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

RE-INTRODUCING WEB-BASED SIMULATION

Steven W. Reichenthal

Boeing
3370 Miraloma Ave.
Anaheim, CA 92803, U.S.A.

ABSTRACT

This paper re-introduces web-based simulation from a web development point of view by first comparing the goals, structures, operations, and communication mechanisms on the web with those of current distributed simulation technology, and then synthesizing a new web-based simulation paradigm that more closely resembles the technology found on the web than Java-HLA solutions. The resulting paradigm is expressed through the Simulation Reference Markup Language (SRML) and Simulation Reference Simulator (SR Simulator) developed through research at Boeing.

1 INTRODUCTION

I feel though that we want something very declarative, rather than procedural, as a base. I would like it to support lazy [evaluation], so that when a bit of embedded video becomes covered by another window, the bandwidth can be saved, or if I can't see the output of a simulation, the CPU can be saved. "If you never have a dream, then you never have a dream come true". (Berners-Lee 1994)

Web technologies and distributed simulation technologies have grown up largely independently, and though distributed simulation preceded the web, we see that the web is now influencing distributed simulation. In the past, web-based simulation was introduced through Java operating over the High-Level Architecture (HLA) (Page 1998). However, let us re-introduce web-based simulation by first comparing the goals, structures, operations, and communication mechanisms on the web with those of current distributed simulation technology, and then by synthesizing a new web-based simulation paradigm that more closely resembles the technology found on the web.

2 GOALS COMPARISON

The web focuses on information accessibility, with the primary goal to "provide a universe of network-accessible information, the embodiment of human knowledge" (WWW 1992). Simulation, however, focuses on information creation and analysis. In this context, the primary goal is to provide knowledge, or answers to important questions through experimentation and modeling, which would otherwise be too difficult or expensive to obtain—i.e., calculate knowledge. Simulation is a process that attempts to predict aspects of the behavior of some system by creating an approximate model of it. Both provide knowledge through computing, however since they focus on different aspects of knowledge, functional requirements and performance may differ significantly.

3 STRUCTURAL COMPARISON

A structural look at the web reveals a foundation built upon the Internet. Standardized web browsers operate at the top of the architecture providing universal information accessibility tools. Browsers receive information in standard HTML documents, parse the documents and generate a standardized hierarchical memory structure named the Document Object Model (DOM). The memory structure includes not only objects contained in the documents but also a fixed set of standardized objects in the browser itself. Web servers operate at distributed locations, and respond to requests for HTML documents using a standardized Hypertext Transport Protocol (HTTP). Many standard and ad-hoc objects flow from a web server to a browser in a client-server fashion. Objects of this nature include: HTML objects, graphics objects, script objects, text objects, and other objects registered as Multi-Purpose Mail Extensions (MIME). Other web automation applications besides web browsers and servers have emerged within the web, for example: proxies, redirectors, filters, robots (BOTs), spiders, and crawlers. Notice that these objects all support the goal of information accessibility. In addition, higher levels of information fidelity operate in the web environment using multimedia and virtual reality techniques.

Supporting the goal of knowledge through modeling, distributed simulation architectures have a slight resemblance. Internet technology operates at the foundation of distributed simulation, however for performance reasons

some simulations may be restricted to an intranet or subnet. We find within the HLA domain, simulation actors characterized as federates (simulations) that operate within a peer-to-peer federation. Federates may participate in the function of a federation or act as observers, logging or visualizing information that passes through the federation. Although the HLA has been standardized and embodied in a combination of software known as the Run-Time Infrastructure (RTI) and the Federate Object Model file (FOM), federates are not standardized. Federates are typically build to suit the needs of the simulation. One reason is that HLA is based on a premise that no simulation can satisfy all uses and users. However, the premise ignores the idea that one standardized simulation tool can satisfy most simulation users, just as one standardized web browser satisfies most web users.

4 OPERATIONAL COMPARISON

Web operation centers on the generation and display of HTML documents--documents that include both data and functionality. Authors generate HTML statically using editors, and software developers generate HTML dynamically with the assistance of program code. Part of the development includes organizing data into documents or other structures such as relational databases. Developers also insert scripts into the HTML stream to provide dynamic behavior in the universal browser. Scripts operate on a standard runtime environment provided by the browser that includes both browser objects and document objects. A document's functionality is tested by loading it into a browser, and perhaps using a script debugger tool. Typically the creation of HTML, development and testing of scripts, takes place in a short development cycle, is considered relatively easy, with little software engineering discipline required. Web browsers retrieve data for display through a process of connecting to a web server, submitting a query, downloading HTML. After retrieving the data, document loading proceeds as the browser parses the HTML data, assembles DOM objects, and compiles the scripts using plug-in language parsers. As the document continues to load, the browser instantiates pre-built objects, downloads graphics, and instantiates plug-in objects. Compiled script behavior also executes as the document loads, and continues to execute after the loading has completed, as user-initiated events or timed events.

Turning to the operations that take place in distributed simulation, we see that HLA has a process for developing federations, called the FEDEP (Lutz 2001). Typically the process includes traditional software development stages such as designing, coding, and testing. Building a federation can take a significant amount work and require a high degree of engineering skill, as the application development interface (API) of the RTI itself includes over 150 members. As federates are developed they are tested both independently and with other federates. Data used in the simulation is pre-processed and associated with the program logic as federates initialize and join the running federation. Federates interchange data using publish-and-subscribe mechanisms built into the HLA. Federates also send and receive messages, called interactions, to other federates. Finally, federates resign from the federation. Once complete, analysts examine the data generated from the simulation.

5 COMMUNICATION COMPARISON

Communication between a browser and a server takes place using the standardized hypertext protocol (HTTP) that operates over TCP/IP. The browser establishes a short client-server connection, sends and receives data, then disconnects. Response time as perceived by the user plays a semi-important role, as web users have grown accustomed to unpredictable wait times. Thus, infrequent user-initiated bursts of data characterize the pattern of communication. Users do not tolerate random errors or omissions in text, thus HTTP communication is lossless.

Communication in HLA is mostly opaque and non-standard. One RTI may implement communication using a proprietary protocol, another may use CORBA, and yet another may simply use inter-process communications. Regardless of the protocol, the communication is peer-peer, which affords much greater flexibility than client-server. Since simulation demands high computational performance, the internet may slow a simulation down significantly. Frequent short bursts of data characterize the pattern of communication in distributed simulation. Distributed simulation may also cause problems in corporate internets where firewalls may block the communication, or where IT systems shut down due to network flooding. Thus HLA simulation typically takes place within a single subnet, rather than across the internet. Furthermore, the flexibility available in some RTI implementations permits both lossless and lossy communications.

6 INTRODUCING WEB-BASED SIMULATION

The foregoing comparisons set the stage for web-based simulation. Ideally web-based simulation has the best of what the web has to offer and the best of what HLA has to offer. The notion of web-based simulation is probably as old as the web itself. However, past representations characterized web-based simulation as the Java programming language operating over HLA. Unfortunately, that architecture bears only a slight resemblance to the web. Therefore, rather than looking at the web-based simulation from the eyes of a simulation developer, we look at web-based simulation from the eyes of a web developer. Given this point of reference, the primarily goal of web-based simulation is to enable simulations to be served, received, and processed in a standard fashion using Internet technologies

and the World Wide Web. We present web-based simulation using the Simulation Reference Markup Language (SRML), and the Simulation Reference Simulator (SR Simulator). SRML and the SR Simulator were both developed at Boeing and are currently in use on several major simulation projects across the company. Just as HTML represents web documents, SRML represents simulation models. Likewise, as the web browser represents a universal client application, so the SR Simulator represents a universal simulator. As HTML contains both declarative and procedural definitions, SRML binds the declarative with the procedural.

7 DECLARATIVE AND PROCEDURAL

SRML is an XML-based language that provides generic simulation markup for adding behavior to arbitrary XML documents. XML documents are fundamentally declarative. XML consists of elements specified using tags (words bracketed by '<' and '>'). Elements can have attributes (described by a name-value pairs), and can contain embedded elements:

```
<elementname1 attributename1='value1'
        attributename2='value2' ...>
    <embedded-element1 ...>
        ...
    </embedded-element1>
    ...
</elementname1>
```

These constructs can naturally describe hierarchically related and networked data—literal data and meta-data. An XML Schema is an XML document that permits a modeler to specify rules about element relationships and attributes that can be validated with software. XML is said to be a declarative language because the tags specify "what" without specifying "how". In other words data structure is explicitly represented while operations are either applied or implied externally to the declaration. Procedural languages like Java, JavaScript, and C++ explicitly represent operations as algorithms. A semantic bridge in XML between the declarative and the procedural lies in this construct: "`<elementname1 ...>`". Programming languages that process XML, provide the meaning or behavior of elementname1, but in a separately compiled program. SRML, also allows the semantics and behavior of elementname1 to be specified within the XML constructs using Script tags. This resembles the way that script is added to HTML, yet much more flexible. Rather than just the document scope and procedure scope provided by HTML, SRML introduces item and item class scopes. The advantage of these new scopes enables the broad functionality requirements needed by simulations, which would otherwise be cumbersome in HTML.

8 WEB-BASED SIMULATION STRUCTURE AND OPERATION

In web-based simulation the universal SR Simulator is a software component that can be plugged-in to a web browser or other tool. Simulations can run in many environments including spreadsheets and even custom embedded applications. A single software component can support all of these environments. A modeler can choose any host environment that is capable of creating a Simulation object and using its interface. Possible hosts include Microsoft Internet Explorer, Microsoft Office applications, Microsoft Visual Basic, and Java. An SR Simulator provides a class of Simulation objects. Each Simulation object encapsulates a simulation instance (or run), and is primarily responsible for providing the runtime environment. Since a Simulation is an object that embodies a particular execution of a simulation, many simultaneous simulations can exist on the same computer, or distributed across a network as a client or server. Figure 1 shows the architecture of the SR Simulator in its external environment.

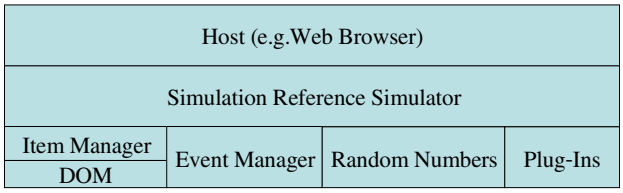

Figure 1: SR Simulator Architecture

The SRML runtime environment is a collection of software objects that can read SRML input, build and manage the corresponding simulation items, connect those items together, and provide an event-driven mechanism for items to communicate. It also provides simulation support in the form of a random number generator, simulation primitives such as time averages and data structures, math functions, statistics functions, and the ability to generate outputs as defined by the model.

Developing a simulation in SRML is like building a web page, and thus the modeling process for a web-based simulation resembles the web page authoring process, whereas the distribution process resembles the FEDEP with an HLA plug-in for distribution. In that way, federates can be rapidly developed and naturally extended when HLA is desired or required. Simulation models combine XML, scripts, links to sub-models, and plug-in declarations. One main difference between SRML and HTML is that SRML is used with domain-specific XML documents as the means for providing simulation behavior. When a model is loaded, the component pieces are assembled, much the same way that an HTML page consisting of data specified externally is assembled. The resulting assembly has a memory representation within the DOM used by the simulator. During simulation execution, the HLA plug-in

operates seamlessly with the simulator so that the model has minimal awareness that other objects exist externally. In addition to functionality provided by HLA, additional objects move from simulator to simulator using HTTP or similar protocol.

9 SRML LANGUAGE CONCEPTS

The things a modeler describes in SRML are referred to as items. An item can represent a physical thing, such as a piece of equipment, or a person, or an entire system of other items. An item may also represent a process or a step in a process. Items can have properties. For example, a piece of equipment might have a 'serial number' property, and a process might have 'duration' property. SRML provides a natural way to express items, their properties, and complex item relationships, using the grammar of XML with a small set of pre-defined elements and attributes that have specific meanings and rules. Items and properties correspond to elements and attributes in the XML. With the exception of a few pre-defined element names, such as ItemClass, SRML uses the elements and attribute names that a modeler defines. This makes it possible to create a domain-specific XML schema to validate a system's structure and to provide data types for the attributes. A domain-specific schema works in conjunction with the SRML schema when using XML namespaces. The following code shows a sample model in SRML.

```
<Simulation Name='ControllerSimulation'>
        <Controller Name='TS293847' Operable='1'
        Pings='0' Health='1'>
                <Script Type='text/javascript'>
                <![CDATA[
                function poll()
                        {
                        Items(1).queryStatus();
                        Items(2).queryStatus();
                        }
                function receive(Name, Value)
                        {
                        Health=math.min(Health,
                        Value);
                        Pings++;
                        }
                ]]>
                </Script>
                <Sensor Name='TSM5865'
                    Quantity='2'>
                        <Script
                        Type='text/javascript'>
                        <![CDATA[
                        function queryStatus()
                                {
                                Location.
                                receive(Name,
                                Random());
                                }
                        ]]>
                        </Script>
                </Sensor>
        </Controller>
</Simulation>
```

Figure 2 shows a simplified conceptual model of an item. Internally an item has a unique system-generated ItemID and a script. It has an association with a DOM node—from the element. It can both serve as a location, and belong at a location along with other items. It can have links to other items, and be a target for a link. Also, it can belong to an item class, which in turn can have super-classes. An item gets its properties directly from DOM attributes, and its behavior from script. The ItemClass element allows the modeler to generalize groups of common items, yet an item does not need to have a corresponding ItemClass.

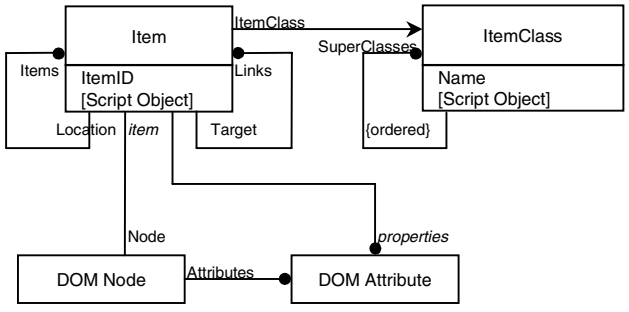

Figure 2: Simplified Item Conceptual Model

The following example shows the definition of a Vehicle element and Name attribute in XML, which corresponds to a Vehicle item that has a Name property with a value of 'Challenger', and a VIN of 'N3462983':

```
    <Vehicle Name='Challenger' VIN='N3462983'/>
```

10 ITEM BEHAVIOR

The modeler provides an item's behavior inside a Script element using a language such as JavaScript or VBScript, and specifying the Type attribute as either 'text/javascript', 'text/vbscript', or any other simulator-supported language. Refer to the following example:

```
<Vehicle ID='AirforceOne' Running='1'>
        <Script Type='text/javascript'>
        <![CDATA[
        var Occupied=0;
        function turnOn()
            {
            Running=1;
            }
        function turnOff()
            {
            Running=0;
            }
        ]]>
        </Script>
    </Vehicle>
```

11 ITEM CLASSES

While SRML allows the modeler to provide specific attributes and behavior for individual items, it also provides a

generalization mechanism for creating classes of items. An item class is analogous to class in object-oriented terms, in that it describes the attributes and behavior for its instances while also providing a type of inheritance. Any number of item classes may exist in a model by using an ItemClass element and specifying a unique Name. Within the class is an embedded instance prototype element that has a tag name which matches the Name attribute of the item class. This prototype can have attributes with default values and data types validated by an XML schema. Refer to the general form of an ItemClass:

```
<ItemClass Name='itemclassname1'
    SuperClasses='itemclass1 itemclass2…'
    property1='value1' …>
    <!-- The following element defines the
    behavior for the class; not individual
    instances -->
        <Script Type='script-language'>
        code
        </Script>
        <!-- The following element defines the
        prototype for the instances -->
        <itemclassname1 property1='value1' …>
            <!-- The following element defines the
            behavior for the instances -->
                <Script Type='script-language'>
                code
                </Script>
        </itemclassname1>
    </ItemClass>
```

The highlighted area indicates the prototype item, and the bold text shows the correspondence between the name of the prototype and the name of the item class. The specific example below, which also highlights the prototype item, demonstrates the definition of an item class called Counter. Each of the ten instances created from this class will have a method called Increment and will have its own Count. As an item itself, the ItemClass can also have properties and behavior that will be shared by the instances. The example defines an Instances property for the class that keeps track of the number of individual Counter instances. Any of the Counter instances can access its item class, through its intrinsic ItemClass property. Notice how the item accesses the Instances property in its item class in the code. In the note below, the code within the instance is directly updating the Instances value of the item class.

```
<Simulation>
    <ItemClass Name='counter' Instances='0'>
        <Counter Count='0'>
            <Script Type='text/javascript'>
            <![CDATA[
            ItemClass.Instances++;
            Function Increment ()
                {
                Count++;
                }
            ]]>
            </Script>
        </Counter>
    </ItemClass>
    <Counter Quantity='10'/>
</Simulation>
```

12 ITEM QUANTITIES

A situation may arise when a large quantity of a particular item or structure needs to occur. Within any element, the modeler can provide the Quantity attribute, which instructs an SRML simulator to duplicate the element and its contents. For example, the following structure defines a total of 240 eggs, in a total of 20 egg cartons in two refrigerators.

```
<Refrigerator Quantity='2'>
    <EggCarton Quantity='10'>
        <Egg Quantity='12' />
    </EggCarton>
</Refrigerator>
```

13 LINKS

XML provides an ideal structure for describing hierarchical containment relationships; however, situations may arise when items at various locations in a simulation need to have non-hierarchical connections to other items. Therefore SRML provides two linking mechanisms. The first involves using the Link element where the modeler provide the name and an XPath (Clark and DeRose 1999) expression to identify the target. The second mechanism uses the Links element where the modeler provides a name and supply inner links. When the modeler uses a schema and provides attributes of type IDREF or IDREFS (Bray, et. al. 2000), SRML treats them in the same way it treats links.

14 INTEGRATION, DISTRIBUTION, AND COMMUNICATION

With the inherent modularity provided by XML, a single document may suffice for representing a large simulation. Many simulated items may take similar forms, and thus the mechanism of item classes provides the integration of common behavior. However, managing a single monolithic XML document does not work well when the pieces are developed independently, or when they need to be reused. External scripts provide a simple way to detach the behavior from the structure, thus allowing code reuse at the cost of some encapsulation. These are specified by adding a Source attribute to a Script element. For example:

```
<Script Type='text/javascript'
Source='file:///Vehicle.js'/>
```

The entire behavior of an item or item class can thus be specified separately. The reason that encapsulation is slightly compromised comes from the fact that the structural aspects of the item, being in XML, have been separated from the behavioral aspect, which is in a separate script. A more sophisticated technique known as embedded

externals, allows portions of one simulation model to be extracted from another, thus preserving encapsulation at the potential cost of a larger runtime footprint. You can specify an embedded external in several ways:

```
<Vehicle
Source='http://srml.boeing.com/Vehicle.xml'/>
<ItemClass Name='Vehicle'
Source='file://Vehicle.xml'/>
<Vehicle Source='VehicleComponent.Tank'/>
```

In the first case, the vehicle is defined completely in the XML document: Vehicle.XML, which is similar to creating an instance from an external item class—many such identical vehicles can be added to the simulation model using the same source file. In the second case, the item class defined in the external file is included in the current model, thus making it available for instancing or sub-classing. The third case shows how separately compiled component-based objects, known as external items, are plugged into the simulation. External items coexist with items in a simulation model, and also provide the capability for entire simulation models to communicate using any combination of CORBA, SOAP, COM, or HLA.

15 CONCLUSION

This paper re-introduced web-based simulation from a web development point of view using the SRML and the SR Simulator as models that parallel the goals, structure, and behavior of the web, while enabling the benefits of distributed simulation through the HLA.

REFERENCES

Berners-Lee, Tim. 1994. Available online via <http://www.w3.org/Bugs/GraphicalComposition.html> [accessed June 1, 2002].

Bray, Tim, et. al. 2000. Extensible Markup Language (XML) 1.0 (Second Edition). Available online via <http://www.w3.org/TR/2000/REC-xml-20001006> [accessed June 1, 2002].

Clark, James and DeRose, Steve. 1999. XML Path Language (XPath) Version 1.0. Available online via <http://www.w3.org/TR/xpath> [accessed June 1, 2002].

Lutz, Bob. 2001. HLA Federation Development and Execution Process (FEDEP). Available online via <http://www.sisostds.org/stdsdev/fedep/index.htm> [accessed June 1, 2002].

Page, Ernest H. 1998. Web-Based Simulation. Available online via <www.mitre.org/pubs/edge/august_98/wbs.html> [accessed June 1, 2002].

WWW. 1994. Available online via <http://www.w3.org/WWW/> [accessed June 1, 2002].

AUTHOR BIOGRAPHY

STEVE REICHENTHAL is a simulation developer at Boeing with the Phantom Works organization and is an adjunct professor at the California State University in Fullerton. He received his Masters degree in Computer Science in 1993 and his MBA degree in 2000. His email address is <steven.w.reichenthal@boeing.com>.