DESIGN OF AN INTERLINGUA FOR DATA DISPLAY SYSTEMS

Burak Meric, Michael Graul, and Ronald Fernandes
Knowledge Based Systems, Inc.
College Station, TX 77840.
Email: {bmeric, mgraul, rfernandes}@kbsi.com

Charles H. Jones
412 TW/ENTI
Edwards AFB, CA 93524-8300

ABSTRACT

This paper presents the description of a new XML-based data display language called Data Display Markup Language (DDML) that can be used as an interlingua for different data display configuration formats. Translation of data display configuration between various vendor- formats can be accomplished by translating in and out of DDML. The DDML can also be used as a vendor-neutral format for archiving and retrieving display configurations in a test and evaluation (T&E) configuration repository.

KEYWORDS

Data Display Configuration, Neutral Format, XML, Automated Translators

INTRODUCTION

Data display systems are critical components of a T&E environment. They need to be quickly assembled, programmed, and tested for both real-time and post-test analysis. Data displays have a wide range of parameters, attributes, dynamics, and data sources. The T&E systems, at different locations, need to transfer and share telemetry data among each other. In the traditional approach, each of these systems would apply its unique display setups; each data display system itself being quite complex. To compound this situation, there are a variety of data display vendors, each requiring its own data display specification. The time required to set up and check these data displays is significant. Currently, the only way to transfer data displays between display applications is to manually recreate the displays. Also, a change in one of the displays

requires manual changes in the other related displays. Thus, the absence of automated translators cause delays in test programs because of these manual setup procedures.

Even if the display setup process were to use a set of automated translators, the absence of a neutral format would require a total of $n(n\text{-}1)$ translators to be built, where n is the number of data display systems. On the other hand, the use of a neutral format would require two unidirectional translators between the neutral format and each vendor-specific format, or $2n$ translators. For example, the availability of a neutral format in a T&E environment of six data display systems require only 12 unidirectional bridges as compared to 30, if a neutral format was not available, as shown in Figure 1. An additional advantage of having a neutral format is that a change in one of the vendor formats requires the recoding of only the translators between that format and the neutral format.

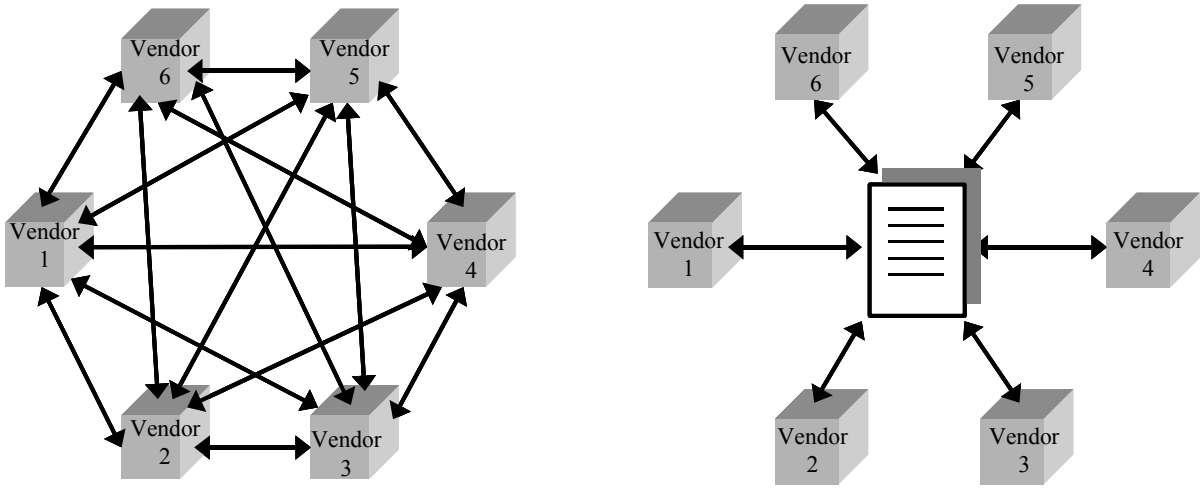

Figure 1: Code Development Effort for Translators With and Without a Neutral Format

Our vision of automating the translation of the data display configuration involves two major steps. The first is to define, validate, and verify the data display neutral format. Because XML is a widely accepted language specification format that provides the infrastructure to define domain-specific self-defining tags [1], the data display neutral format has been defined in XML, and it is called DDML. The DDML is to data display models what the TeleMetry Attributes Transfer Standard (TMATS) is to telemetry data [2]. The second step is to develop a framework under which various translators between the vendor formats and DDML can be integrated. We call this translator software suite the Data Display Translator Framework (DDTF).

While DDML and DDTF have applicability in generic T&E environments, they are especially beneficial for T&E interoperability in joint service programs such as Joint Strike Fighter (JSF) and Joint Air-to-Surface Standoff Missile (JASSM), where common data displays are required for flight-testing, regardless of the testing location or the data acquisition and display system. First, data displays can be specified in DDML to be a part of a well-defined test procedure. By specifying graphical displays and their dynamics in a neutral format, T&E procedures can be defined independently of software implementation. Second, current models in an existing vendor's data display system, e.g., Instrumentation, Loading, Integration, Analysis, and Decommutation (ILIAD) [3], can be saved in DDML by using the appropriate translator. This

translated DDML model can be made part of the T&E standard procedure. Finally, translators from DDML to any (other) vendor's display system can be used for real-time test and post-test T&E analysis. Thus, DDML along with the translators is beneficial in obtaining platform independence and hence, in facilitating the standardization of T&E procedures.

DATA DISPLAY MARKUP LANGUAGE

The DDML has been developed to be generic enough to encompass various vendor-specific data display formats and at the same time be unified (not a loose grouping of XML-ized vendor formats). In addition, it supports reusable concepts such as data variables and data sources within the same display model; is robust (e.g., use of cross references); and supports future objects without warranting a change of the core DDML format. The DDML is defined in terms of four layers: graphics components, dynamics, variables, and data sources; and closely parallels a typical software layered architecture composed of graphics resources, visualization and user interfaces, information management and persistence, respectively, as shown in Figure 2.

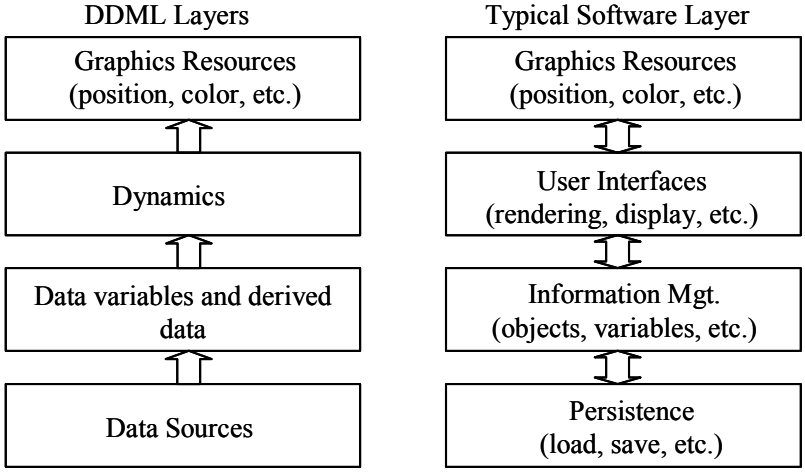

Figure 2: DDML Layers and their Comparison with Generic Layered Software Structure

In DDML, the graphics resources layer includes the visual components of a data display system such as sliders, plots, strip charts as well as low level graphic elements such as lines, rectangles, etc. Basic graphical shapes are modeled using W3C's recommended specification for a 2-dimensional (2-D) vector called Scalable Vector Graphics (SVG) [4]. High-level objects such as plots or sliders can also be represented as a composition of basic shapes using SVG. In DDML, SVG elements are differentiated from core DDML tags by the 'SVG' namespace.

The dynamics layer handles the behavior of an object. It manages the rules and the variable instances attached to an object. The Data Variables layer forms the links between the objects and the data sources. Data variables can be atomic or derived. Derived variables may use other derived or atomic variables in a mathematical expression. Finally, the Data Sources layer handles various data sources such as text files, ODBC, network ports, and ports on DAQ cards.

The DDML layered structure is intended to be a generic structure for various data display tools, even though different data display tools have different ways of specifying the graphics resources, dynamics, variables, and data source layers. For example, in Symvionics' Interactive Analysis and Display System (IADS) [5], the graphics resources and dynamics are tightly integrated by embedding the dynamics rules into the graphical objects. These rules are not exposed in the IADS configuration file. Also, the data sources are not exposed in the configuration file, but they are handled intelligently by the system. On the other hand, ILIAD has a different layered structure. In ILIAD, data formats and data sources are handled by the core ILIAD software itself whereas the graphics and dynamics parts are handled by embedding third-party data display software such as Sherrill-Lubinski's Graphical Modeling System (SL-GMS) [6], Quinn Curtis' Real-Time Graphics, or National Instruments' LabVIEW.

Because of these differences in vendor implementations, DDML has been designed to be generic enough to handle different data display architectures. At the same time, it is not a standard that loosely connects disparate islands of architectural frameworks under the cover of XML. In order to design DDML, a data dictionary was first developed in order to list all possible elements of various flavors of data display systems as well as relations between these elements. From this, logical concepts and relationships were extracted and were syntactically represented using XML. In addition, specific constructs were provided in DDML to allow the encoding of vendor-specific data in a generic way. For example, if a data display format uses very specific characteristics of its graphical resources or dynamics, these can be easily represented using generic "param" XML tags of DDML that allow custom data to be specified in DDML without changing the DDML schema.

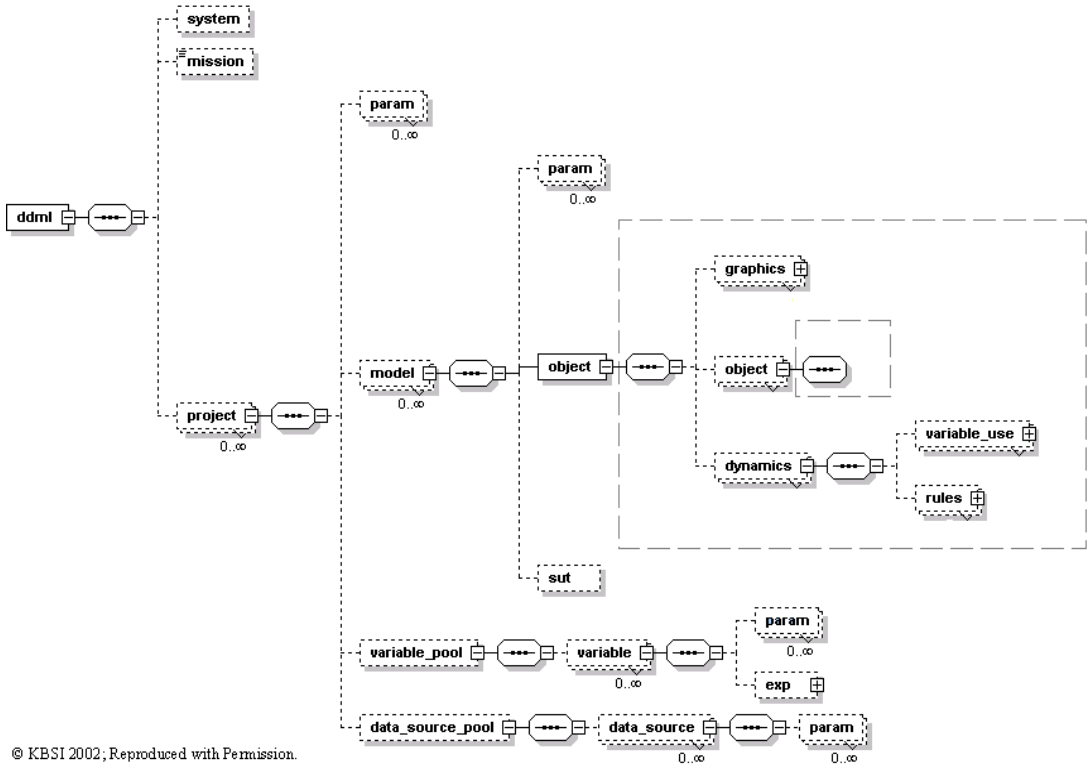

Figure 3: DDML Tree View

The DDML hierarchical structure is shown in figure 3. As shown in the figure, an object element can have another object element, which provides the flexibility to group multiple objects recursively. The object element also has a graphics element and a dynamics element. The "graphics" element provides the ability to represent basic SVG elements as well as optional display parameters for the display object. The "dynamics" element provides the necessary tags for logical operators for building rules and using instances of data variables. There are two types of pool items to represent template objects—one for variables and one for data sources. The variable pool provides template variables to be used in the dynamics part. The variables in the variable pool can be either atomic variables connected to data sources or derived variables composed of atomic or other derived variables. The DDML data model using the IDEF1X entity-relationship diagram is shown in figure 4. This model depicts the relationship between various DDML elements and displays their attributes.

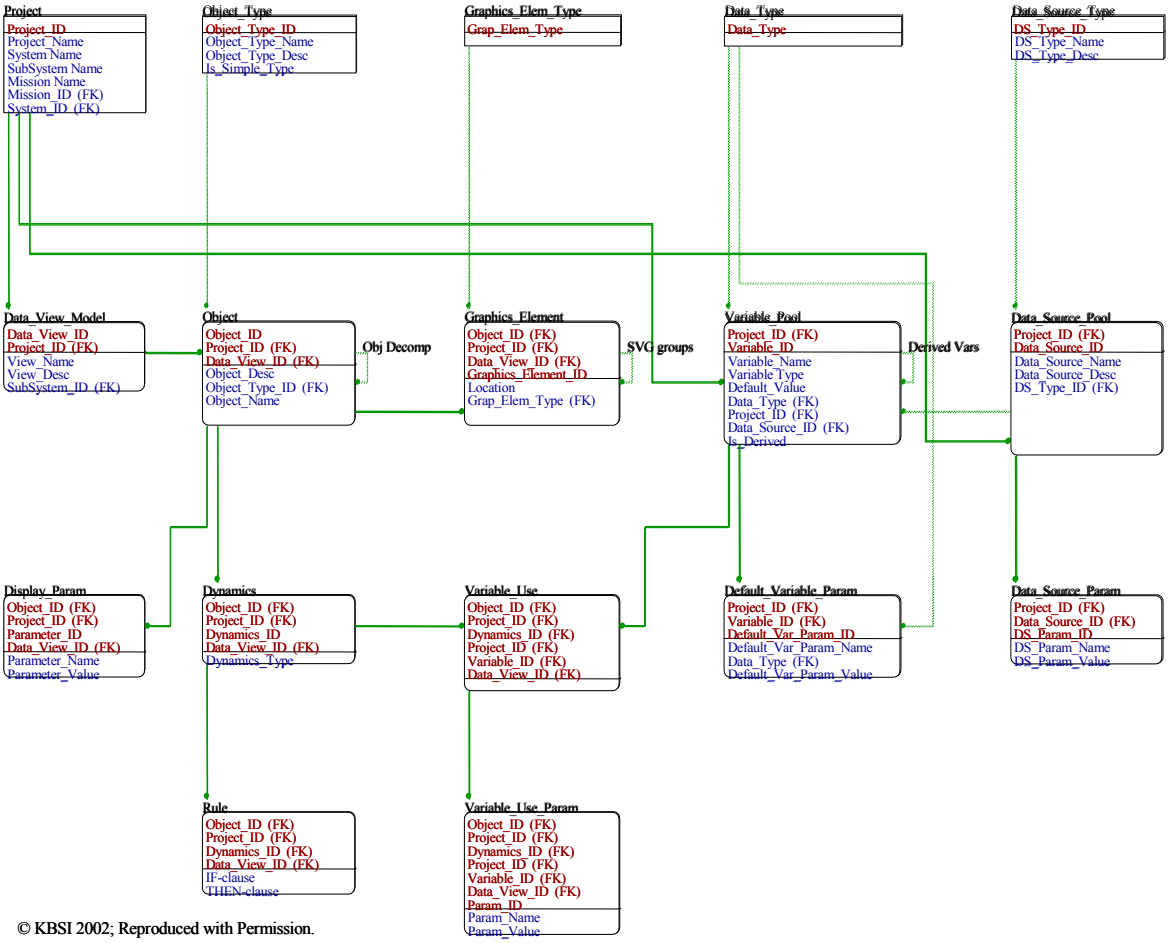

Figure 4: The IDEF1X Data Model for DDML

The DDML also provides the necessary tags for encoding mathematical expressions for derived variables. The semantics of the encoding is that the operator 'oper' is first applied to the variable 'var_ref,' then the power is applied and finally the result is multiplied with the coefficient 'coef.' Following are two examples showing how to represent mathematical expressions in DDML.

Example: $3x^2 - 2(\sin y)^2$

```
<exp
  <exp coef="3" power="2" var_ref="x" />
  <exp coef="-2" power="2" oper=SIN var_ref="y" />
</exp>
```

Example: $4x^2 - 5(\sin \sqrt{(y^2 + z^2)}$

```
<exp
  <exp coef = "4" power = "2" var_ref = "x" />
  <exp coef = "-5" power="1" oper=SIN>
    <exp coef = "1" power = "0.5" >
      <exp coef="1" power="2" var_ref="y" />
      <exp coef="1" power="2" var_ref="z" />
    </exp>
  </exp>
</exp>
```

The ability to support user-defined object types and parameters is crucial to DDML specification. It allows the adding of nonstandard, vendor-specific graphical objects and attributes or custom elements and data without changing the DDML schema. An example of a generic object tag is:

<object name="MyMeter" type="Mike's custom meter"> ,

and an example of a user-defined parameter for an object is:

<param name="Mike's custom attribute" value="some value" type="string">.

DATA DISPLAY TRANSLATION FRAMEWORK

To test the versatility of DDML, three bi-directional bridges between DDML and data display vendor formats were developed, and they were integrated into a common translation framework called DDTF. These applications are ILIAD/SL-GMS, IADS, and DataViews [7]. Figure 5 shows the main panel that depicts the hub and spoke concept. The DDML is the hub language in the middle, and the various target languages surround it. To translate a data display configuration between two vendor languages, the user proceeds from the first language to DDML and then from DDML to the second language. The status and any errors are displayed in the appropriate boxes at the bottom. On the DDML side, the translators use the Document Object Model (DOM) Level 2 application program interface (API) to parse and generate documents.

For each target vendor language, a map file can be used to customize the translation between DDML and the target language. Map files can be edited using a DDTF Map File Editor shown in figure 6, which is accessible from the main DDTF panel shown in figure 5. The DDML elements and attributes on the left tree view are mapped to those of the target language on the right. The map file can be customized for each end-user; this provides flexibility for customized translation of data display configurations.

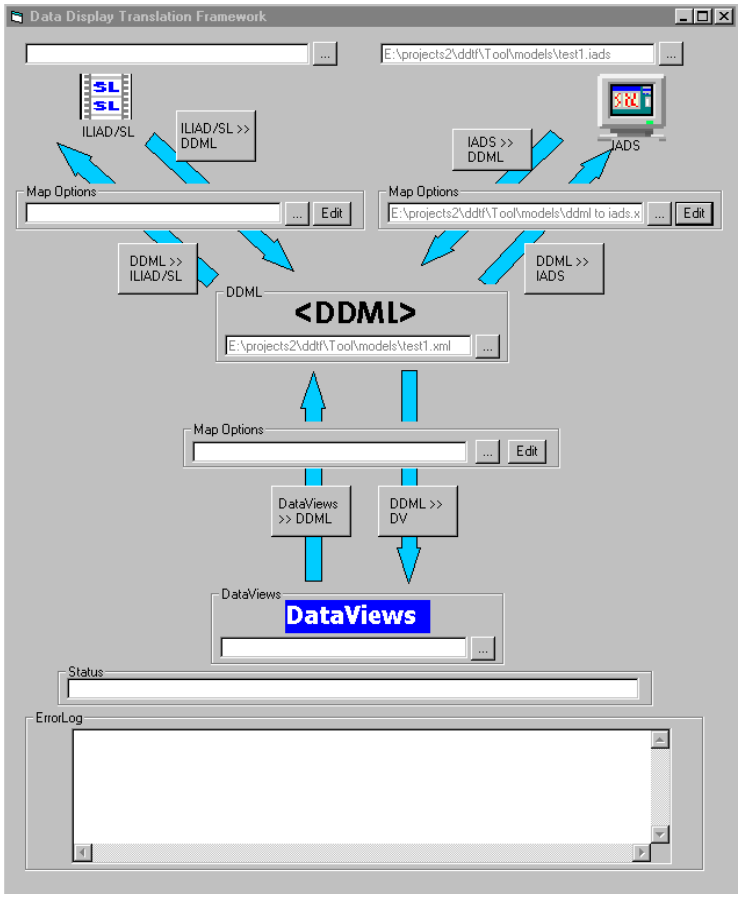

Figure 5: DDTF Main Panel

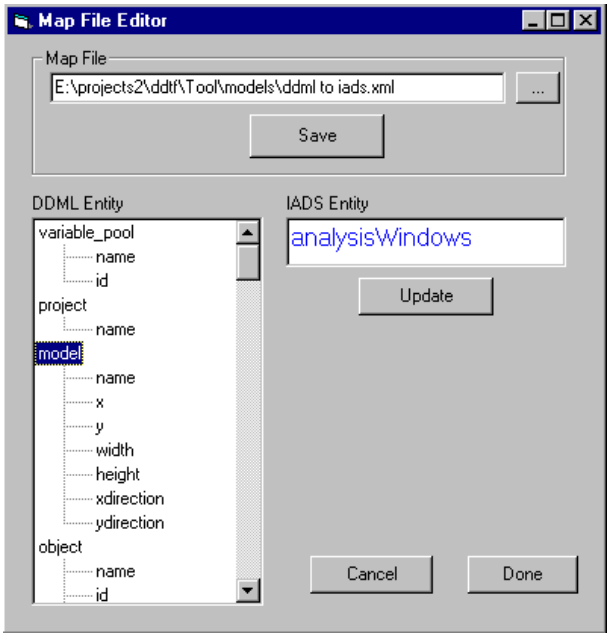

Figure 6: Map File Editor

CONCLUSIONS

The DDML has proven to be an effective neutral format for developing translators among different data display vendor formats. The re-use of existing display system configurations facilitated by DDML and DDTF has proven to significantly reduce time and effort in carrying out various data display functions in T&E, command/control, and process control environments. The DDML and DDTF can play an important role in the standardization of test procedures for joint service programs such as JSF and JASSM, where common data displays are required for flight-testing, regardless of testing location, data acquisition, and display system.

The role of standards in facilitating interoperability and knowledge sharing is well documented. For example, the National Institute of Standards and Technology (NIST) Process Specification Language (PSL) standard [8] facilitates seamless information transfer between various process-centric tools. In the data display world, DDML can play a similar role in the interchange of data display configurations.

REFERENCES

[1] Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation, http://www.w3.org/TR/REC-xml, October 2000.

[2] Telemetry Attributes Transfer Standard, IRIG Standard 106-96, Chapter 9, http://jcs.mil/RCC/manuals/tmstd/chap-9.htm.

[3] The Instrumentation Loading, Integration, Analysis and Display (ILIAD) Toolset, http://www.goiliad.com.

[4] Scalable Vector Graphics (SVG) 1.1 Specification, http://www.w3.org/TR/SVG11/.

[5] Symvionics' Interactive Analysis and Display System (IADS™), http://www.symvionics.com/products/iadshome.htm/.

[6] Sherrill-Lubinski's Graphical Modeling System, http://www.sl.com/sl_gms_solutions.html.

[7] GE Fanuc Automation's DataViews, http://www.gefanuc.com/dataviews/.

[8] The NIST Process Specification Language, http://ats.nist.gov/psl/.