xCalendar: An XML Schema for Calendaring and Scheduling in e-Business

Rajiv Hiremagalur[1], Billy Lim[2]

[1,2] Applied Computer Science Department, Illinois State University
Normal, IL 61790-5150

Abstract

As the Internet becomes more and more ubiquitous, there is a growing need for consumers and businesses to exchange information electronically over the Internet. For example, when businesses merge or when new dealers are added to a business, the parties involved need to share and exchange information. Calendaring is a representative sample application where there is a need for exchanging information across business units and enterprises. A standard format for calendaring allows all parties to easily view each other's schedule and arrange meetings. The challenge here is to find a technology that is platform and language independent to serve as the base for developing standard formats to exchange information.

In recent years, XML has already been shown to be one such technology that can be used in the above scenarios. However, loosely defined XML documents are not very useful and cannot be used to address the aforementioned problem. The documents must be structured based on formats developed with consensus. XML Schema, a recent W3C recommendation, is one such framework that allows one to define structures on XML documents. This paper describes an effort to use XML and XML Schema technologies to develop *xCalendar*, an exchange format for calendaring and scheduling using XML Schema. The constraints expressed and enforced in xCalendar are also discussed. In addition, although XML Schema is very powerful, it has its limitations. The paper also describes the use of XSLT and JAXP technologies to overcome the limitations of XML Schema to make xCalendar even more flexible and powerful.

1. Introduction

The need to exchange information in businesses dates back to the early days of data processing. For example, in the health care industry, patient records, insurance claims, and schedules are exchanged between the health care providers, patients, and insurance agencies. In the airline industry, the flight plans, weather information, and passenger information are exchanged between the airline companies, FAA (Federal Aviation Administration), ATC (Airline Training Council) and NWS (National Weather Services). Lastly, in the manufacturing industry, purchase orders, invoices, quotes, and service plans are exchanged between the manufacturers, dealers, and consumers. With the advent of the Internet, there is a growing need for consumers and businesses, regardless of the industry they belong to, to exchange information electronically.

Today this information is exchanged using proprietary ASCII or binary formats. Some businesses still rely on paper format for information exchange. This means of exchange is inflexible and is not amenable to change, whether it has to do with mergers, new dealers, customers, or agents. When such a change occurs, the proprietary formats utilized can easily cause conflict. The information exchange needed should be flexible; it must not only be easy to accept a change, it must also be easy to propagate the change too. In addition, when information is exchanged across expansive networks, there is always a chance for errors. Error checking will have to be performed while sending and receiving information. If this error checking code is duplicated then it will be difficult to propagate change.

XML (eXtensible Markup Language) [1] is a language that allows structured representation of information. It also allows the definition of custom tags, using which information can be structured as elements that are intuitive to businesses and that business applications can easily process. XML dictates some very simple rules to construct elements. Although these rules are simple, they are rigid. The specification does not allow the XML parsers to relax any rule. This deterministic approach helps to develop a strong foundation on which other technologies can be built.

In today's businesses, it is possible that business groups within the same vertical industry define different custom tags for the same thing when using XML technology. In this scenario, the purpose of a common format is defeated. To better the exchange of information, there is a need for a way in which one can define and enforce this common format based on common consensus among the businesses involved. A common format permits the exchange of information consistently within business units of an enterprise and across enterprises.

Various technologies exist to allow the structure of an XML document to be defined so that XML documents exchanged make sense. For many years, the most popular technology has been DTD (Document Type Definition). DTD is an EBNF-based language with the specific purpose of defining XML document structure. There are, however, many shortcomings of DTD. First, it is decidedly not an XML document and as such, XML processing programs (e.g., parsers) cannot process a DTD easily. Second, there is no data typing in DTD and thus all the contents are treated as strings. This serious shortcoming puts the difficult task of constraint enforcement in the hands of the application developers to verify that the contents of the XML documents are indeed valid. Finally, DTD does not allow element names to be reused on the same document. This lack of support for namespaces unnecessarily constraints the use of element names and forces the designers to choose names that may not be intuitive or natural. All these have led to the development of XML Schema.

2. XML Schema

XML Schema [2], a recent W3C recommendation, is a framework that allows one to define structures on XML documents. It permits one to express shared vocabularies and allows machines to carry out rules made by people. This provides a means for defining the structure, content, and semantics of XML documents. In addition to satisfying the needs of a common exchange formats, XML and XML Schema offer numerous other advantages. They include automated change propagation, strong support for XML in the developer commun ity, automated error checking, and an incredibly rich set of APIs for XML-based processing and messaging. For all these reasons, XML has become the lingua franca of data exchange, replacing the complex and antiquated EDI (Electronic Data Interchange) syste ms.

Because of its flexibility and power, XML Schema is emerging as a replacement for DTD. The reasons for this shift are because XML Schema is written in XML and thus it is extensible to future additions, it is much richer and more useful than DTDs, it supports more than 44 data types, it can create custom data types, and it supports namespaces.

The main purpose of XML Schema is to define the legal building blocks of XML documents. XML Schema defines the following:

- Elements that can occur in a document
- Attributes that can occur in an element
- Child elements of an element
- Sequence in which child elements can appear
- Number of child elements
- Data types of the elements and attributes
- Default values of elements and attributes

An example of an XML schema is shown below in Figure 1.

Schema

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="Person">
   <xsd:complexType>
      <xsd:sequence>
          <element name="FirstName " type="xsd:s tring"/>
          <element name="LastName " type="xsd:string"/>
          <element name="Phone" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

Instance

```
<Person>
<FirstName >BILL</FirstName>
<LastName >JOY</Last Name >
<Phone> 800-412-3569 </Phone>
</Person>
```

Figure 1: Simple and complex elements in XML Schema. An XML schema and an instance of the schema are shown.

A person's information is represented in the XML instance document. The corresponding schema depicts the structure of XML documents to be instantiated from this schema. Note that two types of elements can be defined in an XML schema -- simple and complex elements. Simple elements are those that do not have any child elements whereas complex elements are the ones that do have child elements. In Figure 1, FirstName, LastName, and Phone are simple elements and Person is a complex element.

Custom data types can also be created from base data types by restricting one or more facets. For example, a new data type can be created from the String base data type by restricting the range of values, restricting the length of the value, or restricting the pattern the value has to match. This allows a large collection of constraints to be specified and enforced so that XML documents can be validated automatically by XML Schema.

3. xCalendar: Basic Structures

To examine XML Schema is detail and to develop a proof of concept for this burgeoning technology, this project investigates calendaring systems, where calendaring applications such as Microsoft Outlook, Lotus Notes, and the ones typically found in PDA and cellular Phones are faced with the challenge of interoperability. The existing ASCII standards like vCalendar [3] and iCalendar [4] have not been completely successful in establishing a common exchange format.

To address the interoperability problem of calendaring systems, this paper proposes the use of *xCalendar* (for XML Calendar), an XML Schema specification with external constraints. xCalendar is structured in similar way when compared to the vCalendar specification. The typical structure of calendaring information is shown below.

```
Calendar
   +--- Event
          +--- Summary
          +--- Description
          +--- Attendees
          +--- Location
          +--- Reminder Information
                 +--- Audio Alarm
                        +--- Audio Content
                        +--- Run Time
                        +--- Snooze Time
                        +--- Repeat Count
                 +--- ...
          +--- Recurrence Information
                 +--- Recurrence Number
                 +--- Recurrence Rule
                        +--- Daily
                        +--- Weekly
                        +--- Monthly By Day
                        +--- ...
                 +--- Recurrence Date
                 +--- ...
          +--- Start Date and Time
          +--- End Date and Time
          +--- ...
   +--- To Do Item
   +--- Free/Busy Time
   +--- Journal Item
```

The xCalendar XML instance file has the name and location of the xCalendar XML schema specified within the document. This enables the receiver to fetch the XML schema dynamically and then process the instance document. The validation of the instance document is a five-step process:

1. Verify that the instance document is well formed.
2. Fetch the XML Schema (XSD File).
3. Verify the XML Schema is well formed.
4. Verify the XML Schema is valid as per the W3C Vocabulary.
5. Verify the instance document is valid as per the XML Schema.

As stated earlier, as powerful as XML is, the value of this technology is limited if no structure is imposed on XML documents. Thus, structures and constraints need to be integrated into the definition of what constitute a legal XML document. The xCalendar schema uses the XML Schema language features like complex types and custom data types to enforce constraints on the instance documents. Some examples are listed below.

One of the elements specified in the xCalendar schema is the DayLightSaving time element. It is depicted in the schema as an anonymous type, shown below.

Schema
```
<xsd:element name="DaylightSaving" minOccurs="0">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="StartDate" type="xsd:dateTime"/>
      <xsd:element name="EndDate" type="xsd:dateTime"/>
      <xsd:element name="TimeZone" type="TimeZoneType"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Instance
```
<DaylightSaving>
<StartDate>2001-09-26T00:00:00</StartDate>
<EndDate>2001-10-28T02:00:00</EndDate>
<TimeZone>-05:00</TimeZone>
</DaylightSaving>
```

Another element specified in the xCalendar schema deals with alarm information. The alarm information is depicted using a named (as opposed to anonymous, as in the case of DayLightSaving) complex type as it can be used in many alarm scenarios (e.g., audio alarm, mail alarm, display alarm, and procedure alarm).

Schema
```
<xsd:complexType name="AlarmType">
  <xsd:sequence>
    <xsd:element name="RunTime" type="xsd:dateTime"/>
    <xsd:element name="SnoozeTime" type="xsd:time"/>
    <xsd:element name="RepeatCount" type="xsd:nonNegativeInteger"/>
  </xsd:sequence>
</xsd:complexType>
```

Instance
```
<AAlarm>
  <RunTime>0000-00-00T00:00:15</RunTime>
  <SnoozeTime>00:10:00</SnoozeTime>
  <RepeatCount>3</RepeatCount>
  <AudioContent>Chimes.wav</AudioContent>
</AAlarm>
```

The DayOfYear data type is created by restricting the range of values of positive integers.

XML Schema
```
<xsd:element name="DayOfYear">
  <xsd:simpleType>
    <xsd:restriction base="xsd:positiveInteger">
      <xsd:minInclusive value="1"/>
      <xsd:maxInclusive value="366"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

Instance
```
<DayOfYear>254</DayOfYear> Is Valid
<DayOfYear>0</DayOfYear> Is Invalid
<DayOfYear>367</DayOfYear> Is Invalid
```

The days of the week data type is created by restricting the values of the String data type. It uses the enumeration feature of XML Schema.

Schema
```
<xsd:simpleType name="WeekDayType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="SUNDAY"/>
    <xsd:enumeration value="MONDAY"/>
    <xsd:enumeration value="TUESDAY"/>
    <xsd:enumeration value="WEDNESDAY"/>
    <xsd:enumeration value="THURSDAY"/>
    <xsd:enumeration value="FRIDAY"/>
    <xsd:enumeration value="SATURDAY"/>
  </xsd:restriction>
</xsd:simpleType>
```

Instance
```
< WeekDayType >SUNDAY</ WeekDayType > Is Valid
< WeekDayType >HOLIDAY</ WeekDayType > Is Invalid
```

The time zone data type is created by applying a *pattern* constraint on the string data type.

Schema
```
<xsd:simpleType name="TimeZoneType">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="[-+](0[0-9]|11):[0-5][0-9]"/>
  </xsd:restriction>
</xsd:simpleType>
```

Instance
```
<TimeZone>-05:00</TimeZone> Is Valid
<TimeZone>*50:00</TimeZone> Is Invalid
```

The above examples showcase a number of features of XML Schema that have been used in xCalendar to enforce the constraints specified in the calendaring application considered. Some other XML Schema language constructs have also been used to enforce other constraints but are not detailed here.

While one can implement most of the constraints using XML Schema, there are constraints that may seem trivial but are beyond the capability of XML Schema. For example, a constraint that states that the start date of daylight saving time must be before the end date of daylight saving time is a condition that cannot be expressed using XML Schema.

After examining several ways in which these conditions can be tested, XSLT (XML Stylesheet Language Transformation) and JAXP (Java API for XML Processing) emerged as the top contenders for addressing this shortcoming of XML Schema. Thus, xCalendar is complementing the power of XML Schema with two well- supported external technologies.

XSLT [5] is written in XML and it provides a vocabulary to navigate the XML documents. Action can be assigned based on matching elements. The language has features to perform conditional action and iteration. Output generated is text or any other textual derivative. As shown in Figure 2 below, the XSL processor takes two files as input. The XML schema instance file and the XSL file, which has the actual processing instruction.

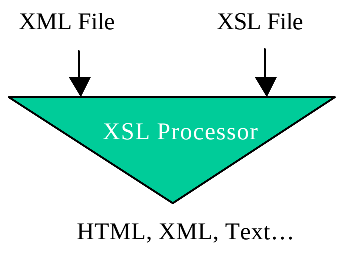

Figure 2: XSLT-based approach to complementing XML Schema for constraint checking.

JAXP [6] is one of the latest additions to the Java API. It supports processing of XML documents using DOM (Document Object Model) and SAX (Simple API for XML processing). It allows applications to process XML documents independent of any particular XML processing platform. A Java program will have to be written for each XML Schema. In the program, the instance file can be loaded into a DOM object. In this case, the application is a Java program and the output can be text (plain, HTML, XML, etc.) or binary. Figure 3 depicts such a process.

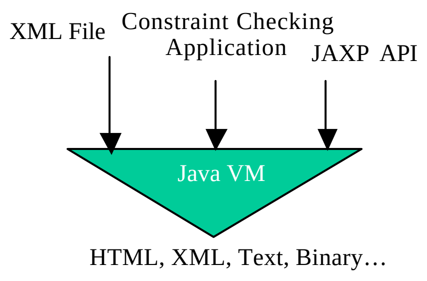

Figure 3: JAXP-based approach to complementing XML Schema for constraint checking.

4. xCalendar: Architecture and Operation

The above describes each of the system components individually. Now let's examine how these components can be assembled to reliably exchange xCalendar documents. This paper described only the architecture that uses XSLT to check non-schematic constraints (see Figure 4 below). Here, non-schematic constraints simply refer to the ones that cannot be expressed using XML Schema. The JAXP architecture is not detailed in this paper.

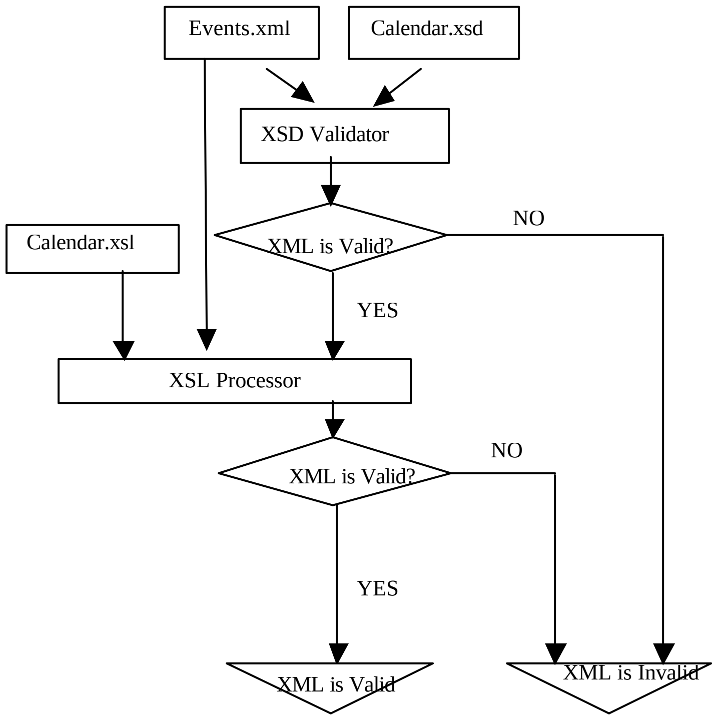

Figure 4: Constraint validation using the combination of XML Schema and XSLT approach.

The XML instance document received is first validated for well-formedness by the XSD Validator. After that, the XSD Validator uses the schema to check if the XML instance document obeys all the rules specified in the xCalendar XML schema. For example, if one changes the DaylightSaving element to HaylightSaving, the XSD Validator will throw a validation error.

Of course, this simple error can be fixed by simply changing HaylightSaving to DaylightSaving. After validating the instance document against the schema, one can run the XSLT processor to transform the instance document to find non-schematic errors. The XSL Processor streams the output to a designated output file in XML format (out.xml is the name used in Figure 5 below). If the XML instance file is valid, the first message in out.xml is "Complete". This indicates that the transformation was successful and the instance file is valid.

```
C:\xCalendar>run-xt Birthdays.xml Calendar.xsl out.xml

--------------------------------------------------------
Running the XSL Processor, XT, with the following specifications:
  - XML File: Birthdays.xml
  - XSL File: Calendar.xsl
  - Output File: out.xml
--------------------------------------------------------
Done.
C:\xCalendar>type out.xml
<?xml version="1.0" encoding="utf-8"?>
<output>
<message>Complete</message>

</output>
```

Figure 5: Testing the XSLT approach with an XML instance file and a XSL transformation file. Here, the XML file is valid and the result shown is "complete"

If the DaylightSaving start date was after the DaylightSaving end date, then the constraint is violated and an error message would first appear in the XML output file before the "Complete" message. This is depicted in Figure 6 below.

```
C:\xCalendar>run-xt Birthdays.xml Calendar.xsl out.xml

-----------------------------------------------------------
Running the XSL Processor, XT, with the following specifications:
  - XML File: Birthdays.xml
  - XSL File: Calendar.xsl
  - Output File: out.xml
-----------------------------------------------------------
Done.
C:\xCalendar>type out.xml
<?xml version="1.0" encoding="utf-8"?>
<output>
<message>Start Date greater than End Date at Day Light Saving Date
Range</message>
<message>Complete</message>
</output>
```

Figure 6: Testing the XSLT approach with an XML instance file and a XSL transformation file. Here, the XML file is invalid and an error message is shown.

5. Summary and Conclusions

In this paper, we have described XML and XML Schema as a technology for information exchange. The hope is that we (and others) can build upon the work done here to develop platform and language independent standards for information exchange in general and calendaring exchange specifically. We will see the need for many more common standards for this to happen. Once these common standards have been established, there will be easy exchange of information. That is when we will fully utilize the extensive connected networks that are so prevalently used in our society today.

Based on the experience of this project it is fair to say that there is a tremendous potential for XML and XML Schema in the arena of information interchange. There are already technologies like SOAP and Web Services being developed based on this framework. Even with all these developments and the excellent technologies made available, it depends on the people and the companies to work together to make this dream a reality.

References

1. Extensible Markup Language (XML). Sep 2001. URL: http://www.w3.org/XML/
2. XML Schema. Sep 2001. URL: http://www.w3.org/XML/Schema
3. vCalendar Overview. Dec 2001. URL: http://www.imc.org/pdi/vcal-10.doc
4. Internet Calendaring and Scheduling Core Object Specification (iCalendar). RFC 2445. Nov 1998. URL: http://www.ietf.org/rfc/rfc2445.txt
5. Extensible Style Sheet, http://www.w3.org/Style/XSL/
6. Java API for XML Processing (JAXP). Dec 2001. URL: http://java.sun.com/xml/jaxp/index.html