

OneSAF: Management, Normalization, and Verification and Validation of Parametric Data: A Flat File Data Solution

Jeffrey Ketts Jr.
Innovative Management Concepts, Inc.
Orlando, FL 34787
jketts@imcva.com

Boaventura DaCosta
Dynamics Research Corporation
Orlando, FL 32817
bdacosta@drc.com

ABSTRACT

The One Semi-Automated Forces (OneSAF) Objective System (OOS) is a next-generation Computer Generated Forces (CGF) simulation that is intended to represent a full range of operations, systems, and control processes from individual combatant level and platform level to fully automated BLUFOR battalion level and fully automated OPFOR brigade level. Unlike past systems, OneSAF has had the benefit to be built from the ground-up utilizing eXtensible Markup Language (XML) as the means to define, store, share, and interchange data. Most OneSAF data is stored as XML, including parametric and initialization data. Given the magnitude of the data, and the simple fact that XML data is stored as text, some of the biggest challenges facing OneSAF have been the data normalization, verification, validation, and management of XML documents.

This paper describes the methods of managing flat files during development in such a way as to maintain the flexibility of a truly data centric simulation without introducing fragility into the environment. For the purpose of this paper, the term flat files refers to files that contain data in a tabular or text based format like a text file, a spreadsheet or a comma delimited file. The process of taking data from this flat file format to XML, then into the final form as parametric data, is detailed including transformation, data normalization, and verification and validation. The use of eXtensible Style Sheets (XSL), XQuery, and XPath is discussed, as well as the incorporation of Perl scripting. In particular, how these technologies can be used to manage flat files in a manner similar to an RDBMS. To facilitate a better understanding of OneSAF, background knowledge is given, providing a brief overview of the OneSAF Data Architecture with emphasis as to how data is read, written, stored, and currently managed. Lastly, the introduction of these methods and technologies into the current OneSAF Data Architecture is discussed, outlining benefits and drawbacks.

ABOUT THE AUTHORS

Jeffrey Ketts is a Senior Data Engineer for Innovative Management Concepts working as a Team lead on the One Semi-Automated Forces Objective System, Knowledge Acquisition/Knowledge Engineering Tools Team. The KA/KE Tools Team is responsible for managing the repository of parametric data used in OOS including data created by the OOS KA/KE Team, submitted from AMSAA, and re-used from other programs. Mr. Ketts has recently completed the requirements for the Training Simulation Graduate Certificate program at the University of Central Florida. He holds a Bachelors Degree in Business Administration, Management Information Systems from California State University, Northridge. Mr. Ketts' has over 10 years of multi-disciplinary experience in data engineering and data management.

Boaventura DaCosta is a Senior Software Engineer with Dynamics Research Corporation in Orlando, FL. He is currently supporting work for the Future Combat Systems (FCS) embedded training effort. He is the chief architect for an internal database to archive FCS task data that will be used for developing Collective Training Support Packages (TSPs) for the FCS equipped Unit of Action. He received an M.A. in Instructional Systems and B.S in Computer Science from the University of Central Florida.

OneSAF: Management, Normalization, and Verification and Validation of Parametric Data: A Flat File Data Solution

Jeffrey Ketts Jr.
Innovative Management Concepts, Inc.
Orlando, FL 34787
jketts@imcva.com

Boaventura DaCosta
Dynamics Research Corporation
Orlando, FL 32817
bdacosta@drc.com

INTRODUCTION

eXtensible Markup Language (XML) is no stranger to modeling and simulation. Released as a recommendation by the World Wide Web Consortium (W3C) in 1998, XML has become a commonly used vehicle to structure data both within applications and via the Web. Countless papers have been published on its utilization, particularly in the area of data interchange. Some systems have already adopted the use of XML to some capacity. In some instances incorporating XML into their existing architecture to represent behaviors and other simulation-specific data. In other instances, XML has been seamlessly integrated into the architecture from the ground-up as the means to define, store, share, and interchange data, as in the case with the One Semi-Automated Forces (OneSAF) Objective System (OOS).

The entry-point of XML into many of the systems developed to date has been different. More commonly, however, are the lessons learned from using XML. Most OneSAF data is stored as XML, including parametric and initialization data. Given the magnitude of the data, and the simple fact that XML data is stored as text, some of the biggest challenges facing OneSAF and potentially other simulation programs have been the data normalization, verification and validation, and management of XML documents.

This paper describes the methods of managing flat files during development in such a way as to maintain the flexibility of a truly data centric simulation without introducing fragility into the environment. The process of taking data from source format to XML, then into the final form as parametric data, is detailed including transformation, data normalization and verification and validation. The use of eXtensible Style Sheets (XSL), XQuery, and XPath is discussed, as well as the incorporation of Perl scripting. In particular, how these technologies can be used to manage flat files in a manner similar to a Database Management System (DBMS). To facilitate a better understanding of OneSAF, background knowledge is given, providing a

brief overview of the OneSAF Data Architecture with emphasis as to how data is read, written, stored, and currently managed. Lastly, the introduction of these methods and technologies into the current OneSAF Data Architecture is discussed, outlining benefits and drawbacks.

A BRIEF LOOK AT THE CURRENT DATA ARCHITECTURE

The OneSAF Data Architecture has been implemented as a Repositories Framework, providing System Repository Services (SRS) to access all data and meta-data stored across the OneSAF Data Repositories represented as file-based groupings of similar, logical areas. Think of the SRS as an Application Programming Interface (API), providing a consistent, uniform vehicle with which to access, create, update, delete, and manage data across all of OneSAF. Even though the SRS provides XML-centric services to access any and all data represented as XML, additional format-independent services are also provided allowing non-XML represented data to be accessed as well. As for the XML-centric services, both domain-neutral core and value-added data services are provided. Core services are just that, fundamental services providing primitive access to the data, such as the ability to read, write, and copy data. Value-added services, on the other hand, are coupled with specific types of data, such as in the access of OneSAF Simulation Scenario data (OneSAF Repository Component Layer 2003). All OneSAF products, tools, and utilities utilize the SRS to work with data and meta-data housed within the system and it is the only valid means in which to access the data.

Most of the data is represented as XML. This data is stored in flat files, as opposed to being stored in models or other binary representations and is organized on disk in logical groups making up the seven OneSAF Repositories. These repositories are the Software Repository (SWR), Knowledge Acquisition/Knowledge Engineering (KA/KE) Repository, System Composition Repository (SCR), Military Scenario Repository (MSR), Environment Repository (ER), Parametric and

Initialization Repository (PAIR), and the Simulation Output Repository (SOR). All seven repositories serve different purposes and are considered data repositories, except for the SWR and the KA/KE Repositories, which are primarily used for the development of OneSAF and are not accessible via the SRS. The remaining repositories strictly house data used by the OOS and are only accessible via the SRS (DaCosta 2003).

MANAGING FLAT FILES TO MAINTAIN FLEXIBILITY

The processes and procedures described throughout the remainder of this paper have been developed in order to manage data during development. The tools described here have matured as the processes and procedures have matured. These processes, procedures, and tools could be used as easily during fielding as they are to manage data during development.

OneSAF is implemented as a Product Line Architecture, meaning, it is composed of a set of products as opposed to a single system or application. Each product is further decomposed into components. These components consist of composers, tools, utilities, and editors, which are tightly coupled with the specific data they interact with (Courtemanche 2002, Rieger 2002, Wittman 2001). This parametric and initialization data, stored as XML documents, is clearly structured with grammars defined through XML Schemas. This data has been “cooked” during development through transformations, or created by using the appropriate component and is ready for use by the system.

This “cooking” process involves the storage, management, massaging, and transformation of source data into producer side data that eventually is transformed into parametric data. This source data is “raw” data which is received from multiple data producers (groups, organizations, and individuals) in multiple native file formats. These files, therefore, must be managed as to not introduce fragility into the environment. Why is this so important? Why must the data be kept organized, synchronized, and redundancy, and conflicting data identified and eliminated?

Let us consider the example of the ammunition, 7.62mm cartridge, called a “cartridge762”. The ammunition can be used in several different weapon systems like the AK47 or an SKS. Using this ammunition with different weapons or mounts creates different inputs to the models. These input combinations have corresponding data in other tables that show target accuracy, vulnerability, probability of kill value and others that

will create different results and output from the simulation.

Given this example, if a new type of high velocity 7.62 mm cartridge (cartridge762HV) were invented for all systems that used the previous ammunition, the data would need to be changed. So, how does this happen utilizing a file-based management system?

The subsequent sections outline the process of “cooking” source data. This includes normalizing the data, the transformations involved, and validating and verifying the data so that it can be approved for the finalized transformation into parametric data. The technologies involved are discussed and how they are used to manage the data to aid in keeping the information organized, easily accessible, synchronized and in the process helping in the elimination of redundancy and conflicts. In essence, how can a level of data integrity across a flat file system be achieved, but at the same time keep flexibility incorporated into the process.

An Overview of the Data Management Process

The process depicted in Figure 1 displays the flow of source data to producer side XML, then into the finalized form as parametric data reposed to the PAIR for use by the system models.

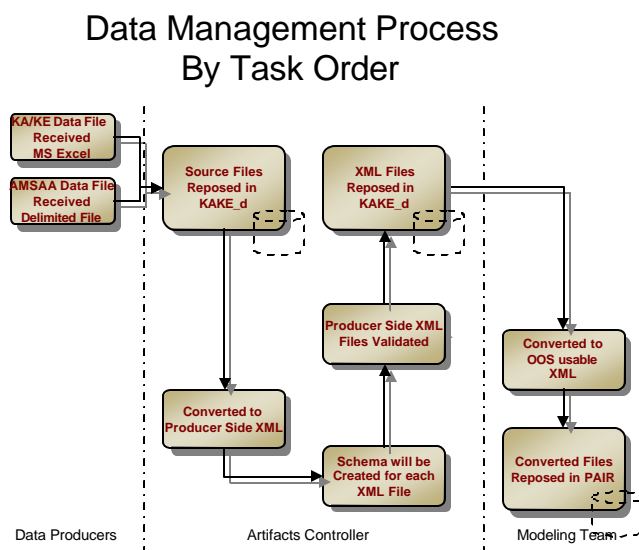
The process begins with the arrival of data from the data producers. Source data is currently received in three formats. KA/KE Data is received in Excel spreadsheets. AMSAA (U.S. Army Material Systems Analysis Activity) data is received in delimited (colon or comma) text files (as a .dat, .txt) or as Individual Unit Action (IUA) tables, while the third format is received as XML. XML data may be legacy data acquired from WARSIM or CCTT. This artifact data is delivered to the Artifacts Controller (the KA/KE Tools Team). The data itself can be delivered directly in the KA/KE Repository via CVS (Concurrent Versioning System) by the data producers themselves, by disk, or email.

Once archived into the KA/KE Repository, the data files are checked manually for data completeness and structure. This manual check is performed to catch any discrepancies or problems with the structure before the transformation process begins. If any of the data is missing upon this check, or the structure incorrect, the data is rejected by the Artifacts Controller and returned to the data producer. The data can be later re-submitted once the noted problems have been resolved. In addition, all data producers must submit data definitions prior to, or with the data submitted. These

data definitions clearly define the structure of the data, allowing for validation later in the process. It also aids the Artifacts Controller in manually checking the data upon delivery to ensure the structure of the data is in fact correct.

Data files passing this check are then transformed into an intermittent state. (See Producer Side XML in Figure 1.) This data, now stored as XML documents are validated against XML Schemas. These XML Schemas are generated from the data definitions supplied by the

data producers. This validation checks the documents for structure, grammar, and specific key fields against enumerations stored in the system. If the validation is successful, the producer side XML documents and accompanying generated XML Schemas are stored in another part of the repository to undergo the next phase. If validation checks fail, the data files are returned to the data producer. The data can be later re-submitted once the noted problems have been resolved. The process then begins all over again.



**Figure 1: OneSAF: The Data Management Process by Task Order
(KA/KE Tools Data Management Process 2003)**

Once validation is successful, a completeness check is performed. This check ensures that all required data is in fact present. This ensures that there is a complete picture of the system (platform, weapon, etc.) once the data is transformed into the final “cooked” formats. Data files passing this completeness check are ready for the final phase of transformation. If the completeness check fails, the data files are returned to the data producer. The data can be later re-submitted once the noted problems have been resolved and the process then begins all over again.

At this point, the producer side data files (XML documents) and coupled XML Schema (generated from the data definitions) are made available to the Models Development Team. Data files are analyzed for structure and performance by the modelers. If the producer side XML Schema is sufficient to meet the needs of the model, the XML documents are reposed to the PAIR “as-is”; however, in most cases, the structure is

modified to match a specific schema needed for the model. In other words, the data in question conforms to specific XML Schemas the system understands and can read during initialization of the simulation. This is typically done to optimize the data for use, in the process reducing redundancies and eliminating fields that are otherwise included for human readability only (Figure 3). Once transformed, this parametric data is reposed to the PAIR for use by the system.

Understanding the Data

With an overview of the entire process given, a discussion on the data itself may be helpful. Even though source and parametric data may have the same content, they look very different. There are several reasons for this. Let’s begin with the simplest to explain. When data is developed and undergoes a OneSAF Peer-Review process, there are fields included that are strictly used for the understanding and support

of the domain experts developing and validating the data. Often times these notes, references, and comments are not part of the data that the models will be using. For size and performance reasons these fields are not transferred into the finalized parametric forms.

The next consideration in regard to size and performance is the presentation of the data. Flat files are not conducive to nesting data. By this we mean if multiple lines of data are presented to represent a platform then some platform data may be repeated, such as the platform's name or identifier. This leads to wasted storage space due to the redundancy of data, not to mention the performance problems when having to query the information. This also lends itself to data

NORMALIZING, TRANSFORMING, AND VALIDATING

As we've discussed, data is normalized, transformed, and validated between the time it is delivered as source data and its final emergence as parametric data. So, how

maintenance complications, particularly if the same data has to be changed in multiple places. So, when transforming data, redundancies are eliminated.

The last consideration in the differences is the needs of the model developer. For example, the model developer might be more interested in focusing on the munitions rather than the platform. In such a case, the data would be organized differently in the final structure than if the emphasis was placed on the platform as opposed to the munitions. When creating a model, the developer will create a sample data file and a supporting XML Schema. The final transformation to "cooked" form will build the appropriate XML documents specified by the coupled XML Schema.

is this actually done? In the following sections we'll be examining the normalization, transformation, and validation of the data. Figure 2 has been provided, which takes the OneSAF Data Management Process (Figure 1) and divides it into each of these phases.

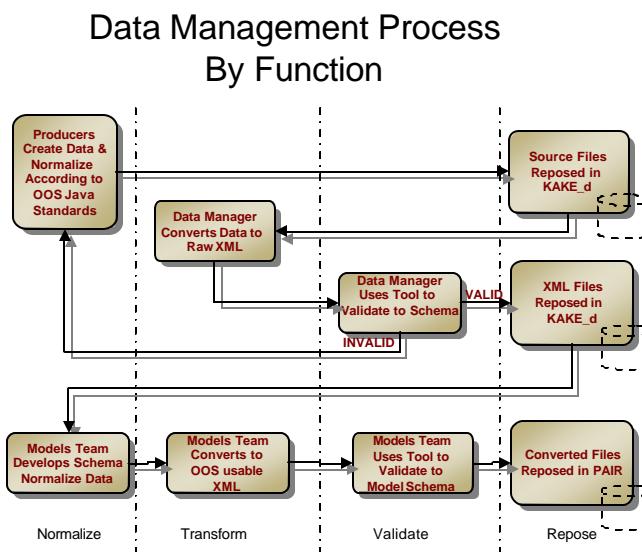


Figure 2: OneSAF: The Data Management Process by Function (KA/KE Tools Data Management Process 2003)

Normalizing the Data

Normalizing data is typically performed to streamline the data to avoid storing extraneous or redundant information. It involves examining and arranging the data to avoid problems when the data must be modified at a later date. OneSAF begins the data normalization process early in development. As shown in Figure 2, data normalization occurs at two distinct intervals

during the entire process. The responsibility for the first part of data normalization is levied on the data producer. In other words, normalization begins at the time the data is being developed.

OneSAF has established a set of rules for the development of data. These rules address a number of concerns. For example, the naming convention for keys as well as default values is clearly outlined. These rules

regarding the naming of key fields are based on the OOS Java Naming Convention (OneSAF Coding Standards 2005). Rules have also been imposed that dictate the use of null values within data. In order to ease the querying process, null values are not allowed within the data. To facilitate this, any field that is not used or is invalid will be assigned a value of "99999" for an integer or float or "NA" for a string data type. If data is not available for a field at the time of development "99999" is used for an integer or float and "TBD" is used for a string data type. OneSAF also uses other policies that regulate the structure of the data. One example is the OneSAF Common Enumeration Policy concerning the use of enumerated sets within data (OneSAF Common Enumeration Policy 2005).

Once data is received from the data producers it is manually inspected by the KA/KE Tools Team. The data is inspected for normalization errors, such as naming conventions, structure, and unit of measure discrepancies. Data failing inspection is returned to the data producer; however, the KA/KE Tools Team works with the data producer to correct any normalization errors. Data passing inspection is reposed to the KA/KE Repository to undergo transformation into producer side XML documents for validation.

Once the source data has been transformed into producer side XML documents and has passed validation, the data and coupled schemas are supplied to the Models Development Team where the second interval of normalization occurs as shown in Figure 2. The Models Development Team ensures that the data is normalized as needed by comparing the XML documents against the XML Schemas in question. If additional normalization is required, the XML Schemas defining the grammar and structure of the parametric data are modified. In other words, any transformations performed will not only transform the producer side data into parametric data, but at the same time will further normalize the data if necessary.

Figure 3 depicts a simple example as to how data is normalized from source data into producer side XML, then into the final form as parametric data. In the example, some of the source data stored in the file is duplicated, such as the platform and corresponding weapons. After the transformation into OOS useable XML, redundancy is removed. Each platform and corresponding weapon only appears once in the file.

Normalization concepts are applied to the schema design (XML Schemas). As an example XML is hierarchical and can be used to capture matrices and

reduce redundancy. It is important to understand that the rules of normalization as they apply to relational theory (1NF to 5NF) cannot be applied explicitly to XML Schemas. However, the rational or thinking behind these can. In essence, when the finalized transformations take place (namely from the producer side XML documents to the parametric XML documents), the following goals are sought:

- Redundancy is eliminated (or minimized)
- Ambiguity is avoided (when possible)
- Data remains consistent (such as the use of enumerations, units of measures, and so forth)

Transforming the Data

From the time of receipt the data will be transformed at two different intervals during the data management process (see Figure 2 with an example depicted in Figure 3) with two different purposes in mind. The first transformation occurs as the source data is transformed into producer side XML documents. This producer side XML is defined as data that is closely matched to the source document. Tags directly map to the fields from the source data. The files themselves are not sorted, combined, or massaged in any way. Instead, tools are utilized which convert the source data (AMSAA delimited files and KA/KE Excel spreadsheets) into tagged XML.

Source data is transformed into producer side XML for the purpose of validating the data. Automated tools are used to convert the source data into producer side XML documents. XML Schemas, derived from the data definitions, are also automatically created for validation purposes by these same automated tools. Once in an XML format, other automated tools are used to validate the data and checks are performed to ensure completeness.

The generation of the XML Schema is a complicated process outside the scope of this paper. To briefly explain; the schema for the producer side XML is delivered from the data developer and includes format, structure, and enumerations used to validate the XML. This is converted from the source form to an XML Schema (XSD). The XML Schemas for the OOS usable XML are manually developed by the model developer.

Source Data

	A	B	C	D
1	platformCommonName	weaponCommonName	mount	supplyCommonName
2	Combatant	machineGun762mm	bipod	cartridge762
3	T72_Tank	machineGun762mm	coax	cartridge762
4	T80_Tank	machineGun762mm	coax	cartridge762

Source Data is transformed into intermittent Producer Side XML using the Universal Translator

Producer Side XML

```

Sample_BDT.xml
18 <csvFile xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
19 xsi:noNamespaceSchemaLocation="Sample_BDT.xsd">
20 <row>
21 <platformCommonName>Combatant</platformCommonName>
22 <weaponCommonName>machineGun762mm</weaponCommonName>
23 <mount>bipod</mount>
24 <supplyCommonName>cartridge762mm</supplyCommonName>
25 </row>
26 <row>
27 <platformCommonName>T72_Tank</platformCommonName>
28 <weaponCommonName>machineGun762mm</weaponCommonName>
29 <mount>coax</mount>
30 <supplyCommonName>cartridge762mm</supplyCommonName>
31 </row>
32 <row>
33 <platformCommonName>T72_Tank</platformCommonName>
34 <weaponCommonName>machineGun762mm</weaponCommonName>
35 <mount>pintle</mount>
36 <supplyCommonName>cartridge762mm</supplyCommonName>
37 </row>
38 <row>
39 <platformCommonName>T80_Tank</platformCommonName>
40 <weaponCommonName>machineGun762mm</weaponCommonName>
41 <mount>coax</mount>
42 <supplyCommonName>cartridge762mm</supplyCommonName>
43 </row>
44 </csvFile>
    
```

Producers side XML is transformed into OOS useable XML using the XSL Translation Tool

OOS Useable XML

```

Sample_BDT_PAIR.xml *
18 <csvFile xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
19 xsi:noNamespaceSchemaLocation="Sample_BDT.xsd">
20 <platformCommonName value="Combatant">
21 <weaponCommonName value="machineGun762mm">
22 <mount value="bipod">
23 <supplyCommonName value="cartridge762mm">
24 </supplyCommonName>
25 </weaponCommonName>
26 </platformCommonName>
27 <platformCommonName value="T72_Tank">
28 <weaponCommonName value="machineGun762mm">
29 <mount value="coax">
30 <supplyCommonName value="cartridge762mm">
31 </supplyCommonName>
32 <mount value="pintle">
33 <supplyCommonName value="cartridge762mm">
34 </supplyCommonName>
35 </weaponCommonName>
36 </platformCommonName>
37 <platformCommonName value="T80_Tank">
38 <weaponCommonName value="machineGun762mm">
39 <mount value="coax">
40 <supplyCommonName value="cartridge762mm">
41 </supplyCommonName>
42 </weaponCommonName>
43 </platformCommonName>
44 </csvFile>
    
```

Figure 1: Data Transformation

The second transformation occurs once the producer side XML documents have been thoroughly checked and the contents validated against the appropriate XML Schemas. The source data files, producer side XML documents and any coupled XML Schemas are reposed to a designated area in KA/KE Repository for access by the model developers. The Models Development Team is notified (through automated notification processes) that the specific model data is ready for consumption. At this point, the Models Development Team decides whether the producer side structure will meet the needs of the models in question or if further transformations are necessary. Meaning, will the producer side XML documents and XML Schemas suffice, or will further transformations have to take place.

Validating and Verifying of the Data

There are several layers to the validation and verification of the data. In terms of data verification, none of the data is scrutinized for quality or content. The data that has come from authoritative data sources or other programs are considered to be "approved" for use and receives no scrutiny. The data that has been manufactured in house by the KA/KE Teams undergoes a peer-review process. These data table are also considered "approved" at the point that it is delivered into the KA/KE Repository for processing.

In terms of data validation, it occurs twice within the data management process as shown in Figure 2. The first time it is performed by the KA/KE Tools Team. Data is received from the data producers and placed in the repository where it is manually inspected. This visual inspection is initially performed to ensure that the data was received to structure as outlined by the data definitions. Checking the structure is a matter of ensuring that the data follows the organization and grammar outlined by the data definitions supplied with the data.

As indicated earlier, each data file is coupled with data definitions. These data definitions are nothing more than Data Interchange Formats (DIFs). They outline the structure and grammar of the data. The data types are also outlined in the DIFs as well as the definition of the column headers. These DIFs are compiled together making the KA/KE Master Data Definition Spreadsheet (MDDS). This is a data dictionary for all data. This artifact contains a list of all column headers and details, including the structure, grammar, definition, and links to the enumerated sets if applicable. There is also a reference to the units of measure used.

This data dictionary is used to create individual XML Schemas used in the automated validation of the producer side XML documents. Once transformed into producer side files, the data is checked for well-formedness and validity. Namely, the XML documents are validated against the XML Schemas utilizing commercially available tools such as Altova's XML Spy.

Once the validation and well-formedness checks are complete, the KA/KE Tools Team checks the files for completeness. These checks ensure that all represented enumerated sets which are required by the model are in fact present. This step is in place to guarantee a complete picture of the system (platform, weapon, etc.). For example, a check against the AMSAA delivery accuracy XML document would ensure that all the delivery accuracy data needed for all weapons on the master weapons list is in fact present. Any discrepancies would appear in the check. If this were the case, the data would be rejected by the Artifacts Controller and returned to the data producer. The data can be later re-submitted once the noted problems have been resolved and the entire processes would begin again.

This completeness check step is currently being automated. Tools are currently being developed which will automatically pinpoint such discrepancies. This includes tools which will allow the checking of the data to verify that all of the necessary information is available for a system to function correctly in OOS. XML documents passing this completeness check are tagged as "accepted" by the OneSAF program and are ready to be handed to the model developers for any final transformations.

Data is validated again once it is received by the Models Development Team. Once the producer side files are transformed into parametric XML documents, the data is again validated against the new XML Schemas defining the new "cooked" structure and grammar. This time the files are checked to verify that the keys used are included in the OOS Enumerations. Any references to subordinate data are also checked at this time.

THE USE OF CUSTOM TOOLS AND OPEN-SOURCE TECHNOLOGIES

A combination of custom developed tools and open-source technologies are used in the data management process described thus far. Tools are currently being developed which will use XQuery and XPath in the

checking of producer side XML documents for completeness. Tools are already in place, which use XSL to transform data into "cooked" data formats, while PERL scripts are used to speed up the process of transforming data. These tools and technologies are described here.

The Universal Translator

The Universal Translator (UT) (Figure 3) is a custom Java tool serving two primary functions. The first is the conversion of source data (delimited files and Excel spreadsheets) into producer side XML documents. The second is the creation of the XML Schemas (derived from the data definitions compiled into the KA/KE MDDS).

In the conversion of delimited data, the UT takes as parameters the location of the source data file, delimiter, and the location of the MDDS. The UT reads the column headers of the data sheet into an array. It then reads each row of the corresponding data as well, writing out the column headers as the tag markings encapsulate the data according to the MDDS.

In the conversion of Excel spreadsheets, the UT takes as parameters the location of the Excel file and the location of the KA/KE MDDS. In addition to building the XML document, the UT takes the sheet name and uses it as meta-data inserting a comment section within the XML document. This aids in the traceability of the source file to the resulting XML document.

Lastly, the UT has the capability to generate XML Schemas. Derived from the KA/KE MDDS, the UT can output an XML Schema that can be used to validate the resulting producer side XML documents for structure, grammar, and against enumerated sets. The UT is used in the data management process by the KA/KE Team to transform source data into producer side XML documents for validation.

The XSL Translation Tool and XSL

The XSL Translation Tool (Figure 3) is used to transform the producer side XML documents created by the UT into parametric XML documents used by appropriate models. Currently there is one style sheet for each producer side XML document transformation that must occur. The style sheets contain information as the location of the producer side XML document, the desired location of the outputted parametric XML document, and the XML Schema used. Since this step of the data conversion is so closely aligned with the

development of the models, the style sheets are developed by the Models Development Team.

Even though commercial tools exist, such as Altova's XML Spy, which can perform XSL transformations, a custom Java tools, similar to that of the UT has been developed for this purpose. The XSL Translation Tool takes as parameters the location of the producer side XML document, the location of the style sheet, and the desired output location of the parametric XML document. The XSL Translation Tool is used in the data management process by the Models Development Team to transform producer side XML documents created with the UT by the KA/KE Team into parametric XML documents for use by the system.

XQuery and XPath

XQuery and XPath are currently being integrated into tools which will automate checking producer side XML documents for completeness. These tools are being designed to query the entire data sets to check for completeness of key elements. Specifically they will verify that all files that refer to a platform, weapon or a munition represent the entire set of each. The current validation only checks to see that each data file represents the key elements as they are stated in the enumerations. These tools are being designed to check for completeness of these data files. These tools are expected to be available for use at the time of this paper's publication.

Perl Scripts

Depending on the transformation in question, the use of XSL can be slow at best. To speed up this process, the Models Development Team has developed a set of PERL scripts. These scripts are used to batch transformations. Meaning, large groupings of producer side XML documents can be transformed together in the process replacing all of the data or logical groupings of data at one time. This has become especially helpful when all the data for a given system (platform, weapon, etc.) must be replaced at one time within the PAIR thus avoid multiple XML documents having to be transformed one by one.

Another use of PERL scripts is in the instance of known sets of data received from a legacy program or a data provider that uses a key other than the OOS BSO (Battle Space Object). (These are key element for the simulation. This is the key by which all things refer to platforms.) A mapping is created between the OOS BSO and the new key. The PERL script is used to run

through the entire set of data replacing the new key with the correct OOS BSO key understood by the system.

THE DATA MANAGEMENT PROCESS WITHIN THE CURRENT DATA ARCHITECTURE

The data management process described in this paper is already in use within the current OneSAF Data Architecture. This process hasn't required changes to the Repository Framework or SRS. The KA/KE Repository and the PAIR have changed slightly in the manner in which data is stored. In the case of the KA/KE Repository, this has been expected considering that all source data files and producer side XML documents are stored within the repository. As for the PAIR, the structure has changed as needed by the models which have been developed.

The development and maturing of this process has led to the creation of tools, however. These tools were not originally planned as part of the OOS fielded system. However, their value to an end user has led to the hardening of these tools and their inclusion in the OOS baseline. This includes the XSL Transformation Tool and the UT. Once available, the XPath and XQuery based tools are also expected to be included.

Another change has been the addition of a source data repository to the OOS baseline. This was a direct result of the data management process. Originally conceptualized that this data would only be available through OneSAF.net, the development of the scripts and tools has shown the benefits the end user would possess in the ability to change data in bulk.

IN SUMMARY

First, with this process, end users can not only trace source data to the actual data used directly by the OOS, but they have the ability to change the source data, follow the process and in the end alter the parametric data they use. The data management process has helped OneSAF in identifying data discrepancies early in the process, avoiding costly problems with delivering potentially bad data to the Models Development Team. In addition, end users also have the ability to use the same checks to ensure the data is valid and complete.

This paper has outlined an approach to the management of data across a flat file system while at the same time trying to introduce a level of data integrity. This data management approach has changed over time, evolving as new requirements, constraints, and

data have been introduced. The result is a process, which allows “raw” data to be “cooked” into forms useable by the OOS fielded system. At the same time, it allows for the data to be checked to ensure it will meet its intended use. As OneSAF matures and new data requirements are introduced, so will this data management process.

REFERENCES

- Courtemanche, A. J., & Wittman, R. L. Jr. (2002) OneSAF: A Product Line Approach for a Next Generation CGF. *11th Computer Generated Forces and Behavioral Representation Conference, Orlando, FL, May 2002.*
- DaCosta, B., Lucas T., Outar R., & Helton, D. (2002) OneSAF Repository Framework: Defining, Storing, and Interchanging XML Data. Spring Simulation Interoperability Workshop, Orlando, FL, March 2003.
- OneSAF KA/KE Tools Task Order (2003). *KA/KE Tools Data Management Process*. Retrieved May 17, 2005, from OneSAF Web site: https://www.onesaf.net/KA-KE_Tools/ProgramMgmt/2003_02_03_OneSAF_KATools_Data_Management_Process.ppt
- OneSAF Architecture and Integration (A&I) Task Order. (2003). *OneSAF Coding Standards*. Retrieved May 17, 2005, from OneSAF Web site: https://www.onesaf.net/Architecture_and_Integration/SoftwareEngineering/Standards/OneSAF_Coding_Standards.2.htm#NamingConventions
- OneSAF Architecture and Integration (A&I) Task Order. (2003). *OneSAF Common Enumeration Policy*. Retrieved May 17, 2005, from OneSAF Web site: https://www.onesaf.net/Architecture_and_Integration/SoftwareEngineering/SDF/Policies/CommonEnum/CommonEnumPolicy.html
- OneSAF Architecture and Integration (A&I) Task Order. (2003). *OneSAF Repository Component Layer*. Retrieved May 24, 2003, from OneSAF Web site: https://www.onesaf.net/Architecture_and_Integration/SystemsEngineering/SystemArchitecture/PLAF/EP/LAF/RepositoryLayer/repository_component_layer.html