ARMY DATA SERVICES LAYER (ADSL) – DATA MEDIATION SERVICE (DMS)

Anthony Petosa
U.S. Army Communications-Electronics Command Life Cycle Management Command (CECOM LCMC)
Software Engineering Center (SEC)
Ft. Monmouth, NJ 07703

Narsim Ganti
Data Intelligence – LLC
Marlton, NJ 08053

William DeMasi
Atlantic Consulting Services, Inc.
Shrewsbury, NJ 07702

ABSTRACT

Data architectures in legacy systems employed proprietary solutions with tightly coupled software components. Point-to-point solutions enabled exchange between systems, but these solutions were costly to maintain. The current migration toward enterprise-wide, service-oriented architectures calls for the elimination of these “stovepipe” solutions and promotes the use of accepted standards and protocols for data exchange. The Data Mediation Service adopts these standards and protocols, and it employs semantic technology to increase its value in providing data mediation services.

1. INTRODUCTION

“The mission of the U.S. Army Communications-Electronics Life Cycle Management Command (CECOM LCMC) Software Engineering Center (SEC) Army Net-Centric Data Strategy (ANCDS) Center of Excellence (CoE), as chartered by the Army Office of the Chief Information Officer (CIO/G6), is to facilitate the execution of the Army’s Net Centric Data Strategy and provide users with common and overarching data products and services to promote interoperability and faster access, retrieval, analysis and utilization of data.”

The ANCDS CoE is developing the Army Data Services Layer (ADSL), which is a virtual data tier that addresses data in a SOA environment. The Data Utilization layer builds composable applications from “atomic” data services. The Data Abstraction layers captures and manages metadata at the structural and semantic levels. The Data Access layer exposes interfaces to search, retrieve and manipulate data. The Data Management layer manages data “at rest” and provides persistence. Lastly, the Data Mediation layer makes data understandable and usable by bridging the gap between various data formats, vocabularies and semantics. (Dirner, et al., 2006) The paper focuses on this layer.

2. DATA MEDIATION SERVICE

The Data Mediation Service (DMS) is a web service that will integrate with the Army Enterprise SOA architecture. It receives a Simple Object Access Protocol (SOAP) message carrying an eXtensible Markup Language (XML) message payload that validates against an XML Schema. The DMS outputs an eXtensible Stylesheet Language Transformation (XSLT) document for a service consumer to effect an XML message transformation. Optionally, the DMS performs the transformation and outputs an XML message conforming to the target domain’s XML Schema.

2.1 The Problem with XSLT

An XSLT restructures an XML document into another one or into some other document format (eg, HTML, XHTML). XSLT template rules instruct an XSLT processor to locate XML elements and output them in some other form. If the target output is an XML document, then it must be well-formed XML. If the source & target XML documents validate against XML Schemas, then they are XML instance documents.

The DMS will produce an XSLT document for some target XML structural form. As a web service, it fulfills a consumer request only if the source & target XML messages share concepts. Humans ascertain if vocabulary terms across domains convey the same meaning. This not only requires an understanding of target & source vocabularies, but it also requires knowledge of the context in which the terms appear. Stated differently, semantics communicate the meaning of vocabulary terms in a given domain. For example, one cannot determine in a machine-readable way if the term “automobile” in one XML Schema means the same thing as “car” in another, solely based on the XML Schema language. In another example, two XML Schema definitions, “CarLot” & “JunkYard”, each contain the XML element “Car”, but the context for “Car” is markedly different for the two definitions. XML Schema carries no semantics; it is purely structural. Subject matter experts across domains must evaluate the terms and their usages before they agree those terms share

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE **01 DEC 2008**	2. REPORT TYPE **N/A**	3. DATES COVERED **-**
4. TITLE AND SUBTITLE **Army Data Services Layer (ADSL) Data Mediation Service (DMS)**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Applied Research, Telcordia Technologies, Inc. Piscataway, NJ 08854**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES
See also ADM002187. Proceedings of the Army Science Conference (26th) Held in Orlando, Florida on 1-4 December 2008

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**UU**	**9**	

a common meaning. Therefore, humans must first semantically map XML Schema terms across domains for the DMS to generate an XSLT properly.

Given these semantic mappings, creating XSLTs presently requires human intervention. Unfortunately, this process is both tedious and error-prone. An XSLT transformation is unidirectional, and a bi-directional transformation of the same XML message across two domains requires two XSLT documents. In fact, a message involved in a bi-directional exchange over *(n-1)* domains requires *n(n-1)* transformations – the *n^2 problem* (Dirner, et al., 2006). Given the potential volume of XSLT documents, human effort is unwieldy at best, even when using software tools for creating XSLTs. Furthermore, a change to one XML Schema invalidates existing XSLT documents derived from that schema. In this case, every domain that interacts with the one changing its XML Schema (ie, potentially *n–1* domains) must recreate XSLTs anew. An efficient solution to this problem establishes a means to semi-automate the process of XSLT generation. Note, semi-automation will not eliminate the *n^2 problem*. Potentially, *(n^2–1)* XSLT documents are required to transform an XML message from each domain to one in every other domain. Yet, it removes redundancy and human error from the equation. We will address "semi-automation" later.

3. XML SCHEMA MAPPING PATTERNS

The DMS initially considered defining XML Schema mapping patterns to establish repeatable relationships between elements within two disparate XML Schemas. (Petosa, et al., 2008) Simple XML element definitions form the basis of these relationships. A simple element refers to an XML instance document element containing literal data only. This differs from a complex element, which contains other elements, attributes and/or literal data. Simple XML element definitions form the building blocks of XML Schema mapping patterns. The first step in pattern development establishes relationships between vocabulary terms across domains.

3.1 Mapping Types

We define two types of mapping criteria for XML Schema mapping patterns: cardinality and hierarchy. Cardinality mapping occurs at the simple XML element level. It is a one-to-one, one-to-many or many-to-one relationship between simple elements across two XML Schemas. A hierarchy mapping derives from XML Schema structural constructs, which provide a context for the elements it contains.

3.2 One-to-One Cardinality Mapping

A simple XML element definition is the building block of XML Schema mapping patterns. It is atomic, since it only holds literal data and does not contain other XML structures. However, an XML element name has no inherent meaning; it requires human intervention to ascribe it meaning. Figure 3 shows a one-to-one cardinality mapping between two XML Schemas, each defining different representations for a simple XML element named "timeComponent".

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
  xmlns:xs=http://www.w3.org/2001/XMLSchema
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <xs:element name="timeComponent"
    type="timeDecimalType" />
  <xs:simpleType name="timeDecimalType">
    <xs:restriction base="xs:decimal">
      <xs:minInclusive value="0.00" />
      <xs:maxInclusive value="23.99" />
      <xs:fractionDigits value="2" />
      <xs:totalDigits value="4" />
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

Figure 1: "timeComponent" XML Schema (ver 1)

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
  xmlns:xs=http://www.w3.org/2001/XMLSchema
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <xs:element name="timeComponent"
    type="timePatternType" />
  <xs:simpleType name="timePatternType">
    <xs:restriction base="xs:string">
      <xs:pattern value="[0-23]{2}:[0-59]{2}:
        [0-59]{2}" />
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

Figure 2: "timeComponent" XML Schema (ver 2)

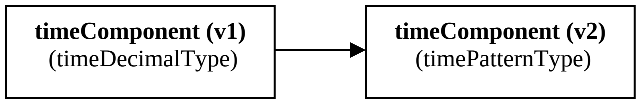


Figure 3: A one-to-one cardinality mapping

Suppose the XML Schemas provided in Figure 1 and Figure 2 represent two different domains, and domain experts agree each version describes the same concept – namely, a time representation. Although the data formats differ, they convey the same intent and both are simple XML elements. Therefore, they are viable candidates for a one-to-one cardinality mapping.

The following XML instance documents illustrate XML messages that validate against the two preceding XML Schemas, respectively, with the actual data highlighted for emphasis.

```
<?xml version="1.0" encoding="UTF-8"?>
<timeComponent
   xsi:noNamespaceSchemaLocation=
      "TimeComponent-One.xsd"
   xmlns:xsi="http://www.w3.org/2001/
      XMLSchema-instance">0.00</timeComponent>
```

Figure 4: "timeComponent" XML instance document (ver 1)

```
<?xml version="1.0" encoding="UTF-8"?>
<timeComponent
   xsi:noNamespaceSchemaLocation=
      "TimeComponent-One_v2.xsd"
   xmlns:xsi="http://www.w3.org/2001/
      XMLSchema-instance">
      00:00:00</timeComponent>
```

Figure 5: "timeComponent" XML instance document (ver 2)

3.3 One-to-Many Cardinality Mapping

One often finds simple XML elements packaged into a complex XML element. A one-to-many cardinality mapping is not concerned with the containership of the simple elements. Rather, it connotes a match from a simple XML element in one domain to two or more simple XML elements in a second domain. That is, subject matter experts for the domains agree that a single, simple XML element encompasses the meaning of two or more simple XML elements. Note, that a one-to-many cardinality mapping can just as easily occur between a single, simple XML element matched to multiple, simple XML elements in another domain's XML Schema.

The XML Schema shown in Figure 6 builds on the "timeComponent" XML schemas presented earlier. In this example, the "many" side of the cardinality mapping presents an XML Schema with a complex type XML element containing the simple XML element components "hours", "minutes" and "seconds".

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
   xmlns:xs=http://www.w3.org/2001/XMLSchema
   elementFormDefault="qualified"
   attributeFormDefault="unqualified">
   <xs:element name="timeComponent"
      type="timeComponentType"/>
```

(continued in next column)

(continued from previous column)

```
   <xs:complexType name="timeComponentType">
      <xs:all>
         <xs:element name="hours">
            <xs:simpleType>
               <xs:restriction base="xs:nonNegativeInteger">
                  <xs:minInclusive value="0"/>
                  <xs:maxInclusive value="23"/>
               </xs:restriction>
            </xs:simpleType>
         </xs:element>
         <xs:element name="minutes">
            <xs:simpleType>
               <xs:restriction base="xs:nonNegativeInteger">
                  <xs:minInclusive value="0"/>
                  <xs:maxInclusive value="59"/>
               </xs:restriction>
            </xs:simpleType>
         </xs:element>
         <xs:element name="seconds">
            <xs:simpleType>
               <xs:restriction base="xs:nonNegativeInteger">
                  <xs:minInclusive value="0"/>
                  <xs:maxInclusive value="59"/>
               </xs:restriction>
            </xs:simpleType>
         </xs:element>
      </xs:all>
   </xs:complexType>
</xs:schema>
```

Figure 6: "timeComponent" XML Schema (ver 3)

Figure 7 conceptually illustrates the Figure 6 one-to-many cardinality mapping.

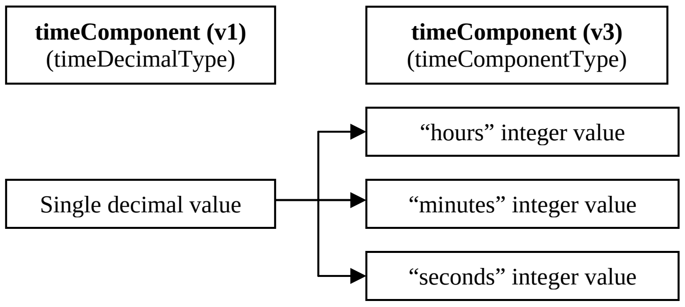


Figure 7: A one-to-many cardinality mapping

3.4 Many-to-One Cardinality Mapping

Alternatively, one may need to consider a one-to-many cardinality mapping in reverse. The "many" side of the mapping collectively describes the same intent as the "one" side, as determined by the domains' respective subject matter experts.

3.5 Cardinality Mapping Summary

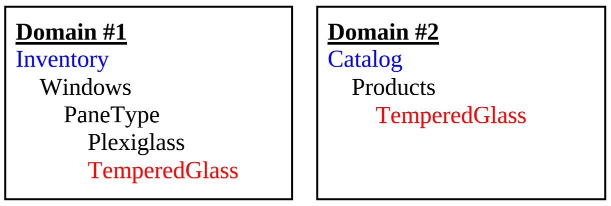
The preceding sections alert us to the following points. First, separate XSLTs are required to enable XML message translations in either direction of a cardinality mapping. This requirement applies for any type of cardinality mapping, but this is more apparent in a one-to-many/many-to-one cardinality mapping. Second, human beings define cardinality mappings. Third, a cardinality mapping only intends to show that the terms on either side of the mapping convey the same meaning across domains. Fourth, for simplicity's sake, the one-to-one cardinality mapping example in Section 3.3 uses identical vocabulary terms across two domains, but this is the exception and not the rule. In fact, a domain may express vocabulary terms in foreign languages or using proprietary internal codes as XML element names. The latter is anathema to human readability, but it is allowable. Furthermore, terms with identical spellings may convey radically different meanings across domains. Fifth, cardinality mapping addresses data mediation in terms of message transformations, but it does not speak to the data format conversion. In an earlier example, we expressed a "timeComponent" cardinality mapping using a decimal value in one domain and a string value patterned from a regular expression in a second domain.

3.6 Hierarchy Mapping

A hierarchy mapping examines the structural nature of XML Schema. Whereas a cardinality mapping occurs at the simple element level, a hierarchy mapping provides the context for a cardinality mapping. Therefore, a hierarchy mapping builds upon a cardinality mapping. An element map usually occurs between nested elements across disparate XML Schemas, so a cardinality mapping alone cannot adequately define a mapping. The exception is mapping two global, simple elements, since they have no nested structures.

Suppose a vocabulary term qualifies for a cardinality mapping across various domains, and XML Schema definitions in various domains nest this term. How would one know if the root levels of two XML messages containing these terms convey the same meaning (ie, form a hierarchy mapping)? Once again, humans determine this knowledge.

The following example represents XML Schema definitions from various domains.



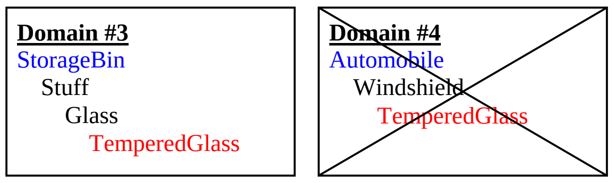


Figure 8: Hierarchy mapping example

Subject matter experts for each domain agree that "TemperedGlass" conveys the same meaning across domains. Assuming "TemperedGlass" is a simple XML element, it is eligible for a cardinality mapping. The subject matter experts also agree that "Inventory", "Catalog" & "StorageBin" convey similar meanings in that each generally describes a collection of things by a weak association. They determine that "Automobile" does not express a similar meaning as the preceding terms, since it is a collection of things having a strong association. That is, an "Automobile" must contain "Windshield" and "TemperedGlass" to be an "Automobile". However, the various other items within the other domains are not inherently part of their top-level descriptions (ie, "Inventory", "Catalog" & "StorageBin"). For example, an empty "StorageBin" is still a "StorageBin". As a result, one establishes inter-domain hierarchy mappings between Domains 1-3. Domain #4 cannot participate in these mappings.

Without the assistance of subject matter experts, how can one determine a hierarchy mapping in a machine-readable way? We have already seen that a cardinality mapping requires subject matter experts to agree that terms involved in the mappings have the same meaning. Conceptually, the same problem arises for a hierarchy mapping. First, we establish containership using the XML Document Object Model. The XML DOM presents an XML document as a tree, and an XML parser can use the XML DOM to "walk the tree". One can discover if an XML element defined in a cardinality mapping is contained within an element slated for a hierarchy mapping. Subject matter experts then use this information to vet the hierarchy mapping.

3.7 Equivalency in a Hierarchy Mapping

Unlike a cardinality mapping, a hierarchy mapping usually does not convey the same meaning; rather, it represents a similar (or equivalent) meaning. Equivalency establishes an additional criterion for a hierarchy mapping. For example, hierarchy mappings between "Inventory", "Catalog" and "StorageBin" in

Figure 8 are similar in that they all describe weak containership structures. However, the concepts are not identical.

There needs to be some way to quantify the degree of equivalency to best match these terms in a hierarchy mapping. For example, is "Inventory" more similar to "Catalog" than it is to "StorageBin"? Perhaps one can attach a numerical "weight" value to a hierarchy mapping, where a larger "weight" value correlates to a higher degree of equivalency. Still, this is highly arbitrary, and it requires subject matter experts to agree on the "weight" values. Consider the following "weight" values in various hierarchy mappings. In this example, "Inventory / Catalog" is the "most equivalent" hierarchy mapping amongst the terms "Inventory", "Catalog" and "StorageBin".

Hierarchy Mapping	Weight Value
Inventory / Catalog	10
Inventory / StorageBin	2
Catalog / StorageBin	3

Table 1: Hierarchy mapping weight values

3.7 Hierarchy Mapping Summary

A hierarchy mapping provides the context for a cardinality mapping in an XML document, but it requires domain experts to qualify it. Hierarchy mappings connote equivalent meanings between terms across domains, and weight values may provide a way to quantify the degree of similarity between terms.

3.8 The Viability of XML Schema Mapping Patterns

There is no inherent semantics built into the XML Schema language; it is purely structural. Thus far, we have seen that subject matter experts must play a role in mapping terms across domains. Therefore, it is impractical to develop XML Schema mapping patterns, since they are not machine-readable. Instead, we turn to semantic technologies to assist our mapping efforts.

4. DMS USE OF SEMANTIC TECHNOLOGIES

As noted earlier, the Data Mediation Service accepts an XML instance document and produces an XSLT document. A service consumer uses this XSLT document to transform the source XML instance document into a target XML instance document. The DMS semi-automates the process of creating XSLT documents, and this is highly advantageous since it removes the necessity to create them manually. Removing the human element not only reduces the susceptibility for errors, but it also greatly increases the turnaround time to generate XSLT documents. Yet, the fact remains that the human element still plays a role, which is why the XSLT-generation process is semi-automatic.

4.1 Creating XSLTs "semi-automatically"

At this point, humans must initially agree upon which terms map across domains, since the XML Schema language cannot semantically express such mappings. Yet, the Data Mediation Service requires XML Schemas to generate XSLTs. While human-generated mappings form the basis of the XSLT-generation process, the rest of the process is machine-generated. Hence, the XSLT generation process is "semi-automated". Subsequent sections discuss the machine-generated concept.

4.2 The Web Ontology Language (OWL) and Related Languages

"Ontology is a data structure that consists of concepts (such as things, events and ideas), relationships between these concepts, and the rules governing when these inter-concept relationships hold and when they do not. These concepts, relationships and rules are used to define a particular domain of knowledge." (Ganti, 2008)

OWL is an ontology modeling language, and it is a World Wide Consortium (W3C) recommendation. It is human-readable, typically written in the language of RDF/XML. It also carries semantics and extends Resource Description Framework Vocabulary Description Language (aka, RDF Schema). To understand OWL, we must first look at RDF and RDFS, briefly discussed next.

Resource Description Framework (RDF) provides a framework to identify resources on the World Wide Web, which is an application that runs on a distributed network (ie, the "Internet"). Each resource has a Uniform Resource Identifier (URI), described in terms of properties with values. The RDF model is a labeled, directed graph. However, like XML Schema, RDF carries no inherent semantics (RDF Primer, 2004), but it is possible to trace the location of a resource by "walking" a path in an RDF graph.

RDFS extends RDF with language constructs that carry semantics describing "groups of related resources and the relationships between these resources." One such construct is <rdfs:Class>. According to the literature, an RDF Class is a resource while "instances" refer to members of the class (RDF Schema, 2004). An RDF datatype is an example of an RDF Class, and it may derive from an XML Schema datatype (eg, <xs:integer>, <xs:string>, etc.). An RDF Class models a concept, whereas an instance describes it concretely. For example, suppose an RDF Class models an XML Schema "nonNegativeInteger" datatype. The set { 0, 1, 2, … n }describes all the possible instances of that class. This is an interesting concept but not especially useful. Yet, the value is evident when coupled with other RDFS language constructs, such as <rdfs:subClassOf>. As one may surmise, <rdfs:subClassOf> models a subset of <rdfs:Class>. Suppose the RDF Class, "music_genre", describes the following instances: Baroque, Bebop, Rock, Trance, Sonata, Symphony, 12-Bar Blues, Concerto, and French Dance Suite. An RDF SubClass, "classical_music" (in its common usage) describes the following set of instances: Baroque, Sonata, Symphony, Concerto, and French Dance Suite. Here, we see that the first set of instances completely contains the second set of instances (ie, the "classical_music" instance set is a subset of the "music_genre" instance set).

RDFS provides language constructs that are useful in describing ontologies. As a declarative ontology language, it encodes domain knowledge. In other words, RDFS describes concepts, but it does not indicate how to implement these concepts (RDF Schema, 2004). The RDFS language also allows a reasoning engine to infer new knowledge from an existing knowledge base modeled as an RDFS document.

In the example earlier, the RDF Class and RDF SubClass language concepts "music_genre" and "classical_music", respectively, semantically describe a necessary but not sufficient relationship. That is, "classical_music" implies "music_genre", but the reverse is not true. Practically speaking, one can infer that all "classical_music" instances (Set Z) are contained within "music_genre" instances (Set A), but not the reverse (see Figure 10). Stated differently, the "classical_music" instances are a subset of "music_genre" instances, but the "music_genre" instances are not a subset of "classical_music" instances. At least one element in Set A is not contained in Set Z. For example, Set A contains "Blues"; Set Z does not. Therefore, Set Z is a proper subset of Set A.

Earlier, we touched upon OWL, but now there is enough background to expound upon it. Briefly, OWL is a declarative ontology language typically written in RDF/XML that semantically extends RDFS. One useful extension to RDFS is OWL's <owl:Class>. An OWL class modifies the semantics of <rdfs:Class> to include class axioms that provide greater specificity for the concept it models. Another useful extension is the OWL language construct, <owl:equivalentClass>.

Let us return to the prior example. Suppose some other domain models an OWL ontology that contains the OWL class, "types_of_music", and this class independently describes a set of instances – "individuals" in OWL parlance – that are the same as the "music_genre" individuals. By virtue of the exact same members contained within each set, the "music_genre" and "types_of_music" OWL class concepts are "equivalent". Again, in OWL parlance, the "music_genre" class extension is identical to the "types_of_music" class extension (OWL Guide, 2004). This scenario represents a necessary and sufficient relationship between OWL classes. Each class meets the definition that describes it as a subset of the other class and each subset is identical in content to its counterpart. In other words, the "music_genre" and "types_of_music" OWL classes each imply the other; they have a logical biconditional relationship. Figure 9 illustrates this relationship. Let us define "music_genre" as Set A and "types_of_music" as Set B. Therefore,

$$(A \subseteq B) \equiv (B \subseteq A)$$

, where A = { Baroque, Bebop, Rock, Trance, Sonata, Symphony, 12-Bar Blues, Concerto, French Dance Suite } and B = { Baroque, Bebop, Rock, Trance, Sonata, Symphony, 12-Bar Blues, Concerto, French Dance Suite }

Figure 9: A necessary and sufficient condition

The following mathematically expresses the relationship between "music_genre" (Set A) and "classical_music" (Set Z).

$$(Z \subseteq A) \not\equiv (A \subseteq Z)$$
$$\therefore (Z \subsetneq A)$$

, where A = { Baroque, Bebop, Rock, Trance, Sonata, Symphony, 12-Bar Blues, Concerto, French Dance Suite } and Z = { Baroque, Sonata, Symphony, Concerto, French Dance Suite }

Figure 10: A necessary but not sufficient condition

Although the "extensional" meaning of "music_genre" and "types_of_music" is the same (ie, each class extension is identical), the "intensional" meaning (ie, the abstract OWL class definition) is not necessarily the same. Let us modify the "music_genre" class to include the notion of composers, where a composer value is optional. For example, "Baroque" music and "Baroque" music "composed by Johannes Sebastian Bach" are valid extensional meanings of "music_genre". It just so happens that the latter individual includes an optional composer value. If one exercises this option at least once in the "music_genre" class extension, then the "types_of_music" class extension described earlier is no longer identical to it. In this case, the "music_genre" and "types_of_music" OWL classes are no longer equivalent. In practice, a reasoning engine tests for "consistency" to ensure the individuals contained in an ontology adhere to the semantics described in the OWL language constructs. In the prior example, the reasoning engine deems the ontology "inconsistent", because the class extensions are not identical for the OWL classes asserted to be equivalent.

Recall the "music_genre" & "classical_music" concepts in one ontology and the "types_of_music" concept in a second ontology. We asserted "classical_music" is a subclasss of "music_genre" and the "music_genre" & "types_of_music" concepts are equivalent. We did not assert a relationship between "classical_music" and "types_of_music". In applying a reasoning engine to the ontologies, the engine will infer that "classical_music" is a subclass of "types_of_music". Hence, the reasoning engine acquires new knowledge from the existing knowledge base. Herein is in the real power of ontologies.

4.3 OWL Species

The Data Mediation Service leverages OWL DL ontologies to semi-automate the process of creating XSLT documents. OWL is available in three species: OWL Lite, OWL DL and OWL Full. The Web Ontology Language Description Logic (OWL DL) and OWL Lite species utilize the decidable fragments of first-order logic for its logical formalism (Horrocks). Both language species are computationally complete, which guarantees that an inference engine can compute a result when querying an OWL DL/Lite ontology. However, OWL DL is more expressive than OWL Lite. For example, OWL Lite only defines an intersection set operation, while OWL DL also adds the union and complement set operations. What level of expressivity is necessary for a given problem? One common OWL DL (version 1.0) complaint is that it does not allow qualified cardinality restrictions. A cardinality restriction specifies the number of allowable values for a property, but it cannot address the type of value (Schreiber, 2004). OWL 2.0 proposes to allow qualified cardinality restrictions, which includes a type qualification on the cardinality restriction (OWL 2, 2008). Figure 11 and Figure 12 show an example of unqualified & qualified cardinality restrictions, respectively, using Manchester OWL syntax.

```
hasDigit exactly 5
```

Figure 11: Unqualified cardinality restriction

```
(hasDigit exactly 4 Finger) and (hasDigit exactly 1 Thumb)
```

Figure 12: Qualified cardinality restriction

Both describe a human hand, but the qualified cardinality restriction is more expressive.

OWL 2.0 also intends to allow a more expressive data range by including facets. The following example (see Figure 13) restricts the "hasAge" property to teenage years (Knublauch, 2007).

```
hasAge some xsd:int[>=13, <20]
```

Figure 13: Data range facet

Unfortunately, the trade-off for greater expressivity is intractability. At some point, an ontology language can become too expressive, to the point that computations on the model may require more time than is reasonably acceptable.

The Data Mediation Service uses OWL DL (version 1.0) in its present incarnation, although it may adopt OWL DL 2.0 in the future.

4.3 Ontology Alignment in the DMS

"Given two ontologies, aligning one ontology with another one means that for each entity (concept, relation, or instance) in the first ontology, we try to find a corresponding entity, which has the same intended meaning, in the second ontology. An alignment therefore is a one-to-one equality relation. Obviously, for some entities no corresponding entity might exist." (Ehrig, 2005)

This definition is similar to the one-to-one cardinality mapping discussed in Section 3.2. Conceptually, both ideas semantically align vocabulary terms across domains. Subject matter experts produce one-to-one cardinality mappings between XML elements. This is not always the case when aligning concepts across ontologies; not every ontology alignment requires human intervention. Given the inherent semantics built into an ontology, some alignments are later inferred. For example, consider the following concepts modeled in various domains' ontologies (see Table 2).

Domain Ontology	Concept
A	**Paper**
B	Newspaper
C	**WhitePaper**
D	Wallpaper
E	**ScholarlyWork**
F	**TechnicalAnalysis**

Table 2: Concepts used in OWL equivalencies

A subject matter expert analysis determines that A:Paper conveys the same meaning as C:WhitePaper & F:TechnicalAnalysis. A second and independent subject matter expert analysis reveals that E:ScholarlyWork is semantically equivalent to A:Paper. Using these results, ontologists assert OWL DL equivalencies to align the preceding concepts. This process is similar to the one-to-one cardinality mapping process, since they each require human intervention. However, given these asserted equivalencies, a DL-compliant inference engine discovers all the concepts are mutually equivalent.

4.4 The Data Mediation Service Concept

The DMS end goal is to produce an XSLT document that transforms an incoming XML message to one that validates against a target domain's XML Schema. An external repository shared by various communities of interest stores the XSLT documents required by the DMS. Therefore, the first order of business is to query this repository for an XSLT document that meets a DMS service consumer request. If it finds one, the Data Mediation Service retrieves it from the repository and outputs the XSLT document. If not, it generates an XSLT document and saves it to the repository.

The semantic component of the DMS establishes an ontology "core", and it further models the core into an "inner core" and an "outer core of cores". In its present incarnation, the DMS inner core ontologically models concepts common to the communities of interest that will access the service. The primary reason for building an inner core is to address the cascading imports problem in OWL (version 1.0) ontologies. An OWL ontology may decide to import other OWL ontologies, much like an XML Schema may decide to include/import other XML Schemas. In fact, any OWL ontology can potentially import other OWL ontologies (ie, the cascading import problem). Since the inner core will focus on common inter-domain concepts, it is desirable to prevent cascading imports. As such, the DMS inner core will not import other OWL ontologies to ensure it does not add other concepts inadvertently.

The “outer core of cores” intends to house several ontologies serving as “proxies” to the inner core. These may already be in common use, where one or more communities of interest reference them within their own ontologies. Therefore, they are excellent candidates for inclusion in the DMS outer core. Outer-core ontologies will proxy the DMS inner core concepts, while carrying additional concepts relevant to one or more communities of interest. Subject matter experts and ontologists evaluate these “outer-core” ontologies and may decide to align them to each other further. The extent of these alignments is yet to be determined. For example, future performance testing may define the outer-core alignment criteria. With the exception of inner-core concepts, it is likely that outer-core ontologies may share very few, if any, concepts. However, an inference engine may reveal unanticipated but desirable relationships between unaligned outer-core ontologies.

The following demonstrates how the proxy concept works. The outer-core ontology imports the inner-core ontology. Suppose the inner core models “Hours”, “Minutes” and “Seconds” concepts and the outer core models a “TimeOfDay” concept. Suppose “TimeOfDay” is a military time concept that exactly contains the other concepts. The OWL DL “unionOf” language construct fits this scenario. An example is show in N3 (Notation3) format.

```
:TimeOfDay
    a     owl:Class ;
    owl:unionOf (IC:Hours IC:Minutes IC:Seconds) .
```

Figure 14: A concept by proxy

IC is a prefix that represents the DMS inner core. Since the unionOf operation exactly describes the TimeOfDay class extension, this operation forms an equivalency between TimeOfDay and an anonymous class containing the unionOf operation. In fact, any OWL set operation is a syntactic shortcut for describing an equivalency. In this example, “TimeOfDay” is acting as a proxy to the other three concepts. Conceptually, the DMS “hides” inner core concepts. In reality, OWL cannot prevent an ontology from directly using a concept imported from another ontology. An external ontology (ie, one not contained within the DMS core) approved for use with the DMS must use an outer-core ontology as a proxy to inner core concepts. The intent is to maintain a black-box approach for the DMS core. Any changes to the DMS core should not invalidate existing external ontology alignments to the core.

In summary, “external” ontologies align to one or more outer-core ontologies. These, in turn, may possibly align to other outer-core ontologies. Ultimately, external ontologies align to the DMS inner-core ontology via outer-core ontology proxies. This model provides the semantics the DMS requires, while preventing the addition of unanticipated concepts to the DMS inner core.

On its own, the DMS semantic component cannot produce XSLTs, but it can determine how to generate an XSLT based upon ontological alignments. This requires DMS-approved ontologies to encode XML Schemas within them. An inference engine determines any equivalency alignments between the source and target ontologies. These alignments provide the semantics necessary for the DMS to extract the appropriate source and target XML Schema elements for building a source-to-target XSLT document.

CONCLUSIONS

XML is a good first step toward making data widely understandable. It promotes net-centricity by providing a common language in place of point-to-point solutions to share data. XML Schema describes how to structure XML data, but the XML Schema language lacks semantics. Yet, XML technology is at the heart of service-oriented architecture implementations in terms of messages sent across services. The XSLT language specifies how to transform XML messages, but it also lacks semantics.

Ontologies provide the missing semantic piece to this puzzle. Using OWL DL, a W3C-recommended ontology modeling language, the Data Mediation Service semi-automates the process of generating the XSLTs needed to transform source XML messages to their target XML message structural format. Subject matter experts initially align ontologies across domains, but inference engines draw on the logical formalism behind OWL DL to gain newly acquired knowledge (ie, new alignments are discovered). The new knowledge enables the DMS to extract the relevant parts of source and target XML Schemas in order to create appropriate XSLT documents.

The Data Mediation Service provides value to the warfighter in that it makes data more understandable across the entire Army enterprise and beyond. By removing the human element, the DMS not only eliminates the propensity for errors to creep into XSLT documents, but it also reduces the turnaround time, from requesting an XSLT to generating an XSLT, dramatically. A quicker flow of information provides value to the warfighter.

REFERENCES

Dirner, M., Mansell, M. and Wronko, M., 2006: Army Data Services Layer (ADSL) – Data Mediation Providing Data Interoperability and Understanding in a SOA Environment, 4.

Ehrig, Marc, 2005: Ontology Alignment – Bridging the Semantic Gap, Springer Science+Business Media, LLC, 19.

Ganti, Narsim, 2008: Army Net-Centric Data Strategy Center of Excellence Semantic Technology Studies Ontology Languages Analysis Report, 3.

Horrocks, Ian, OWL: A Description Logic Based Ontology Language, http://www.cs.man.ac.uk/~horrocks/Slides/cisa06.ppt#831,1,OWL: A Description Logic Based Ontology Language.

Knublauch, Holger, 2007: OWL 1.1 Support in TopBraid Composer 2.0, http://composing-the-semantic-web.blogspot.com/2007/02/owl-11-support-in-topbraid-composer-20.html.

Petosa, A. and DeMasi, W., 2008: Data Harmonization Mapping Patterns, Part 1: XML Schema Language Alone is Inadequate for Data Harmonization.

Schreiber, Guus, 2004: Qualified Cardinality Restrictions (QCRs): Constraining the Number of Property Values of a Particular Type.

Various authors, 2004: RDF Primer: W3C Recommendation 10 February 2004, World Wide Web Consortium.

Various authors, 2004: RDF Vocabulary Description Language 1.0: RDF Schema: W3C Recommendation 10 February 2004, World Wide Web Consortium.

Various authors, 2008: OWL 2 Web Ontology Language: Primer, World Wide Web Consortium.

Various authors, 2004: OWL Web Ontology Language Guide: W3C Recommendation 10 February 2004, World Wide Web Consortium.