# Visualization Service Bus

**Abstract**— In this research, we are applying modern Service-Oriented Architecture (SOA) technologies to make complex visualizations realizable without intensive graphics programming; in fact, without writing code at all. A Visualization Service Bus is the integration of two powerful Extensible Markup Language (XML) technologies, Extensible 3D (X3D) and an Enterprise Service Bus (ESB), which allows users to integrate data and develop user defined visualizations together. Analysts use graphical interfaces to construct visual elements, assemble a dynamic scene, connect to and transform data from a variety of sources, and map scientific data to the scene. The development and implementation of this visualization architecture allows non-programmers to develop their own visualization applications for their specific needs.

**Index Terms**—X3D, XML, ESB, BPEL, XSLT, XSD, SOA, JBI, JMS, and XPATH.

## 1 Introduction

In the aviation research community, scientists analyze, mine, verify, manipulate and visualize data in many ways to understand complex problems and to convey research results. Analysts today use a wide variety of visualization tools to demonstrate their research findings. Typically, these tools are designed to read in a specific type of data in a proprietary format and transport protocol. These visualizations tend to be inherently rigid, and large software efforts are needed to make changes or implement new features. However, recent advances in software technology, architecture, and methods provide an opportunity to improve this process.

A common design pattern is unfolding with the rise of Service-Oriented Architecture (SOA). The SOA design pattern is the decoupling of application-specific functionality into standardized functional nodes described in XML. A generalized engine that knows how to turn those nodes into functional logic is a recurring theme in SOA. This design pattern allows for greater flexibility between application user and software developer. In the SOA paradigm, a set of standard services are defined rather than specific end user requirements. Thus, developers focus on implementing these standard services, which are then wired together to meet a business need by the end user. The focus of this research is to use existing standards to provide a service framework such that end users can query, create and translate data into user-defined visualizations.

### 1.1 Enterprise Service Bus

An Enterprise Service Bus is a middleware platform that implements a set of standardized interfaces for portability, security, connectivity, transformation, and communication, providing the capability to integrate systems without the need to write code. The VSB described in this paper runs within the Open Enterprise Service Bus (Open ESB)[1]. The VSB is more formally an X3D Service Engine running within a Java Business Integration (JBI) container [9] Open ESB is a JBI container created and maintained by Sun Microsystems and is an open source ESB that leverages standards built on the JBI model.

#### 1.1.1 Java Business Integration

The JBI model is a standard created under the Java Community Process (JCP) to implement SOA. This standard defines components that consume and provide services as binding components and service engines. JBI components, once installed in the JBI environment, interact with each other using message exchange documents published by the JBI component providing the services.

These documents fully describe the message exchange using Web Service Definition Language (WSDL) [2]. WSDL is an XML-based language for describing Web services and how to access them. The JBI specification defines two types of components: binding components and service engines.

### 1.2 Binding Components

Binding components communicate with external services. These components access services over a known protocol and data format. Examples include HTTP, SOAP, JMS, TCP, FTP, and SMTP. The binding component is responsible for communicating with external services over a specified protocol and converting that data into the data format that is used in the Normalized Message Router (NMR). Since each component communicates with the NMR, the components are decoupled from other components communication protocol.

#### 1.1.1 Service Engines

Service Engines are components within the JBI runtime environment which provide data transformation, business logic, processing, and routing services. Binding components and service engines provide the generalized services that end users assemble to implement a business process as a service assembly.

#### 1.1.2 Service Assemblies

Service assemblies contain the configuration information that tells the JBI container the names of the components that each service unit will be deployed. A service unit is the configuration information that is to be deployed to a component.

### 1.3 Extensible 3D (X3D)

X3D is an open standard format that describes 3D scenes and objects in XML, it is based on the Virtual Reality Modelling Language (VRML). X3D is a scene graph description of a scene, which is a directed and acyclic tree structure. The specification is defined in an Extensible Schema Definition (XSD) file, which can be used by any XML binding software to convert to programming code. This flexibility allows software developers to program in their favorite programming language without portability concerns.

X3D supports 3D graphics, 2D graphics, CAD data, animation, audio and video, user interaction, navigation, user-defined objects, scripting, and networking. Dynamics can be added through ROUTE nodes statically or by writing software that uses the Scene Access Interface (SAI) to access the scene at runtime. X3D provides nodes to draw and perform tasks to do virtually anything desired in a 3D world. X3D files are read in by a generalized browser which displays the description of the scene.

The rest of this paper will show that by creating an X3D Service Engine in the Open ESB implementation of the JBI standard, users are then able to construct business logic, connect data sources, and construct visualizations without the need to write code thus providing a codeless visualization programming framework.

## 2 VSB DESIGN

In the Open ESB project all binding components and service engines have two module projects which make up the design of a component. Open ESB is implemented as a Netbeans plug-in. A Netbeans module project is a plug-in project for the Netbeans Integrated Development Environment (IDE). One module project provides an editor interface for the user to construct the XML and configurations required to tell the service engine how it needs to behave. The other project is to build the generalized service engine.

The VSB design follows this pattern. It is made up of two components. The first is an X3DEditor which allows users to define their visualization and how the incoming data will map to the scene. The second is an X3D Service Engine, which reads the XML data produced by the X3DEditor, constructs the dynamic scene, and updates it as incoming messages are received from the NMR.

### 2.1 X3D Editor

The X3DEditor was built using the Netbeans API [10] provided by Sun Microsystems. The editor has four separate views: Tree Graph, Data Map, 3D, and XML. Together they allow the designer to construct how the visualization will look and behave in response to incoming data. .

#### 2.1.1 Tree Graph View

The user will first construct the static part of the visualization using the tree view. A view of the X3D file in a graphical tree is provided so users can drag and drop X3D nodes from a palette into the graphical tree. Users of this tree view need to understand how X3D works, but they do not need to understand XML syntax. The majority of users in our community will be merging together X3D files that represent airports, airspace models, airplanes, and other National Airspace System (NAS) models. They will not be trying to construct complex geometric models using this editor. Tools such as Maya [3], Flux [4] or Blender [5] can be used to construct complex geometric object models. Instead this editor is used to create a simple tree view of many complex models enabling users to construct and change complex scenes quickly. In contrast, for a traditional 3D application a software developer will need to spend significant effort making software changes to achieve the desired result.

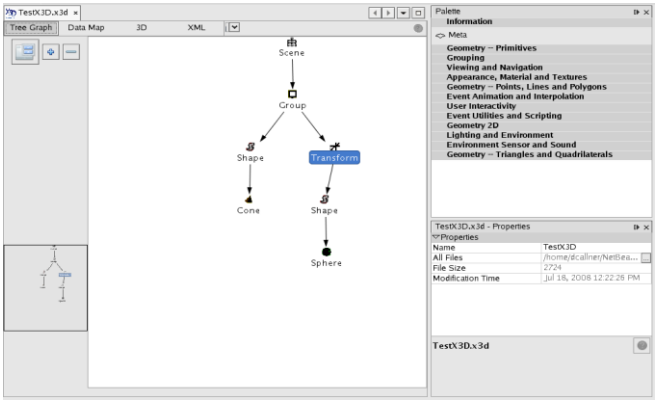


Fig. 1. Tree Graph View X3D Editor

#### 2.1.2 Tree Graph Canvas

The Tree Graph Canvas is the graphical area onto which users can drop X3D nodes from the palette. Users can zoom in and out to change the focus and viewing area of large trees. To edit the static data for each node the user can right click on the node and edit node parameters. Changes to the tree will only be written to the XML file once the file has been saved.

#### 2.1.3 Tree Graph Palette

The Palette contains all the nodes contained within the X3D specification and are outlined based on a similar editing tool, X3D-Edit [6], for uniformity. In future releases this palette will be linked with a database enabling users to drag and drop predefined X3D files that contain objects like airplanes, airports, etc.

#### 2.1.4 Tree Graph Toolbar

The tree representation of the scene can get quite large. A magnifying glass is provided for users to scroll across the tree with the mouse. The level of magnification can be altered by pressing the plus and minus buttons in the top left corner. A window pane navigation window is provided in the bottom left corner so users can change the view of the tree without using scroll bars.

#### 2.1.5 Data Map View

Aviation researchers need to visualize a wide variety of data. Example topics include noise modelling, airspace redesign, and traffic flow management algorithms. Visualization helps researchers analyze this data and verify it against known models. The X3DEditor allows an analyst to quickly assemble a scene, and then map scientific data to it using the Data Map View on the X3DEditor. Once the user has a static tree view of the scene they can define how data is going to be mapped into this scene. The data map view enables users to dynamically add X3D nodes to a scene or to update existing nodes. The editor can then automatically generate an XSLT file with embedded JavaScript to define the data mapping.

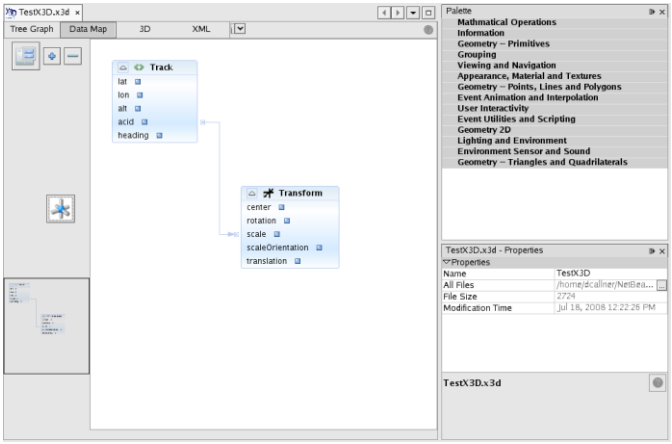


Fig. 2. Data Map View X3D Editor

#### 2.1.6 Data Map Canvas

The Canvas provides a graphical interface for users to construct mappings from incoming requests to X3D nodes. These incoming requests come from the NMR. A WSDL file is needed to construct a WSDL file to describe the data formatting and what operations are available to either update or add X3D nodes into the scene. To populate the canvas with nodes, users can drag and drop a WSDL file from the Projects Tab, drag and drop X3D and Function Nodes from the Palette, or select an X3D node from the Tree View. Each attribute of a node is shown in the node box and an arrow can be drawn from each field to represent the mapping. Function nodes can sit between the mappings to indicate additional logic to be applied before the mapping takes place.

The user first constructs a WSDL file using the Netbeans IDE. Netbeans provides an easy setup wizard for constructing a WSDL file without directly editing XML. Once the user constructs the incoming messages for this instance of an X3D Service Engine, then the user constructs an XML Schema Definition (XSD) which describes the structure of the XML data received in these incoming messages. Netbeans also provides a wizard and GUI to construct an XSD file without the need to directly write XML.

The user can drag the newly constructed WSDL onto the data map canvas, which will automatically create data nodes to represent the incoming data.

Often when a user wants to map data into a visualization scene the user will want to apply a mathematical operation to the incoming data. This view provides a palette of nodes that represent mathematical, string manipulation, and XPATH expressions on the incoming data. These nodes can be drag-and-dropped from the palette and sit between the incoming data nodes and visualization X3D nodes. Each of these Function Nodes auto-generates JavaScript to be embedded into the auto-generated XSLT file.

When a user wants to add 3D elements to a scene based on incoming messages then the user can drag and drop X3D nodes from the palette onto the canvas. On the tree view users can select which tree branch node this data will be added to. If no node is selected on the tree view then the first Group node will be the parent for the incoming data. Each X3D nodes' static data can be edited by right clicking on the node. For example, if a WSDL node containing a message that represents an alert in a geographic airspace region is added to the canvas, then the user would be able to draw an arrow from the WDSL node to the X3D sphere node. This associates each incoming message to map to adding a sphere node to the scene whenever this message arrives. Editing the X3D sphere node's static properties by setting the radius indicates to add a sphere of specified size to the scene every time an incoming message arrives.

### 2.1.7 Data Map Toolbar

The Canvas provides a graphical interface for users to construct mappings from incoming requests to X3D nodes. These incoming requests come from the NMR. A WSDL file is needed to describe the data formatting and what operations are available to either update or add X3D nodes into the scene. To populate the canvas with nodes, users can drag and drop a WSDL file from the Projects Tab, drag and drop X3D and Function Nodes from the Palette, or select an X3D node from the Tree View. Each attribute of a node is shown in the node box and an arrow can be drawn from each field to represent the mapping.

### 2.1.8 3D View

This view provides VSB users with a way to visualize the results of their mappings and tree editing. The 3D view is a visualization of the X3D file and shows the scene changing dynamically based on incoming requests. This view allows for network connectivity to deployed X3D Service Engines, and provides a basic browser for navigation and user input. During the development of the X3DEditor it was discovered that X3D lacks networking standards. We plan to work with the Web3d Consortium to standardize the networking for X3D and to provide an implementation in the Xj3D open source project. [7]

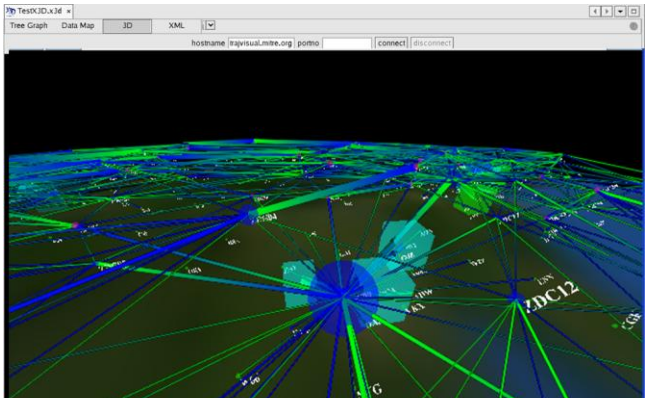


Fig. 3. 3D View X3D Editor

### 2.1.9 3D Browser

The 3D browser is an embedded Xj3D [7] browser which displays the X3D-defined scene and provides navigation controls. The Xj3D browser is an open source toolkit that can be used as a standalone browser or be embedded into a Java application using its Application Programming Interface (API). This view displays the X3D file generated using the tree view and can also register with a deployment of the X3D Service Engine to take incoming requests and turn them into SAI function calls to update the scene dynamically.

### 2.1.10 3D Networking

The connect button on the 3D view allows the user to connect to a deployed X3D Service Engine in a service assembly. When messages are translated in the deployed application server then those messages will be sent to registered browsers. The incoming data is in X3D and this browser will convert those incoming messages into SAI requests to either update the scene or add new data to the scene.

### 2.1.11 3D Navigation

The Xj3D browser provides a rich set of navigation controls. Users can fly, pan, rotate, walk, go home, and zoom into a selected object. Users can navigate to any point in their 3D world with these controls.

## 2.2 X3D Service Engine

The X3D Service Engine implements the appropriate JBI interfaces to perform as a service engine. When a user deploys a X3D Service Engine configuration as part of a service assembly, an XSLT file is deployed and read by the X3D Service Engine. XSLT files are created from the Data Map view in the X3DEditor and can contain embedded JavaScript to define complex mathematical operations on XML nodes. This service engine also creates a TCP socket connection that the Netbeans IDE connects to when running a service assembly. When the X3D Service Engine receives data the engine will translate the incoming XML data based on the XSLT file and apply the JavaScript to produce a final set of X3D XML nodes for display. This output is sent over TCP to the registered browsers. In future releases of X3D Service Engine a web service will provide a HTML page from which users can launch a 3D browser. This browser is unique because it can retrieve the X3D messages and convert those messages into SAI calls to either update or add X3D nodes into the existing scene.

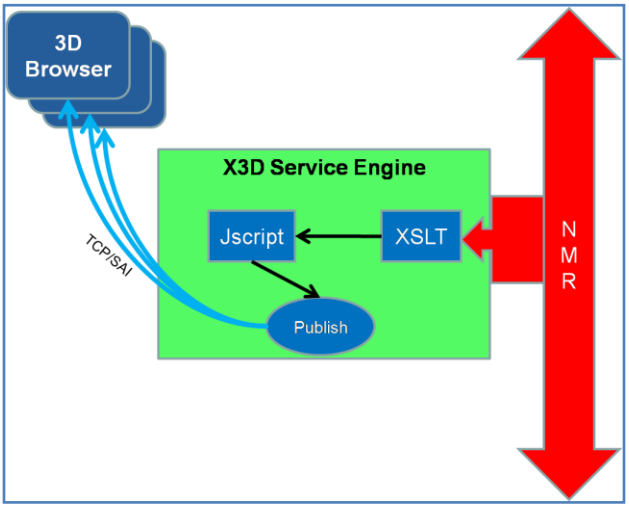


Fig. 4. X3D Service Engine Concept Design

### 2.2.1 Data Translation

The X3D Service Engine's primary role is to map the incoming messages into the visualization scene. Our original idea was to use a XSLT service engine, but it was soon realized that XSLT does not support complex mathematical operations well or at all. The idea for a new mapping language was considered, but in the end it was decided to build on the existing XSLT technology. Embedding JavaScript into XSL files is not a new idea, but most implementations do not support this level of the standard (XSLT 1.1), and the future of embedded scripts is not clear since it may compromise the portability of XML to embed a specific language. However, no solid alternatives were found.

The XSLT processor used in the X3D Service Engine supports XSLT 1.0, so the processing must be done in two steps. First, the incoming XML messages are translated into new messages based on the XSLT mapping file provided in the deployment. Next, the XSLT file is scanned for embedded JavaScript, and for each node with embedded JavaScript, the script is run to produce new nodes that are added back to the XML using a DOM parser. The result is then sent to registered browsers and converted into SAI calls to update the scene.

### 2.2.2 Networking

Networking seems to be the least supported feature of X3D in the browser community. The Web3D Consortium has made a proposal for direct networking for which all of the configurations for network communication would be edited in an X3D node rather than embedded scripts. Browsers may support this functionality in the future. For the time being, a TCP network connection is made between browsers and the deployed X3D Service Engine. The service engine translates the incoming messages into X3D nodes and publishes the updates to the registered browsers. The registered browsers convert the X3D requests into SAI function calls to update the scene. CAASD plans to help define the standards with the Web3D Consortium and the networking functionality in the Xj3D browser.

## 3 Examples of VSB Implementations

### 3.1 Escorted Aircraft Landing

To show the ease of assembling a complex scene, a collection of 3D models were downloaded from the NPS Scenario Authoring and Visualization for Advanced Graphical Environments (SAVAGE) [8]. SAVAGE is a library of open-source X3D models used for defense simulation. For this scene construction example, the X3D Service Engine was not used; rather data points (e.g., aircraft position histories) were built by hand using the X3DEditor.

### 3.1.1 Add and Position 3D Models

The first step in constructing the scene was to add the 3D models. Each model was referenced by an Inline node, and usually placed underneath a Transform node so that the position and orientation of the model can be set and controlled. Initially, the Transform node contains default values. The user can use the 3D view as a reference while editing the Transform fields to correctly position the model. Alternative techniques for adding models are currently being explored.

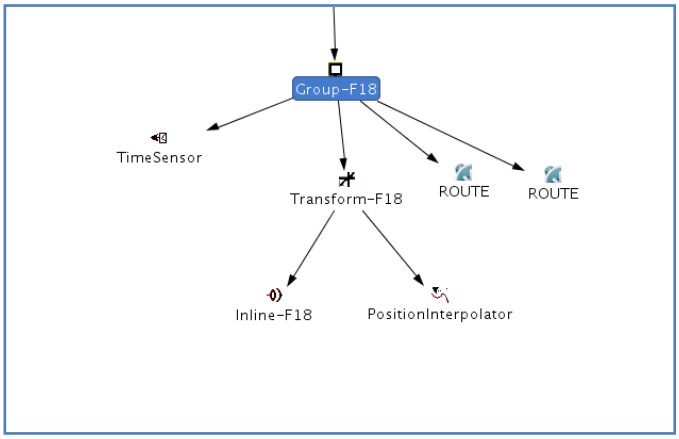


Fig. 5. Scene in Tree View

Figure 5 shows the result of using the X3DEditor to put together a scene of a 747 aircraft being escorted by two fighter jets. This was done by creating a few Inline nodes that point to the downloaded 3D models, and creating ROUTE nodes to move them in the 3D world. In total, 30 Inline nodes were required to assemble the visualization depicted in Figure 6.

### 3.1.2 Animate 3D Models

X3D's specifies animation simply by allowing most fields to be modified at runtime. Changes to field values are known as events. Three types of nodes were needed to animate this scene:

- ROUTE nodes connect the output field of one node to the input field of another node.
- The PositionInterpolator node linearly interpolates a 3-vector value output based on the key and a keyValue fields.
- The TimeSensor node is the trigger to an event.

An example of X3D XML using these nodes is shown below.

```
    <Transform DEF="f18" scale="39.3113 39.3113 39.3113" translation="-
78040.8 11000 50000" rotation="0 1 0 .65">
        <Inline url="'file:F18.x3d'"/>
    </Transform>
    <PositionInterpolator keyValue="-78940.8 11626.5 50000 -40000  1000
14000 -15000 10000 2000" key="0.0 0.5 1.0" DEF="PIP1"/>
    <TimeSensor cycleInterval="90.0" loop="true" DEF="PTS1"/>
    <ROUTE toField="set_fraction" toNode="PIP1"
fromField="fraction_changed" fromNode="PTS1"/>
      <ROUTE toField="set_translation" toNode="f18"
fromField="value_changed" fromNode="PIP1"/>
```

As shown above, a Transform, Inline, PositionInterpolator, TimeSensor and two ROUTE nodes are needed to fly an F18 in the 3D scene. These nodes were added using the X3DEditor tree view. The Transform node was used to place the F18 at a starting position

in the scene. The PositionInterpolator gives the flight path of the F18. Only three data points were picked for the flight: starting position, mid flight position and ending position. The TimeSensor is used to trigger the animation, and the ROUTE nodes indicate how the fields are mapped. The aircraft position was smoothly interpolated between positions by the X3D browser as time passed in the animation. Note that users of the VSB will rarely animate scenes in this way. Instead, they will use the data map view on the X3DEditor to map data from another source into the static visualization picture. This example is shown here to demonstrate how easily one can assemble a complex 3D scene with the X3DEditor. The next example will show another alternative, where a running simulation can be used to animate a scene using the VSB.

### 3.1.3 Final 3D Scene

This dynamic scene (Figure 6 shows a still view) was constructed in only a few hours. We plan to assemble a database of 3D NAS models so that our aviation researchers can easily construct scenes like the one below, or scenes containing abstract representations of air traffic management structures (e.g., sector boundaries). This effort will be expedited with object model converter tools. These tools provide the ability to convert from one object model into the X3D format.

Fig. 6. Escorted Aircraft Landing in Hawaii

## 3.2 Traffic Flow Network

This example is a two step process. The first step takes geometric airspace data and constructs a static X3D file to display the traffic flow network. The second step runs another assembly which maps incoming data from a live simulation into the generated X3D scene to add the dynamic elements to the scene.

### 3.2.1 Static Flow Network

The static flow network is the X3D scene that contains spheres, boxes, and links projected onto an earth object. These objects are static because no animation nodes are used to animate these 3D objects. The dynamics of the scene will be added in the next section. The spheres represent sector center points. Sectors are volumes of airspace that an air traffic controller team is responsible for. Each box represents an airport. A flight will take off from one airport and travel through many sectors and arrive at another airport. The number and direction of flights passing between airports and sectors are represented by links. These links are scaled by size and color gradient to represent the number and direction of flights that travel each link within a designated time interval. The simulation is given parameters for a particular airport and time range which, when run, generates a large XML document file containing sector center points, airport locations, and flight count data by time interval for sectors, airports, and links between them. This visualization gives insight into how traffic is flowing in and out of airports and how air traffic controllers are affected by the traffic flows over this time period.

### 3.2.2 Network Flow Mappings

The mapping of data in the translation file for a sector center point creates an X3D tree branch as shown in Figure 7. Each sector center point contains a name, latitude, longitude, and altitude that are translated into a GeoLocation node's latitude, longitude and altitude. The DEF name for the GeoLocation contains the name of the sector plus GeoLocation to uniquely identify this node. DEF is a field that all X3D Nodes contain which is used to uniquely indentify the node. DEF fields can also be used in conjunction with USE fields so that fields defined with a DEF field can be copied, which is far more efficient than creating new nodes. The GeoLocation node contains two children, a Text child to display the sector name, and a Sphere child to represent this sector. A Billboard node is utilized so the Text child will always face the user, and the Sphere child is given a Material node since the desired behavior is to change the color of the sphere based on changes to the flow over time. Rather than embedding animation nodes in the scene as in Example 3.1, we are creating the animation dynamically utilizing the X3D Service Engine which updates the scene using the SAI. Airports are described by a very similar tree branch, except that a Box node is used instead of a Sphere node.

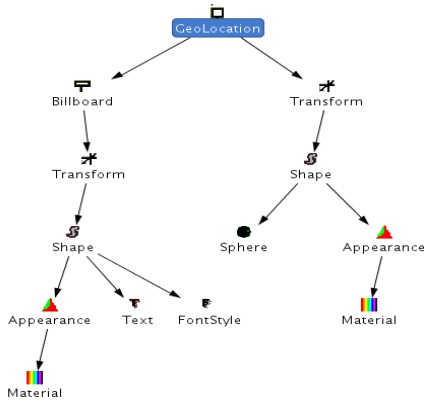


Fig. 7. Tree View of Network Flow Scene

Figure 8 shows the branch view of the flow links between the adjoining sectors and airports. The IndexedLineSet contains the coordinates for the sector or airport ends. The LineProperties node is added to change the size of the links based on the amount of traffic. The entire Tree View for this scene, once all sectors, airports, and nodes have been created, is extremely large. However, the Zoom and Satellite views make it easy to navigate the tree.

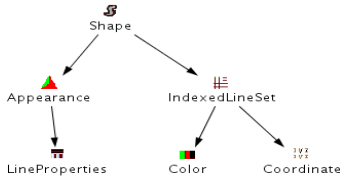


Fig. 8. Flow Link

### 3.2.3 Static Flow Network Assembly

It was beneficial to create a service assembly for this network flow, otherwise several scripts and commands would have been necessary and manually invoked whenever new sets of data were produced. With this service assembly, when a user runs a simulation for a given

time period for a particular airport, the service assembly picks up the airspace data written to disk and creates a static network flow XSD file. The service assembly for this example is shown in Figure 9.

In this example, the simulation generates airspace data in XML and creates a file on disk. Next, a BPEL process is created using the Netbeans IDE. This process reads the XML airspace data using the file binding component. It then sends the data to the XSLT service engine that was configured to convert the data into an X3D scene.

The dynamics of the scene can be accomplished by either using the animation techniques shown in the first example, producing a standalone X3D file, or through the use of the X3D Service Engine. An additional benefit to using the X3D Service Engine is that users can connect to live simulations and view the data as the simulation is running.

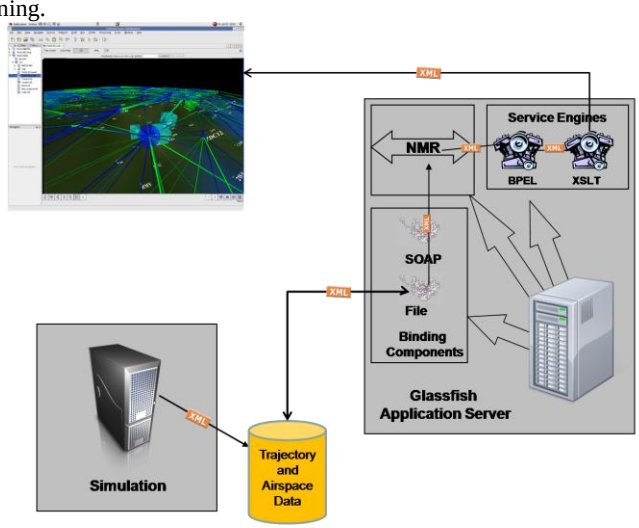


Fig. 9. Static Traffic Flow Service Assembly Diagram

### 3.2.4 Dynamic Traffic Flow Network

Figure 10 shows the completed running service assembly. In this example, a simulation is running outside the JBI environment and sends data over TCP. A BPEL process was configured to read the incoming simulation data using a TCP binding component, and then sends the incoming messages to the X3D service engine. The X3D Service Engine translates the data and sends the data to the Netbeans IDE where the user can view the updates. In future versions, users will be able to launch a browser from an HTML page on the application server to display the scene. This will allow users to share the scene with others in remote locations.

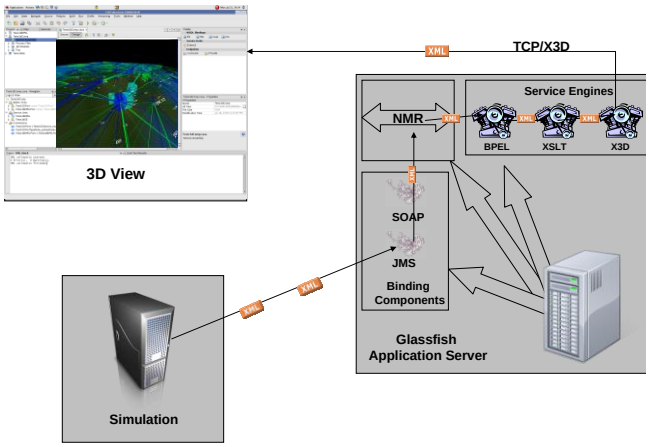


Fig. 10. Dynamic Traffic Flow Service Assembly Diagram

### 3.2.5 Dynamic Update Mappings

The first step in using the Data Map view on the X3DEditor is to construct a WSDL file which contains the messages to update the scene. For the scene in this example there are messages to update the size and color of the sectors, airports, and flow links. Once the user constructs a WSDL file they can drag and drop it onto the Data Map view canvas, which causes a jbi.xml file to be created for configuring this service engine into the JBI framework. The canvas is populated with the messages and schema types that are defined in the WSDL file.

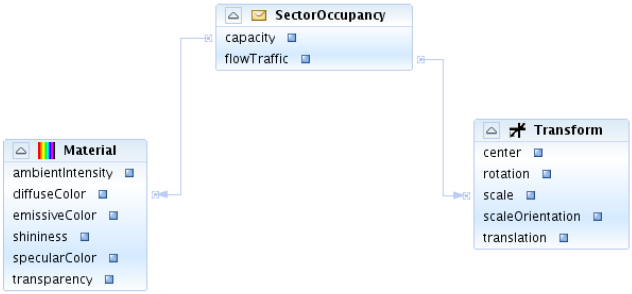


Fig. 11. Sector Mappings

The SectorOccupancy box in Figure 11 is a message type from the WSDL created for this scene. This message contains information about the given sector and updates the correct Sphere based on the DEF field. A mapping is shown that updates the Transform node's scale attribute which changes the Sphere's size. The capacity message part changes the color of the sphere. This mapping generates an XSLT file that is deployed into the X3D Service Engine, so that whenever this message arrives the sphere's color and size changes based on the incoming message parameters. The rest of the mappings are done using similar techniques. The dominant direction of flow is shown by a blue-to-green transition of the connected lines. The airports' traffic count increases the size of the boxes. The color of the airport box transitions from blue when most of the traffic is for departures, to green when most of the traffic is for arrivals. While the simulation is running, the scene gives users an indication of where the heavy traffic flows are occurring, and how those flows are impacting air traffic controllers by implying increased workload. Figure 3 shows a snapshot of this visualization.

## 4 Conclusion

In this paper we have shown that by applying modern SOA technologies we are able to make complex visualizations realizable without writing code. The paper described how by integrating X3D and ESB technologies we give users the ability to use graphical interfaces to construct visual elements, assemble a dynamic scene, connect to and transform data from a variety of sources, and map the data to the scene. The examples described in Section 3 are use cases that demonstrate the potential power and flexibility of this technology.

The power of this type of framework is in the elegance of the GUI interface. However, we have observed that, for widespread adoption of this technology, significant effort will be needed to improve the user interface that we have developed. Currently, the user needs to understand too much of the underlying technology to easily use the provided tooling, but with improvements to the user interface we believe that this type of visualization framework will enable users of all technological levels to develop sophisticated visualizations.

The combination of the X3D Service Engine and the X3DEditor with Open ESB provides users the ability to wire in data from external sources and construct visualizations without writing code, thus providing a codeless visualization programming environment for scientists, analysts, and non-programmers.

## REFERENCES


[1] Sun Microsystems. About Us: Open ESB. Open ESB. [Online] July 20, 2008. [Cited: July 20, 2008.] https://open-esb.dev.java.net/AboutOpenEsb.html.

[2] JBI Technical Component Overview. Open ESB. [Online] February 2008. [Cited: July 1, 2008.] https://open-esb.dev.java.net/kb/v2/jbiag.html..

[3] Rawkee. An Open Source X3D Plug-in for Maya. Rawkee. [Online] 2004. [Cited: July 21, 2008.] http://rawkee.sourceforge.net/..

[4] Media Machines. Developers Resources. Media Machines. [Online] 2008. [Cited: July 21, 2008.] http://www.mediamachines.com/developer.php.W.-K. Chen, *Linear Networks and Systems.* Belmont, Calif.: Wadsworth, pp. 123-135, 1993. (Book style)

[5] Blender. Blender Home Page. Blender. [Online] 2008. [Cited: July 21, 2008.] www.blender.org.

[6] Brutzman, Don and Daly, Leonard. Extensible 3D Graphics For Web Authors. Extensible 3D Graphics For Web Authors. San Francisco : Morgan Kaufmann, 2007, p. 441.

[7] Hudson, Alan. Xj3D Developer Documentation. Xj3D. [Online] 6 26, 2007. www.xj3d.org/arch.html

[8] Brutzman, Don. Scenario Authoring and Visualization for Advanced Graphical Environments. Savage. [Online] April 23, 2008. https://savage.nps.edu/Savage/index.html

[9] Sun Microsystems. Java Business Integration JSR 208. [Online] March 31, 2009. [Cited: March 31, 2009] http://jcp.org/en/jsr/detail?id=208

[10] Sun Microsystems. Netbeans API [Online] March 31, 2009. [Cited: March 31, 2009] http://bits.netbeans.org/dev/javadoc/index.html