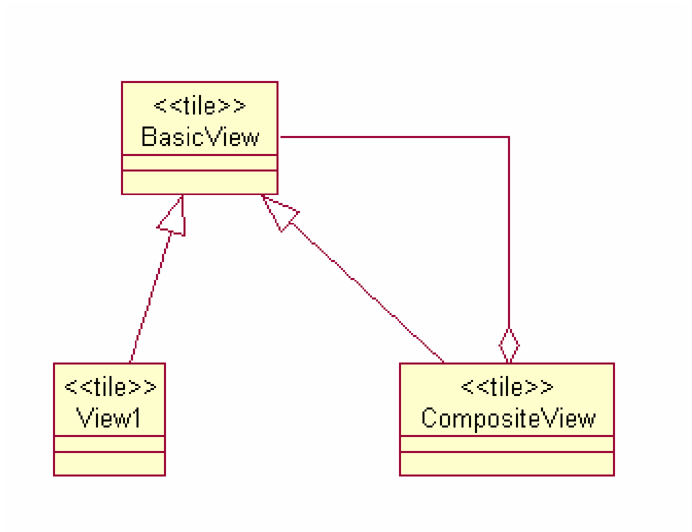


Figure 25. Class diagram illustrating Composite View pattern

Code Listing 2 illustrates the Composite View pattern, by defining a *layout* that all the pages in RVG Web application must follow. The individual screen definitions are done in separate JSP files, which are gathered together dynamically to form the composite page that is displayed to the user.

```
<%@ taglib uri="/WEB-INF/struts-tiles.tld" prefix="tiles" %>
<table>
<tr><td> <tiles:insert attribute="header"/></td></tr>
<tr ><td><tiles:insert attribute="leftMenu"/></td></tr>
<tr><td><tiles:insert attribute="rightMenu" /></td></tr>
<tr><td><tiles:insert attribute="body" /></td></tr>
<tr><td><tiles:insert attribute="footer" /></td></tr>
</table>
```

Code Listing 2. Code sample illustrating Composite View Pattern

4.4. Business Tier

This section discusses how RVG Web application implements the design patterns of the Business Tier (see Section 3.5).

4.4.1. Session Façade

Figure 26 illustrates how Session Façade pattern is implemented in RVG Web application. VariateGenerator EJB acts as the Session Façade in the application, hiding the twenty-seven entity beans that correspond to the twenty-seven probability distributions.

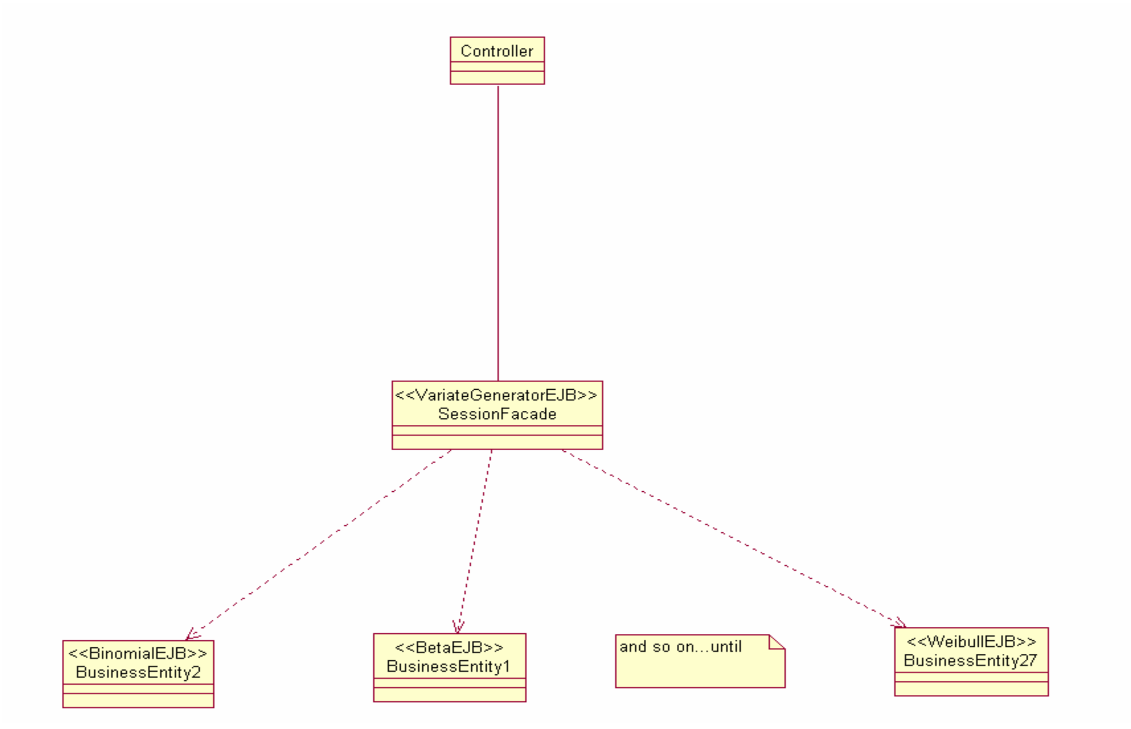


Figure 26. Class Diagram illustrating Session Facade pattern

Figure 27 illustrates the class interaction that takes place in the generation of random variates for the RVGs in the distribution cart. The distribution cart has Beta, Binomial and Uniform distributions in the example below.

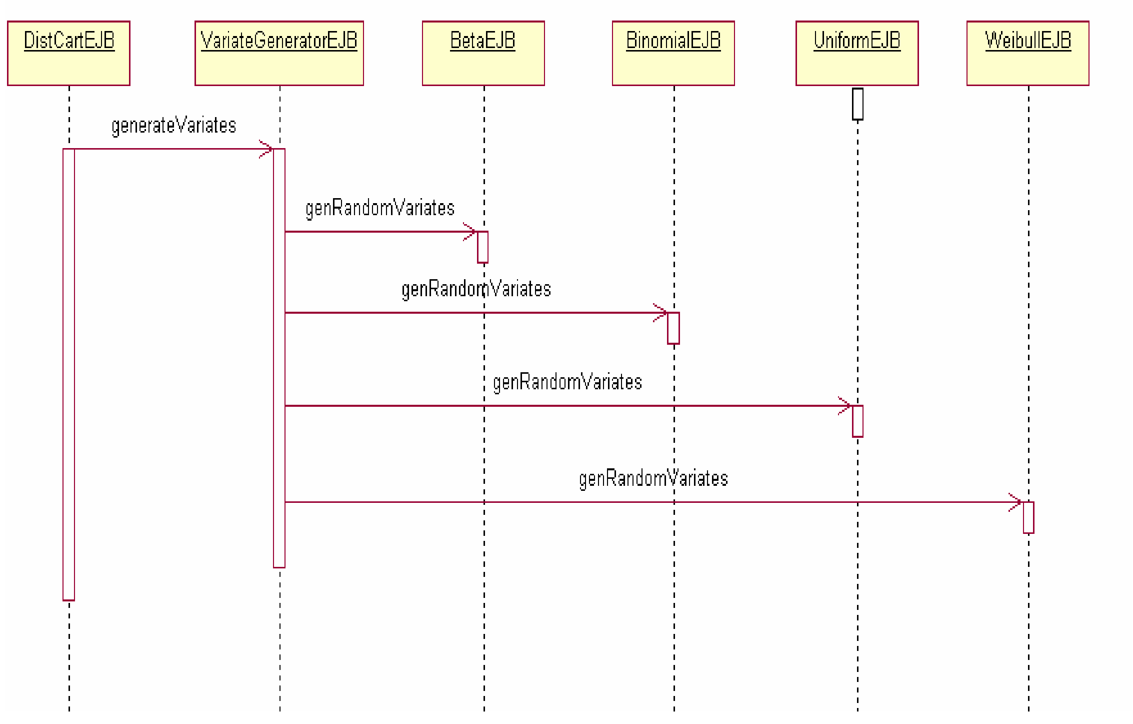
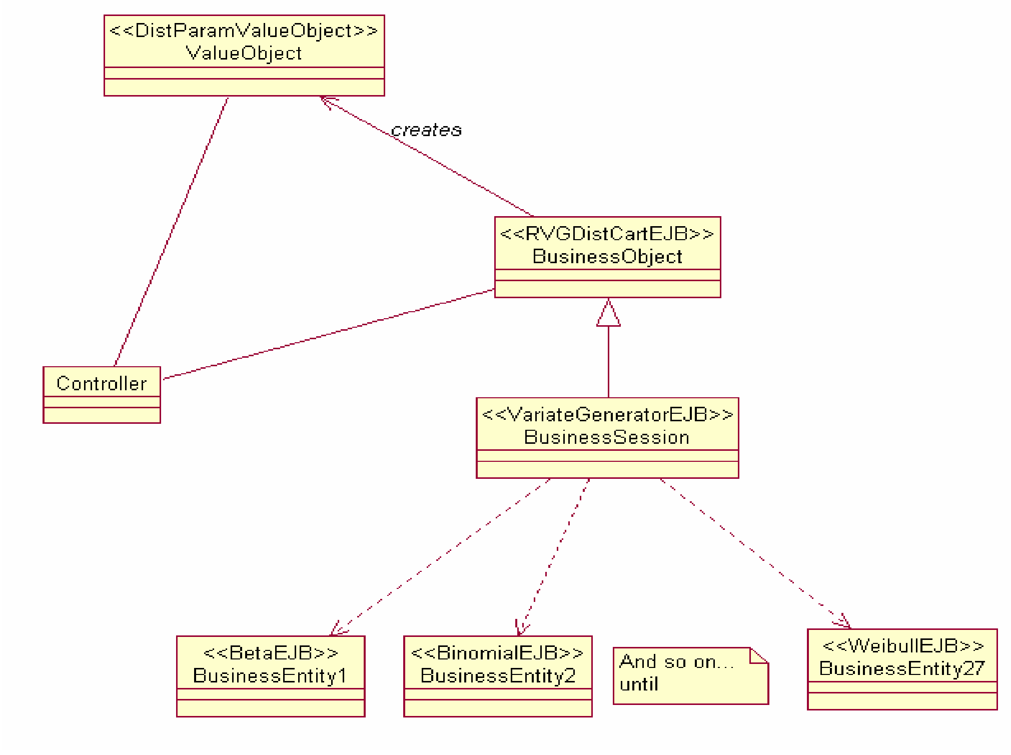


Figure 27. Sequence Diagram illustrating Session Facade pattern

4.4.2. Value Object/Data Transfer Object

Figure 28 shows the classes that implement the Value Object pattern in the RVG Web application. RVGDistCart EJB returns a value object to the Controller, which in turn creates another value object (essentially a JavaBean of class DistParamValueObject), and places it in session scope. This session-scoped JavaBean is used by the next page being displayed to the user.



<<DistParamValueObject>> ValueObject
creates
<<RVGDistCartEJB>> BusinessObject
Controller
<<VariateGeneratorEJB>> BusinessSession
<<BetaEJB>> BusinessEntity1
<<BinomialEJB>> BusinessEntity2
And so on... until
<<WeibullEJB>> BusinessEntity27

Figure 28. Class Diagram illustrating Value Object pattern

Figure 29 illustrates the sequence of messages that flow between classes involved in Figure 28. It is relevant to point out that displayList.jsp uses the Client copy of DistValueParamObject, which has been placed in session scope by BetaSubmitAction.

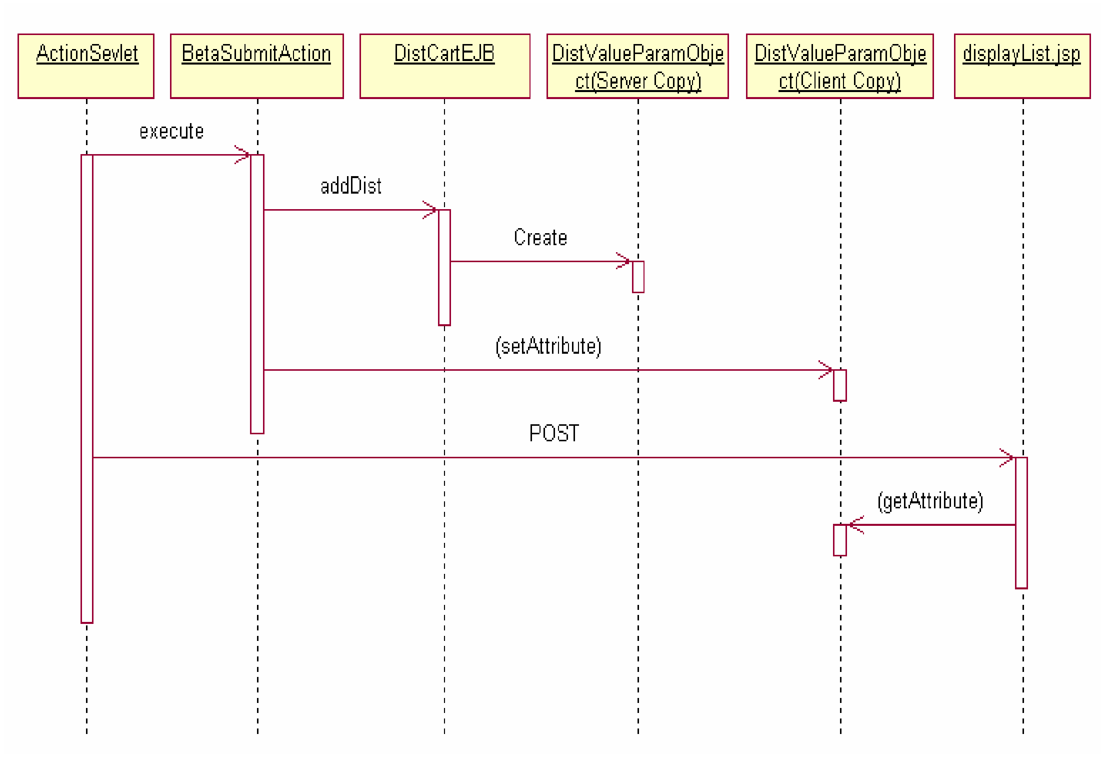


Figure 29. Sequence Diagram illustrating Value Object pattern

4.4.3. Server Session State

In J2EE, “the two most common techniques for storing *Server Session State* are using http session and using a stateful session bean” [Fowler 2002] . RVG Web application makes use of both the techniques.

The http session stores the object in the Web server by a simple ID that can be used to later retrieve the object. RVG Web application makes use of DetailDistValueObject, DistParamValueObject and DistributionBean as value objects, to be stored in session scope for access by multiple pages in the applications. Furthermore, the instance of the distribution cart implemented as a stateful session bean – RVGDistCart EJB – is stored in http session, so that each session has a unique copy of distribution cart.

The distribution cart is implemented as a Stateful Session bean – RVGDistCart EJB - and hence is stored by the EJB container, which handles all persistence and passivation for the enterprise bean.

4.5. Integration Tier

This section discusses how the RVG Web application implements design patterns of the Integration Tier, mentioned in Section 3.6.

Domain Model

Building a domain model was perhaps the most significant part of the work, because of the options available and their trade-offs. RVG Web application has different business rules for the 27 probability distributions. There is a need to make some of the fields persistent in the database for later access. There are two ways to handle persistence: using Plain Old Java Objects (POJOs) or using entity beans. The advantages of using POJOs over entity beans are:

- POJOs support inheritance, and entity beans do not.
- POJOs are generally easier to build and test, as opposed to entity beans.
- POJOs incur lesser performance overhead than entity beans

However, RVG Web application uses entity beans to encapsulate business logic and data that needs persistence for the following reasons:

- The performance of CMP in EJB 2.0 has been improved vastly over pre-EJB 2.0 implementations. So much so that CMP is actually the preferred approach over Bean Managed Persistence (BMP), both in terms of ease of development and deployment, and in performance. That justifies the use of CMP over BMP in RVG Web application.
- Using entity beans with CMP makes the bean free of any SQL or JDBC. JDBC is generated at deployment time by the EJB container. The fields to be mapped to a table of the database, and their persistence properties, are specified declaratively in an XML file.
- "Domain model is at its best when you use fine grained objects" [Fowler 2002]. That justifies having twenty-seven entity beans, corresponding to the probability distributions that are supported.
- Use of a stateful session bean – VariateGenerator EJB - to act as session façade over the entity beans, as well as use of local interfaces, does away with most of the performance-related issues associated with entity beans.
- Last, but certainly not the least, using entity bean with CMP frees the Domain Model from the database, as the CMP approach acts as a *Data Mapper* [Fowler 2002].

The fields that are made persistent in the database are:

- Stream Identifier – primary key
- Original Seed
- Last Seed
- Distribution Name

Chapter 5. RVG Web Service

This chapter provides a description of the RVG Web service, created from the RVG Web application.

5.1. Overview of WSDL

"Web Services Description Language (WSDL) provides a model and an XML format for describing Web services" [WSDL 2003]. It describes a standard format for specifying interfaces, and essentially works as a contract between the client (Web service requestor) and server (Web service provider). Using a standard XML-based format to specify the interface makes possible the automatic generation of client proxies from the WSDL document in a language- and platform-independent manner.

WSDL Document Structure

The WSDL document can be divided into two sections:

- Abstract Definitions – These define the 'what' of the functionality that is offered. Abstract definitions define the SOAP messages that flow between the Web service requestor and Web service provider 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 Web service, and also the endpoint where the Web service is accessible.

The Abstract Definitions section of WSDL document is made up of the following element, in order:

- import – This element imports other WSDL documents or XML schemas.
- types – This element specifies all application-specific datatypes exchanged between the Web service requestor and provider.
- message – This element defines one-way messages, and contains one or more 'part' elements that correspond to message parameters or return values.
- portType – This element defines the operations of the Web service, and specifies a complete one-way or roundtrip operation. A 'portType' element declares the name of the method (using 'message' elements mentioned before), and types (using 'part' elements defined in 'message' element), within one or more 'operation' elements.

The Concrete definitions are made up of the following elements, in order:

- binding – This element specifies the binding of each 'operation' element in 'portType' element. Binding refers to the SOAP message formats and serialization schemes.

- service – This element specifies the address for invoking the Web service, for example, a URL. In other words, it specifies the address of each 'binding' element. A 'service' element contains one or more 'port' elements, each 'port' element corresponding to a Web service.

Figure 30 shows the relationships between the different elements in a WSDL document. The 'definitions' element forms the root of all WSDL documents. The order of other elements is: import, types, message, portType, binding, and service.

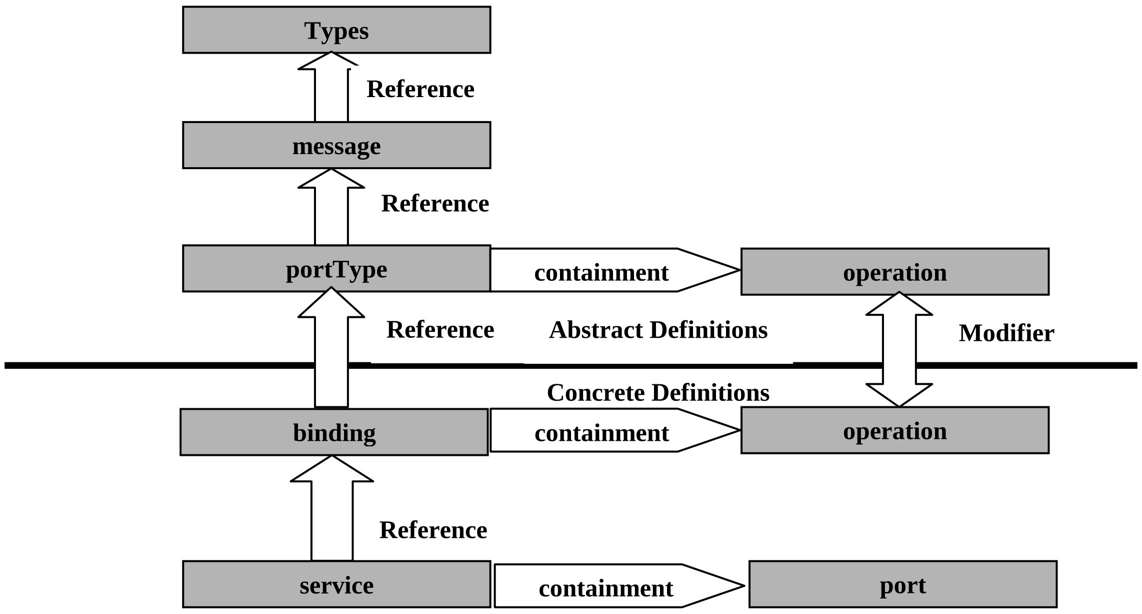


Figure 30. WSDL: Abstract and Concrete Definitions [Tapang 2001]

5.2. Overview of SOAP

"SOAP Version 1.2 provides the definition of the XML-based information which can be used for exchanging structured and typed information between peers in a decentralized, distributed environment" [SOAP 2003]. In other words, SOAP is a XML-based protocol for exchanging information between two computers. The salient features of SOAP are:

- Simplicity – SOAP's simplicity makes it very flexible, and highly beneficial to interoperability.
- Underlying protocols – SOAP can be used over any transport protocol like HTTP, FTP, or SMTP.
- Flexible to programming model – Unlike CORBA and RMI, SOAP is not tied to the 'Remote Procedure Call' ('rpc') programming model for invoking remote procedures on distributes objects; it can be used even in messaging

systems. SOAP defines a model for processing individual, one-way, request-response, solicit-response, notifications or long-running peer-to-peer conversations.

5.2.1. SOAP Message

"A one-way message, a request from a client, or a response from a server is officially referred to as a SOAP message" [Cerami 2002]. Figure 31 shows the main elements of a SOAP message.

- Envelope – This forms the root of a SOAP message, and is mandatory for every SOAP message.
- Header – This element, if present, comes as the first child of Envelope. The header can be used for providing authentication, transaction and authorization services.
- Body – This mandatory element comes right after the header, if the header is present, and represents the payload of the SOAP packet. It represents the message that is sent between the Web service requestor and the Web service provider.
- Fault – This element, if present, is included as a sub-element of Body. It gives information about error conditions or faults encountered in the invocation or operation of the Web service.

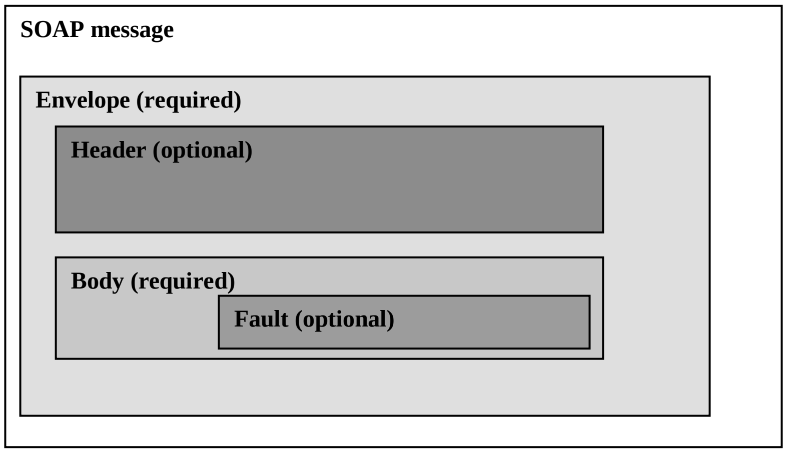


Figure 31. Main elements of an XML SOAP message [Cerami 2002]

5.2.2. SOAP Message Formats and Serialization / Deserialization

SOAP provides the following message formats for the Web service:

- Document – The body of SOAP packet contains an XML document whose format the sender and receiver agree upon (through an XML Schema).
- Remote Procedure Call (RPC) – The body of SOAP packet contains an XML representation of a remote method call. The element is named with the name of the remote procedure, and has elements for each parameter of that procedure.

Applications that have to serialize/deserialize the data in transit, have to decide upon the serialization format for the Web service. SOAP provides the following serialization formats:

- SOAP Encoding – This is the set of serialization rules that specify how “objects, structures, arrays and object graphs should be serialized” [Ewald 2002]. The SOAP processor runs through the various SOAP-defined encoding rules at runtime to determine the proper serialization of the body of SOAP packet.
- Literal – Data is serialized according to an XML schema, with the latter defining the format of the body of SOAP packet.

However, SOAP message format *does not* dictate the programming model. For example, one can use document message format with rpc-style programming model to access remote methods on distributed objects.

RVG Web Service makes use of document/literal message format. The salient features are:

- Each ‘message’ element in WSDL has one ‘part’ sub-element, which points to a schema element declaration that describes the contents of the entire body.
- In ‘binding’ element, style attribute is ‘document’, and use attribute is ‘literal’.

5.3. WSDL for RVG Web Service

The WSDL file for RVG Web service is generated from the session EJB – VariateGenerator – using WebSphere Studio Application Developer’s (WSAD’s) Web service creation wizard. This sections walks through the different sections of the WSDL file for RVG Web Service, highlighting the salient points.

Code Listing 3 shows the abstract definitions in 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. Appendix A contains the request and reply schemas.
- There are two messages, in their own separate 'message' elements that correspond to the request and reply messages in Web service invocation. The bodies of the respective SOAP messages have to follow the imported schema specification.
- 'portType' element declares the one operation exposed in the Web service: getVariateFromXML, along with the input and output messages that flow.

5.3.1. Abstract Descriptions

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="VariateGenerator"
   targetNamespace="http://Session.RVG.src.vt.edu.wsdl/VariateGenerator/"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:tns="http://Session.RVG.src.vt.edu.wsdl/VariateGenerator/"
   xmlns:xsd=http://www.w3.org/2001/XMLSchema
   xmlns:xsd1="http://Session.RVG.src.vt.edu/">

   <import location="VariateRequestSchema.xsd" namespace="http://Session.RVG.src.vt.edu/"/>
   <import location="VariateReplySchema.xsd" namespace="http://Session.RVG.src.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>
```

Code Listing 3. RVG Web Service WSDL: Abstract Descriptions

Appendix A shows the request and reply schemas.

5.3.2. Concrete Description: Bindings

Code Listing 4shows the bindings of 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/deserialization for the SOAP messages that flow between the Web service requestor and the Web service provider. The binding is document/literal.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="VariateGeneratorBinding"
   targetNamespace="http://Session.RVG.src.vt.edu.wsdl/VariateGeneratorBinding/"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:interface="http://Session.RVG.src.vt.edu.wsdl/VariateGenerator/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://Session.RVG.src.vt.edu.wsdl/VariateGeneratorBinding/">

 <import location="VariateGenerator.wsdl"
     namespace="http://Session.RVG.src.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.src.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.src.RVG.Session.VariateGenerator" use="literal"/>
       </output>
     </operation>
</binding>
</definitions>
```

Code Listing 4. RVG Web Service WSDL: Concrete Description - Bindings

5.3.3. Concrete Description: Services

Code Listing 5 shows the service description of RVG Web Service. The salient points to note are:

- It imports the WSDL file fragment that contains bindings described in 5.3.2.
- The 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.src.vt.edu.wsdl/VariateGeneratorService/"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:binding="http://Session.RVG.src.vt.edu.wsdl/VariateGeneratorBinding/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://Session.RVG.src.vt.edu.wsdl/VariateGeneratorService/">
   <import                                   location="VariateGeneratorBinding.wsdl"
namespace="http://Session.RVG.src.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>
```

Code Listing 5. RVG Web Service WSDL: Concrete Description - Service

5.4. Justification for document style in RVG Web Service

WSDL contains a switch that switches the mode from 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 case of RVG Web service, since we make use of request and reply schemas, SOAP encoding is avoided, and the SOAP binding is 'literal'.

There are many reasons why we opted to go in for document-style Web service, as opposed to RPC-style Web service in our work [McCarthy 2002]:

- *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" [McCarthy 2002] 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 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.

5.5. RVG Web Service Client

The client of the RVG Web service is a Web-based simulation application that needs random variates to run the model. To be able to invoke the RVG Web services, the client needs to download the WSDL document of RVG Web service presented in Section 5.3. This WSDL document contains the following important pieces of information that are needed in order to use the RVG Web service:

- Request Schema, as shown in A1.
- Reply Schema, as shown in A2.
- Address of the RVG Web service, as shown in Section 5.3.3.
- Name of the RVG Web service, as shown in Section 5.3.3.
- SOAP binding: document; SOAP encoding: literal. (See Section 5.3.2.)

Based on the information above, the client application can invoke the RVG Web service. The first step for the client application is to create an XML document encapsulating the request. Code Listing 6 shows an example of a request XML document compliant with request schema shown in A1, to be sent to the RVG Web service.

```
<?xml version="1.0" encoding="UTF-8"?>
<rv:variateRequest xmlns:rv=" http://Session.RVG.src.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>222.0</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>1983</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>
```

Code Listing 6. Sample RVG Request

The next step after having prepared an XML document based on the published Request Schema, is to invoke the Web service. Environments like WSAD provide Web service proxy generation tools, which the client can use to invoke the RVG Web service. Code Listing 7 shows the proxy for a sample client of RVG Web service, created using WSAD's Web service client generation wizard. The following are the salient points about the client proxy:

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

- The proxy hides the communication details with the remote RVG Web service from the calling application. That is, the calling application (RVG Web service client) merely calls the “getVariateFromXML” method of the client proxy passing it org.w3c.dom.Element (inputXML parameter of “getVariateFromXML” method) obtained after parsing the RVG request XML file shown in Code Listing 6. The proxy then takes responsibility for invoking the remote method of RVG Web service.
- The Call object “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 shown below, where string URL 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 an org.w3c.dom.Element obtained after parsing the returned XML document from the RVG Web service. The following lines illustrate it.

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

All RVG Web service clients follow the code structure shown in Code Listing 7; the Web service clients, which are Web-based simulation applications, can be in any language and running on any platform.

```
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.src.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
```

Code Listing 7. RVG Web Service Proxy

RVG Web service allows the calling application to request for random variate generation in batches or streams, over a period of time. This is supported by having fields for stream Identifier and Seed number in both the request and the reply schemas.

- Request Schema: “streamID”, and “seed”.
- Reply Schema: “streamID”, and “nextSeed”.

The combination of the two fields encapsulates the calling application, and the source of randomness for which random variate generation is required. For example, let us consider a scenario when the calling Web-based simulation application, let’s say A wants to start random variate generation for a particular source of randomness in its system. A selects a “streamID”, and an initial seed “seed”, and passes it to the RVG Web service. For reasons of performance or needs of the application, A may not request for all the random variates corresponding to this “streamID” to be generated at one go by the RVG Web service, but may request for a small number, say for 5000 random variates. At the end of the generation of 5000 random variates, the RVG Web service passes to A the “nextSeed” that A must use if it wants to continue random variate generation of that particular “streamID” later. On the next occasion, if A passes the returned “nextSeed” as “seed” in the request, the RVG Web service continues random variate generation starting with the last seed that it had used in the previous occasion. Otherwise, random variation starts afresh, using the new “seed” that A entered.

Chapter 6. Verification and Validation

This chapter deals with activities concerned with verification and validation of the RVG Web application and service.

6.1. Verification

This section highlights the efforts made to assure *transformational accuracy* during the project life cycle. The random variate generators for the 27 probability distributions are documented in [Law and Kelton 2000]. The following techniques were used to make sure that the implementation of the random variate generators in Java is correct:

- Desk-checking,
- Code walkthrough, and
- Inspection.

6.2. Validation

This section highlights the efforts made to assure the *behavioral accuracy* of the RVG Web application and the RVG Web service.

6.2.1. Need for Validation

The RVG Web application essentially accepts input parameters for distributions, and generates random variates for the same. In addition, the application also optionally produces a scatter plot and a histogram from the random variates generated. The scatter plot is a test of independence and randomness of the random variates, while the histogram shows the distribution of the random variates over different intervals.

To assure that the results achieved by the application are correct, we did *comparison* testing with ExpertFit software, which determines which probability distribution a data set fits into. Furthermore, ExpertFit also generates a scatter plot and a histogram from the data set fed in. Hence, the scatter plot and histogram generated by RVG Web application are also validated by comparing them with those generated from ExpertFit.

6.2.2. Expert Fit

ExpertFit [ExpertFit 1995] is a tool that is used extensively in discrete-event simulation studies of real-world systems for analysis. It allows one to determine which probability distribution a particular data set belongs to (Data Analysis). In addition to data analysis, ExpertFit may also be used to specify a probability distribution for a general activity when no data is available (General Activity Model). Yet another use of

ExpertFit is to model the random downtimes of a machine when no downtime data are available (Machine Breakdown Model). We used ExpertFit to do Data Analysis.

6.2.3. Validation Results

ExpertFit supports all of the 27 probability distributions that have been implemented in RVG Web application. To do the validation test, RVG Web application was requested for 7000 random variates corresponding to each distribution, and this data was fed into ExpertFit to validate whether the data set actually represents the probability distribution of interest. This operation of generating 7000 random variates from RVG Web application, and feeding into ExpertFit was carried out 5 times for each distribution.

The results of the validation with ExpertFit are shown in Table 3 through Table 29.

Table 3. Validation with ExpertFit: Beta Probability Distribution

Distribution	Run	Distribution Fit (%) – Top 3		Histogram Fit	Scatter Plot Fit
Beta	1	Beta	83.33	Yes	Yes
		Johnson SB	75.00		
		Uniform	25.00		
	2	Beta	91.67	Yes	Yes
		Johnson SB	66.67		
		Uniform	25.00		
	3	Beta	83.33	Yes	Yes
		Johnson SB	75.00		
		Uniform	25.00		
	4	Beta	93.75	Yes	Yes
		Johnson SB	75.00		
		Uniform	43.75		
	5	Beta	81.25	Yes	Yes
		Johnson SB	87.50		
		Johnson SB	43.75		

Table 4. Validation with ExpertFit: Binomial Probability Distribution

Distribution	Run	Distribution Fit (%) – Top 3		Histogram Fit	Scatter Plot Fit
Binomial	1	Binomial Poisson Discrete Uniform	100.00 66.67 22.22	Yes	Yes
	2	Binomial Poisson Discrete Uniform	100.00 75.00 50.00	Yes	Yes
	3	Binomial Poisson Discrete Uniform	100.00 75.00 50.00	Yes	Yes
	4	Binomial Poisson Discrete Uniform	100.00 75.00 50.00	Yes	Yes
	5	Binomial Poisson Discrete Uniform	100.00 75.00 50.00	Yes	Yes

Table 5. Validation with ExpertFit: Discrete Uniform Probability Distribution

Distribution	Run	Distribution Fit (%) – Top 3		Histogram Fit	Scatter Plot Fit
Discrete Uniform	1	Discrete Uniform Negative Binomial Poisson	100.00 75.00 41.67	Yes	Yes
	2	Discrete Uniform Geometric Logarithmic Series	100.00 50.00 0.00	Yes	Yes
	3	Discrete Uniform Geometric Logarithmic Series	100.00 50.00 0.00	Yes	Yes
	4	Discrete Uniform Geometric Logarithmic Series	100.00 50.00 0.00	Yes	Yes
	5	Discrete Uniform Geometric Logarithmic Series	100.00 50.00 0.00	Yes	Yes

(All the distributions other than Exponential in Table 6 are special cases of Exponential Distribution.)

Table 6. Validation with ExpertFit: Exponential Probability Distribution

Distribution	Run	Distribution Fit (%) – Top 3	Histogram Fit	Scatter Plot Fit
Exponential	1	Gamma 95.83 Erlang 92.71 Exponential 92.71	Yes	Yes
	2	Gamma 97.50 Weibull 92.50 Erlang 75.00	Yes	Yes
	3	Weibull 88.24 Weibull 85.29 Gamma 82.35	Yes	Yes
	4	Weibull 100.00 Weibull 89.71 Gamma 89.71	Yes	Yes
	5	Weibull 100.00 Gamma 87.50 Exponential 68.74	Yes	Yes

Table 7. Validation with ExpertFit: Extreme Value Type A Probability Distribution

Distribution	Run	Distribution Fit (%) – Top 3	Histogram Fit	Scatter Plot Fit
Extreme Value Type A	1	Extreme Value Type A 100.00 Johnson SB 94.44 Weibull 88.89	Yes	Yes
	2	Extreme Value Type A 100.00 Johnson SB 94.44 Weibull 88.89	Yes	Yes
	3	Extreme Value Type A 96.88 Johnson SB 96.88 Weibull 87.50	Yes	Yes
	4	Extreme Value Type A 98.53 Johnson SB 95.59 Weibull 88.24	Yes	Yes
	5	Extreme Value Type A 98.44 Johnson SB 95.31 Weibull 87.50	Yes	Yes

Table 8. Validation with ExpertFit: Extreme Value Type B Probability Distribution

Distribution	Run	Distribution Fit (%) – Top 3		Histogram Fit	Scatter Plot Fit
Extreme Value Type B	1	Extreme Value Type B Johnson SU Gamma	96.88 96.88 87.50	Yes	Yes
	2	Extreme Value Type B Johnson SU Gamma	100.00 93.75 87.50	Yes	Yes
	3	Johnson SU Extreme Value Type B Gamma	98.44 95.31 87.50	Yes	Yes
	4	Johnson SU Extreme Value Type B Gamma	98.44 95.31 85.94	Yes	Yes
	5	Gamma Extreme Value Type B Erlang	96.74 94.22 92.39	Yes	Yes

Table 9. Validation with ExpertFit: Gamma Probability Distribution

Distribution	Run	Distribution Fit (%) – Top 3		Histogram Fit	Scatter Plot Fit
Gamma	1	Weibull Gamma Exponential	96.88 90.63 68.75	Yes	Yes
	2	Gamma Gamma Weibull	100.00 94.74 88.16	Yes	Yes
	3	Gamma Gamma Weibull	100.00 95.00 90.00	Yes	Yes
	4	Gamma Erlang Gamma	100.00 95.45 90.91	Yes	Yes
	5	Gamma Johnson SB Erlang	100.00 97.22 91.67	Yes	Yes

Table 10. Validation with ExpertFit: Geometric Probability Distribution

Distribution	Run	Distribution Fit (%) – Top 3		Histogram Fit	Scatter Plot Fit
Geometric	1	Geometric Negative Binomial Poisson	83.33 83.33 22.22	Yes	Yes
	2	Geometric Negative Binomial Poisson	83.33 83.33 33.33	Yes	Yes
	3	Geometric Negative Binomial Poisson	72.22 72.22 55.56	Yes	Yes
	4	Geometric Negative Binomial Poisson	72.22 72.22 55.56	Yes	Yes
	5	Geometric Negative Binomial Poisson	72.22 72.22 55.56	Yes	Yes

Table 11. Validation with ExpertFit: Inverse Gaussian Probability Distribution

Distribution	Run	Distribution Fit (%) – Top 3		Histogram Fit	Scatter Plot Fit
Inverse Gaussian	1	Inverse Gaussian Lornormal Weibull	100.00 87.50 75.00	Yes	Yes
	2	Inverse Gaussian Lornormal Weibull	100.00 87.50 75.00	Yes	Yes
	3	Inverse Gaussian Lornormal Weibull	100.00 95.24 90.48	Yes	Yes
	4	Inverse Gaussian Lornormal Weibull	100.00 87.50 75.00	Yes	Yes
	5	Inverse Gaussian Lornormal Weibull	100.00 88.89 77.78	Yes	Yes

Table 12. Validation with ExpertFit: Inverted Weibull Probability Distribution

Distribution	Run	Distribution Fit (%) – Top 3		Histogram Fit	Scatter Plot Fit
Inverted Weibull	1	Log Logistic Inverted Weibull Lognormal	100.00 93.27 93.27	Yes	Yes
	2	Inverted Weibull Gamma Log Logistic	100.00 95.65 91.30	Yes	Yes
	3	Inverted Weibull Log Logistic Erlang	100.00 93.48 91.30	Yes	Yes
	4	Inverted Weibull Gamma Erlang	100.00 94.57 92.39	Yes	Yes
	5	Inverted Weibull Log Logistic Log Laplace	100.00 90.91 81.82	Yes	Yes

Table 13. Validation with ExpertFit: Johnson SB Probability Distribution

Distribution	Run	Distribution Fit (%) – Top 3		Histogram Fit	Scatter Plot Fit
Johnson SB	1	Johnson SB Johnson SB Power Function	100.00 66.67 33.33	Yes	Yes
	2	Johnson SB Beta Power Function	83.33 75.00 41.67	Yes	Yes
	3	Johnson SB Beta Johnson SB	87.50 68.75 56.25	Yes	Yes
	4	Johnson SB Beta Power Function	83.33 75.00 41.67	Yes	Yes
	5	Johnson SB Johnson SB Beta	93.75 68.75 62.50	Yes	Yes

Table 14. Validation with ExpertFit: Johnson SU Probability Distribution

Distribution	Run	Distribution Fit (%) – Top 3		Histogram Fit	Scatter Plot Fit
Johnson SU	1	Johnson SU Extreme Value Type A Weibull	100.00 93.33 83.33	Yes	Yes
	2	Johnson SU Extreme Value Type A Weibull	100.00 93.75 87.50	Yes	Yes
	3	Johnson SU Extreme Value Type A Weibull	100.00 91.18 85.29	Yes	Yes
	4	Johnson SU Extreme Value Type A Weibull	100.00 91.18 85.29	Yes	Yes
	5	Johnson SU Logistic Weibull	100.00 93.75 87.50	Yes	Yes

Table 15. Validation with ExpertFit: Laplace Probability Distribution

Distribution	Run	Distribution Fit (%) – Top 3		Histogram Fit	Scatter Plot Fit
Laplace	1	Laplace Johnson SU Log Laplace	100.00 93.75 85.94	Yes	Yes
	2	Laplace Johnson SU Log Laplace	100.00 93.75 85.94	Yes	Yes
	3	Laplace Johnson SU Log Laplace	100.00 93.75 84.38	Yes	Yes
	4	Laplace Johnson SU Log Laplace	100.00 93.75 87.50	Yes	Yes
	5	Laplace Johnson SU Log Laplace	100.00 93.75 85.94	Yes	Yes

Table 16. Validation with ExpertFit: Logistic Probability Distribution

Distribution	Run	Distribution Fit (%) – Top 3		Histogram Fit	Scatter Plot Fit
Logistic	1	Logistic Johnson SU Normal	100.00 93.75 85.94	Yes	Yes
	2	Johnson SU Logistic Normal	100.00 93.75 87.50	Yes	Yes
	3	Johnson SU Logistic Normal	98.44 95.31 85.94	Yes	Yes
	4	Logistic Johnson SU Normal	96.88 96.88 87.50	Yes	Yes
	5	Logistic Johnson SU Normal	96.88 96.88 87.50	Yes	Yes

Table 17. Validation with ExpertFit: Log Laplace Probability Distribution

Distribution	Run	Distribution Fit (%) – Top 3		Histogram Fit	Scatter Plot Fit
Log Laplace	1	Log Laplace Log Laplace Log Logistic	100.00 95.00 90.00	Yes	Yes
	2	Log Laplace Log Logistic Log Laplace	100.00 92.86 92.86	Yes	Yes
	3	Log Laplace Log Laplace Log Logistic	100.00 95.84 91.67	Yes	Yes
	4	Log Laplace Log Laplace Log Logistic	100.00 94.05 91.67	Yes	Yes
	5	Log Laplace Log Logistic Log Laplace	100.00 83.33 66.67	Yes	Yes

Table 18. Validation with ExpertFit: Log Logistic Probability Distribution

Distribution	Run	Distribution Fit (%) – Top 3		Histogram Fit	Scatter Plot Fit
Log Logistic	1	Log Logistic Log Logistic Lognormal	100.00 95.65 89.13	Yes	Yes
	2	Log Logistic Inverse Gaussian Random Walk	100.00 86.11 72.22	Yes	Yes
	3	Log Logistic Log Logistic Log Laplace	100.00 95.65 88.04	Yes	Yes
	4	Log Logistic Log Logistic Random Walk	100.00 91.67 88.10	Yes	Yes
	5	Log Logistic Log Logistic Log Laplace	100.00 95.65 88.04	Yes	Yes

Table 19. Validation with ExpertFit: Lognormal Probability Distribution

Distribution	Run	Distribution Fit (%) – Top 3		Histogram Fit	Scatter Plot Fit
Log Normal	1	Johnson SB Lognormal Random Walk	98.91 96.74 91.30	Yes	Yes
	2	Lognormal Weibull Inverse Gaussian	100.00 80.00 60.00	Yes	Yes
	3	Lognormal Lognormal Random Walk	98.53 94.12 89.71	Yes	Yes
	4	Lognormal Weibull Gamma	100.00 87.50 75.00	Yes	Yes
	5	Lognormal Pearson Type V Pearson Type VI	98.81 96.43 90.48	Yes	Yes

Table 20. Validation with ExpertFit: Negative Binomial Probability Distribution

Distribution	Run	Distribution Fit (%) – Top 3		Histogram Fit	Scatter Plot Fit
Negative Binomial	1	Negative Binomial Geometric Poisson	83.33 83.33 22.22	Yes	Yes
	2	Negative Binomial Geometric Poisson	100.00 66.67 22.22	Yes	Yes
	3	Negative Binomial Geometric Poisson	100.00 66.67 22.22	Yes	Yes
	4	Negative Binomial Geometric Poisson	100.00 55.56 33.33	Yes	Yes
	5	Negative Binomial Geometric Poisson	100.00 44.44 44.44	Yes	Yes

Table 21. Validation with ExpertFit: Normal Probability Distribution

Distribution	Run	Distribution Fit (%) – Top 3		Histogram Fit	Scatter Plot Fit
Normal	1	Normal Johnson SB Weibull	100.00 94.12 88.24	Yes	Yes
	2	Johnson SB Normal Weibull	98.53 95.59 88.24	Yes	Yes
	3	Johnson SB Normal Weibull	98.44 95.31 87.50	Yes	Yes
	4	Johnson SB Normal Weibull	98.44 95.31 85.94	Yes	Yes
	5	Normal Johnson SB Weibull	98.44 95.31 87.50	Yes	Yes

Table 22. Validation with ExpertFit: Pareto Probability Distribution

Distribution	Run	Distribution Fit (%) – Top 3	Histogram Fit	Scatter Plot Fit
Pareto	1	Pareto 100.00 Extreme Value Type B 80.00 Logistic 50.00	Yes	Yes
	2	Pareto 100.00 Extreme Value Type B 80.00 Logistic 50.00	Yes	Yes
	3	Pareto 100.00 Extreme Value Type B 80.00 Logistic 55.00	Yes	Yes
	4	Pareto 100.00 Extreme Value Type B 80.00 Logistic 55.00	Yes	Yes
	5	Pareto 100.00 Extreme Value Type B 80.00 Logistic 50.00	Yes	Yes

Table 23. Validation with ExpertFit: Pearson Type V Probability Distribution

Distribution	Run	Distribution Fit (%) – Top 3	Histogram Fit	Scatter Plot Fit
Pearson Type V	1	Pearson Type V 100.00 Inverted Weibull 80.00 Log Logistic 50.00	Yes	Yes
	2	Pearson Type V 100.00 Lognormal 94.05 Pearson Type VI 50.00	Yes	Yes
	3	Pearson Type V 100.00 Johnson SB 96.30 Pearson Type VI 92.59	Yes	Yes
	4	Pearson Type V 100.00 Gamma 91.00 Log Logistic 91.00	Yes	Yes
	5	Pearson Type V 100.00 Lognormal 95.00 Inverse Gaussian 93.00	Yes	Yes

Table 24. Validation with ExpertFit: Pearson Type VI Probability Distribution

Distribution	Run	Distribution Fit (%) – Top 3		Histogram Fit	Scatter Plot Fit
Pearson Type VI	1	Weibull Pearson Type VI Pareto	100.00 80.00 50.00	Yes	Yes
	2	Pareto Pearson Type VI Pareto	93.18 90.91 88.64	Yes	Yes
	3	Pearson Type VI Pareto Pareto	100.00 87.50 82.50	Yes	Yes
	4	Pearson Type VI Pareto Pareto	100.00 90.91 81.82	Yes	Yes
	5	Pearson Type VI Pareto Pareto	100.00 88.64 84.09	Yes	Yes

Table 25. Validation with ExpertFit: Poisson Probability Distribution

Distribution	Run	Distribution Fit (%) – Top 3		Histogram Fit	Scatter Plot Fit
Poisson	1	Binomial Poisson Geometric	100.00 66.67 22.22	Yes	Yes
	2	Poisson Negative Binomial Discrete Uniform	88.89 77.78 22.22	Yes	Yes
	3	Poisson Discrete Uniform Geometric	100.00 66.67 82.50	Yes	Yes
	4	Poisson Discrete Uniform Geometric	100.00 66.67 82.50	Yes	Yes
	5	Poisson Discrete Uniform Geometric	100.00 66.67 82.50	Yes	Yes

Table 26. Validation with ExpertFit: Random Walk Probability Distribution

Distribution	Run	Distribution Fit (%) – Top 3		Histogram Fit	Scatter Plot Fit
Random Walk	1	Johnson SB Random Walk Lognormal	98.91 96.74 91.30	Yes	Yes
	2	Random Walk Weibull Gamma	100.00 90.00 80.00	Yes	Yes
	3	Random Walk Weibull Pearson Type VI	100.00 90.91 81.82	Yes	Yes
	4	Random Walk Weibull Gamma	100.00 90.00 80.00	Yes	Yes
	5	Random Walk Gamma Weibull	97.50 92.50 80.00	Yes	Yes

Table 27. Validation with ExpertFit: Triangular Probability Distribution

Distribution	Run	Distribution Fit (%) – Top 3		Histogram Fit	Scatter Plot Fit
Triangular	1	Triangular Beta Johnson SB	98.33 95.00 95.00	Yes	Yes
	2	Triangular Beta Johnson SB	97.50 95.00 95.00	Yes	Yes
	3	Triangular Johnson SB Beta	98.33 95.00 91.67	Yes	Yes
	4	Triangular Inverted Weibull Lognormal	100.00 90.91 72.73	Yes	Yes
	5	Triangular Weibull Erlang	100.00 96.00 90.00	Yes	Yes

Table 28. Validation with ExpertFit: Uniform Probability Distribution

Distribution	Run	Distribution Fit (%) – Top 3		Histogram Fit	Scatter Plot Fit
Uniform	1	Beta Uniform Power Function	97.32 95.54 93.75	Yes	Yes
	2	Uniform Johnson SB Lognormal	100.00 91.67 79.17	Yes	Yes
	3	Uniform Johnson SB Random Walk	98.33 95.00 81.67	Yes	Yes
	4	Uniform Johnson SB Erlang	98.33 95.00 85.00	Yes	Yes
	5	Uniform Johnson SB Erlang	98.33 95.00 85.00	Yes	Yes

Table 29. Validation with ExpertFit: Weibull Probability Distribution

Distribution	Run	Distribution Fit (%) – Top 3		Histogram Fit	Scatter Plot Fit
Weibull	1	Rayleigh Weibull Rayleigh	96.74 94.57 94.57	Yes	Yes
	2	Gamma Weibull Gamma	91.18 88.24 82.35	Yes	Yes
	3	Weibull Weibull Weibull	100.00 95.65 91.30	Yes	Yes
	4	Weibull Weibull Weibull	100.00 95.65 91.30	Yes	Yes
	5	Weibull Weibull Gamma	98.61 95.83 87.50	Yes	Yes

Chapter 7. Conclusions and Future Work

7.1. Conclusions

RVG Web Service provides random variate generation Web services to Web-based simulation models, with its support for 27 univariate probability distributions. The clients of the RVG Web service are Web-based simulation applications carrying out statistical and simulation experiments, that are in need of a source of random variates to feed into their models. The request comes into the RVG Web service in the form of an XML document, which can be validated by the client before sending across through the well-defined request schema. The RVG Web service responds back to the calling simulation application by returning another XML document containing the generated random variates, according to a well-defined reply schema which the client can use to validate it. The RVG Web service is created from the RVG Web application, which is a distributed, multi-tier J2EE application for random variate generation. The architecture of the RVG Web application has been based on best practices applicable to the J2EE platform. In addition to generation of random variates for 27 probability distributions, the RVG Web service also supports generation of statistical data, histogram and scatter plot on the generated random variates; the histogram and scatter plot are created and displayed as SVG, an XML-based grammar for graphics.

7.2. Contributions

- *RVG Web Application:* Underlying the RVG Web service is a J2EE-based Web application that has been developed based on the recommended best practices for the J2EE platform. Apart from forming the bedrock of the RVG Web service, the RVG Web application can be used in isolation, for research and teaching purposes, to illustrate the properties of the probability distributions that it supports.
- *RVG Web Service:* The functionality of random variate generation has been exposed as a Web service so that it can be invoked by Web-based simulation and statistical applications. The RVG Web service has the following salient features:
    - *Probability distributions:* RVG Web service supports 27 probability distributions.
    - *Supported operations:* RVG Web service supports the following operations:
        - *Random variate generation,*
        - *Generation of general statistics,*
        - *Generation of histogram,* and
        - *Generation of scatter plot.*

- *XML Request Schema:* RVG Web service has a well-defined XML request schema that defines the structure of the request to the Web service.
- *XML Reply Schema:* RVG Web service has a well-defined XML reply schema that defines the structure of the reply to the Web service requestor.
- *SVG Graphs:* RVG Web service generates histogram and scatter plots on the generated random variates in SVG, which can be parsed and validated according to a XML Schema just like any other XML document.
- *Document-literal style invocation:* RVG Web service supports the document-literal type of Web service serialization, with the body of the request and reply inserted into the body of the SOAP message as is, with no encodings and no wrappers.

7.3. Future Work

The current implementation demonstrates the core of the design of the RVG Web application. The following are suggested for future work:

- *Web application - User ID*: Presently, there is a Stream Identifier field that is used to decide whether the random variate generation takes place using the user-entered seed value or the last seed value stored in the database. This can be enhanced by having a User Identifier field to remember a returning user. The primary key will then be User ID + Stream ID, instead of being just the Stream ID as of now.
- *Web application - Asynchronous Communication*: Presently RVG Web application supports only synchronous communication. This can be extended to support asynchronous communication through the use of middleware like IBM MQ-series and message-driven beans, which are a new kind of enterprise bean supported in EJB 2.0.
- *Web application Security*: RVG Web application allows anyone to request for random variates as of now. This can be enhanced to included authentication and authorization of the user.
- *Web Service Faults*: The exposed Web Service as of now has a minimal interface that allows a client application to encapsulate the random variate generation request in a SOAP packet, and get the reply in another SOAP packet. Both the request and reply follow their own particular XML grammars, based on the request and reply schemas. Generation of faults in the reply message is minimal. The reply can be extended to provide more detailed fault conditions to the requesting application.
- *Web Service Document-style, Asynchronous communication*: Presently the SOAP interaction between the client and the server follows a literal-encoded RPC-style, with all communication being synchronous. Asynchronous

communication between the requester and the provider can be supported in the future version.

- *Web Service Security*: The Web service can be enhanced to support authentication and authorization of the calling application in the future version.
- *SVG Animation*: The histogram and scatter plot are displayed in SVG, and are static as of now. The future versions can embed animation into these SVG graphs.

Bibliography

Alur, D., Crupi, J., and Malks, D. (2001), *Core J2EE Patterns: Best Practices and Design Strategies*, Prentice-Hall, Palo Alto, CA.

Apache (2003), “Apache Struts project”, http://jakarta.apache.org/struts

Banks, J., Ed. (1998), *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice*, JohnWiley & Sons, New York, NY.

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

Batik (2003), “Batik SVG Toolkit project”, http://xml.apache.org/batik/

Cavaness, C. (2002), *Programming Jakarta Struts*, O’Reilly, Sebastopol, CA.

Cerami, E. (2002), *Web Services Essentials*, First Edition, O’Reilly, Sebastopol, CA.

DOM (1998), “DOM Level 1 Specification”, http://www.w3.org/TR/REC-DOM-Level-1

Englander, R. (2002), *Java and SOAP*, O’Reilly, Sebastopol, CA.

Ewald, T. (2002), *The Argument Against SOAP Encoding*, http://msdn.microsoft.com/Webservices/default.aspx?pull=/library/en-us/dnsoap/html/argsoape.asp

Fowler, M. (2002), *Patterns of Enterprise Application Architecture*, First Edition, Addison Wesley, Palo Alto, CA.

Fowler, M. (1997), *Analysis Patterns: Reusable Object Models*, First Edition, Addison Wesley, Palo Alto, CA .

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994), *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994.

Graham, S., Simeonov, S., Boubez, T., Davis, D., Daniels, G., Nakamura, Y. and R. Neyama (2002), *Building Web Services with Java*, First Edition, Sams, Indianapolis, IN.

Hall, M. (2000), *Core Servlets and JSPs*, Sun Microsystems Press, Palo Alto, CA.

Haas, H (2003), *Web Services Glossary*, http://www.w3.org/TR/2003/WD-ws-gloss-20030514/

Husted, T., Dumoulin, C., Franciscus, G., and Winterfeldt, D. (2003), *Struts in Action*, Manning Publications, Greenwich, CT.

Java Blueprints (2003), “Java Blueprints Website”, http://java.sun.com/j2ee/blueprints/

J2EE Specifications (2003), “Java Servlet 2.3, Enterprise JavaBeans 2.0 and JavaServer Pages 1.2 Specifications”, http://www.jcp.org/aboutJava/communityprocess/final/jsr053

JFreeChart (2003), JFreeChart project, http://www.jfree.org/jfreechart/index.html

Kreger, H. (2001), *Web Services Conceptual Architecture (WSCA 1.0),* IBM Software Group, http://www-3.ibm.com/software/solutions/Webservices/pdf/WSCA.pdf

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

Law, A.M., and Vincent, S. (1995), *ExpertFit User’s Guide*, First Edition, Averill M. Law & Associates, New York, NY.

Marinescu, F. (2002), *EJB Design Patterns*, John Wiley & Sons, New York, NY.

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

McLaughlin, B. (2000), *Java and XML*, O’Reilly, Sebastopol, CA.

Monson-Haefel, R. (2001), *Enterprise JavaBeans*, O’Reilly, Sebastopol, CA.

Roman, E., Ambler, S., and Jewell, T. (2002), *Mastering Enterprise JavaBeans*, Wiley, New York, NY.

Schmeiser, B. (1980), “Random Variate Generation: A Survey,” In *Proceedings of the 1980 Winter Simulation Conference,* A.I. Oren, C.M. Shub, and P.F. Roth, Eds. IEEE, Piscataway, NJ, 79-90.

Schmeiser, B. (1981), “Random Variate Generation, ” In *Proceedings of the 1981 Winter Simulation Conference*, A.I. Oren, C.M. Shub, and P.F. Roth, Eds IEEE, Piscataway, NJ, 227-242.

Shohoud, Y. (2003), *RPC/Literal and Freedom of Choice*, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnWebsrv/html/rpc_literal.asp

Singh, I., Stearns, B., Johnson, M. and the Enterprise Team (2002), *Designing Enterprise Applications with J2EE Platform*, Addison-Wesley, Palo Alto, CA.

Skonnard, A. (2003), *Understanding SOAP*, http://msdn.microsoft.com/Webservices/understanding/Webservicebasics/default.aspx?pull=/library/en-us//dnsoap/html/understandsoap.asp

SOAP (2003), "SOAP Specification (Primer)", http://www.w3.org/TR/soap12-part0/

SVG (2003), "SVG Specification", http://www.w3.org/TR/SVG11/

Tapang, C.C. (2001), *Web Services Description Language (WSDL) Explained*, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnWebsrv/html/wsdlexplained.asp

Tiles (2003), Tiles project, http://jakarta.apache.org/struts/userGuide/dev_tiles.html

Turner, J., Bedell, K. (2002), *Struts Kick Start*, Sams Publishing, Boston, MA.

Wahli, U., Brown, I., Ferraz, F., Schumacher, M., and Sjostrand, H. (2003), *WebSphere Studio Application Developer Version 5 Programming Guide*, IBM Redbook, International Technical Support Organization, New York, NY.

Wahli, U., Denayer, W., Schunk, L., Shaddon, D., Welss, M. (2003), *EJB 2.0 Development with WebSphere Studio Application Developer,* IBM Redbook, International Technical Support Organization, New York, NY.

Wahli, U., Drobnic, M., Gerber, C., Ochoa, G. G., and Schramm, M. (2003), *WebSphere Version 5 Web Services Handbook*, IBM Redbook, International Technical Support Organization, , New York, NY.

WSDL (2003), "WSDL Specification (Core Language)", http://www.w3.org/TR/wsdl12/

XML Schema (2001), "XML schema Specification (Primer)", http://www.w3.org/TR/xmlschema-0/

Appendix A: XML Schema for Request and Reply

A1. Request Schema

```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
 targetNamespace="  http://Session.RVG.src.vt.edu/"
 xmlns:rv=" http://Session.RVG.src.vt.edu/">

     <element name="variateRequest" type="rv:VariateRequestType"/>

   <complexType name="VariateRequestType">
      <sequence>
         <element name="distribution" type="rv:DistributionType" maxOccurs="unbounded"/>
      </sequence>
   </complexType>

   <complexType name="DistributionType">
      <sequence>
         <element name="sequenceID" type="positiveInteger"/>
         <element name="genParameters" type="rv:BaseParameterType"/>
         <element name="genStatistics" type="boolean" default="false" minOccurs="0"/>
         <element name="genHistogram" type="rv:HistogramType" minOccurs="0"/>
         <element name="genScatterPlot" type="boolean" default="false" minOccurs="0"/>
      </sequence>
   </complexType>

   <complexType name="HistogramType">
     <sequence>
              <element name="genHist" type="boolean" default="false"/>
              <element name="numberIntervals" type="positiveInteger"/>
     </sequence>
   </complexType>

   <complexType name="BaseParameterType" abstract="true">
     <sequence>
              <element name="nameDist" type="rv:DistributionNameType"/>
              <element name="streamID" type="positiveInteger"/>
              <element name="numberOfVariates" type="positiveInteger"/>
              <element name="seed" type="double"/>
     </sequence>
   </complexType>

   <complexType name="betaParameterType">
     <complexContent>
              <extension base="rv:BaseParameterType">
                       <sequence>
                                <element name="aValue" type="double"/>
                                <element name="bValue" type="double"/>
                                <element name="shapeValue1" type="double"/>
                                <element name="shapeValue2" type="double"/>
                       </sequence>
```

```
            </extension>
 </complexContent>
</complexType>

<complexType name="binomialParameterType">
 <complexContent>
            <extension base="rv:BaseParameterType">
                        <sequence>
                                    <element name="pValue" type="double"/>
                                    <element name="tValue" type="integer"/>
                        </sequence>
            </extension>
 </complexContent>
</complexType>

<complexType name="dUniformParameterType">
 <complexContent>
            <extension base="rv:BaseParameterType">
                        <sequence>
                                    <element name="lowerValue" type="integer"/>
                                    <element name="upperValue" type="integer"/>
                        </sequence>
            </extension>
 </complexContent>
</complexType>

<complexType name="exponentialParameterType">
 <complexContent>
            <extension base="rv:BaseParameterType">
                        <sequence>
                                    <element name="locationValue" type="double"/>
                                    <element name="scaleValue" type="double"/>
                        </sequence>
            </extension>
 </complexContent>
</complexType>

<complexType name="extremeAParameterType">
 <complexContent>
            <extension base="rv:BaseParameterType">
                        <sequence>
                                    <element name="locationValue" type="double"/>
                                    <element name="scaleValue" type="double"/>
                        </sequence>
            </extension>
 </complexContent>
</complexType>

 <complexType name="extremeBParameterType">
 <complexContent>
            <extension base="rv:BaseParameterType">
                        <sequence>
                                    <element name="locationValue" type="double"/>
                                    <element name="scaleValue" type="double"/>
                        </sequence>
            </extension>
 </complexContent>
</complexType>

<complexType name="gammaParameterType">
 <complexContent>
		<extension base="rv:BaseParameterType">
			<sequence>
				<element name="locationValue" type="double"/>
				<element name="scaleValue" type="double"/>
				<element name="shapeValue" type="double"/>
			</sequence>
		</extension>
 </complexContent>
</complexType>

 <complexType name="geometricParameterType">
 <complexContent>
		<extension base="rv:BaseParameterType">
			<sequence>
				<element name="pValue" type="double"/>
			</sequence>
		</extension>
 </complexContent>
</complexType>

<complexType name="invGaussianParameterType">
 <complexContent>
		<extension base="rv:BaseParameterType">
			<sequence>
				<element name="locationValue" type="double"/>
				<element name="scaleValue" type="double"/>
				<element name="shapeValue" type="double"/>
			</sequence>
		</extension>
 </complexContent>
</complexType>

 <complexType name="invWeibullParameterType">
 <complexContent>
		<extension base="rv:BaseParameterType">
			<sequence>
				<element name="locationValue" type="double"/>
				<element name="scaleValue" type="double"/>
				<element name="shapeValue" type="double"/>
			</sequence>
		</extension>
 </complexContent>
</complexType>

 <complexType name="johnsonSBParameterType">
 <complexContent>
		<extension base="rv:BaseParameterType">
			<sequence>
				<element name="aValue" type="double"/>
				<element name="bValue" type="double"/>
				<element name="shapeValue1" type="double"/>
```

```
                        <element name="shapeValue2" type="double"/>
                </sequence>
        </extension>
 </complexContent>
</complexType>

<complexType name="johnsonSUParameterType">
 <complexContent>
        <extension base="rv:BaseParameterType">
                <sequence>
                        <element name="locationValue" type="double"/>
                        <element name="scaleValue" type="double"/>
                        <element name="shapeValue1" type="double"/>
                        <element name="shapeValue2" type="double"/>
                </sequence>
        </extension>
 </complexContent>
</complexType>

<complexType name="laplaceParameterType">
 <complexContent>
        <extension base="rv:BaseParameterType">
                <sequence>
                        <element name="locationValue" type="double"/>
                        <element name="scaleValue" type="double"/>
                </sequence>
        </extension>
 </complexContent>
</complexType>

<complexType name="logisticParameterType">
 <complexContent>
        <extension base="rv:BaseParameterType">
                <sequence>
                        <element name="locationValue" type="double"/>
                        <element name="scaleValue" type="double"/>
                </sequence>
        </extension>
 </complexContent>
</complexType>

<complexType name="logLaplaceParameterType">
 <complexContent>
        <extension base="rv:BaseParameterType">
                <sequence>
                        <element name="locationValue" type="double"/>
                        <element name="scaleValue" type="double"/>
                        <element name="shapeValue" type="double"/>
                </sequence>
        </extension>
 </complexContent>
</complexType>

<complexType name="logLogisticParameterType">
 <complexContent>
        <extension base="rv:BaseParameterType">
```

```
                <sequence>
                        <element name="locationValue" type="double"/>
                        <element name="scaleValue" type="double"/>
                        <element name="shapeValue" type="double"/>
                </sequence>
        </extension>
 </complexContent>
</complexType>

<complexType name="logNormalParameterType">
 <complexContent>
        <extension base="rv:BaseParameterType">
                <sequence>
                        <element name="locationValue" type="double"/>
                        <element name="scaleValue" type="double"/>
                        <element name="shapeValue" type="double"/>
                </sequence>
        </extension>
 </complexContent>
</complexType>

<complexType name="negBinomialParameterType">
 <complexContent>
        <extension base="rv:BaseParameterType">
                <sequence>
                        <element name="pValue" type="double"/>
                        <element name="sValue" type="integer"/>
                </sequence>
        </extension>
 </complexContent>
</complexType>

<complexType name="normalParameterType">
 <complexContent>
        <extension base="rv:BaseParameterType">
                <sequence>
                        <element name="meanValue" type="double"/>
                        <element name="stddevValue" type="double"/>
                </sequence>
        </extension>
 </complexContent>
</complexType>

<complexType name="paretoParameterType">
 <complexContent>
        <extension base="rv:BaseParameterType">
                <sequence>
                        <element name="locationValue" type="double"/>
                        <element name="scaleValue" type="double"/>
                </sequence>
        </extension>
 </complexContent>
</complexType>

<complexType name="poissonParameterType">
 <complexContent>
```

```
            <extension base="rv:BaseParameterType">
                        <sequence>
                                    <element name="meanValue" type="double"/>
                        </sequence>
            </extension>
 </complexContent>
</complexType>

<complexType name="pearson5ParameterType">
 <complexContent>
            <extension base="rv:BaseParameterType">
                        <sequence>
                                    <element name="locationValue" type="double"/>
                                    <element name="scaleValue" type="double"/>
                                    <element name="shapeValue" type="double"/>
                        </sequence>
            </extension>
 </complexContent>
</complexType>

 <complexType name="pearson6ParameterType">
 <complexContent>
            <extension base="rv:BaseParameterType">
                        <sequence>
                                    <element name="locationValue" type="double"/>
                                    <element name="scaleValue" type="double"/>
                                    <element name="shapeValue1" type="double"/>
                                    <element name="shapeValue2" type="double"/>
                        </sequence>
            </extension>
 </complexContent>
</complexType>

<complexType name="randomWalkParameterType">
 <complexContent>
            <extension base="rv:BaseParameterType">
                        <sequence>
                                    <element name="locationValue" type="double"/>
                                    <element name="scaleValue" type="double"/>
                                    <element name="shapeValue" type="double"/>
                        </sequence>
            </extension>
 </complexContent>
</complexType>

<complexType name="triangularParameterType">
 <complexContent>
            <extension base="rv:BaseParameterType">
                        <sequence>
                                    <element name="minValue" type="double"/>
                                    <element name="maxValue" type="double"/>
                                    <element name="modeValue" type="double"/>
                        </sequence>
            </extension>
 </complexContent>
</complexType>
```

```
<complexType name="uniformParameterType">
 <complexContent>
		<extension base="rv:BaseParameterType">
			<sequence>
				<element name="lowerValue" type="double"/>
				<element name="upperValue" type="double"/>
			</sequence>
		</extension>
 </complexContent>
</complexType>

<complexType name="weibullParameterType">
 <complexContent>
		<extension base="rv:BaseParameterType">
			<sequence>
				<element name="locationValue" type="double"/>
				<element name="scaleValue" type="double"/>
				<element name="shapeValue" type="double"/>
			</sequence>
		</extension>
 </complexContent>
</complexType>

<simpleType name="DistributionNameType">
 <restriction base="string">
			<enumeration value="Beta"/>
			<enumeration value="Binomial"/>
			<enumeration value="DUniform"/>
			<enumeration value="Exponential"/>
			<enumeration value="ExtremeValueA"/>
			<enumeration value="ExtremeValueB"/>
			<enumeration value="Gamma"/>
			<enumeration value="Geometric"/>
			<enumeration value="InverseGaussian"/>
			<enumeration value="InvertedWeibull"/>
			<enumeration value="JohnsonSB"/>
			<enumeration value="JohnsonSU"/>
			<enumeration value="Laplace"/>
			<enumeration value="Logistic"/>
			<enumeration value="LogLaplace"/>
			<enumeration value="LogLogistic"/>
			<enumeration value="LogNormal"/>
			<enumeration value="NegativeBinomial"/>
			<enumeration value="Normal"/>
			<enumeration value="Pareto"/>
			<enumeration value="Pearson5"/>
			<enumeration value="Pearson6"/>
			<enumeration value="Poisson"/>
			<enumeration value="RandomWalk"/>
			<enumeration value="Triangular"/>
			<enumeration value="Uniform"/>
			<enumeration value="Weibull"/>
		</restriction>
 </simpleType>
```

```
</schema>
```

Code Listing 8. RVG Web Service: Request Schema

A2. Reply Schema

```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
 targetNamespace=" http://Session.RVG.src.vt.edu/"
 xmlns:rv=" http://Session.RVG.src.vt.edu/">

<element name="variateReply" type="rv:VariateReplyType"/>

<complexType name="VariateReplyType">
      <sequence>
          <elementname="distributionReply"type="rv:DistributionReplyType"
maxOccurs="unbounded"/>
      </sequence>
</complexType>

<complexType name="DistributionReplyType">
      <sequence>
          <element name="sequenceID" type="positiveInteger"/>
          <element name=”streamID” type=”double”/>
          <element name=”nextSeed” type=”double”/>
          <element name="variateValues" type="rv:VariateValuesType"/>
          <element name="variateStatistics" type="rv:VariateStatisticsType" minOccurs="0"/>
          <element name="variateHistogram" type="string" minOccurs="0"/>
          <element name="variateScatterPlot" type="string" minOccurs="0"/>
      </sequence>
</complexType>


<complexType name="VariateValuesType">
     <sequence>
                <element name="randomVariate" type="double" maxOccurs="unbounded"/>
     </sequence>
</complexType>

<complexType name="VariateStatisticsType">
     <sequence>
                <element name="typeOfValues" type="rv:ValueType"/>
                <element name="distributionName" type="rv:DistributionNameType"/>
                <element name="numberOfValues" type="positiveInteger"/>
                <element name="minValue" type="double"/>
                <element name="maxValue" type="double"/>
                <element name="meanValue" type="double"/>
                <element name="medianValue" type="double"/>
                <element name="varianceValue" type="double"/>
                <element name="skewnessValue" type="double"/>
                <element name="coeffOfVariation" type="double" nillable="true"/>
                <element name="lexisRatio" type="double" nillable="true"/>
     </sequence>
</complexType>
```

```
<simpleType name="ValueType">
     <restriction base="string">
                  <enumeration value="IntegerValued"/>
                  <enumeration value="RealValued"/>
     </restriction>
</simpleType>

<simpleType name="DistributionNameType">
     <restriction base="string">
                               <enumeration value="Beta"/>
                               <enumeration value="Binomial"/>
                               <enumeration value="DUniform"/>
                               <enumeration value="Exponential"/>
                               <enumeration value="ExtremeA"/>
                               <enumeration value="ExtremeB"/>
                               <enumeration value="Gamma"/>
                               <enumeration value="Geometric"/>
                               <enumeration value="InverseGaussian"/>
                               <enumeration value="InvertedWeibull"/>
                               <enumeration value="JohnsonSB"/>
                               <enumeration value="JohnsonSU"/>
                               <enumeration value="Laplace"/>
                               <enumeration value="Logistic"/>
                               <enumeration value="LogLaplace"/>
                               <enumeration value="LogLogistic"/>
                               <enumeration value="LogNormal"/>
                               <enumeration value="NegativeBinomial"/>
                               <enumeration value="Normal"/>
                               <enumeration value="Pareto"/>
                               <enumeration value="Pearson5"/>
                               <enumeration value="Pearson6"/>
                               <enumeration value="Poisson"/>
                               <enumeration value="RandomWalk"/>
                               <enumeration value="Triangular"/>
                               <enumeration value="Uniform"/>
                               <enumeration value="Weibull"/>
                  </restriction>
</simpleType>

</schema>
```

Code Listing 9. RVG Web Service: Reply Schema

Appendix B: RVG Web Application Components

This appendix lists the components of RVG Web application.

B1. Enterprise Beans

Enterprise beans represent business logic and business data.

The session enterprise beans that RVG Web application uses are shown in Table 30, along with a short description of their purposes.

Table 30. RVG Web Application: Session Beans

Enterprise Bean Name	Purpose
RVGDistCart	(Stateful) Maintains contents of user's virtual distribution cart
VariateGenerator	(Stateless) Provides access to random variate generation algorithms

The entity beans are shown in Table 31. All entity beans use Container-Managed Persistence, and track stream ID (primary key), original seed value and last seed value used.

Table 31. RVG Web Application: Entity Beans

Enterprise Bean Name	Purpose
Beta	Provides random variate generation for Beta probability distribution
Binomial	Provides random variate generation for Binomial probability distribution
Duniform	Provides random variate generation for Discrete Uniform probability distribution
Exponential	Provides random variate generation for Exponential probability distribution
Extremea	Provides random variate generation for Extreme Value Type A probability distribution
Extremeb	Provides random variate generation for Extreme Value Type B probability distribution
Gamma	Provides random variate generation for Gamma probability distribution
Geometric	Provides random variate generation for Geometric probability distribution
Inversegauss	Provides random variate generation for Inverse

	Gaussian probability distribution
Invweibull	Provides random variate generation for Inverted Weibull probability distribution
Johnsonsb	Provides random variate generation for Johnson SB probability distribution
Johnsonsu	Provides random variate generation for Johnson SU probability distribution
Laplace	Provides random variate generation for Laplace probability distribution
Logistic	Provides random variate generation for Logistic probability distribution
Loglaplace	Provides random variate generation for Log-Laplace probability distribution
Loglog	Provides random variate generation for Log-Logistic probability distribution
Lognormal	Provides random variate generation for Lognormal probability distribution
Negbinomial	Provides random variate generation for Negative Binomial probability distribution
Normal	Provides random variate generation for Normal probability distribution
Pareto	Provides random variate generation for Pareto probability distribution
Pearson5	Provides random variate generation for Pearson Type V probability distribution
Pearson6	Provides random variate generation for Pearson Type VI probability distribution
Poisson	Provides random variate generation for Poisson probability distribution
Rwalk	Provides random variate generation for Random Walk probability distribution
Triangular	Provides random variate generation for Triangular probability distribution
Uniform	Provides random variate generation for Uniform probability distribution
Weibull	Provides random variate generation for Weibull probability distribution

B2. JSPs

Table 32 lists JSP pages that are common to all the screens of the RVG Web application.

Table 32. RVG Web Application: JSP pages (Common)

JSP Name	Purpose
header.jsp	Displays banner at top of screen
footer.jsp	Displays message in page footer
displayDetail.jsp	Displays random variate generation details of a clicked distribution
displayList.jsp	Displays contents of the distribution cart
errorPage.jsp	Displays error message
genInfoPage.jsp	Displays information that variate generation has succeeded
leftMenu.jsp	Distribution cart options
menu.jsp	Main menu (Home, Add RVG, Generate RVGs, Empty Cart)
selection.jsp	Displays distribution selection drop box

Table 33 lists JSP pages that accept input parameters for different distributions.

Table 33. RVG Web Application: JSP pages (Input Pages)

JSP Name	Purpose
betaInput.jsp	Displays input page for Beta probability distribution
binomial.jsp	Displays input page for Binomial probability distribution
dUniform.jsp	Displays input page for Discrete Uniform probability distribution
exponentialInput.jsp	Displays input page for Exponential probability distribution
extremeAInput.jsp	Displays input page for Extreme Value Type A probability distribution
extremeBInput.jsp	Displays input page for Extreme Value Type B probability distribution
gammaInput.jsp	Displays input page for Gamma probability distribution
geometricInput.jsp	Displays input page for Geometric probability distribution
invGaussianInput.jsp	Displays input page for Inverse Gaussian probability distribution
invWeibullInput.jsp	Displays input page for Inverted Weibull probability distribution
johnsonSBInput.jsp	Displays input page for Johnson SB probability distribution
johnsonSUInput.jsp	Displays input page for Johnson SU

	probability distribution
laplaceInput.jsp	Displays input page for Laplace probability distribution
logisticInput.jsp	Displays input page for Logistic probability distribution
logLaplaceInput.jsp	Displays input page for Log-Laplace probability distribution
logLogisticInput.jsp	Displays input page for Log-Logistic probability distribution
logNormalInput.jsp	Displays input page for Lognormal probability distribution
negBinomialInput.jsp	Displays input page for Negative Binomial probability distribution
normalInput.jsp	Displays input page for Normal probability distribution
paretoInput.jsp	Displays input page for Pareto probability distribution
pearson5Input.jsp	Displays input page for Pearson Type V probability distribution
pearson6Input.jsp	Displays input page for Pearson Type VI probability distribution
poissonInput.jsp	Displays input page for Poisson probability distribution
rWalkInput.jsp	Displays input page for Random Walk probability distribution
triangularInput.jsp	Displays input page for Triangular probability distribution
uniformInput.jsp	Displays input page for Uniform probability distribution
weibullInput.jsp	Displays input page for Weibull probability distribution

Table 34 lists the layout page of RVG Web application.

Table 34. RVG Web Application: JSP page (Layout)

JSP Name	Purpose
myLayout.jsp	Contains layout of all the screens of RVG Web application.

Table 35 lists whole screens used in RVG Web application– screens that are composed of other JSP pages.

Table 35. RVG Web Application: JSP pages (whole screens)

JSP Name	Purpose
WBetaInput.jsp	Displays input screen for Beta probability distribution
WBinomial.jsp	Displays input screen for Binomial probability distribution
WDUniform.jsp	Displays input screen for Discrete Uniform probability distribution
WExponentialInput.jsp	Displays input screen for Exponential probability distribution
WExtremeAInput.jsp	Displays input screen for Extreme Value Type A probability distribution
WExtremeBInput.jsp	Displays input screen for Extreme Value Type B probability distribution
WGammaInput.jsp	Displays input screen for Gamma probability distribution
WGeometricInput.jsp	Displays input screen for Geometric probability distribution
WInvGaussianInput.jsp	Displays input screen for Inverse Gaussian probability distribution
WInvWeibullInput.jsp	Displays input screen for Inverted Weibull probability distribution
WJohnsonSBInput.jsp	Displays input screen for Johnson SB probability distribution
WJohnsonSUInput.jsp	Displays input screen for Johnson SU probability distribution
WLaplaceInput.jsp	Displays input screen for Laplace probability distribution
WLogisticInput.jsp	Displays input screen for Logistic probability distribution
WLogLaplaceInput.jsp	Displays input screen for Log-Laplace probability distribution
WLogLogisticInput.jsp	Displays input screen for Log-Logistic probability distribution
WLogNormalInput.jsp	Displays input screen for Lognormal probability distribution
WNegBinomialInput.jsp	Displays input screen for Negative Binomial probability distribution
WNormalInput.jsp	Displays input screen for Normal probability distribution
WParetoInput.jsp	Displays input screen for Pareto probability

	distribution
WPearson5Input.jsp	Displays input screen for Pearson Type V probability distribution
WPearson6Input.jsp	Displays input screen for Pearson Type VI probability distribution
WPoissonInput.jsp	Displays input screen for Poisson probability distribution
WRWalkInput.jsp	Displays input screen for Random Walk probability distribution
WTriangularInput.jsp	Displays input screen for Triangular probability distribution
WUniformInput.jsp	Displays input screen for Uniform probability distribution
WWeibullInput.jsp	Displays input screen for Weibull probability distribution
WDefaultPage.jsp	Displays distribution selection screen
WErrorPage.jsp	Displays error screen
WGenInfoPage.jsp	Displays information after successful random variate generation screen
WOutputPage.jsp	Displays details of random variate generation for a particular distribution

B3. Servlets

Table 36 lists the servlets used in RVG Web application, along with a short description of their purposes.

Table 36. RVG Web Application: Servlets

Servlet Name	Purpose
ActionServlet	Acts as the controller of the application – handles all requests that end in *.do
RVGDisplayHistogram	Embeds Histogram (SVG) in the output page
RVGDisplayScatterPlot	Embeds Scatter Plot (SVG) in the output page

B4. XML Files

Table 37 lists the XML files in RVG Web application.

Table 37. RVG Web Application: XML files

XML Filename	Purpose
Web.xml	J2EE deployment descriptor for Web-tier components
application.xml	J2EE deployment descriptor for the Web application
ejb-jar.xml	J2EE deployment descriptor for EJB-tier components
struts-config.xml	Struts configuration file
VariateRequestSchema.xsd	Schema definition file for the request
VariateReplySchema.xsd	Schema definition file for reply

B5. Utility & Exception classes

Table 38 lists the Utility and custom Exception classes used in RVG Web application.

Table 38. RVG Web Application: Utility& Exception Classes

Class Name	Purpose
RVGConstants	Stores the connstans used in the application
RVGFileHandling	Contains static methods for file handling
Distribution Bean	Value object for distribution selection
DistParamValueObject	Value object for parameters of distributions
DetailsDistValueObject	Value object for generated distributions
RVGHistogramException	Encapsulates exceptional conditions in histogram creation
RVGScatterPlotException	Encapsulates exceptional conditions in scatter plot creation
RVGHistogramIntervalException	Encapsulates exceptional conditions in the specified number of intervals for histogram creation
RVGInputFormatException	Encapsulates exceptional conditions in the format of input parameters
RVGParameterRangeException	Encapsulates exceptional conditions in the range of input parameters
RVgParametersRelationException	Encapsulates exceptional conditions in the relationship of two parameters

B6. Action classes

Table 39 lists the Action classes that carry out the different actions for user responses in RVG Web application.

Table 39. RVG Web Application: Action classes

Action class name	Purpose
BetaSubmitAction	Comes into play when the user tries to add a Beta distribution to cart
BinomialSubmitAction	Comes into play when the user tries to add a Binomial distribution to cart
DUniformSubmitAction	Comes into play when the user tries to add a Discrete Uniform distribution to cart
ExponentialSubmitAction	Comes into play when the user tries to add an Exponential distribution to cart
ExtremeASubmitAction	Comes into play when the user tries to add an Extreme Value Type A distribution to cart
ExtremeBSubmitAction	Comes into play when the user tries to add an Extreme Value Type B distribution to cart
GammaSubmitAction	Comes into play when the user tries to add a Gamma distribution to cart
GeometricSubmitAction	Comes into play when the user tries to add a Geometric distribution to cart
InvGaussianSubmitAction	Comes into play when the user tries to add a Inverse Gaussian distribution to cart
InvWeibullSubmitAction	Comes into play when the user tries to add a Inverted Weibull distribution to cart
JohnsonSBSubmitAction	Comes into play when the user tries to add a Johnson SB distribution to cart
JohnsonSUSubmitAction	Comes into play when the user tries to add a Johnson SU distribution to cart
LaplaceSubmitAction	Comes into play when the user tries to add a Laplace distribution to cart
LogisticSubmitAction	Comes into play when the user tries to add a Logistic distribution to cart
LogLaplaceSubmitAction	Comes into play when the user tries to add a Log-Laplace distribution to cart
LogLogisticSubmitAction	Comes into play when the user tries to add a Log-Logistic distribution to cart
LogNormalSubmitAction	Comes into play when the user tries to add a Lognormal distribution to cart
NegbinomialSubmitAction	Comes into play when the user tries to add a Negative Binomial distribution to cart
NormalSubmitAction	Comes into play when the user tries to add a Normal distribution to cart
ParetoSubmitAction	Comes into play when the user tries to add a Pareto distribution to cart
Pearson5SubmitAction	Comes into play when the user tries to add

	a Pearson Type V distribution to cart
Pearson6SubmitAction	Comes into play when the user tries to add a Pearson Type VI distribution to cart
PoissonSubmitAction	Comes into play when the user tries to add a Poisson distribution to cart
RwalkSubmitAction	Comes into play when the user tries to add a Random Walk distribution to cart
TriangularSubmitAction	Comes into play when the user tries to add a Triangular distribution to cart
UniformSubmitAction	Comes into play when the user tries to add a Uniform distribution to cart
WeibullSubmitAction	Comes into play when the user tries to add a Weibull distribution to cart
DeleteDistAction	Comes into play when the user tries to delete a distribution from cart
DisplayDistAction	Comes into play when the user tries to view the details of a generated distribution from cart
EditDistAction	Comes into play when the user tries to edit a distribution from cart
GenerateVariatesAction	Comes into play when the user tries to generate random variates from cart
ResetGenerationAction	Comes into play when the user tries to empty the distribution cart
SelectionAction	Comes into play when the user selects a distribution from distribution selection screen

B7. ActionForm classes

Table 40 lists the ActionForm classes that handle validation and capturing request parameters for Action classes to work with in RVG Web application.

Table 40. RVG Web Application: ActionForm classes

ActionForm class name	Purpose
BetaForm	Validates and makes request parameters available for Action class when the user tries to add a Beta distribution to cart
BinomialForm	Validates and makes request parameters available for Action class when the user tries to add a Binomial distribution to cart
DUniformForm	Validates and makes request parameters available for Action class when the user tries to add a Discrete Uniform distribution

	to cart
ExponentialForm	Validates and makes request parameters available for Action class when the user tries to add an Exponential distribution to cart
ExtremeAForm	Validates and makes request parameters available for Action class when the user tries to add an Extreme Value Type A distribution to cart
ExtremeBForm	Validates and makes request parameters available for Action class when the user tries to add an Extreme Value Type B distribution to cart
GammaForm	Validates and makes request parameters available for Action class when the user tries to add a Gamma distribution to cart
GeometricForm	Validates and makes request parameters available for Action class when the user tries to add a Geometric distribution to cart
InvGaussianForm	Validates and makes request parameters available for Action class when the user tries to add a Inverse Gaussian distribution to cart
InvWeibullForm	Validates and makes request parameters available for Action class when the user tries to add a Inverted Weibull distribution to cart
JohnsonSBForm	Validates and makes request parameters available for Action class when the user tries to add a Johnson SB distribution to cart
JohnsonSUForm	Validates and makes request parameters available for Action class when the user tries to add a Johnson SU distribution to cart
LaplaceForm	Validates and makes request parameters available for Action class when the user tries to add a Laplace distribution to cart
LogisticForm	Validates and makes request parameters available for Action class when the user tries to add a Logistic distribution to cart
LogLaplaceForm	Validates and makes request parameters available for Action class when the user tries to add a Log-Laplace distribution to cart
LogLogisticForm	Validates and makes request parameters

	available for Action class when the user tries to add a Log-Logistic distribution to cart
LogNormalForm	Validates and makes request parameters available for Action class when the user tries to add a Lognormal distribution to cart
NegbinomialForm	Validates and makes request parameters available for Action class when the user tries to add a Negative Binomial distribution to cart
NormalForm	Validates and makes request parameters available for Action class when the user tries to add a Normal distribution to cart
ParetoForm	Validates and makes request parameters available for Action class when the user tries to add a Pareto distribution to cart
Pearson5Form	Validates and makes request parameters available for Action class when the user tries to add a Pearson Type V distribution to cart
Pearson6Form	Validates and makes request parameters available for Action class when the user tries to add a Pearson Type VI distribution to cart
PoissonForm	Validates and makes request parameters available for Action class when the user tries to add a Poisson distribution to cart
RwalkForm	Validates and makes request parameters available for Action class when the user tries to add a Random Walk distribution to cart
TriangularForm	Validates and makes request parameters available for Action class when the user tries to add a Triangular distribution to cart
UniformForm	Validates and makes request parameters available for Action class when the user tries to add a Uniform distribution to cart
WeibullForm	Validates and makes request parameters available for Action class when the user tries to add a Weibull distribution to cart
DeleteDistForm	Validates and makes request parameters available for Action class when the user tries to delete a distribution from cart
DisplayDistForm	Validates and makes request parameters available for Action class when the user tries to view the details of a generated

	distribution from cart
EditDistForm	Validates and makes request parameters available for Action class when the user tries to edit a distribution from cart
GenerateVariatesForm	Validates and makes request parameters available for Action class when the user tries to generate random variates from cart
ResetGenerationForm	Validates and makes request parameters available for Action class when the user tries to empty the distribution cart
SelectionForm	Validates and makes request parameters available for Action class when the user selects a distribution from distribution selection screen

B8. Message resources

Table 41 lists the common repository of application-specific messages in RVG Web application.

Table 41. RVG Web Application: Message Resources

Message Resource name	Purpose
ApplicationResources.properties	Provides a maintainable repository of application-specific messages

VITA
Mohammad Sabah

Education

M.S. (Computer Science), August 2003 (GPA: 3.87/4).
Virginia Polytechnic Institute and State University, Blacksburg, Virginia.
Thesis: Random Variate Generation Web Service.

B.E. (Computer Science and Engineering), May 2000.
Regional Engineering College (M.N.R.E.C.), Allahabad, India.

Work Experience

Graduate Teaching Assistant, August 2001 – May 2003.
Department of Computer Science, Virginia Tech, Blacksburg, VA.
Assist students, grade projects and assignments for – Computer Network Architecture, Network Application Programming, Software Engineering, and Introduction to C++.

Software Engineer, June 2000 – June 2001.
River Run Software Group, Noida, India.

CCRMS application for Verizon (Visual C++ 6.0, Win CE, MFC): Developed a resource management application running on Win CE-based handheld PCs. ZModem protocol was used for communication between client and server.

Notification subsystem (Visual Age for Java, WebSphere Application Server, DB2, MQ-Series): Developed and tested a generic system which any WebSphere Everyplace Suite based application can use to implement PUSH functionality.

Software Intern, May 1999 – August 1999.
Variable Energy Cyclotron Center, Kolkata, India.
Developed a tool to automate design of High Voltage two-phase transformer using C to optimize efficiency and performance.

Relevant Projects

Sliding Window, Routing Protocols (C, Linux): Implemented Go-Back-N algorithm for reliable transmission. Implemented Distance Vector and Link State routing protocols for a simulated network of nodes and connections.

C-Interpreter (Java, Linux): Developed the back end (semantic check) of a C-language interpreter. It formed part of a C programming environment with customizable subsets of language features. ANTLR was used as lexer/parser/tree walker.

Performance Evaluation of RMI and JavaSpaces (Java, Windows NT): Through a simulated distributed application of a bmp-jpeg converter, running times of RMI and JavaSpaces were compared and evaluated.

Object Oriented Requirement Analysis and Design (UML, Rational Suite): Robust and scalable design done for an internet e-commerce site having a business model. Requirements were gathered through use cases/scenarios.

Firewall (C, Java, Linux): Developed a firewall that incorporated a packet filter and a proxy server. Packet filter working at network layer was coded in C. Proxy server working at application layer was coded in Java.

Relevant Courses

System Simulation, Distributed Operating Systems, Theory of Algorithms, Software Engineering, Computer Network Architecture, Software Design & Quality, Programming Languages, Computer Architecture, Compiler Design, Operating Systems, Database Management Systems, Data Structures.

Relevant Skills

Languages	: C, C++, Java, UML, XML
Operating System	: Windows NT/2000/CE, Linux, UNIX
Developer tools	: Visual Studio 6.0, WebSphere Studio Application Developer 5.0, Visual Age for Java 3.5
Databases/Servers	: DB2, WebSphere Application Server 5.0.
Miscellaneous	: J2EE, Struts, JUnit, MFC, XSLT, SAX/DOM, SOAP, WSDL, SVG, Rational Rose, ANTLR.

Honors

National Talent Search Examination (NTSE) Scholarship, 1994.

Secured 12th position in Regional Mathematics Olympiad in 1994.

Placed in the top 10% in Indian Standard Examination in Physics in 1996.