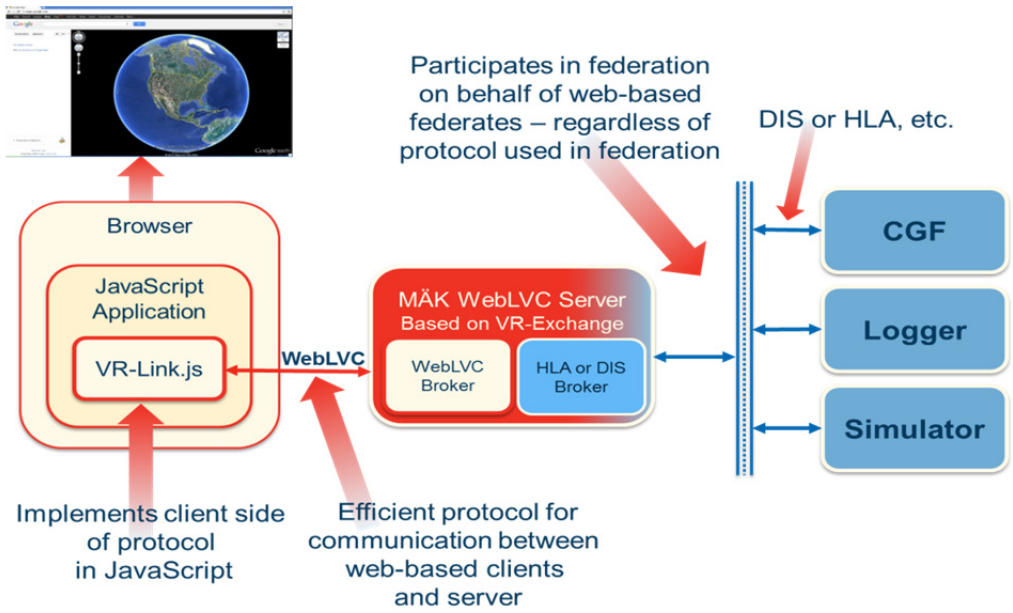
MÄK WebLVC Server 1.0 Release Notes

Introduction

MÄK WebLVC Server is the linchpin that enables web-based simulation to interoperate with traditional distributed simulation federations. Web-based client applications connect to MÄK WebLVC Server using WebSockets, and exchange information with the server using the WebLVC Protocol. MÄK WebLVC Server translates simulation data between the WebLVC Protocol and traditional M&S protocols such as DIS and HLA.

MÄK WebLVC Server is based on MÄK's VR-Exchange interoperability platform, which supports many-to-many translation between various interoperability protocols through the use of "Brokers" for each protocol. When you install MAK WebLVC Server, you are essentially installing a copy of VR-Exchange pre-configured with a Broker for DIS and a Broker for the WebLVC protocol.



In the diagram above, the federates on the right may be using DIS, HLA 1.3, HLA 1516, HLA Evolved, TENA, or any other protocol for which a VR-Exchange Broker exists. See documentation for more information on the VR-Exchange architecture, and on the configuration of various Brokers.

Installation

To install MÄK WebLVC Server, simply run the MÄK WebLVC Server installer, and follow the prompts.

Included in the installation

- The MÄK WebLVC Server software itself – pre-configured with Brokers for DIS and WebLVC
- A set of client-side libraries called vrlink.js to help you implement the client-side of the WebLVC protocol in your web applications
- A set of JavaScript example applications, with source code, to get you started in building your own web-based client applications
- A draft of the WebLVC Specification (0.2).

Getting Started

To try out MÄK WebLVC Server:

1. Launch MÄK WebLVC Server from the Start Menu: Programs -> MÄK Technologies -> MÄK WebLVC Server x.y.z -> WebLVC Server. The standard VR-Exchange GUI should come up, allowing you to see that there are brokers running for the DIS and WebLVC protocols.
2. Start the example OpenLayers PVD Example client from the Start Menu: Programs -> MÄK Technologies -> MÄK WebLVC Server x.y.z -> OpenLayers PVD Example. This should launch your default web browser, and automatically load a pvd.html file that contains the sample application.
3. At this point, you should see a simple Web-based map application running in your browser. But you will not see any entities; because you aren't running any DIS federates yet.
4. Launch any DIS application that publishes entity data. For example, you can run the VR-Link F-18 example application, VR-Forces, or a MAK Data Logger configured to play back a sample Logger file.
5. You should now see entities listed in the MÄK WebLVC Server GUI to indicate that there is data coming into the Server from the HLA or DIS side, and you should see entity icons being displayed in the web browser overlaid on the 2D map.

Important Notes:

- In order for an HLA Broker to connect to an HLA federation, you will need to have an HLA Run-time Infrastructure (RTI) installed. You can download the MÄK RTI for free here: http://www.mak.com/products/link-simulation-interoperability/hla-rti-run-time-infrastructure.html. MÄK RTI version 4.1.1 or later is recommended.
- If you do not have an HLA RTI, MÄK WebLVC Server will report that it cannot connect to an HLA federation, but you will still be able to interoperate with DIS federates.

- MÄK WebLVC Server communicates with web-based clients using the WebLVC protocol, using WebSockets. Therefore you must use a browser that supports WebSockets. We have tested our sample web applications with recent versions of Firefox, Chrome, and Safari (including on iOS), which all support WebSockets. Internet Explorer version 9 and earlier do not support WebSocket. WebSockets are supported in the Internet Explorer 10 release.
- MÄK WebLVC Server fully supports deployments on closed networks behind firewalls, with no internet access. Browser-based federates will be able to communicate with traditional DIS/HLA federates running on the same network, through MÄK WebLVC Server. However, our sample OpenLayers PVD application requires an internet connection in order to access the terrain and imagery data that it displays on the map. If you are trying out MÄK WebLVC Server for the first time on a machine without access to the internet, we recommend that you run the Message Inspector sample web application, so that you can at least see that data is flowing from DIS/HLA federates to your browser through MÄK WebLVC Server. Recognize that it is entirely possible to build web-based PVDs, Stealth viewers, maps applications, etc. without requiring internet access. But you would need a locally deployed terrain and imagery server such as ArcGIS Server, Google Enterprise Server, or MÄK's VR-TheWorld Server in order to serve the data.
- By default MÄK WebLVC Server launches the WebLVC Broker (configured to use port 9101) and the DIS broker (configured to use port 3000). Additional brokers such as the HLA Evolved Broker (configuration dictated by the RTI) can be added at runtime and configured to launch automatically.

Configuring the Brokers

- The MÄK WebLVC Server documentation (accessible in the .\doc directory or from the Start menu) covers how to configure the DIS and HLA brokers for your environment.
- By default, the web socket server comes up on port 9101. To change to a different port, you can right-click on the WebLVC Broker from the main VR-Exchange GUI, and choose Configuration. When the WebLVC Broker configuration dialog comes up, change the WebSocket Port parameter.
    - When you click apply, the WebLVCBroker.bcf broker configuration file is written to the bin directory.

Relationship between MÄK WebLVC Server and Your Web Server

In the Getting Started example above, the browser loaded our sample web application by loading a local file. In a real deployment, you will likely want your web applications to be served by a web server such as Apache or IIS – especially if your web applications exist within the context of a web site.

To try this out, copy the html folder from the MÄK WebLVC Server installation directory to the folder in which your web server typically looks for HTML files (e.g. C:\inetpub\html for IIS, and C:\Program Files\Apache\html for Apache). Our html folder contains the MÄK WebLVC Server sample applications, along with the vrlink.js libraries that the applications use to implement the WebLVC protocol. You should now be able to bring up a web browser, and browse to http://yourwebserver/olPvd/pvd.html, where yourwebserver is the URL of your web server. (olPVD/pvd.html is the location of our Open Layers-based sample 2D viewer application. Other sample applications can be found under the html folder as well).

Note that the approach above assumes that you will be running MÄK WebLVC Server on the same machine as your web server, because our sample applications are hard-coded to initiate WebLVC (WebSocket) connections to the same machine that served the HTML web page containing the application. If you would like to run MÄK WebLVC Server on a separate machine (not your web server), please contact support@mak.com, and we can walk you through the configuration process.

Note also, that our sample apps are hard-coded to initiate their WebLVC (WebSocket) connections on port 9101 – since that is the port used by default in MÄK WebLVC Server. If you want to use a different port, you will need to change the port used by the server as described above, and then pass that port number t our sample applications via URL parameters. For example, enter the following URL in your browser: http://yourwebserver/olPvd/pvd.html?wsPort=9202.

For those who working in network environments that allow traffic only on port 80, we do support web proxying, which allows clients to use port 80 for both their static HTTP traffic and the WebLVC WebSocket traffic. If you are interested in such a configuration, please contact support@mak.com, and we can walk through the configuration process.

Extending WebLVC with Automatic FOM Translation

The MÄK WebLVC Server currently has built-in translators to convert the handful of message types contained in the current WebLVC Protocol Specification between WebLVC and either DIS or HLA (e.g. Entity, Aggregate, Fire, and Detonate). But MÄK WebLVC Server can also automatically generate extensions to the WebLVC protocol based on the information contained in an HLA FOM, and automatically perform translation between HLA and those WebLVC extensions.

In other words, MÄK WebLVC Server is self-extending: MÄK WebLVC Server reads the names and data types of classes, attributes, parameters, structures and enumerations directly from an HLA FOM, and maps them to corresponding JSON elements. (No such similar capability exists for WebLVC <-> DIS translation, because there is no DIS equivalent to a FOM).

The MÄK WebLVC Server's Automatic FOM Translation capability relies on VR-Exchange's HLA Broker's "generic" or pass-through translation capability. When an HLA Broker is configured to perform generic translation, attributes and parameters of various FOM classes will pass through the VR-Exchange portal in "raw" form, without the HLA Broker converting the data into any kind of VR-Exchange protocol-independent representation. Once the raw values get across the portal, the WebLVC Broker can import a FOM document to understand how to interpret the data, and can convert the binary data to the WebLVC JSON representation.

To use this Automatic FOM Translation capability, you must do the following:

1) Configure your HLA Broker to choose which object and interaction classes should be sent across the VR-Exchange portal using generic translation. For a detailed explanation of setting up generic translation, see the VR-Exchange Users Guide.

2) Configure your WebLVC Broker with the name of the file that defines your FOM data. This is either an HLA-1.3-compliant OMT file or the xml-based file from HLA 1516 or HLA Evolved. Set the value of the "omtFiles" variable in your WebLVCBroker.bcf configuration file to point to the desired FOM document. The default FOM file is "VR_Link.omt". You can define more than one FOM file when using HLA Evolved modules, by separating the files using a semi-colon. For example: vrlink.xml;vrlinkextensions.xml;vrforcesextensions.xml.

Mapping HLA Data Representations to WebLVC Data Representations

The Automatic FOM Translation capability maps HLA types to JSON/WebLVC objects according to the following scheme:

The following HLA types translate to a JSON string (e.g. "HelloWorld")
- string
- HLAcString
- HLAstring
- HLAcount
- handle
- timeType

The following HLA types translate to a JSON number string (e.g. "1234")
- HLAbyte
- HLAfederateHandle
- HLAinteger16BE
- HLAinteger16LE
- HLAinteger32BE
- HLAinteger32LE
- HLAinteger64BE
- HLAinteger64LE
- HLAmsec
- HLAoctet
- HLAoctetPairBE
- HLAoctetPairLE
- HLAseconds)
- HLAunicodeChar
- UnsignedInteger16BE
- UnsignedInteger32BE
- UnsignedInteger64BE
- UnsignedShort
- VerfierIntegerTime

int
long
long long
octet
short short
unsigned int
unsigned long
unsigned long long
unsigned short

The following HLA types translate to a JSON floating point string (e.g. "1234.5")
HLAfloat32BE
HLAfloat32LE
HLAfloat64BE
HLAfloat64LE
double
float

The following HLA types translate to a single JSON character (e.g. "a")
HLAASCIIchar
char

The HLA Boolean type translates to a JSON Boolean string (e.g. “False”)

HLA enumerations map to a JSON string name:

```
var BoolEnum = "HLAFalse"
```

HLA arrays translate to arrays of corresponding JSON types,

```
var arrayOfLongs = [ "1","2","3" ]
```

Composite types are composed from primitive JSON types and other composite types, for example:

```
var messageLocation = {
    X : "-2696500",
    Y : "-4430500",
    Z : "3701900"
};

var messageToPublish = {
   MessageKind: 1,
   ObjectType:
       "ObjectRoot.BaseEntity.PhysicalEntity.Platform.Aircraft",
   ObjectName: "From Web",
   WorldLocation: messageLocation
};
```

Automatic FOM Translation Example: Web Bounce

The MÄK WebLVC Server includes an example named Web Bounce that uses Automatic FOM Translation to translate a simple FOM. This example is based on the HLA Bounce example that comes with the MAK RTI (see section B.5 "HLA Bounce" in the MAK RTI Reference Manual for details on this example).

The Web Bounce example demonstrates how to configure a simple FOM to be generically translated to WebLVC. The HLA Bounce example application publishes one or more bouncing ball objects in an HLA federation. The Web Bounce web client app listens for and displays these published objects in a web browser.

Important Note: Web Bounce client requires MAK RTI 4.1.1 (This restriction only applies to this specific example. If you develop your own web clients to work with an Automatic FOM Translation, then you can use a different version of the MAK RTI.)

To run the Web Bounce example:

1. Start the HLA Bounce example application (HLA Evolved version). This is described in detail in section B.5.1 "Running HLA Bounce" in the RTI Reference Guide. Configure the HLA Bounce application to publish some number of bouncing balls.
2. Start MÄK WebLVC Server.
3. In the Connections pane in MÄK WebLVC Server, right-click and choose "Add New Connection". In the "Add New Connection" popup dialog, set the following:
    a. Set the Connection Name to "HLA Bounce".
    b. Set the Broker Executable to hla1516eBroker.
    c. Click on the Advanced Options button and specify the Configuration File as "HLA Bounce.bcf".

    Click "Add and Configure…" to complete the process. A new connection named "HLA Bounce" appears in the Connections pane.
4. In the Connections pane in MÄK WebLVC Server, right-click on the HLA Bounce connection and select Enable Connection.
5. In the Connections pane in MÄK WebLVC Server, right-click on the WebLVC connection and select Edit Connection Information. In the "WebLVC" popup dialog,
    a. Click on the Configure Icon.
    b. Expand WebLVC Settings attribute.
    c. Set the OMT Files attribute to MakHlaBounce1516e.xml.
    d. Click on the OK button.

    A popup dialog prompting the user to restart the broker is shown. Click on the Yes button to restart the broker with the new settings. Right-click to disable and then re-enable the broker.
6. Start the Web Bounce federate by choosing MAK Technologies→MAK WebLVC Server→Web Bounce. A web browser window will open with the Web Bounce client app.

In the Web Bounce federate window, you should see a mirror image of the bouncing balls being published by the HLA Bounce example federate.

You can also examine the JSON-based WebLVC extensions that are auto-generated from the MakHLABounce1516e FOM by opening the Message Inspector client app, available from the start menu by choosing MAK Technologies→MAK WebLVC Server→Message Inspector.

As of RTI version 4.1.1, you must run the HLA Evolved version of HLA Bounce example. There are known bugs in the HLA1.3 and HLA1516 versions of this example. These bugs will be fixed in the next RTI release. These bugs only affect the HLA Bounce examples for those protocols. The MÄK WebLVC Server Automatic FOM Translation mechanism itself is not affected.

Known Problems

1. In rare instances the WebLVC Broker may not shut down correctly and continues to run after the main executable closes. In this case the broker must be killed from the Task Manager.

Fixed

1. The server may not properly detect when web clients close if there is no other traffic going through the server.
2. Objects published by the HLA Bounce web client do not appear correctly in non-web HLA Bounce clients (they appear with the wrong color). This was due to a problem in mapping enumerated values in the FOM translation.