International Journal of Advanced Computer Research (ISSN (print): 2249-7277 ISSN (online): 2277-7970)
Volume-3 Number-4 Issue-13 December-2013

Instrument Cluster Driver Information Display Database and Viewer Tool

Adnan Shaout[1], Dominic Colella[2]

Abstract

Instrument Clusters have grown increasingly complex in the last decade to include multi-character displays as standard features. Automotive Original Equipment Manufacturers (OEMs) all too often leave the types of messages displayed in their Instrument Cluster's up to the component suppliers, this accounts for huge overhead costs in software design, text string translations, validation, and maintenance. This paper presents a design of a database for an Automotive OEM's Instrument Cluster Driver Information Display as well as a tool for viewing the database and validation. This paper focuses on allowing an OEM to design its own database in house using inexpensive tooling to avoid the costly overhead of having a component supplier design the database in addition to designing the software to run the Instrument Cluster.

Keywords

Eclipse; Eclipse Modeling Framework; Sashimi Software Design Methodology; Instrument Cluster; JavaScript; XML; XSD; XSL;OEM

1. Introduction

As the automotive industry steadily begins to merge with the consumer electronics industry, the ratio of analog gauged Instrument Clusters to those with LCD displays begins to shift to the latter. As early as 2010, Ford Motor Co. unveiled their 2011 F-Series trucks with an integrated 4.2" LCD display. The display performed functions such as providing the Odometer, various vehicle information readouts, and a message center to deliver important, vehicle related information to the user. Figure 1 shows a sample instrument cluster with LCD display. These Driver Information Displays typically have different levels of priority for the messages displayed, usually determined by the color of the message, as well as including audible cues and visual images of the piece of information being delivered [1].

Adnan Shaout, Professor at the Electrical and Computer Engineering Department, The University of Michigan – Dearborn.
Dominic Colella, Graduate student in the Software Engineering MS program at The University of Michigan – Dearborn.

The text color, audible cue, and image all work cooperatively to begin teaching the driver to understanding what is going on with their vehicle without diverting their attention from operating the vehicle through learned reactions to any combination of the message's attributes. The driver can begin to associate taking a certain level of action in response to a message based on what color the text is or what the sound is. This elements help to increase the driver's awareness of what their vehicle is doing while maintaining their ability to operate it safely [2]. The biggest problem with the development of a Driver Information Display is that Automotive OEMs typically leave the design of the databases for these messages up to the component supplier's delivering the Instrument Clusters. At best, OEM's will deliver spreadsheets with lists of message text in them with little to no formal review of the text for correctness.

Figure 1. Sample Instrument Cluster with LCD Display

These suppliers then take the spreadsheets and assess the level of work involved in creation of a database to maintain these message lists, hire translation firms to translate the messages into the languages that the Instrument Cluster hardware will support, hire testing staff to validate the message text in all the translated languages, and then provide a financial quotation to the OEM which usually includes at least a 10% increase in cost to ensure their investment is returned. OEMs can avoid huge costs by designing these databases in house using the skill of employees they usually already have on staff to build the message lists, determine the correct tooling for handling the work, and building of tools to for validation of the message lists. Once the database in house is designed

then the database will be a living element that will grow as the features in the vehicle continue to grow and mature. As each new message is added to the database, the associated database viewer can be used to ensure the messages convey the correct information to the driver and also check the translations of the messages to ensure information is not lost in conversion from one language to the next.

This paper will present a design of a database for an Automotive OEM Instrument Cluster Driver Information Display as well as a tool for viewing the database and validation. The software design process in this paper focuses on the use of the Sashimi software design methodology, an offshoot of Waterfall method, for the planning, design, code, and testing of the database and database viewer elements. The format used for the database is eXtensible Markup Language (XML) and the tooling designed to allow the database to be viewed is a Hyper Text Markup Language (HTML). The Integrated Development Environment (IDE) used is the free and open source Eclipse package with the Eclipse Modeling Framework (EMF) plug-in used for database modeling. To the knowledge of the authors no papers has been published with a design of a database for an Automotive OEM Instrument Cluster Driver Information Display with a tool for viewing the database and validation.

2. Objectives of This Paper

The first objective in this paper is to develop the Driver Information Display message database. The second object is to develop a tool for viewing the database to validate text and translations.
For the database, the requirements are going to be the following:

1. The database should allow for a message's title, text, color, sound, and image parameters to be added for each entry.
2. The format used for the database should be easily read by both computers and humans while being common enough that any Instrument Cluster supplier would know how to work with it.
3. The database should allow all text (both title and message text) to be entered in the six most common languages for vehicles shipped to the United States and Western Europe English, French, Spanish, German, Italian and Polish.
4. The database format should be flexible enough to handle data entered as strings and integers.
5. The database should be constrained and validated against these constraints to ensure clerical errors in entries do not lead to product defects.
6. The database shall allow new entries to be added easily.

For the database viewer, the requirements are going to be the following:

1. The viewer should be easy to use and include instructions and additional information.
2. The viewer should not require extra software to run outside of the normal suite of programs included with any Microsoft Window's PC.
3. The viewer should be developed using an IDE that is inexpensive and wide spread enough such that the software could be updated though future maintenance and possibly by new team members not included in the original development.
4. The viewer should provide an output to the user that matches the three Automotive OEM Instrument Cluster Display sizes.
5. The viewer should model the audible feedback to the user that would accompany the visual display of the Drive Information Display message.

After meeting with the customer, the software team can make in initial proposal for a solution.

A. Proposed Solution

Based on requirements 2 and 4, as stated above for the Driver Information Display message database, the most suitable option would be to define the database using eXtensible Markup Language or XML. XML is a subset of what is known as Standard Generalized Markup Language or SGML. The usefulness of XML in this application comes in its flexibility. Since the author of the XML document defines the "tags" used for each "node" or entry in the database, any combination of parent and child elements can be created. While an XML document can be used on its own, the creation of an XML Schema Definition file (XSD) allows constraints to be placed on the elements within an XML file. The XML file can then be validated against the constraints to ensure scenarios like integer values going out of bounds or character limits are not exceeded, the creation of a well formed XSD file would satisfy requirement 5 above. To meet the needs of requirements 1, 3, and

5; the structure of the XSD and XML files will be built to accommodate five parameters per entry and support six languages over the title and text parameters [3]. The proposed solution for the Driver Information Display database viewer compliments the versatility of the XML format used for the database. The viewer will use Extensible Stylesheet Language (XSL) to convert the XML format into Hyper Text Markup Language (HTML). XSL on its own is a stylesheet language used to render text in XML documents into a different format, for this case the new format will be HTML. The scripts in the XSL provide template and formatting information that feeds to a process known as XSL Transformation (XSLT). Most modern web browsers include the capability to perform XSLT when provided with XML document with a link to an XSL that contains the formatting instructions [4 and 5].

When viewing an XSL document that performs an XSLT to HTML, the first assumption is most likely that you are reading a traditional HTML document but further inspection proves this to be untrue. The confusion is caused by that face which XSL documents typically contain elements of the XSL namespace, denoted by elements starting with "<xsl:", and elements of the target format of the XSLT, in this case traditional HTML elements such as <body> or <table>. When the XSLT transformation is performed, the XSL elements are transformed to the new format while the HTML elements are untouched. This flexibility allows a well formed and styled web page to be created within an XSL document and the final XSLT performed by your web browser or XML editor results in a well composed web page that also includes the parsed elements from your XML document. This combination allows data in XML databases to be parsed and viewed easily in web browsers without requiring expensive IDEs for the development of the viewer or specialized software or hardware for usage of the viewer; these facts fulfil requirements 2 and 3 of the database viewer [3, 5, and 6]. For requirement 1 of the viewer, various user interface elements that leverage the flexibility in the combination of JavaScript and HTML will be used to create a web page that provides the look and feel of an embedded software application. HTML is another markup language similar to XML, both of which are based on the original markup language of SGML. Strategies for HTML and XML differ in many ways, two of the key differences are:

- XML defines what the data type of the current element is and the value for the current element of a data type.
- HTML defines the visual organization of data within a web page and how the data is styled when viewed in a browser.

Simple HTML documents typically include embedded styling instructions to adding formatting to how the data looks but added styling information accompanies more complex HTML

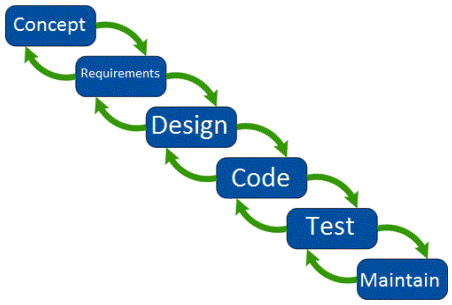

Figure 2. Progression of phases in Sashimi Methodology [7].

documents in the form of a Cascading Style Sheet (CSS). CSS scripts attach styling information to various headers or section ids within an HTML document to perform formatting like varying text font, color, size, or style or adding structural details to the HTML elements such as centering, padding, bordering, or adding tables for information within the HTML element. JavaScript is considered to be the most prevalent scripting language used in web pages and internet data transfer. An important idea about in learning JavaScript is that HTML on its own is static, meaning that once your internet browser has processed your HTML the data will remain unchanged if left purely up to the HTML code. What JavaScript provides is a scripting language to add dynamicelements to HTML web pages such as visual or audio reactions to user interface elements such as buttons or menus or data processing based on input values to a form or input field. HTML elements and JavaScript functions can be developed to provide tangible interface items such as different button types and drop down menus to provide a simple means to add the functionality of an installed application to a web page. To improve the user interface further, the recently created HTML5 markup language can provide dynamic web page elements such as image elements known as "canvases" and the HTML5 "audio" element [3, 4, and 5].

In HTML5, the "canvas"' element provides the web page designer with the ability to draw dynamic graphic within an organized portion of the screen using scripts, usually JavaScript. The "canvas" elements are defined within the body of the HTML as a sort of empty container, there are no inherent drawing functions owned by the element itself. JavaScript methods are used to populate the container with images, draw geometric shapes, or input scripted text objects into the "canvas". The HTML5 "audio" element allows a web page to play audio on the client's PC without the use of external audio software. The audio data itself comes in the form of an audio file in one of several supported formats: .wav, .mp3, or .ogg. Since the target browser to be used is Microsoft Internet Explorer, the audio format supported is .mp3. An "audio" element has several methods to control the play of the audio such as providing a play command through the use of an on screen button or through JavaScript that plays the audio file in response to events such as on the first load of the web page or when the function controlling the script to play the file is run [4 and 6].

B. *Software Metodology Employed*

Due to the straightforward nature of the objectives of this project in this paper, the Sashimi model was chosen. The Sashimi model matches the Waterfall model in the organization and progression of phases but each current phase actually overlaps with the previous phase. In the Sashimi model, the progression is more similar to Concept/ Requirements, Requirements/Design, Design/Code, Code/Test, and end with the Test/Maintain phase. This progression enables more information sharing from one phase to the next such as applying lessons learned from current phase to the previous phase or adjusting scheduling for the next phase due to information found in the current phase. Figure 2 shows the model allowing each current phase to overlap on to the next phase, which in turn provides feedback to the previous phase while in the current phase. Since the requirements for this paper are relatively straight forward and should remain static over the course of development, the Sashimi model's logical progression and feedback capabilities make it an efficient choice [7].

3. Sashimi Phases

Since the Sashimi or "Waterfall with feedback" software design methodology was chosen for this paper, a layout of the phases is in order before we begin.

A. *Concept*

Our initial concept of creating a machine readable database of Driver Information Display messages in XML format with validation and robustness provided by an associated XSD file is a feasible task. Tooling that would be appropriate for work of this type would be the free and open source Eclipse Integrated Development Environment with the associated Eclipse Modeling Framework Plug-in. Eclipse provides a unique graphical approach to modeling the structure of XSD and XML data and conveniently ties the creation of a new XML document automatically to a pre-selected XSD file to validate the XML data accordingly. The popularity of the Eclipse IDE is also a contributing detail that adds to the feasibility of using this tool on this project. An increased popularity in an IDE guarantees that more users will already be acquainted with the tool enough to be able to being using without much training or help to train others, either case suits to increase the feasibility of a successful project.

The next feasibility analysis would be on the creation of a database viewer for XML databases that output to HTML. The choice in the initial proposed design was to use an XSL Transform within an Internet Browser to view the database. Since the most popular internet browser on business computers is Microsoft Internet Explorer, we'd need to ensure compatibility between XSLT and Internet Explorer versions. Since the release of Internet Explorer v.6 in 2001, browser support has been available for XSLT and since each Microsoft Windows release since Windows XP has included Internet Explorer v.6 or greater, there is no potential that a user of the viewer would not have the correct browser on their PC to use our software [8].

For the tooling used to create the Driver Information Display viewer, Eclipse Modeling Framework includes and extensive set of plug-ins to provide efficient created of XSL documents that result in generated in HTML. Eclipse also allows for development of the necessary JavaScript and CSS files to improve the quality of the viewer.

B. *Requirements*

For the requirements of the paper, we can begin by referencing the initial requirements from the customer and build off those to perform analysis to lead to the end result of a set of product requirements. The requirements for the database are designed as follows:

- Each database entry shall have a parameter for message text, message title, message sound, message color, and message image.
- Each database entry shall be flexible enough to display a title/text combination in all of the following languages: English, French, Spanish, German, Italian, and Polish.
- The database title and text lengths shall be small enough to fit on the following display resolutions appropriately:
    - 800px x 400px
    - 300px x 400px
    - 250px x 150px
- The title field shall be limited to 25 characters with a minimum of 5.
- The text field shall be limited to 75 characters with a minimum of 5.
- There shall be up to 16 sounds supported.
- There shall be up to 16 colors supported.
- There shall be up to 32 different images supported

Based on the preceding requirements for the database, design can begin on how to structure the XSD and XML data. For the Driver Information Display database viewer, the following requirements were developed:

- The viewer shall be implemented as a web page that can parse XML databases.
- The viewer be coded in a format that lends itself to viewing in Microsoft Internet Explorer.
- The viewer shall adhere to all Driver Information Display form and behavioral guidelines that Instrument Cluster Suppliers must adhere to when delivering a complete product to ensure consistent operation of the end product and the database validation tooling:
    - All message titles shall be no longer than 25 characters
    - All message text fields shall be no longer than 75 characters
    - Title and message text fields shall be color coded with the following RGB color triplets:
        - Color Code 1: #00FF00 – Green
        - Color Code 2: #FFFF00 – Yellow
        - Color Code 3: #FF0000 – Red
        - Color Code 4: #FF9900 – Orange
    - The Viewer shall implement 21 of the approved vehicle telltale light images as noted in Federal Motor Vehicle Safety Standard 101.
        - Each of the 21 telltale light images shall be colored in any of the four colors approved for the Driver Information Display.
    - The viewer shall implement all approved motor vehicle chime sounds.
        - Sound 1: A 750Hz Beep tone repeated 4 times with a 200ms separation of pulses.
        - Sound 2: A 750Hz amplitude decay tone at 1200ms in duration, repeated 5 times, with the last tone decaying with a 2400ms duration
        - Sound 3: A 2000Hz amplitude decay tone at 500ms in duration, repeated 3 times.
        - Sound 4: A 2000Hz Beep tone repeated 4 times with a 200ms separation of pulses.
    - The viewer shall allow the user to validate all messages in all 6 supported languages.
    - The viewer shall allow a user to validate all messages in all three display resolution formats:
    - At the smallest display size the Viewer shall not display an image to maximize display for the text.

C. Design

While the requirements phase is underway, the design phase can begin on the XSD files to define the Driver

```
<xsd:element name="didmsgdbc">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="didmsg" maxOccurs="unbounded"
                minOccurs="1">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="msgtitle" maxOccurs="1"
                        <xsd:element name="msgtext" maxOccurs="1"
                        <xsd:element name="msgcolor" maxOccurs="1"
                        <xsd:element name="msggraphic" maxOccurs="1"
                        <xsd:element name="msgsound" maxOccurs="1"
                    </xsd:sequence>
                    <xsd:attributeGroup ref="msginfo"></xsd:attributeGroup>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

Figure 3. "didmsg" Element Definition in XSD Code. (Child Element Definitions Collapsed to Save Space)

Information Display database. When designing an XML Schema, the user first defines what the Root element in their XML will be and then follows that by defining what each one of the Child elements in the database will be. When an XML document is created, the XSD file is used to ensure the structure of the XML file follows the requirements in the XSD. To create a database in XML, the associated XSD file will typically define a top level Root element and then a single Child element that can occur an unbounded number of times but is required at least once to make up a complete database. For our purposes, the Root Element will be called "didmsgdbc" and will have Child elements called "didmsg".

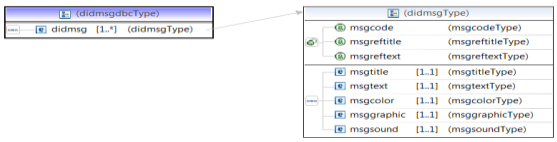

Figure 4. "didmsg" Graphical Representation in Eclipse EMF

Each didmsg element is a separate entry in our Driver Information Display database. To differentiate multiple elements of the same type within an XML file, each element can use attribute values which act as identifiers within the XML document. For each didmsg element, we will include three attributes: "msgcode", "msgreftext", and "msgreftitle". The message code field is used to identify each database entry and is unique and non-repeatable. The message reference text field is used to add reference text to the database entry for traceability; the message reference title field is used for the same purposes. The message code is defined as an integer data type that is constrained between 0 and 256, the reference title field is a string with a length between 5 and 25, and the reference text field is a string between 5 and 75 characters in length. Within each didmsg element, there are 5 child elements: "msgtext", "msgtitle", "msgcolor", "msggraphic", and "msgsound". Each of these elements is defined as an integer data as shown in figures 3 and 4.

For the message text and title parameters, we are using an integer value instead of a string because what our design calls for is known as a relational database which will be explained in the next paragraph. The message color parameter is defined as an integer between 0 and 15, the message sound parameter is define as an integer between 0 and 15, and the message image parameter is an integer between 0 and 31. These constraints ensure that the values for each of the parameters can be validated by the XSD file when an XML file is created. Figure 5 shows examples of constraint definitions in XSD. This ensures each element within an XML file follows the validation requirements set forth in the XSD file, which produces a more robust and error free set of data. To add further robustness, the message sound, message title, and message color parameters are also defined by sets of enumerated values. For the message title, the design only requires supporting 4 different message titles and thus the title parameter only allows a choice of integer values between 0 and 4, with 0 being a test value. For the message color, the design only requires supporting 4 color codes and thus the color parameter only allows a choice of integer values between 1 and 4, the sound parameter operates in the same manner with the integer value constrained to between 1 and 4 [3].

To allow the Message Database to correctly supply text strings to the Driver Information Display, another XML database is employed that only relates the text and title code parameters to the actual strings of characters that are displayed. What are defining now is known as a Multilingual User Interface (MUI) database, which is a collection of text strings in various languages all bound together by a common string ID. The concept functions as follows: within our message database we supply an integer value for the msgtitle

```
<xsd:element name="msgsound" maxOccurs="1"
    minOccurs="1">
    <xsd:simpleType>
        <xsd:restriction base="xsd:int">
            <xsd:minInclusive value="0"></xsd:minInclusive>
            <xsd:maxInclusive value="15"></xsd:maxInclusive>
            <xsd:enumeration value="1">
            </xsd:enumeration>
            <xsd:enumeration value="2">
            </xsd:enumeration>
            <xsd:enumeration value="3">
            </xsd:enumeration>
            <xsd:enumeration value="4">
            </xsd:enumeration>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:element>
```

Figure 5. Example of Constraint Definitions in XSD

```
<xsd:sequence>
    <xsd:element name="didmuimsg" maxOccurs="unbounded" minOccurs="1">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="eng" maxOccurs="1" minOccurs="1">
                <xsd:element name="esp" maxOccurs="1" minOccurs="1">
                <xsd:element name="frn" maxOccurs="1" minOccurs="1">
                <xsd:element name="ger" maxOccurs="1" minOccurs="1">
                <xsd:element name="itl" maxOccurs="1" minOccurs="1">
                <xsd:element name="pol" maxOccurs="1" minOccurs="1">
            </xsd:sequence>
            <xsd:attribute name="txtcode" use="required">
                <xsd:simpleType>
                    <xsd:restriction base="xsd:int">
                        <xsd:minInclusive value="0"></xsd:minInclusive>
                        <xsd:maxInclusive value="255"></xsd:maxInclusive>
                    </xsd:restriction>
                </xsd:simpleType>
            </xsd:attribute>
            <xsd:attribute name="txttype" use="required">
                <xsd:simpleType>
                    <xsd:restriction base="xsd:string">
                        <xsd:enumeration value="text"></xsd:enumeration>
                        <xsd:enumeration value="title"></xsd:enumeration>
                    </xsd:restriction>
                </xsd:simpleType>
            </xsd:attribute>
        </xsd:complexType>
    </xsd:element>
</xsd:sequence>
```

Figure 6. MUI Database Definition in XSD

constrained between 0 and 99 and further constrained within enumeration constants of 1-4. When the Message Database is parsed, the result is a set of five integer values for title, text, color, sound, and image. Two of these values, message text and title, are then searched for in the MUI database according to an attribute value in each entry known as the "txtcode". Once one of the values is matched, string for the currently chosen language is chosen based on the element title within the MUI database XSD file. This allows all the actual translations to be separated into one database and the functional behavior to be defined in another database. This is known as a relational model, which links elements of databases to elements in another to produce an end result that is a combination of the previous two databases. The MUI database XSD file is composed of a Root element known as "didmuidbc" with children elements known as "didmuimsg". Each didmuimsg element contains two attributes: "txtcode" that links the incoming message title or message text code to the MUI database element and the "txttype" attribute which defines if the current MUI element contains text strings for a title element or a text element. Within the MUI database, txtcodes 0-99 are reserved for message titles, and 100-255 are reserved for message text. Within each didmuimsg element, there are six child elements for strings in each of the languages supported. The XSD definition for the MUI database can be seen in figure 6.With the XSD files defined for the DID message and MUI message database, the populating of the database can be handled by hand during the coding phases. The next step in the design phase is the design of the viewer. From the requirements, we know the viewer has to be an XSL document whose end result after transformation is into an HTML web page with proper scripting to allow the following interactions:

- A UI element to allow selection of one of the six supported languages.
- A UI element to allow selection of one of the three supported display resolution sizes.
- A UI element to allow selection of one of the Driver Information Display database entries.
- The ability to produce audio to the user to simulate vehicle chimes.
- The ability to control a portion of the screen to dynamically populate it with colored text, vary the text size, and impose an image under the text.

To handle the UI elements for selection of language and display sizes, we will choose to use what are known as radio buttons. Figure 7 shows examples of Radio button design. Radio buttons allow a specific center of interactions from the user: all selections are mutually exclusive and selections of the same option cannot be made twice. These two characteristics are unique to radio buttons and are not found in more traditional button elements or check boxes. Within the HTML code, a form type with the label of radio is used to define a radio button. When a radio button is selected a JavaScript function is called to handle the dynamic reaction to the button. The each element in the radio button form has an attribute of "onclick=" that defines what JavaScript function is called when the option is selected. To handle the display of the different database entries to the user,

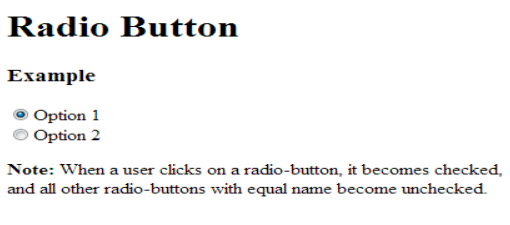

Figure 7. Example Radio Button Design

the most efficient choice is through a combo box or better known as a drop-down menu. Drop-down menus allow lists of information to be contained in a single field which contains a long list but only displays a small footprint on the visual portion of the web page. Figure 8 shows example of Combo box design. Similar to the radio buttons, the HTML information for the drop-down is defined within a form and the attribute of "onchange=" defines what JavaScript function is called when a new option is chosen. To provide the ability to produce audio, the design decision was made to use an HTML5 audio element. Within HTML code and audio element is defined using the code "<audio>" as seen in figure 9. The audio element can have several attributes such as defining whether or not it plays automatically at page load, whether the web

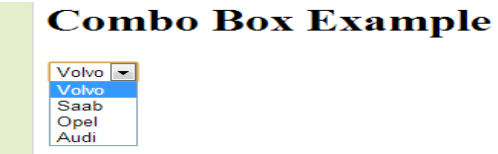

Figure 8. Example of Combo Box Design

page includes visual display of play/pause controls, or if the audio is looped or not. To dynamically populate a portion of the screen with text or images, we'll be using another new HTML5 element: the canvas. An HTML5 canvas element is only constructed by the

```
<!DOCTYPE html>
<html>
<body>

<audio controls>
  <source src="test.ogg" type="audio/ogg">
  <source src="test.mp3" type="audio/mpeg">
  You do not have audio element support in this browser.
</audio>

</body>
</html>
```

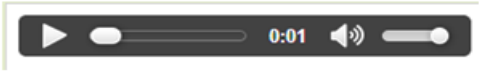

Figure 9. HTML5 Audio Element Code and Chrome Brower Supported Audio Controls

HTML code with details for initial size and location, the actually dynamic drawing or filling that occurs with the canvas is handled by JavaScript as seen in figure 10. The canvas is located within the HTML code by the element "id=" attribute, the JavaScript creates an object for it by using the "document.getElementById("myCanvas")" constructor. Once the canvas object is created, a context for it is created and then the area can be manipulated by the JavaScript.

```
<!DOCTYPE html>
<html>
<body>

<canvas id="myCanvas" width="200" height="100" style="border:1px
Your browser does not support the HTML5 canvas tag.</canvas>

<script>

var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
ctx.font="30px Arial";
ctx.fillText("Hello World",10,50);

</script>

</body>
</html>
```

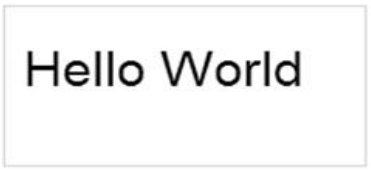

Figure 10. HTML and JavaScript Canvas Code and Result in Web Page

With the design of the web page elements defined, the next set of work is to code the DID message and MUI databases and code the XSL document to be transformed to HTML.

D. D.Code

For the first half of the coding phase, the message database will be populated. There are 26 initial Driver Information Display messages to create entries for within the database as shown in figure 11. This process is greatly simplified through the use of the Eclipse IDE which verifies each new addition to your XML document by checking it against the requirements in the schema first.

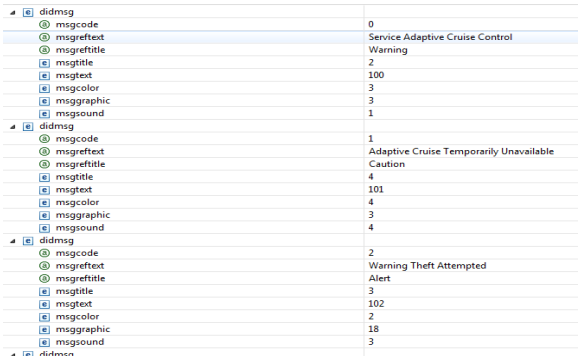

Figure 11. Graphical Representation in Eclipse IDE of Driver Information Display XML Database

Once each of the database entries are added to the message database using the XSD file as guidance with

the Eclipse IDE tool, the next step is to populate the MUI database with each of the text stings. As in the previous database, using the XSD file and Eclipse automatically verifies to additions to the XML file according to the schema as shown in figure 12.

After the MUI database is populated, both XML files are validated according to their XSD files to ensure all parameters are filled out correctly. The first step in coding the XSL files to create the saved XSL file and then create a link to the file within the XML file

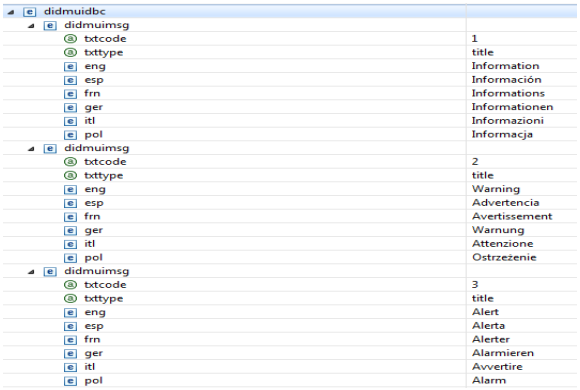

Figure 12. Graphical Representation of MUI Database in Eclipse

using the XSL style sheet link element as follow:
<?xml-stylesheet type=*"text/xsl"* href=*"Viewer_ECE574.xsl"*?>
This element within the XML file is what notifies your Internet Browser that an XSL exists for this XML and the link to where the XSL can be found. Without this link in the XML document, the browser would not be able to associate this XML with any XSL documents.

When beginning an XSL file, the first step after creating the initial headings is to declare the first template element. Within the templates are where the XSL elements and the HTML elements are combined. During the transform the browser ignores the HTML elements and only focuses on the XSL elements, denoted by the XSL namespace "<xsl:". The templates within an XSL usually follow the hierarchy of the nodes in an XML document, identified by the node path within the match attribute as follow:
<xsl:template match=*"/didmsgdbc"*>
The preceding template is the highest level template, which will encompass the bulk of the traditional HTML code within the XSL. Inside this template is where the <head> and <body> elements are defined for the HTML. In the <head> element are defined the two links: one to the JavaScript file which will contain all of our functions titled "my_java.js" and one link to the Cascading Style sheeting titled "mystyle.css".

After the head element, the body element defines the HTML to create the drop down menu, the two series of radio buttons, and the HTML5 canvas elements. The drop down menu in HTML is defined by the following code:

```
<form name="comboBoxForm" size="1" method="POST">
<select name="comboBoxMenu" onchange="getComboBox()">
<xsl:apply-templates select="didmsg" mode="combo_item"/>
</select>
</form>
```

The code defines the combo box form that will populate the dropdown menu. The "onchange=" attribute is what notifies the JavaScript function "getComboBox" to be called when the user changes the value of the drop down menu, the size attribute determines that only 1 option is displayed at once, and the method attribute determines how to transfer the form data. The menu itself is populated by an XSL template as follow:

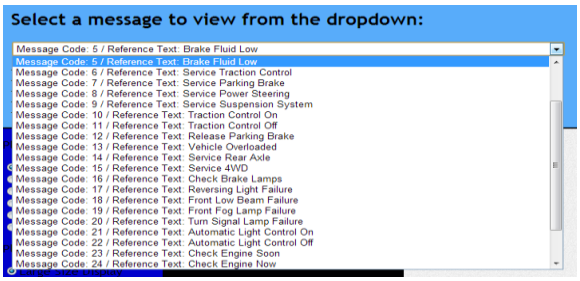

Figure 13. Output of XSL Transform of Drop down Form/Template

```
<xsl:template match="didmsg" mode="combo_item">
<xsl:element name="option">
<xsl:attribute name="value">
Value attributes excluded for brevity of paper.
</xsl:attribute>
<xsl:value-of select="concat('Message Code: ', concat(@msgcode,concat(' / Reference Text: ',@msgreftext)))"/>
</xsl:element>
</xsl:template>
```

What the XSL code above does it to apply the XSL Transformation dictated by the template while the transformatin is in "combo_mode". This portion of the transform is what populates the actual drop down menu, dictated by the "value-of" element. This populates the drown down menu with a static string of "Message Code: " concatenated with a dynamic string populated by the XSL transform of the message code text concatednated with the static text of " / Reference Text: " and in the end concatenated by the dynamic string populated by the XSL trasform of the message reference text, both pieces of information parsed our of the message database XML file. A screen shot of the HTML output of the transform is shown in figure 13.

After the drop down menu HTML form, the form for the two sets of radio buttons is found in the menu section. The language selection radio buttons are defined as follow:

```
<form >
<input type="radio" name="lang" id="eng"
onclick="chklang(this.value)"
value="0,1" checked="checked"/>English<br/>
<input type="radio" name="lang" id="esp"
onclick="chklang(this.value)"
value="2,3" />Spanish<br/>
<input type="radio" name="lang" id="frn"
onclick="chklang(this.value)"
value="4,5" />French<br/>
<input type="radio" name="lang" id="ger"
onclick="chklang(this.value)"
value="6,7" />German<br/>
<input type="radio" name="lang" id="itl"
onclick="chklang(this.value)"
value="8,9" />Italian<br/>
<input type="radio" name="lang" id="pol"
onclick="chklang(this.value)"
value="10,11" />Polish
</form>
```

From the radio button HTML, you can see that each button is leasted as part of the form under an "input" element. Each input element of a radio button is listed as "radio" type and radio button inputs from the same set of buttons all require the same name. Each button calls this "chklang()" fuction upon being clicked or selected and when the function is called it passes the inputs value property as the input argument to the chklang() function. In the following sections we will review the JavsScript functions to see how this operates. The radio button HTML for the display size resolution buttons is as follow:

```
<p>Please select a display size:</p>
<form >
<input type="radio" name="size" id="big"
onclick="cvsBig()"/>Large Size Display<br/>
<input type="radio" name="size" id="med"
onclick="cvsSmall()" checked="checked"/>Mid
Display Size<br/>
<input type="radio" name="size" id="sml"
onclick="cvsTiny()" />Small Display Size<br/>
</form>
```

From the above HTML, you can see that when one of the display size radio buttons is selected, their own JavaScript function is called. These functions each activate to dynamically adjust the size of the HTML canvas element when called as seen in figure 14.

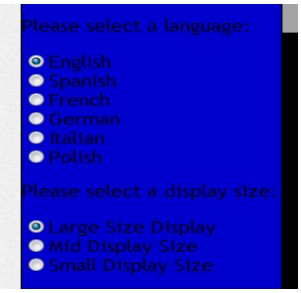

Figure 14. Language and Display Size Radio

After the radio button definitions, the HTML5 canvas elements are defined. Driver Information Display supports two fields, one for the message title and one for message text; we will use two overlapping canvases with one defined for the title text and one defined for the message text as follow:

```
<canvas id="myCanvas" width="300" height="400"
style="border:1px solid #000000;
background-color:#000000;position: absolute; left:
0px; top: 0px; z-index: 0;">
Your browser does not support the HTML5 canvas
tag.
</canvas>

<canvas id="myCanvas2" width="300" height="30"
style="border:1px solid #000000;
background-color:#A4A4A4;position: absolute; left:
0px; top: 0px; z-index: 1;">
Your browser does not support the HTML5 canvas
tag.
</canvas>
```

You can see in the previous code that the canvas elements use the "z-index" attribute to define which layer the canvas sits on. Canvas id myCanvas is the

large canvas defined for the message text and is 300px x 400px and is on layer 0. Canvas id myCanvas2 is the small canvas defined for the message title and is 300px x 30px and is on layer 1 and thus on top of the previous canvas.
After the canvas elements are defined, the HTML5 audio element is defined as follow:

```
<audio id="one" src=""></audio>
```

The audio element is identified by the id attribute, the src attribute defines where the path is for the Internet Browser to find the audio files. The src attribute is blank by default because it is populated by the "getComboBox()" JavaScript function when the drop down menu changes state.

Now that the HTML elements of the XSL have been reviewed, the focus can shift to the JavaScript functions in the my_java.js file included with the software. The JavaScript file contains seven functions:

- startup()
- chkLang(radioValue)
- textColor(colorCode)
- getComboBox()
- cvsBig()
- cvsTiny()
- cvsSmall()

The startup() function is called at web page load when the body element of the HTML loads. This function assigns the default language to English and calls the chkLang() function at on load.

The chkLang() function accepts the radio button values and then writes two global JavaScript variables: ttl and msg. These variables determine which language to use when making a change to the language radio buttons. After the radio buttons are clicked, the JavaScript calls the getComboBox function to update the value within the canvas elements to the new language.

The textColor() function accepts a color code value when called from the getComboBox function and sets the global variable of clr to the HTML color triplet value. The getComboBox() function is called when the drop down menu changes state, when the canvas changes size or when the language changes. When called, the function starts by pulling the values from the combo box form within the HTML. After this the form data populates fields within the HTML code that notify the user of elements of data such as the message title, message text, message color, file path to the message audio file, and file path to the message image file. Once the HTML values are assigned, the function uses the combo box form data to fill myCanvas and myCanvas2 with the data from the MUI database.

Before the text string is written to myCanvas, it is processed through an algorithm to determine the correct text size to set the font size that will maximize the space in the canvas. After the message text is written to myCanvas, the image path is determined by concatenating a string of the message color code and a string containing the image code form the message database and then a string with the static text of ".gif". This allows 4 different sets of each image to be created in each of the 4 colors with the file names differing by the leading digit, which is the color code. Within the database, the same codes can be used for all messages and the combination of the color code and the image code is what leads the HTML to receiving the correct path to the image file. After the image algorithm, the message sound parameter from the message database is written to the audio element within the HTML to ensure one of the 4 audio files is found. After the src property is written for the audio element, the play method is used to start the audio file. This ensures that each time the getComboBox function is called, the audio file will be simulated. The final 3 JavaScript functions are cvsBig(), cvsSmall(), and cvsTiny(). Each of these functions is called when the display resolution radio buttons are selected to change the size properties of the canvas elements to the correct resolution.

E. Test

The testing phase of the design methodology is relatively simple for a project of this size. Testing was done by creating a

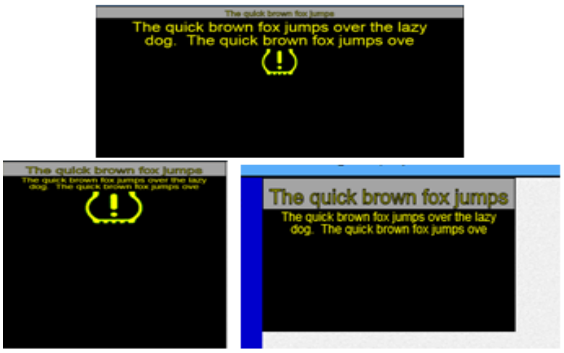

Figure 15. Test Message in all Display Resolution Sizes

test message within the message and MUI databases that contained the maximum length text string of 75 characters and title string of 25 characters. The text used is the classical test text string of "The quick brown fox jumps over the lazy dog." By creating a message like this we can test each of the images, all 4 colors, all 4 sounds, and the entire length of the title and text field for any errors that could be introduced into the viewer. Figure 15 shows test messages in all display resolution sizes.

The testing phase did not reveal any issues and was completed ahead of schedule. Since no refactoring was required after the testing phase, there were no updates to make in the Maintenance phase.

4. Conclusion

This paper presented a Driver Information Display database in XML format and associated HTML viewer for the XML data. The feasibility for an Automotive OEM's employees to perform the work was laid out in the paper. The paper allows an organization to reduce cost from Instrument Cluster suppliers. The paper followed the Sashimi software design methodology. The paper exhibited one of the many useful functions of markup languages and the versatility of the Eclipse Integrated Development Environment.

References

[1] Levine, Mike, "Ford F-150 Getting More Productive for 2011", On line at http://news.pickuptrucks.com/2010/04/ford-f150-getting-more-productive-for-2011.html (as of January 5, 2014).

[2] Howard, Bill, "Digital dashboard: Why your car's next instrument panel will be one big LCD", On line at http://www.extremetech.com/extreme/131485-digital-dashboard-why-your-cars-next-instrument-panel-will-be-one-big-lcd (as of January 5, 2014).

[3] "Extensible Markup Language (XML) 1.0 (Fifth Edition)" On line at http://www.w3.org/TR/2008/REC-xml-20081126/ (as of December 19, 2013).

[4] "JavaScript Reference", On line at https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference (as of Decemeber 19, 2013).

[5] "XSL Transformations (XSLT)", On line at http://www.w3.org/TR/xslt (as of December 19, 2013).

[6] Berjon, Robin, Travis Leithead, Erika Doyle Navara, Edward O'Connor, Silvia Pfeiffer (editors), "HTML5 A vocabulary and associated APIs for HTML and XHTML W3C Candidate Recommendation 17 December 2012", On line at http://www.w3.org/TR/html5/ (as December 19, 2013).

[7] Rising, Jim;, "Sashimi Waterfall Software Development Process", On line at http://www.managedmayhem.com/2009/05/06/sashimi-waterfall-software-development-process/ (as December 10, 2013).

[8] Warburton, Richard, "The Resurgence of Apache and the Rise of Eclipse", On line at http://java.dzone.com/articles/resurgence-apache-and-rise (as December 19, 2013).

Dr. Adnan Shaout is a full professor in the Electrical and Computer Engineering Department at the University of Michigan – Dearborn. At present, he teaches courses in fuzzy logic and engineering applications and computer engineering (hardware and software). His current research is in applications of fuzzy set theory, embedded systems, software engineering, artificial intelligence and cloud computing. Dr. Shaout has more than 30 years of experience in teaching and conducting research in the electrical and computer engineering fields at Syracuse University and the University of Michigan - Dearborn. Dr. Shaout has published over 140 papers in topics related to electrical and computer engineering fields. Dr. Shaout has obtained his B.Sc., M.S. and Ph.D. in Computer Engineering from Syracuse University, Syracuse, NY, USA, in 1982, 1983, 1987, respectively.

Dominic Colella is a graduate student in the College of Engineering and Computer Science at the University of Michigan – Dearborn.