X3D Distributed Interactive Simulation (DIS) Implementation and Run-Time Discovery of New Entities using X3DOM

McGregor, Don; Harder, Byron; Brutzman, Don

X3D Distributed Interactive Simulation (DIS) Implementation and Run-Time Discovery of New Entities using X3DOM

Don McGregor, Byron Harder and Don Brutzman
Modeling, Virtual Environments Simulation (MOVES) Institute
Naval Postgraduate School, Monterey California USA 93933-5000

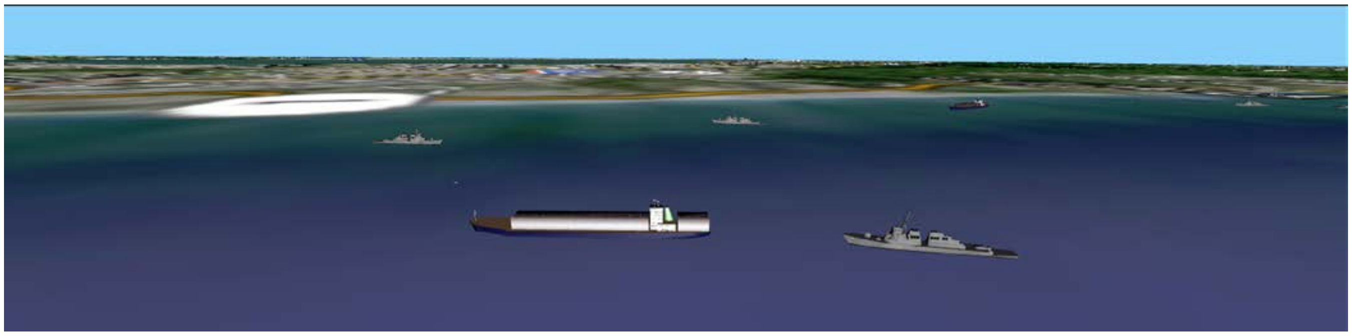

Figure 1: *X3D ship models driven by Distributed Interactive Simulation (DIS) protocol packets, originating from an Automatic Identification System (AIS) commercial tracking service, rendered in browser-based X3DOM player.*

Abstract

New capabilities in web browser JavaScript implementations including networking, improved graphics performance, and improved speed allow the implementation of Networked Virtual Environments (NVEs) inside the web browser. An NVE can be written in JavaScript, which enables deployment in the enterprise entirely from a web server without the use of browser plugins. We discuss one implementation of this idea using X3DOM, an open-source implementation of the X3D standard written in JavaScript. The Open-DIS library for the IEEE Distributed Interactive Simulation (DIS) network protocol is used to create a partial implementation of the X3D standard's DIS profile. Mechanisms for using the X3D DIS Profile DISEntityTypeMapping and DISEntityManager to enable run-time discovery and launching of new entities are discussed. Measurements of the capabilities and performance aspects of Websockets for network transport demonstrate excellent results.

CR Categories: I.3.3 [Computer Graphics]: Three-Dimensional Graphics

Keywords: X3D, JavaScript, DIS, X3DOM, Websockets, NVEs

*e-mail: mcgredo@nps.edu
brharder1@nps.edu
brutzman@nps.edu

Web3D '15, June 18 - 21, 2015, HERAKLION, Greece
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-3647-5/15/06…$15.00
DOI: http://dx.doi.org/10.1145/2775292.2775318

1 X3D and Networked Virtual Environments

The Extensible 3D (X3D) standard provides an XML-based format that can be used to describe 3D objects and scenes [Brutzman and Daly 2007]. X3D format files can be downloaded from a web server and rendered inside a web browser by software dedicated to this purpose. While the X3D scene in each browser may implement time-based animation via an event model combined with features such as *TimeSensor* and *Interpolator* nodes, the animations in each web browser to which the scene has been downloaded are not coordinated or synchronized with scenes loaded in other browsers—the animation in each application's scene proceeds independently of the animation of other scenes loaded in other browsers.

Games and simulations require the ability to coordinate object movement in one browser's scene with other applications. For example we might cause a vehicle in a scene on one computer to move, while observing the approximately synchronized movement of that object in a scene on another host. The vehicle should be in the same position and have the same orientation in both scenes. This class of application is sometimes called a Networked Virtual Environment (NVE).

NVEs require that changes to a scene's state be sent to other hosts. This makes NVEs inherently more difficult to implement than animation in a single scene on a single host. Events may arrive asynchronously, network latency and reliability may affect the application state updates, and applications need to agree on coordinate systems, message formats, and many other issues arise.

Sending and receiving state updates from inside a web browser has traditionally required the use of a browser plugin, a requirement that often runs afoul of enterprise security policies. But the adoption of recent web standards means that web pages can now use JavaScript, executed in the downloaded HTML page, to communicate scene graph state changes to other hosts. This paper discusses one implementation of this idea.

2 Web Standards

Together the X3D standard, existing Institute for Electrical and Electronic Engineers (IEEE) standards, and Internet Engineering Task Force (IETF) and World Wide Web Consortium (W3C) standards can be used to implement a NVE in X3D running inside a web browser.

2.1 JavaScript and WebGL

Today nearly all web browsers contain a JavaScript engine that allows scripts to run inside the web page. Competition between browser vendors over the course of the last few years has resulted in improved JavaScript implementations, and now for some tasks JavaScript applications running in a web browser are nearly in the same performance class as Java or C++ programs. In addition the WebGL standard provides native OpenGL bindings for JavaScript that can exploit the accelerated graphics hardware that may be present on the host, which allows significant 3D scenes can be displayed with good performance. The X3DOM project [Behr et al. 2011] has implemented an X3D viewer entirely in JavaScript using WebGL. This means a client browser can download the X3D rendering software at page load time, download the X3D scene, and the entire HTML page can be viewed in a web page without a plugin.

2.2 Distributed Interactive Simulation (DIS)

Moving an entity in a distributed 3D scene requires at a minimum an agreed-upon standard for describing the position and orientation of the entities in the scene graph. The military did much of the early research on interactive and distributed 3D graphics, and has been using NVEs in training for decades. One standard in widespread use by the defense industry is Distributed Interactive Simulation (DIS), which has been standardized by the IEEE. The DIS standard encapsulates many of the lessons learned for NVE implementations, such as the use of coordinate systems, dead reckoning interpolation of movement, and a system of identifiers to simplify the exchange of semantics. The DIS standard describes a binary format for the content of state update messages, along with semantics such as identifiers, coordinate systems, and entity discovery. Formats other than binary have also been discussed [McGregor et al. 2006] [Swan 2014].

A JavaScript implementation of the DIS standard, Open-DIS, has been written by the authors and is available at GitHub and SourceForge [McGregor et al. 2008] [McGregor et al. 2006].

The X3D standards group recognized the usefulness of DIS for this problem domain and approved the optional “DIS profile” for use in X3D. The profile describes the X3D nodes used to interoperate with DIS [Brutzman and Daly 2012].

2.3 WebSockets, WebRTC Networking

While the DIS profile of X3D describes an interface to DIS, it is silent about the transport mechanism used to send DIS state updates. In the past this task has been exceedingly difficult to achieve from inside a web page. Originally X3D rendering inside the web browser had to be implemented by a Java or C++ browser plugin. Since the plugin was written in a conventional language, a TCP/IP protocol socket might be opened inside the plugin to send messages to an arbitrary address on the Internet. However such low-level access often conflicts with enterprise and individual browser security policies, and the networked state updates often had problems traversing host and enterprise firewalls. As a result 3D software was often difficult to deploy in an enterprise environment. Asynchronous JavaScript and XML (AJAX) provides an alternative transport method, but because AJAX’s architecture is based on the web page polling the web server it does not provide message latency low enough to be useful for most interactive NVEs.

The W3C and the IETF have cooperated on a new standard, Websockets [Wang et al. 2013], [Kapetanakis et al. 2013], [McGregor 2012]. This allows JavaScript to open a TCP socket to a server and exchange data using a simple API. The Websocket connection was designed with modern network proxy architectures in mind, and typically connects to port 80 or 443 on the destination server, which allows it to traverse most host and enterprise firewalls and network architectures. The Websocket upgrades an HTTP connection to a TCP socket that is able to transmit arbitrary data without wrapping it inside the HTTP protocol. The Websocket transport method provides the advantages of TCP sockets (reliability, in-order delivery, rate limiting) but is also limited by TCP socket disadvantages (higher jitter, poor performance in networks that exhibit packet loss).

The W3C and IETF have cooperated on another standard for JavaScript access to socket-based communication between hosts, WebRTC [Johnston and Burnett, 2012]. WebRTC is primarily intended for audio and video conferencing, but also allows arbitrary data to be sent between hosts. The WebRTC standard is closely matched to the TCP/IP suite’s UDP sockets, just as Websockets are to TCP sockets, but again with some added JavaScript API features. In contrast to TCP sockets UDP sockets may have unreliable message delivery and may deliver messages out of order. NVE state updates, which are frequent and need not be delivered reliably, have traditionally used UDP transport.

The governing IETF WebRTC Request For Comments (RFC) documents are still in draft status at the current writing, and browser support for the WebRTC standard in deployed web browsers is more limited than that of Websockets.

Either of these networking standards can be used to send application state updates and implement a NVE. Chen-Fu Hsiao [2014] demonstrated a 3D NVE that made use of both Websockets and WebRTC in an effort to benchmark performance for both options. Typical stream throughput of thousands of state-update PDUs per second can support quite large interactive NVEs.

There is one significant limitation in web-based networking: there is no concept of broadcast or multicast, which allows a single copy of a message to be sent to many recipients. Neither of these concepts can be supported in TCP, the underlying technology for Websockets. WebRTC does not support broadcast or multicast at this writing, and it is unlikely to in the near future.

2.4 Geospatial Considerations

DIS uses a Cartesian coordinate system with the origin at the center of the earth. This might seem like an odd choice, but an earth-centered coordinate system is more mathematically tractable if one wishes to convert it to one of the several more commonly used coordinate systems. In contrast X3D uses a Cartesian coordinate system with the origin at an arbitrary point, not necessarily tied to a particular place on the earth. It is unrealistic to use simply place the origin of the X3D scene at the center of the earth because the entities on the surface of the earth would be placed several million meters out on the axes. We can

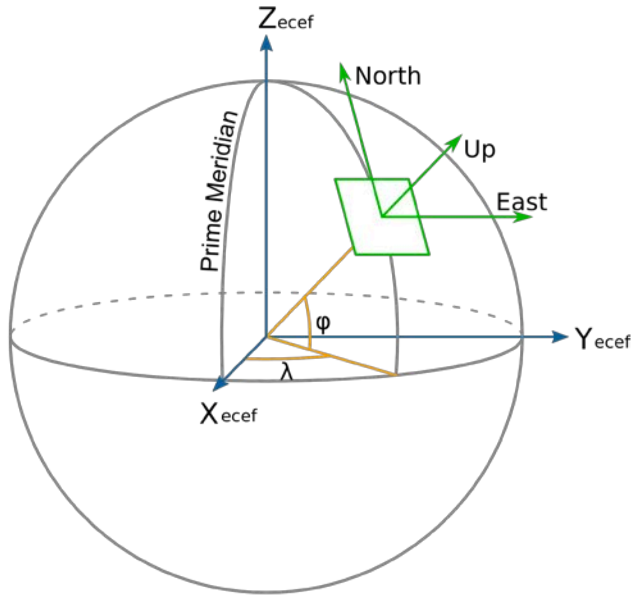

Figure 2: *DIS and scene coordinate systems*

accommodate the two coordinate systems by placing the origin of the X3D coordinate system at a reference point in the DIS coordinate system, and then converting between the two coordinate systems. For example, we can place the origin of the X3D coordinate system at a given latitude, longitude, and altitude, and create a plane tangent to the earth. The plane corresponds to the X3D coordinate system. When state updates are sent the coordinates of the object's X3D coordinates are converted to the DIS coordinates and used in the state update message.

Many software simulations assume a flat earth. This is not an issue for sufficiently small applications or homogeneous NVEs. But use cases typically call for different simulation types within the NVE—each of which may or may not assume a flat earth. As demonstrated in Figure 3, this can result in incorrect entity altitudes. The simplest fix to this, projecting the entities to ground level by zeroing their local y-coordinates (in the case of spheroid-to-flat transforms), results in location error in the xz plane or in a warped terrain map, depending on where the developers choose to take the error. Entity velocities are also affected. These issues can become visually apparent over surprisingly small distances, resulting in artifacts such as entity collisions that make no visual sense.

One of the most significant benefits of a 3D simulation is the ability to visualize the impact of terrain on a situation—for example, on intervisibility or trafficability. All of today's virtual environments have some kind of terrain representation to describe elevation deviations from the perfect oblate spheroid, but there is little standardization on terrain representation. Even if two terrain models use the same level of precision in their data, they will not necessarily agree at each point or in how to interpolate between points. In a networked simulation environment, this results in entities that seem to tunnel through the earth or levitate. Since terrain representations are often tightly coupled within simulations, restricting applications a single terrain data service is usually not a viable solution. The only option left, assuming that realistic-looking behavior is a priority on the user's screen, is to use *terrain clamping*. Similar to the flat earth-projection described above, terrain clamping translates an entity up or down (on the y-axis or from the center of the earth) until it touches the ground. Terrain clamping can also adjust rotations to minimize visual artifacts.

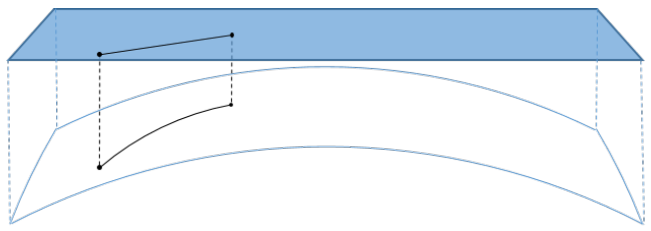

Figure 3: *Flat earth issues*

X3DOM does not currently have a terrain-clamping feature. This work is investigating the feasibility of a generally useful module to support this. The obvious starting point for such a module is to support terrain data presented in the X3D Geospatial component's format. The geospatial nodes are available in X3DOM's current development release. It should prove useful to keep terrain features decoupled from direct DIS-to-X3D coordinate translation if different terrain approaches will be supported in the future.

Flat-earth issues and terrain representation are two different questions that can manifest in many different combinations. Although we could pick a single standard and ignore all others, that approach may not be supportive enough of the current modeling and simulation environment to inspire greater X3D adoption. One experimental alternative is to include attribute settings in the X3D DIS module that allow users to account for the most common possibilities. Some X3D specification work is needed to align the DIS and Geospatial components completely.

2.5 Modeling Conventions

While X3D provides a standard for describing 3D objects, there are more issues that can cause problems when actually using a model in a networked virtual environment. The essence of a virtual environment is multiple 3D objects integrated into one scene. Models may all be in the X3D format, but may still have different scales or different orientations or different ideas about which way is "up" for the model if conventions and requirements are not followed. For example the conventional way to model aircraft is to place the local origin at the center of gravity with the Z axis pointing down, the X axis pointing out the nose, and the Y axis out the right wing. Other models, such as vehicles, may assume that "up" for the object points through the roof of the vehicle. Two X3D model creators, each working within their respective modeling domain assumptions, may create entirely valid models that when integrated in the same scene appear to be upside down. The X3D Scene Authoring Hints document the correct right-hand rule guidance: X axis is vehicle direction, Y axis is up, and Z axis is right (starboard) side of vehicle.

This can be addressed by both convention and by including meta information in the X3D model. Conventions may include agreements about where to place the object local coordinate system—for example at the center of the object, rather than at one end. Some of the model assumptions can be encoded in the X3D file itself. Since X3D includes strictly typed metadata, developers can easily include metadata that describe each model's assumptions and the parameters necessary to integrate the object into a larger scene. The Savage Model Analysis Library (SMAL) conventions provides one such approach.

3 Implementation

3.1 Networking

We chose to use the Websockets standard to implement the transport layer of DIS networking for X3D. This requires that a

Websocket standards-compliant server be present to accept client connections. We implemented a central server in Jetty, a Java-based modular web application container. Clients specify a connection point URL, the websocket client/server handshake is performed, and an underlying TCP socket is established to the websocket server.

The Websocket server implementation acts as a central hub for message distributon; the server repeats a copy of each message sent by a client to any other client that has established a connection to the server. The Websocket server can also act as a conventional web server to provide HTML, JavaScript, and X3DOM files to clients via HTTP or HTTPS requests.

The JavaScript implementation of X3DOM uses information specified in the X3D file to establish a connection to the Websocket server at the time the page is loaded in the client browser. All data on the client web page is loaded from the webserver: HTML pages, including the JavaScript X3DOM implementation, the X3D models, and the DIS implementation.

3.2 Implementing X3D DIS Component Nodes

X3DOM is a partial implementation of the X3D standard [ISO/IEC 2013] and is written entirely in JavaScript and WebGL. This means that X3DOM can be downloaded from a web server, run in a web browser, and display X3D content without a plugin. The optional DIS X3D profile, documented in Chapter 28 of the X3D standard, is not implemented by X3DOM. We have implemented the most important nodes described in the X3D DIS Component as a proof of concept extension to X3DOM, with further completion of remaining nodes expected to follow.

The *DISEntityManager* node in the X3D DIS profile is responsible for network communications and managing all entities added from the network. When a state update from the network comes in this node is responsible for decoding the message and sending the state update to a corresponding *EspduTransform* node.

DIS uses the concept of an "entity type" record, which consists of several numbers that, taken together, are a descriptor of a particular piece of military or civilian hardware, as shown in Figure 4.

```
[kind:1, domain:1, country:224, category:1,
subcategory:1, specific: 1, extra:0]
```

Figure 4: *Entity type settings*

By prior agreement these numbers in the Entity Type record describe a particular type of vehicle, person, or other entity. The field values are arbitrary, but need to be agreed upon by all applications cooperating in a NVE. A mapping between the values and the semantic meaning—the particular type of entity described by the numbers—is specified in the Enumerated and Bit Values (EBV) document from the Simulation Interoperability Standards Organization (SISO). Different mappings between the Entity Type record values and objects may be created other than the EBV document, but for the sake of consistency with existing simulations this is strongly discouraged.

DIS state update messages are sent every few seconds for every entity in the world and contain the entity's position, orientation, and speed. Also included are a unique identifier for that entity and an EntityType record. An application listening to DIS state update messages can, over a period of a few seconds, build a picture of the position, orientation, and type of all the entities in the scene.

The *DISEntityTypeMapping* node contains EntityType records, and in addition a link to a URL where an X3D model corresponding to the values of the Entity Type record can be found. Thus a simulation can, at runtime, find a 3D model that corresponds to a given Entity Type record and load it into the scene.

The *EspduTransform* node represents one entity in the world. If this X3D document creates and controls the entity the *EspduTransform* node instance will send periodic state updates to the network. If the X3D scene receives an update for an entity it has not encountered before, a new *EspduTransform* node will be created at runtime and added to the DOM tree. Subsequent updates will modify that same *EspduTransform* node—for example, position and orientation values change as it moves.

A simple scene fragment is shown below in Figure 5.

```
<DisEntityManager
websocketUrl="ws://10.1.1.100:80"
localCoordinateSystemOrigin="36.6 -121.9
1.0" applicationID="23">
  <DISEntityTypeMapping
  url="https://savage.nps.edu/tank.x3d"
  category="1" country="225" domain="1"
  kind="1" specific="3"
  subcategory="1" extra="0">
  </DISEntityTypeMapping>
</DisEntityManager>
<EspduTransform marking="X3D Entity"
entityID="42" siteID="2" entityKind="1"
entityDomain="1" entityCountry ="225"
entityCategory="1" entitySubCategory="1"
entitySpecific ="1" entityExtra="1"
networkMode="networkWriter" writeInterval =
"2000"> <!-- times in milliseconds -->
  <!-- no child shapes/models needed -->
</EspduTransform>
```

Figure 5. *Scene Fragment Using a Modified X3D DIS Profile*

The DISEntityManager element in Figure 5 declares that the server it will be sending and receiving DIS messages from is at the IP 10.1.1.100 on TCP port 80. A DISEntityTypeMapping is declared between a entity type record and a URL from which a model corresponding to that entity type can be retrieved.

This differs from the approved X3D DIS Profile in several ways. First of all, the *DisEntityManager* node specifies a "websocketUrl" attribute, with a value corresponding to the websocket server address. This attribute is not specified in the X3D DIS profile; the semantics of Websockets demanded that slightly different terms be used. Second, there is a "localCoordinateSystemOrigin" attribute. The attribute specifies the geo-referenced location of the X3D scene's coordinate system origin. DIS location coordinates from the network state updates are converted to local scene coordinates. Likewise a local entity informing the NVE of its position has its location converted to DIS coordinates before a state update is sent to the network. In the current example, an entity describing its position in earth-centered coordinates such as (-2678632, -4371130, 3781849) had its position converted to the scene's X3D flat, rectilinear coordinate system with an origin at latitude 36.6, longitude -121.9, and altitude 1 meter. Note that for simplicity of the initial implementation, the demonstration X3D scene assumes a flat earth. This is an approximation that is certainly not true over large distances.

The modifications the authors made to X3DOM do not yet completely implement the X3DOM DIS profile. The developmental source code is available online at http://github.com/mcgredonps/x3dom. Included in the source code is an implementation of DIS in JavaScript, utilities for converting between an earth-centric coordinate system and a local, scene-based, rectilinear coordinate system, and a sample HTML file. The Jetty websocket server is also available at http://sourceforge.net/open-dis.

3.3 Demonstration

The sample HTML file demonstrates some of the DIS functionality described above. In this naval scenario, civilian vessels and friendly warships are threatened by terrorist craft, which look like civilian vessels, as well as small pirate skiffs. For convenience, the training scenario uses the San Francisco bay area as its geographic location. When a user follows a link to the scene, the browser displays a 3D-navigable window into the running simulation in real-time (minus network latency). The user can pan and zoom into different parts of the map to watch the ships as they sail around the bay and interact with each other. The models are smoothly animated from point to point and oriented in the correct direction.

The HTML page is mostly comprised of an X3D scene with a DisEntityManager node. The scene includes no static entity nodes; all entities are created dynamically by the X3DOM module as the browser receives DIS traffic on the Websocket. All model URLs used in these DISEntityTypeMapping nodes are from the freely available SAVAGE X3D model repository found at https://savage.nps.edu/Savage. The only static object explicitly defined in the scene is the "terrain"—which in this simple example is just a satellite map image placed on a rectangle. The purely naval scenario avoids some of the difficulty of serving detailed ground elevation, but does not resolve the curvature of the earth problem discussed above. We use the simplest approach for this early prototype, clamping all models to sea level in local coordinates. If the DIS source is from a spheroid (that is, not flat) earth model, entities far from the center of the map appear to move a bit more slowly in the browser.

4 Performance

Hsiao Chen-Fu [2014] investigated performance of NVEs using the *Three.js* Javascript 3D framework, along with Websockets and WebRTC sockets. He used broadly similar technology, including the same implementation of DIS for JavaScript over websockets and WebRTC, but primarily used Three.js along with some X3D exemplars to implement the 3D portion. Benchmarking showed the ability to send over 5,000 DIS entity state updates per second.

DIS JavaScript decoding benchmarks are available at http://jsperf.com/JavaScript-dis-native-vs-json/2. The results showed significant differences depending on the browser implementation and version, a result that reflects varying performance of the different JavaScript engines. For example Firefox version 33 is several times faster than Firefox version 32 when performing the same message decoding task.

Ultimately performance is gated more by 3D scene size and complexity than by the ability to send and receive messages. Small to medium NVE scenes can be implemented in the web browser.

5 Conclusions and Future Work

The X3D DIS profile should be extended to handle different methods of message transport. At the time the standard was developed the only options for transport were conventional sockets accessed by plugins, a fact that resulted in simplifications in the specification. Recent technology changes have increased the number of transport options, and the DIS profile should reflect the new choices available to implementers.

More support is needed for different terrain projections and formats, perhaps including support for KML and other features. Curvature of the earth issues in scenes larger than a few kilometers can be avoided through proper implementation and integration of the X3D Geospatial Component.

There are many practical scalability issues to be investigated, including cloud deployments and using the server as a filter to forward only the state update messages that can be used by a particular client, instead of all messages.

We have shown the feasibility of enabling X3DOM to support NVEs, a challenge that has become achievable due to today's improved web browser 3D capabilities. NVEs come with several technical challenges, including heterogeneous models, asynchronous messaging with real-time demand, prohibitive security policies, and competing geographical representations, but these are not insurmountable.

The prime use case for our work is the addition of nodes into an NVE without need for any traditional client software deployment—not even browser plug-ins. With a websocket server in support of the NVE, a user needs only a typical modern web browser, a URL, and login credentials to join. For example, an unanticipated visitor to a military simulation-based exercise could be provided immediate access to view the action in real-time, even from halfway across the world.

With enough development of the downloadable (i.e. JavaScript) client software, the browser-based clients could begin to replace traditional thick client nodes altogether, not only viewing but injecting own entity actions. This would result in significant savings in software configuration management, and it suggests a powerful tradeoff between the thick client and thin client models: changes are deployed only to the servers, but all of the costly graphics processing happens on the client side.

An interesting view of this approach to NVEs is that it begins to separate modeling from simulation, with regard to both entity visualization and terrain representation. Since these models are loaded from URLs, and could be hosted anywhere, there is an opportunity to break the coupling of graphical representation, behavior logic, and geography—all three of which are bound together in a single application in today's systems. Perhaps this kind of modularity will someday lead us to employing NVEs as the norm, rather than NVEs that are painfully strapped together for single-use applications. Important future work continues.

References

BHER, J., JUNG, Y., DRVENSEK, T., ADERHOLD, A., "Dynamic and Interactive Aspects of X3DOM," *Proceedings of the 16th International Conference on 3D Web Technology*, ACM, 2011.

BRUTZMAN, D., AND DALY, L., *X3D: Extensible 3D Graphics for Web Authors*, Morgan Kaufmann Publishing, 2007. Course resources, slides, videos and example scenes available at http://www.x3dgraphics.com

BRUTZMAN, D., "X3D Graphics and Distributed Interactive Simulation (DIS) Networking," tutorial slides, 2014. Available at http://x3dgraphics.com/slidesets/X3dForAdvancedModeling/DistributedInteractiveSimulation.pdf

BRUTZMAN, D., X3D Basic Examples for Distributed Interactive Simulation (DIS), model archive and documentation, http://www.web3d.org/x3d/content/examples/Basic/DistributedInteractiveSimulation

BRUTZMAN, D., X3D Scene Authoring Hints: Coordinate Systems, authoring guidance. Available at http://www.web3d.org/x3d/content/examples/X3dSceneAuthoringHints.html

HSIAO, CHEN-FU, *Development of a web-based distributed interactive simulation (DIS) environment using JavaScript*, Masters Thesis, Naval Postgraduate School (NPS), Monterey California, September 2014. Available at https://calhoun.nps.edu/handle/10945/43928

ISO/IEC, 2013. ISO/IEC *19775-1:2013: Extensible 3D (X3D) Standard*. Geneva, Switzerland. Available at http://www.web3d.org/standards

JOHNSTON, A.B., AND BURNETT, D. C., *WebRTC: APIs and RTCWEB Protocols of the HTML5 Real-Time Web*, Third Edition, Digital Codex LLC, 2014.

KAPENAKIS, K., PANAGIOTAKIS, S., AND MALMOS, A.G., "HTML5 and WebSockets; Challenges in Networked 3D Collaboration," *Proceedings of the 17th Panhellenic Conference on Informatics*, pp. 33-38, 2013.

MCGREGOR, D., AND BRUTZMAN, D., Open-DIS Open-Source software implementation of the Distributed Interactive Simulation (IEEE-1278) standard in C++, C-Sharp, Objective-C, Java, Javascript and XML. Available at https://sourceforge.net/projects/open-dis

MCGREGOR, D., BRUTZMAN, D., AND JOHN GRANT, S., "Open-DIS: An Open Source Implementation of the DIS Protocol for C++ and Java," *Simulation Interoperability Workshop (SIW)* of *Simulation Interoperability Standards Organization (SISO),* paper 08F-SIW-051, Orlando Florida, 15-19 September 2008.

MCGREGOR, D., 2012, "WebSockets for Networked Virtual Environments (NVEs)," presentation, *Fall 2012 Simulation Interoperability Workshop (SIW) of Simulation Interoperability Standards Organization (SISO)*. Available at http://hdl.handle.net/10945/44385

MCGREGOR, D., BRUTZMAN, D., ARNOLD, A., BLAIS, C. L., FALASH, M. AND POLLACK, E., "DIS-XML: Moving DIS to Open Data Exchange Standards," *Proceedings of the Fall Simulation Interoperability Workshop (SIW)*, paper 06S-SIW-132, Orlando Florida, 2006. Available at http://calhoun.nps.edu/bitstream/handle/10945/31188/06S-SIW-132-PDF.pdf

RAUCH, T., *Savage Modeling and Analysis Language (SMAL) metadata for tactical simulations and X3D visualizations*, Masters Thesis, Naval Postgraduate School (NPS), Monterey California, March 2006. Available at http://calhoun.nps.edu/handle/10945/2971

RAUCH, T., BLAIS, C., AND BRUTZMAN, D., Savage Modeling and Analysis Language (SMAL) Resources, online support, available at https://savage.nps.edu/Savage/Tools/SMAL/SMAL.html

SIMULATION INTEROPERABILITY STANDARDS ORGANIZATION (SISO), *Reference for Enumerations for Simulation Interoperability*, SISO-REF-010-2015 Version 21, 17 March 2015. Also referred to as Enumeration Byte Values (EBV) document. Available at http://www.sisostds.org/ProductsPublications/ReferenceDocuments.aspx

SWAN, P., "WebLVC—An Emerging Standard and New Technology for Live, Virtual, and Constructive Simulation on the Web," *Proceedings of the 2014 Winter Simulation Conference, IEEE*, pp. 4217-4218, 2014.

WANG, V., SALIM, F., AND MOSKOVITS, P., *The Definitive Guide to HTML5 WebSockets*, Apress, 2013.

WIKIPEDIA, "Automatic Identification System (AIS)," reference article, available at https://en.wikipedia.org/wiki/Automatic_Identification_System