SIGAda 2001 Workshop, “Creating a Symbiotic Relationship Between XML and Ada”

Robert C. Leif
Ada_Med, a Division of Newport Instruments
5648 Toyon Road, San Diego, CA 92115-1022, USA
E-mail rleif@rleif.com
www.newportinstruments.com
www.Ada-Med.com
+1 (619)582-0437

Abstract:

The purpose of the workshop was to organize the Ada community to take advantage of the opportunity to create Ada applications that are operating systems independent because they are based on a web technology, XML, Extensible Markup Language. The commercial use of the Internet is the driving force behind XML. Four elements of XML, which together are sufficient to build a web application, and all employ the same syntax were described. These are XML; its schema; the Extensible Stylesheet Language, XSL; and the XML mechanism for forms, XForms. XML concerns the data objects that are included on the web page and their order of presentation. The schema contains the information on the types and objects for XML. Schemas are roughly equivalent to an Ada specification without the subprograms.

Fortunately, the programing language that has the best fit with XML is Ada. XML has visibility and scoping rules, which are similar to Ada. XML has strong typing and has single inheritance similar to Ada. A mutually beneficial symbiosis requires the creation of applications in Ada that use and support XML, as well as, the use of XML to create Ada environments including XML based tools. These applications include: automated translation of Ada data types and objects in a specification to an XML schema;. and conversely, automated translation of the data types and elements in an XML Schema to an Ada specification.

Introduction:

Both the after-effects of the horror of September 11, 2001 and the scheduling of the workshop on the first day of tutorials resulted in only two presentations. William Currie Colket described the use of HTML to create a genealogy and Robert C. Leif presented an abbreviated precursor of the document below, which included a brief introduction to: XML, integrating Ada and XML, and using XML to build an Ada environment.

The introduction of commercially viable personal computers by IBM and Apple has resulted in successive generations of operating systems. In the case of the IBM PC and its very successful clones, there have been two complete generations of operating systems and the evolution of the third generation is presently occurring. The first generation was the text based DOS, which was created by Microsoft for IBM. The next generation started with a short-lived contest between two Apple Macintosh inspired graphical user interface operating systems, IBM OS/2 and Microsoft Windows. IBM facilitated Microsoft’s victory by providing the perception that only IBM brand PCs could be guaranteed to run OS/2. The third generation is the result of the Internet. A browser, such as Internet Explorer or Netscape, serves as a graphical user interface, GUI, for web pages, other web objects, and provides resources for traditional programs. Since the services needed by web pages and standard programs very significantly overlap, much of the software for both sets of services can be common. This particularly makes sense when two versions of the same program exist one that runs on the PC and the other that is executed in client-server mode. The control of this third generation of operating system has not been settled. One possibility is the proprietary Java system invented by Sun and championed by IBM. The second is Microsoft Windows and Internet Explorer, which does include XML with some proprietary elements. The third possibility is a non-proprietary GUI based on the standards provided by a not-for-profit entity, the World Wide Web Consortium (1).

Before this discussion precedes further, it is necessary to provide a very abbreviated history of markup languages. The first truly commercially viable standardized mark-up language was SGML (Standard Generalized Markup Language) (2). Unfortunately, SGML was large and initially required manual insertion of its codes. At the same time, there was a very commercially successful product that employed a comparatively simple, automated markup, WordPerfect. This proprietary markup was referred to as codes, which the user could modify or change. SGML was similar to Ada in that it was too much and too early. The present web is based on a very successful but limited subset of SGML, HTML (Hypertext Markup Language) (3). Unfortunately, HTML did not provide a mechanism to create specific markup tags for the numerous types of entities in a web page. HTML, because of its very limited number of types, primarily operates in a procedural manor. The successor and ultimate replacement for HTML is XML (4).

Presently, XML is driven by the exploding use of the Internet and the economic advantage of having the same software operate on both the client and server. The historic Ada was a gambler. Her namesake, the computer language, has just had her luck change very much for the better. The commercial use of the Internet is the driving force behind XML. Fortunately, the programing language that has the best fit with XML is Ada. The purpose of the Workshop was and of this paper is to organize the Ada community to take advantage of the opportunity afforded by the close semantic relationship of Ada and XML. This requires creating applications in Ada that use and support XML; and using XML to create Ada environments including XML based tools. As demonstrated below by the Goals for XML, the Ada and XML communities have much in common.

W3C Goals (Requirements) for XML (4):

1. **XML shall be straightforwardly usable over the Internet.**
2. **XML shall support a wide variety of applications.**
3. **XML shall be compatible with SGML.**
4. **It shall be easy to write programs which process XML documents.**
5. **The number of optional features in XML is to be kept to the absolute, minimum, ideally zero.**
6. **XML documents should be human-legible and reasonably clear.**
7. **The XML design should be prepared quickly.**
8. **The design of XML shall be formal and concise.**
9. **XML documents shall be easy to create.**
10. **Terseness in XML markup is of minimal importance.**

Except for item 3, which specifies backwards compatibility, and 7, which is a perceived economic necessity, the other 8 goals could describe Ada.

The XML Standards Puzzle

Multiple Standards

Because of its derivation from SGML and HTML and the evident desire of W3C to employ modern software engineering technology, XML often has both a preexisting and a completely new standard for the same purpose. Since interfacing Ada with all of the XML standards is not within the capabilities of the Ada community, a reasonable approach is to limit our efforts to those parts that share the same syntax and have the closest similarity to Ada. These are primarily XML itself, schema, XSL (the Extensible Stylesheet Language), and XForms. Unfortunately, one major commonality that these standards have with their predecessors is that they are case sensitive.

DTD

The “Document Type Definition”(4, 5) is a carryover from SGML and has inherited from SGML its own arcane syntax, which does not include strong typing. DTDs are part of the XML 1.0 specification (4). A DTD allows a developer, or standards body, to specify what elements and attributes may be used in a particular type of XML document and what their structure and nesting may be. This is also called the *content model* or *schema (generic use of the term)* of an XML document.

If an XML document conforms with the content model defined by a DTD, it is said to be *valid* with respect to that DTD. However, DTDs can be incomplete in that they can not be directly translated into a schema, which is the new alternative to the DTD, see below.

XML Schema:

The “XML Schema” is an ongoing effort by the W3C to supplant DTDs with a more flexible and powerful system to describe the structure of conforming XML documents, including provisions for defining data types including ranges and inheritance. Schema allow a software based validation of document structure. Schemas are written in XML element syntax. Therefore, a correct XML schema definition is a well-formed XML document.

Schemas are described in three W3C documents: The first is the XML Schema Part 0: Primer (6). It is described as “a non-normative document intended to provide an easily readable description of the XML Schema facilities, and is oriented towards quickly understanding how to create schemas using the XML Schema language.” Although Part 0 is a good introduction, it is far from a sufficient description to permit serious use of XML schemas. XML Schema Part 1: Structures (7) is a complex document, which “specifies the XML Schema definition language, which offers facilities for describing the structure and constraining the contents of XML 1.0 documents, including those which exploit the XML Namespace facility. The schema language, which is itself represented in XML 1.0 and uses namespaces, substantially reconstructs and considerably extends the capabilities found in XML 1.0 document type definitions (DTDs).” The third document, XML Schema Part 2: Datatypes (8), “defines facilities for defining datatypes to be used in XML Schemas as well as other XML specifications. The datatype language, which is itself represented in XML 1.0, provides a superset of the capabilities found in XML 1.0 document type definitions (DTDs) for specifying datatypes on elements and attributes.” Since Schema Part 2 does include the definitions of the “Built-in datatypes” and “Derived datatypes”, it is the section most relevant to interfacing with Ada.

XML Schema Part 2: Datatypes, Section 2 Type System: A datatype was defined as

- “a 3-tuple, consisting of a) a set of distinct values, called its value space,” (integers, strings, enumerated, boolean, etc.)
- “b) a set of lexical representations, called its lexical space”·(valid textual, literals for a datatype),
- “and c) a set of facets that characterize properties of the value space, individual values or lexical items” (ranges, formatting, default values).

Facets: "A fundamental facet is an abstract property which serves to semantically characterize the values in a value space." Even equality, which is not explicitly defined in the Ada Reference Manual, is defined for schema data types in Section 4.2.1.

- for any a and b in the value space, either a is equal to b, denoted a = b, or a is not equal to b, denoted a != b
- there is no pair a and b from the value space such that both a = b and a != b
- for all a in the value space, a = a
- for any a and b in the value space, a = b if and only if b = a
- for any a, b and c in the value space, if a = b and b = c, then a = c
- for any a and b in the value space if a = b, then a and b cannot be distinguished (i.e., equality is identity).

Other facets such as: ordered, bounded, and cardinality are also rigorously defined.

Types can be “derived by restriction from another datatype when values for zero or more constraining facets are specified that serve to constrain its value space and/or its lexical space to a subset of those of its base type.” Because schemas primarily define types and objects, it is not directly evident whether the result of this derivation corresponds to a new Ada type, subtype, or even, in some cases an extension to a tagged type.

An example is money with a range of $5.00 to $2,000.

```
1 <simpleType name="Fee_Type">
2   <restriction base="decimal">
3     <totalDigits value="6"/>
4     <fractionDigits value="2"/>
5     <minInclusive value="5.00"/>
6     <maxInclusive value="2000.00"/>
7     <pattern value="[0-9]{4}(.[0-9]{2})"/>
8     <!--Four leading digits, decimal point, and 2 trailing digits.-->
9   </restriction>
10 </simpleType>
```

XML describes attributes and their values in a similar manner to the Ada formal actual => syntax. Line 1 starts a specification of a simple type by assigning to it the string “Fee_Type”. The Ada arrow format is replaced by the combination of the equal sign and setting the actual value in quotation marks. In line 2, the base type is specified. Lines 3 to 6 specify the equivalent of the instatiation of an Ada Decimal type. However, “the number of digits must be less than or equal to ·totalDigits”; and fractionDigits means the number of digits to the right of the decimal point. The range is specified by the minInclusive and maxInclusive attributes. Line 7 describes a simple but very powerful, general means to specify a format, which in this case is used to present a decimal number. Four numeric characters are followed by a decimal point, which is in turn followed by two more numeric characters. Line 8 is a comment, which is shown by a beginning `<!--` and ending `-->`. Comments can have lengths greater than one line.

XML numeric types including decimal have two constraining facets, which do not presently exist in Ada. These are `maxExclusive` and `minExclusive`, which are respectively the next value greater than the maximum of the range and the next value less than the minimum of the range. Fortunately, these are only specified for appropriate derived types, since the values of both `minExclusive` and `maxExclusive` must be in the value space of the base type.

A simple type, **`“Fee_Type”, was defined above.`** XML simple types are equivalent to the elementary types in Ada. Schemas can also define complex datatypes, which are like Ada’s composite types composed of component values. However, XML complex types can include element attributes. Attributes are described by simple types and act as modifiers of XML elements.

OO in XML

Creation of a complex type

Below is a schema, Person_Name, which describes the creation of a person type that includes the given (first), middle, and last names of a person.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!--Web: targetNamespace="http://NewportInstruments.com/Person_Name"
3 xmlns:nam_add="http://NewportInstruments.com/Person_Name"-->
4 <schema targetNamespace="file://C:\SIGAda\2001\XML\Person_Name.xsd"
```

```
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:per_nam="file://C:\SIGAda\2001\XML\Person_Name.xsd"
elementFormDefault="unqualified" attributeFormDefault="unqualified">
<annotation>
  <documentation>Person_Name to be used as part of other schemas and is
     mirrored by an equivalent Ada package 2001 Oct. 15.
     DICOM.Person_Names_Bd.Ads
   </documentation>
 </annotation>
 <!--Steps to create a Person_Name_Type-->
 <simpleType name="Given_Name_Type">
   <restriction base="string">
     <maxLength value="50"/>
   </restriction>
   <!--50 characters are needed because it is possible for a person
     to have only one name-->
 </simpleType>
 <simpleType name="Middle_Name_Type">
   <restriction base="string">
     <maxLength value="20"/>
   </restriction>
 </simpleType>
 <simpleType name="Family_Name_Type">
   <restriction base="string">
     <minLength value="1"/>
     <maxLength value="50"/>
   </restriction>
 </simpleType>
 <complexType name="Person_Name_Type">
   <sequence>
     <element name="Given_Name" type="per_nam:Given_Name_Type"/>
     <element name="Middle_Name" type="per_nam:Middle_Name_Type"/>
     <element name="Family_Name" type="per_nam:Family_Name_Type"/>
   </sequence>
 </complexType>
</schema>
```

Line 1 tells the browser that this is an XML file and that the UTF-8 character set is used. UTF-8 is an 8 bit format Unicode character representation. Latin-1 encoding would be specified as ISO-8895-1. Lines 2 and 3 are a comment, which shows the format that would be used for a web based address. However, since one of the major uses for Ada is embedded systems and this paper emphasizes self-hosting on the client, the URL in line 4 is to the local file system. Line 4 tells an XML processor that this is the location of the schema for a specific XML page. It is the place to validate the types and elements for that page and can include pointer(s) to other schema. Line 5 starts with the reserved word **`xmlns`**, which means XML name space. The most common namespace that is made visible is that of the 2001/XMLSchema itself, line 5 This schema describes the basic XML types, elements, and attributes. It is the equivalent of making the package Ada visible. The name space of the above schema is both made visible in line 6 and is assigned the prefix, per_nam, which is also the equivalent of an Ada package renaming. This prefix will be used with the new datatypes and would have been required to be used with the new elements and attributes if the defaults in line 7 had been set to qualified.

Lines 8 through 13 are an annotation, which is essentially a typed comment, which can extend for more than one line. The functionality of annotations particularly those describing required documentation and items to be only checked at compile time should be considered for inclusion in Ada.

Lines 15 through 21 describe the creation of the equivalent of an Ada bounded string with a maximum of 50 characters for given (first) names. This structure is repeated on lines 22 through 26 and 27 through 32 for respectively Middle and Family name types. Lines 33 through 39 describe the creation of a XML complex type, which is equivalent to a record; however, the term **`sequence`** is used instead of record.

Extension of a Complex Type

The schema below extends the Person_Name_Type to include a generation suffix.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <schema targetNamespace="file://
3   C:\SIGAda\2001\XML\Person_Name_w_Generation.xsd"
4 xmlns:per_nam="file://C:\SIGAda\2001\XML\Person_Name.xsd"
5 xmlns:per_nam_gen="file://
6   C:\SIGAda\2001\XML\Person_Name_w_Generation.xsd"
7 xmlns="http://www.w3.org/2001/XMLSchema"
8   elementFormDefault="unqualified" attributeFormDefault="unqualified">
9 <import namespace="file://C:\SIGAda\2001\XML\Person_Name.xsd"
10   schemaLocation="file://C:\SIGAda\2001\XML\Person_Name.xsd"/>
11   <annotation>
12     <documentation>Child of Person_Name. Generation_Type added: Jr.
13       Senior, II, III, etc.</documentation>
14   </annotation>
15   <simpleType name="Generation_Type">
16     <restriction base="string">
17       <enumeration value="Jr."/> <enumeration value="Senior"/>
18       <enumeration value="II"/> <enumeration value="III"/>
19       <enumeration value="IV"/> <enumeration value="V"/>
20       <enumeration value="VI"/> <enumeration value="VII"/>
```

```
21       <enumeration value="VIII"/> <enumeration value="IX"/>
22       <enumeration value="X"/>
23     </restriction>
24   </simpleType>
25   <complexType name="Person_Name_W_Generation_Type">
26     <complexContent>
27       <extension base="per_nam:Person_Name_Type">
28       <sequence>
29         <element name="generation" type="per_nam_gen:Generation_Type"
30           minOccurs="0"/>
31       </sequence>
32       </extension>
33     </complexContent>
34   </complexType>
35   <element name="Person_Name_w_Generation"
36     type="per_nam_gen:Person_Name_W_Generation_Type"/>
37 </schema>
```

The **schema**, Person_Name_w_Generation.xsd, includes a generation enumerated type, which was used to extend the Person_Name_Type. Lines 1 through 7 are essentially the same as the Person_Name schema; except that a third schema's namespace has been included. The Person_Name schema has been made visible by the **import namespace** statement in line 9. Line 10 provides the value of the required **attribute** of **import**, **schemaLocation.** Thus, 3 lines (4, 9, and 10 replace one Ada with and one renames statement. Lines 15 through 24 create the generation enumerated type, which is based upon or derived from the string type (line 16) and only includes the enumerated values (lines 17 through 22). The **complexType Person_Name_W_Generation_Type** is created (lines 25 through 34) by **extension** of the parent **Person_Name_Type** (line 27)and addition of the **generation element** and **type** (line 29). This is essentially the same as extending a tagged type in Ada.

XML Types

XML Primitive datatypes can be divided into four groups: numbers, date & time, strings, and binary data. Like Ada, XML has two floating point types. The one named float is a single-precision 32-bit floating point type. [IEEE 754-1985]. The other named double is a double-precision 64-bit floating point type [IEEE 754-1985]. XML has a decimal type, which differs from Ada's in the limitation that the exponent of 10 has to be less than or equal to zero and the precision is specified as arbitrary. Like Ada, "there must be at least one digit to the right and to the left of the decimal point which may be a zero." The integer types described in Table 1 below are all consider to be "derived from type integer, which is itself derived from type decimal by fixing the value of fractionDigits to be 0."

Table 1. XML Integer Types

Type	minInclusive	maxInclusive
integer	?	?
nonPositiveInteger		0
negativeInteger		-1
long	-(2**64)/2	(2**64)/2-1
int	-(2**32)/2	(2**32)/2-1
short	-(2**16)/2	(2**16)/2-1
byte	-(2**8)/2	(2**8)/2-1
nonNegativeInte-ger	0	?
unsignedLong	0	2**64-1
unsignedInt	0	2**32-1
unsignedShort	0	2**16-1
unsignedByte	0	2**8-1
positiveInteger	1	?

Except for range constraints, the integer types can be readily mapped to Ada. As shown below other XML languages provide the standard numeric functions.

Table 2 describes the date and time types and their formats. As is true with much of XML, these are based on ISO standards and could be based on Ada.Calendar. Type second is a decimal type, which obviously could be modeled with Ada.Decimal.

Table 2XML Time & Date Types

XML Type	XML Description & Format	Type & Standard
duration	is a six-dimensional space where the coordinates designate the Gregorian year, month, day, hour, minute, and second components respectively. These components are ordered in their significance by their order of appearance i.e. as year, month, day, hour, minute, and second.	defined in § 5.5.3.2 of [ISO 8601],
duration.second	Seconds component allows an arbitrary decimal.	
dateTime	is the space of Combinations of date and time of day values. CCYY-MM-DDThh:mm:ss	5.4 of [ISO 8601].
time	represents an instant of time that recurs every day. hh:mm:ss.sss with optional following time zone indicator.	time of day values as defined in § 5.3 of [ISO 8601].

Table 2XML Time & Date Types

date	CCYY-MM-DD. An optional following time zone qualifier is allowed	is the set of Gregorian calendar dates as defined in § 5.2.1 of [ISO 8601]
gYearMonth	represents a specific gregorian month in a specific gregorian year. CCYY-MM. An optional following time zone qualifier is allowed.	is the set of Gregorian calendar months as defined in § 5.2.1 of [ISO 8601].
gYear	represents a gregorian calendar year. CCYY. An optional following time zone qualifier is allowed	is the set of Gregorian calendar years as defined in § 5.2.1 of [ISO 8601].
gMonthDay	is a day of the year such as the third of May. MM-DD. An optional following time zone qualifier is allowed	is the set of calendar dates, as defined in § 3 of [ISO 8601]
gDay	gDay is a gregorian day that recurs, specifically a day of the month. DD. An optional following time zone qualifier is allowed	is the set of calendar dates as defined in § 3 of [ISO 8601].
gMonth	is a gregorian month that recurs every year. MM--. An optional following time zone qualifier is allowed.	set of calendar months as defined in §3 of [ISO 8601].

Strings: XML includes string types similar to those found in Ada and many specialized types of strings that are specific to the operation of XML. These XML specific strings will not be discussed further. The XML string type has the following constraining facets: length, minLength, maxLength, pattern, enumeration, and whiteSpace. Character Set should be added, since it can be specified for the entire document or locally. Length, minLength, maxLength control the length of a string. If the length is set to a fixed value, the result is a standard Ada string. If the length of a string is not specified, then it is any nonNegativeInteger. Thus, it can be modeled as an Unbounded_String. However, the length of the XML string could be greater than Natural'Last. Specification of maxLength results in the equivalent of an Ada Bounded_String. Specification of minLength without maxLength would result in a string that had at least a minimum number of characters. Specification of both minLength and maxLength would result in a string where the range of its length was specified. This might be a useful construct for the next version of Ada.

The use of the pattern facet was shown above in the creation of the decimal type example, **“Fee_Type”.** The pattern facet is a generalized construct of what is presently accomplished by the Ada.Text_IO.Editing for decimals.

The XML enumeration facet is employed to create the equivalent of Ada enumerated types. However, the XML enumerated types are really a group of unique strings. An example of an XML enumerated type was shown above in the creation of **“Generation_Type”** in the Person_Name_w_Generation schema.

The whiteSpace facet has three values preserve, replace, and collapse. Preserve, as expected, means that the values of the string remain unchanged. Replace means Control characters HT (tab), LF (line feed) and CR (carriage return) are replaced with Space. Collapse means that after the 3 control characters are replaced by space, contiguous sequences of spaces are collapsed to a space and leading and trailing spaces are removed.

Binary types: There are 3 binary types: **boolean**, **hexBinary**, and **base64Binary**. **Boolean** is the same as in Ada. The **hexBinary** type is the standard hexadecimal concatenation of two characters ([0-9a-fA-F]) to represent 0 to 255.

The following is basically a quotation from RFC 2045 (9), except for purposes of clarity the material in parenthesis has been added. The encoding process for **`base64Binary`** “represents 24-bit groups of input bits (binary) as output strings of 4 (US-ASCII) encoded characters. Proceeding from left to right, a 24-bit input group is formed by concatenating 3 8-bit input groups. These 24 bits are then treated as 4 concatenated 6-bit groups, each of which is translated into a single digit in the base64 alphabet. When encoding a bit stream via the base64 encoding, the bit stream must be presumed to be ordered with the most-significant-bit first. That is, the first bit in the stream will be the high-order bit in the first 8bit byte, and the eighth bit will be the low-order bit in the first 8bit byte, and so on.”

“Each 6-bit group is used as an index into an array of 64 printable characters (’A’..’Z’,’a..’z’,’0’..’9’, +,/) with ’A’ having the value of 0 and ’/’ have the value of 63. The ’=’ serves “to signify a special processing function”. The character referenced by the index is placed in the output string. This encoding results in only about 33 percent increase in size versus the unencoded data.”

Prefixes:

The two schema shown above were deliberately made to look like Ada. Since only one xmls (name-space) prefix can be a string with zero characters, following the Ada convention of not withing Ada, the namespace for XML itself did not have a prefix and the two new schemas were prefixed. Quite often, this is not the case. In fact, the default for XMLSpy (10), the tool used to create part of this paper and the standard itself is to use the prefix xs. Thus, the beginning of Person_Name schema, shown above, would be as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="file://C:\SIGAda\2001\XML\Person_Name.xsd"
xmlns="file://C:\SIGAda\2001\XML\Person_Name.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="unqualified" attributeFormDefault="unqualified">
  <xs:annotation>
    <xs:documentation>Person_Name to be used as part of other schemas and is
      mirrored by an equivalent Ada package 2001 Oct. 15.
      DICOM.Person_Names_Bd.Ads
    </xs:documentation>
  </xs:annotation>
  <!--Steps to create Person_Name_Type-->
  <xs:simpleType name="Given_Name_Type">
    <xs:restriction base="xs:string">
      <xs:maxLength value="50"/>
    </xs:restriction>
    <!--50 characters are needed because it is possible for a person to have
      only one name-->
   </xs:simpleType>
</xs:schema>
```

In terms of Ada experience, the "xs:" is clutter and in particular the prefixing of a common type, string.

XSL & XSLT

Unlike HTML documents, XML documents by themselves do not contain formatting information. XSL (11), the “Extensible Stylesheet Language” is a powerful, comprehensive formatting language that allows XML documents to be displayed, transformed from one schema to another or into entirely different forms, such as HTML pages, WML cards, or PDF files. XSLT (12) is a language for transforming XML documents into other XML documents, which is designed for use primarily as part of XSL. “In addition to XSLT, XSL includes an XML vocabulary for specifying formatting. XSL specifies the styling of an XML document by using XSLT to describe how the document is transformed into another XML document that uses the formatting vocabulary.”

"A transformation expressed in XSLT describes rules for transforming a source tree into a result tree. The transformation is achieved by associating patterns with templates. A pattern is matched against elements in the source tree. A template is instantiated to create part of the result tree. The result tree is separate from the source tree. The structure of the result tree can be completely different from the structure of the source tree. In constructing the result tree, elements from the source tree can be filtered and reordered, and arbitrary structure can be added."

Since the W3C Recommendation for XSL (11) is 400 pages, it is impossible to provide a complete description. However, the most important fact is that it employs XML syntax. A simple description of the breath of XSL is provided by the material covered in Integration test 2 of the Implementation Report (13) This test included some extended XSL “features that are not necessarily part of the basic level of conformance”.

Formatting structures in Integration test 2:

blocks with various formatting properties including those representing different levels of titles (some including auto-numbering) and paragraphs (including indented block quote paragraphs)

external graphics

bulleted and enumerated lists including nested lists

generated table of contents including leaders and generated page numbers

running headers and footers including use of markers (an extended level feature)

multiple page layouts, folio formats (an extended level feature)

page master alternation (an extended level feature)

spanning (an extended level feature)

various in-line font changes

hyphenation and justification

and various combinations of display spacing

XSL includes both screen objects, which can be scrolled and page objects, which can be printed. The breath of the above tests, perusal of the comprehensive and detailed XSL table of contents, and the capacity for both screen display together with and sophisticated document processing indicate sufficient functionality to display and output Microsoft Office or similar products. The addition of vector graphics (see below) provides the rest of the required functionality.

The standard description of XSL formatting is to have an XML document transformed by a combination of XSLT and the XML Path Language, XPath, (14) into a new tree. XPath is a language for addressing and querying the content of XML documents. This new tree consists of formatting objects and properties, which is then processed by an XSL-compliant rendering-engine to produce the specified output. A very simple example of a section of an XSL stylesheet, which specifies the font-size, indentation, and alignment for paragraphs, named “para” is shown below.

```
<xsl:template match="para">
 <fo:block font-size="12pt" space-before="1pt"
      text-align="justified">
  <xsl:apply-templates/>
 </fo:block>
</xsl:template>
```

XSL resembles a true programing language in that it includes functions and the equivalent of if and case statements.

The following is an example of an **if** statement from XSLT (12).

```
1 <xsl:template match="namelist/name">
  2 <xsl:apply-templates/>
    3 <xsl:if test="not(position()=last())">, </xsl:if>
4 </xsl:template>
```

Line 1 is the instruction to find the name elements in a list of names. Line 2 is the instruction to do this. Line 3 is the **if** statement, which includes the boolean **test.** The boolean condition is that the name is not the last name. The action is to insert a ’,’ after the name except for the last name in the namelist. The **</xsl:if>** is the equivalent of an Ada end if. The fourth line ends the template statement, which is again similar to Ada syntax. The result is a comma separated list: More complex if tests can include assignment of attributes of the element in the if test.

The following is an example of a **choose** statement from XSL. It assumes that a block of text, paragraph, has already been found by a **template match**.

```
1 <fo:block text-align-last="justify">
  2 <xsl:attribute name="margin-left">
    3 <xsl:choose>
      4 <xsl:when test="$level=1">0mm</xsl:when>
      5 <xsl:when test="$level=2">1mm</xsl:when>
      6 <xsl:when test="$level=3">2mm</xsl:when>
      7 <xsl:otherwise>3mm</xsl:otherwise>
    8 </xsl:choose>
  9 </xsl:attribute>
10 </fo:block>
```

Line 1 states that the text in this block will be justified.

Line 2 selects the margin-left **attribute** as the block property to be assigned a value.

Line 3 starts the equivalent of an Ada case statement.

Lines 4, 5, and 6 describe the test condition, which just as in the **if** statement is a boolean. The condition is the paragraph level. The indentation of the left margin is specified in millimeters, mm.

Line 7 employs **otherwise** with the same meaning as in Ada.

Lines 8, 9, and 10 end the nesting.

The rules covering the **choose** statement (12) differ from those of the Ada case statement. “The content of the first, and only the first, xsl:when element whose test is true is instantiated. If no xsl:when is true, the content of the xsl:otherwise element is instantiated. If no xsl:when element is true, and no xsl:otherwise element is present, nothing is created.”

XSL also includes functions which return values. However, the syntax unfortunately neither includes the word function nor a formal actual format. The XSL standard numeric functions (11) include: floor, ceiling, round, min, max, and abs. The operators include: +, -, *, div, and mod.

“The conventions used (11) for the names of XSL elements, attributes, and functions are as follows: names are all lowercase, hyphens are used to separate words, dots are used to separate names for the components of complex datatypes, and abbreviations are used only if they already appear in the syntax of a related language such as XML or HTML."

Tables & XSL

By separating the details of the formatting of tables from the content, the combination of XSL and XML greatly increase the readability of the content of tables and other complex objects. Extensible Stylesheet Language (XSL) Version 1.0, Section 6.7.1.1.1. provides a XML description of the content of a simple table with 3 columns and only one row. This table is displayed including being centered and indented by an XSL Stylesheet. Unfortunately, no schemas including one for this example were provided in this standard. The actual stylesheet comprises about one and one-half pages of the standard. The XML description of the table is shown directly below.

```
1 <doc>
  2 <table>
    3 <caption><p>Caption for this table</p></caption>
    4 <tgroup cols="3" width="325pt">
      5 <colspec colwidth="100pt"/>
      6 <colspec colwidth="150pt"/>
      7 <colspec colwidth="75pt"/>
      8 <tbody>
        9 <row>
          10 <entry><p>Cell 1</p></entry>
          11 <entry><p>Cell 2</p></entry>
          12 <entry><p>Cell 3</p></entry>
        13 </row>
      14 </tbody>
    15 </tgroup>
  16 </table>
17 </doc>
```

Lines 1 and 2 indicate that a table is contained in a document. Line 3 gives the contents of the caption and that it is a p element, as are all of the strings in this example. In a real example, the caption and cells would usually not have the same format and the justification of the caption probably would be centered and the that of the cell contents would depend on their type, such as strings being left and numbers right. Line 4 begins the description of the content of the table. It has 3 columns with a total width of 325 points. Lines 5, 6, and 7 give the widths of the individual columns. Lines 8 through 14 describe the contents of the table, which in this case has only one row. Lines 15 through 17 end the nesting.

CSS

Cascading Style Sheets, CSS (15), are a popular way to control the appearance of HTML documents and can be used with XML. Unfortunately, Cascading style sheets have their own syntax.

XPath

The XML Path Language, XPath (14), is a language for addressing parts of an XML document. It was designed to provide a common syntax and semantics for functionality shared between XSL Transformations, XSLT. XPath is employed to test whether or not a node matches a pattern. The basic facilities for manipulation of strings, numbers and booleans are provided by XPath. "XPath uses a compact, non-XML syntax to facilitate use of XPath within

URIs and XML attribute values." "The primary syntactic construct in XPath is the expression." An expression is evaluated to yield an object, which has one of the following four basic types: node-set (an unordered collection of nodes without duplicates), boolean, a floating-point number, and a string. XPath is also used with XPointer.

XLink and XPointer

XML includes two more languages based on XML syntax. The first is the XML Linking Language, XLink (16), which describes both the unidirectional hyperlinks of today's HTML, as well as more sophisticated links. The second is The XML Pointer Language, XPointer (17), which is a companion standard to Xlink and is a W3C candidate recommendation. XPointer describes mechanisms for addressing particular parts of a document. XPointer is "to be used as a fragment identifier for any URI-reference that locates a resource of Internet media type text/xml or application/xml." It allows for traversals of a document tree and choice of its internal parts based on various properties, such as element types, attribute values, character content, and relative position.

Since only XML, schemas, and XSL are the inputs to produce a formatted document, it may be possible to build an entirely Ada based display system. An XSL-compliant rendering-engine that was written in Ada could be combined with other Ada software that had the functionalities of XSLT, XPath, and XPointer. This functionality could be extended to add XSLT, XPath, and XPointer to its inputs.

SVG

Scalable Vector Graphics, SVG (18,19), is an XML application used to describe 2D vector graphics including animation, text and raster images. This enables vector graphics to be defined solely in XML. SVG is well designed, complete, and powerful. However, SVG is based on a DTD; SVG overlaps both XSL and CSS. The SVG is a large standard (18), 617 pages total. Appendix C of Scalable Vector Graphics (SVG) 1.1 Specification (19) is "The OMG IDL for the SVG Document Object Model definitions." This IDL specification (20) would be a good place to start in the creation of the equivalent Ada specification.

"SVG shares many of its styling properties with CSS and XSL. Except for any additional SVG-specific rules explicitly mentioned in this specification, the normative definition of properties that are shared with CSS and XSL is the definition of the property from the CSS2 specification (24)." The following properties are defined in both SVG in XSL, font properties: font, font-family, font-size, font-size-adjust, font-stretch, font-style font-variant, and font-weight; text properties: direction, letter-spacing, text-decoration, unicode-bidi, and word-spacing; and other properties for visual media: clip, color, overflow, and visibility. Given the overlap in functionality between SVG and XSL, a reasonable approach is to limit the use of SVG to actual vector graphics and to have XSL position and scale the graphic. This places the SVG as a foreign object in the XSL or the XSL as a foreignObject in the SVG.

Adobe® SVG Viewer 3 (21) has been made available in 15 languages on multiple platforms: Win 98 - XP, Mac 8.6 - 9.1, and Mac 10.1. SVG in Mozilla project (22) is available on Windows, MacOS, and Linux. Similar functionality for SVG on any Java 2 system is available from other corporations or organizations (23) including: Apache, CSIRO, Ionic, and X-Smiles. There is also a Java language binding for the SVG Document Object Model definitions (24). This binding consists of 160 small files all of which are 1 or 2 kilobytes except for one of 3 and one of 4 kilobytes for a total of 72 kilobytes; it would be feasible to produce an Ada implementation, which should be considerably faster. SchemaSoft (25) demonstrated that it was possible to build an XSL implementation based on SVG.

VML

Vector Markup Language, VML (26), is a Microsoft creation, which is concise (50 pages) and based on a schema. No mention of XSL was made in this standard.

XFORMS

XForms (27) is an XML based new platform-independent markup language for online interaction between a remote entity, such as a user, and an XForms Processor. XForms contains three schema, a 10 page long one for XForms itself and two small, less than one page long schema that are for XLink and XML Events. Both of these schema are normative parts of XForms and not normative with respect to themselves. XForms conform to either being Basic or Full. Basic Forms are described in a subset of the XForms schema and are "suitable for devices with limited computing power, such as mobile phones, handheld computers, and appliances." The Full conformance level requires implementation of the entire schema. XForms is a means for display and inputting data, without describing how the data will be presented. The data itself can be exported as XML, which simplifies its processing. Because XML is strongly typed, data can be validated on the client-side at the point of entry against the range and other facets described in a schema.

Since the presentation has been separated from the content, an XForm can be used on multiple devices. XForms includes “declarative event handlers such as setFocus, message, and setValue that cover common use cases.” These are equivalent to Ada procedures.

Multiple individual forms can coexist in the same XML page, because each is identified by a model element. An XForms bind element has attributes that: point to a model and can: associate a schema datatype with a field, restrict a value from changing by declaring it read-only, require that a value exists before the data is submitted, indicate that an item in a model is relevant to the rest of the model and thus is suitable for submission, indicate that a field is the result of a calculation performed on other data, describe the conditions required for the data to be considered valid, and indicate both the minimum and maximum number of allowed child elements. One interesting constraint on a model is, “Under no circumstances may more than a single concurrent submit process be under way for a particular XForms Model.” This behavior of models could be represented by an Ada protected type.

XForms includes user Interface controls. These include: free-form data entry of one or more lines; declaring an item to be secret, which results in the entered data being represented by stars or the equivalent; placing text on the screen or other output device based on some internal value; acquiring binary data from an external source or inputting a file; specifying whether only one or one or more items can be selected; specifying the range and step size (delta) of data that are to be entered, declaring that a button will be used for user-triggered actions; and permitting the data in the form to be submitted. These user interface controls have a “class” attribute that provides a tag or format name. These formats can be rendered by an XSL stylesheet; a Cascading Style Sheet, CSS; or even a speech based representation stylesheet. Single or multiple selections could be made by the rendering of a common presentation such as a listbox, checkbox, collection of radio buttons, or a pull-down menu.

XForms has Error Indications, which include errors, such as schemaConstraintsError, traversalError, and invalidDatatypeError. Although these are similar to Ada exceptions, they primarily have to do with event notification.

Ada and XML

XML, Schema, & XSL Roles

XML files contain the data. Schemas are the specifications of types and objects and are thus similar to the type and object declarations in Ada specifications. XSL describes how the output of the data should look and how (XSLT) it should be translated. The combined action of these two languages is some what equivalent to any Ada body. XML, schemas, and XSL employ the same syntax in a similar manner to Ada specifications and bodies have the same syntax. The similarity between Ada and the XML languages includes similar visibility and scoping rules, strong typing, the ending of statements, and single inheritance.

What should be created

Although the similarity of XML to Ada indicates that it would be technically feasible to readily do virtually everything, if not absolutely everything, described above for the XML family of languages in Ada, this is presently not economically feasible. However, it would be feasible to reduce the languages employed for an application with a significant graphical user interface component to effectively two, Ada and XML. This requires the development of two major translation tools: one to translate Ada data types and objects in a specification into an XML schema; and the other to translate XML data types, elements, and attributes in a schema into an Ada specification.

The Ada to XML approach is suitable for conventional software development, such as object oriented development. This tool could employ ASIS. The XML to Ada approach is suitable for storey-boarding and/or the use of graphical (4th generation) web design tools to create screens including the responses to human interaction and to create print outputs. The second tool could either be Ada based or an XSLT application. In the development of many applications, the language used first in development will depend on the nature of specific parts of the application.

Present Ada screen design tools should be enhanced to produce output suitable for an XML environment. This would substitute XML files for Windows Resource files. One approach to implementing this change would be to replace the present Windows binding by an initial binding to SVG, which is presently available on multiple platforms. A longer term approach is to make a binding to XSL, which itself can be partially implemented with SVG. In any event, the XSL design independently of the syntax can serve as a good source for enhancements and improvements to present Ada GUI tools.

A schema for Ada would permit the enhancement of XML based tools to be useful in an Ada software development environment. If the programing editor and documentation tools were both based on XML, it would be possible to create hypertext linkages between the sources and the documentation. Since these XML based development tools will be used in the future with other languages, Ada would greatly benefit by maximizing the availability of software development tools.

An XForms Processor that worked with an Ada based server to render XForms would permit user interaction. This interaction would include the exchange data in XML format. It could be used together with an Ada schema as the basis of a powerful, simple, portable Ada software development environment. After this environment is created, much of its software could be reused for commercial off the shelf applications.

An Ada-XML operating system:

One of the first steps in development is the elimination of unnecessary requirements. Less is often better. Most of the present PC users would be happier if their computer behaved like an appliance. Two significant virtues of DOS were that, unlike a conventional operating system, no extensive recovery was required if the computer was improperly shut down; and the time to boot the system was minimal. The over-complexity of Windows is mirrored on the need to add new keys to the keyboard and the required use of a mouse for text based programs, such as word-processors and spreadsheets. The incredible increase in computing power and decrease in hardware costs has hidden the inefficiencies and complexities of present commercial software.

A document centric system based on XML including schemas, XSL(T), etc. could produce a truly integrated, extensible product that included text, intelligent tables (spreadsheets), images, graphs based on vector graphics, and a database. The file format would be an XML version of a MHT (Web archive single) file (28). The file-manager should be based upon an XML database. Basing a system on a minimum number of extensible reusable components simplifies both the development and the learning of the system.

Compared with the present dominant Microsoft operating systems and applications, a well designed equivalent based on the combination of Ada and XML should be much cheaper, easier to use, open to extension by third parties, inherently multiple sourced, and very reliable. The work by third parties should result in improvements and extensions to the core programs. These could be commercialized and added to the original products (29,30). As previously stated, ASIS offers a means to equitably compensate the developers of Ada software.

Since the rendering would be controlled by XSL and SVG, the device drivers for graphics cards and printers could simply provide XML data. This non-proprietary data would be rendered by microprocessors included in the graphics cards and printers. This is similar to the use of PostScript for the same purpose.

XForms can be used to create the pull-down menus for both the standard commercial applications and the applications specific to the operating system. The present Microsoft object linking and embedding (OLE) approach to compound documents has the deficiency that multiple copies of data from one program have to be embedded in another. For instance a report produced by a word-processor could include multiple graphs from a spreadsheet. In XML, a reference to the spreadsheet’s schema has to be included and only one copy of the spreadsheet data has to be include in the document file produced by the word-processor. The style sheet can control what is shown in the document.

The present technology employed with Ada compilers could be extended to control the libraries for XML languages. In the case of memory limited systems, if the Ada environment controlled the content of the schema and XSL files, it could optimize out dead code and produce smaller documents. Conversely, Ada compilers that interact with XML could use information present in the XML and XSL files to optimize away dead code.

Conclusions:

The Worldwide Web Consortium has applied a Darwinian approach to standards development. This has resulted in a comparatively rapid standards development process, which has both facilitated their adoption and resulted in an alphabet soup of overlapping standards. The quality of the standards that employ XML syntax is good and the Ada standard could benefit from at least adopting the practice of including in the description of a datatype a reference to the appropriate ISO standard. Harmonization of Ada where relevant with XML is also a reasonable approach in the development of future Ada standards. The similarities of Ada and XML will permit a synergistic combination, which could compete in and ultimately dominate the commercial marketplace.

References

1. World Wide Web Consortium (W3C), Massachusetts Institute of Technology, Laboratory for Computer Science, 200 Technology Square, Cambridge, MA 02139, USA; www.w3.org

2. ISO 8879:1986, Information processing -- Text and office systems -- Standard Generalized Markup Language (SGML), ISO Central Secretariat: International Organization for Standardization (ISO) 1, rue de Varembé, Case postale 56 CH-1211 Geneva 20, Switzerland; www.iso.ch

3 HyperText Markup Language Home Page; www.w3.org/MarkUp/

4. Extensible Markup Language (XML) 1.0 (Second Edition), Editors: Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, and Eve Maler, Copyright © 2000 W3C; www.w3.org/TR/2000/REC-xml-20001006.

5. Guide to the W3C XML Specification ("XMLspec") DTD, Version 2.1, major contributors to the DTD design: J. Bosak, T. Bray, D. Connolly, E. Maler, G. Nicol, C. M. Sperberg-McQueen, L. Wood, and J. Clark; www.w3.org/XML/1998/06/xmlspec-report-v21.htm

6. XML Schema Part 0: Primer, David C. Fallside, Editor; W3C® (2001); www.w3.org/TR/2001/REC-xmlschema-0-20010502/

7. XML Schema Part 1: Structures, H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn; Editors, W3C® (2001); www.w3.org/TR/2001/REC-xmlschema-1-20010502/

8. XML Schema Part 2: Datatypes, P. V. Biron and A. Malhotra; Editors, W3C® (2001); www.w3.org/TR/2001/REC-xmlschema-2-20010502/

9. N. Freed and N. Borenstein. "RFC 2045: Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies." (1996); http://www.ietf.org/rfc/rfc2045.txt

10. XMLSpy, Altova, Vienna, Austria and Boston, MA. www.xmlspy.com/

11. S. Adler, A. Berglund, J. Caruso, S. Deach, T. Graham, P. Grosso, E. Gutentag, A. Milowski, S. Parnell, J. Richman, and S. Zilles, “Extensible Stylesheet Language (XSL) Version 1.0", W3C® (2001); www.w3.org/TR/2001/REC-xsl-20011015/

12. XSL Transformations (XSLT) Version 1.1 W3C Working Draft 24 August 2001, J. Clark, Editor; www.w3.org/TR/2001/WD-xslt11-20010824/. Note, this draft will be replaced by XSLT 2.0.

13. Implementation Report -- Implementations of XSL Formatting Objects Specification, XSL Working Group (2001) www.w3.org/2001/08/28-XSL-PR-IR.html;

14. XML Path Language (XPath) Version 1.0 W3C Recommendation, Editors: J. Clark and S. DeRose (1999); www.w3.org/TR/1999/REC-xpath-19991116

15. Cascading Style Sheets, level 2, CSS2 Specification, W3C (1998); Editors: B. Bos, H.W. Lie, C. Lilley, I. Jacobs; www.w3.org/TR/1998/REC-CSS2-19980512

16. XML Linking Language (XLink) Version 1.0, W3C, Editors S. DeRose, E. Maler, and D. Orchard (2001); www.w3.org/TR/2001/REC-xlink-20010627/

17. XML Pointer Language (XPointer) Version 1.0, W3C, S. DeRose, E. Maler, and R. Daniel Jr., Candidate Recommendation (2001); www.w3.org/TR/2001/CR-xptr-20010911/

18. Scalable Vector Graphics (SVG) 1.0 Specification, W3C, Editor: J. Ferraiolo (2001); www.w3.org/TR/2001/REC-SVG-20010904/

19.Scalable Vector Graphics (SVG) 1.1 Specification, W3C Candidate Recommendation 30 April 2002, Editors: D. Jackson, J. Ferraiolo, and J. Fujisawa (2002); www.w3.org/TR/SVG11/

20. Scalable Vector Graphics (SVG) 1.1 Specification, Appendix C: IDL Definitions, www.w3.org/TR/2001/REC-SVG-20010904/idl.zip

21. Adobe SVG Viewer Download Area, www.adobe.com/svg/viewer/install/main.html

22. Scalable Vector Graphics (SVG); http://www.mozilla.org/projects/svg/

23. SVG Implementations; http://www.w3.org/Graphics/SVG/SVG-Implementations

24. Ref. 18, Appendix D: Java Language Binding; www.w3.org/TR/2001/REC-SVG-20010904/java-binding.zip.

25. SchemaSoft: http://www.schemasoft.com/

26. B. Mathews, D. Lee, B. Dister, J. Bowler, H. Cooperstein, A. Jindal, T. Nguyen, P. Wu, and T. Sandal, Vector Markup Language (VML) World Wide Web Consortium Note 13-May-1998 Submission to the World Wide Web Consortium(1998); www.w3.org/TR/1998/NOTE-VML-19980513

27. XForms 1.0, W3C Working Draft 18 January 2002, Editors: M. Dubinko, J. Dietl, L. L. Klotz Jr, R. Merrick, T. V. Raman; www.w3.org/TR/2002/WD-xforms-20020118/

28. J. Palme, A. Hopmann, and N. Shelness, RFC 2557, MIME Encapsulation of Aggregate Documents, such as HTML (MHTML), The Internet Society, (1999) ftp://ftp.isi.edu/in-notes/rfc2557.txt

29. R. C. Leif, “SIGAda '98, Workshop: How do We Expedite the Commercial Use of Ada?." Ada letters XIX, No 1 pp. 28-39 (1999).

30.R. C. Leif, "Ada Developers Cooperative License (Draft) Version 0.3", Ada letters XIX, No 1 pp. 97-107 (1999).

Supplementary Information:

31. P. Walmsley, Definitive XML Schema: New Jersey: Prentice Hall, ISBN 0-13-065567-8; 2002.

32. Antenna House XSL Formatter version 1.1E; www.antennahouse.com/xslformatter.html

33. Apache XML Project, FOP; http://xml.apache.org/fop/

34. RenderX is the first company in the World to roll out a commercial grade XSL FO rendering engine; www.renderx.com

35. W3C Documents Formats Domain, The Extensible Stylesheet Language (XSL); www.w3.org/Style/XSL/

36. P. Grosso and N. Walsh, XSL Tutorial, Concepts and Practical Use Monday, 12 June 2000; http://www.nwalsh.com/docs/tutorials/xsl/

37. XSLT Developer's Guide, Microsoft, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/xmlsdk/htm/xslt_starter_78s4.asp