White Paper on

OASIS CAM v1.1 and W3C Schema v1.1 Insights

August, 2008

Author

David RR Webber

Chair OASIS CAM
Technical Committee

Table of Contents

Introduction

This paper introduces the work of technical committees within the OASIS standards development community that is focused on enhancing interoperability of business information exchanges. Particularly the OASIS Content Assembly Mechanism (CAM) work and comparing and contrasting that with the W3C XSD schema work. The CAM approach emphasizes simple lighter weight syntax for consistent optimized and interoperable XML transaction exchanges. In addition the work of OASIS technical committees focused on semantic tools and technologies is discussed and related to the CAM techniques including templates and context mechanisms.

Background

In 1998 when XML version 1.0 was made a formal W3C recommendation the vision was of simplicity of mark-up technology and particularly that the level of effort needed to create software tooling to manipulate and process the XML composed documents would be minimized.

There however was a complication in this picture. XML evolved from SGML and in that they shared a common piece of technology called a DTD. A DTD is provided to define the valid structural combinations and content model for a given XML document. This is referred to as a document schema. Due to the circumstances of the evolution of XML itself the DTD turned out to be a work in progress. The DTD syntax used a direct character-based notation to provide a quasi visual representation of the structure composition of the associated XML document. There are limitations to this DTD approach as specified for the original XML V1.0. Carry-over syntax components retained from SGML also added further contradictions.

The W3C then decided to design a new schema language replacement for DTD and hence XSD was created. The XSD syntax however is completely different in design and approach. Whereas DTD syntax is minimalist and uses character designators to group together hierarchical structure components the XSD approach uses XML syntax itself. However the most important difference comes in terms of focus. Whereas DTD schema can be hand composed in a text editor and directly interpreted the XSD syntax is designed to require software tooling to interpret and validate the schema and display the structural results.

Another important aspect is the focus and audience. The XSD syntax is based on supporting information modeling theory and object oriented inheritance techniques and as such provides an array of tools for supporting complex relationships and inheritance methods found in developing those information models. Also XSD syntax is focused substantially on document-centric mechanisms designed to support sophisticated page layout and information publishing and rendering techniques. This sets up a natural discontinuity with the other major use of XML content that of business information exchanges.

Business information exchanges are different from document composition and rendering applications in important ways that XSD schema does not fully support. In business exchanges a minimalistic paradigm is preferred combined with interoperability as a paramount objective. These transactions are highly predictable and iterative in their content models and intended for machine processing as for example in a purchase order with one or more line items.

On the other hand publication documents tend to reflect a human use model so they are much less deterministic and can be recursive and include documents within documents. Conversely of particular importance to business documents is the need for context mechanisms to match variations in content to specific business process scenarios and partner relationships and roles. Documents on the other hand have an open use model based on direct interpretation by the user of the content.

Even before XML 1.0 was completed there was identified a requirement for schemas to address the needs of business exchanges. Microsoft and others submitted to the W3C a proposal to extend DTD syntax for business exchanges; their XML-Data standard provided a large set of data types more appropriate to database and application interchanges. For example W3C XSD schema uses date and time representations that are not compatible with traditional database and electronic data information (EDI) exchanges.

This difference between business exchange use and document-centric preparation led to the development of supplemental technologies in addition to W3C schema. The W3C itself developed a XML scripting and manipulation technology called XSLT while a range of other specifications have also been developed by ISO and OASIS particularly and we will consider those next.

Meeting the needs of XML business exchange users

From the business information exchange perspective the W3C schema syntax is missing several important components and characteristics to support interoperability between partners systems. To address the business exchange area specifically the OASIS Content Assembly Mechanism (CAM) work developed[1]. Five years ago when this work first started much of the efforts on Service Orientated Architecture (SOA) based systems and *core component based exchanges* were only in their infancy. Most of this core component work occurring outside the W3C orbit within UN/CEFACT Core Components and ebXML (electronic business XML) initiatives specifically. Now today we are seeing that communities of business users particularly understand how these core component and transaction assembly concepts are relevant to their emerging needs. A simple schema may suffice for limited interchanges between a handful of partners or for defining an internal interface where both ends are managed by the in-house development staff. However interoperability directly affects the ability to manage extended interactions and business processes across diverse partners' systems including scaling out to across industry collaborative community or global context. Therefore for industry communities particularly the need is to define what those interoperable exchanges are. This includes expressing the exchange transactions themselves in a succinct way as templates that can be readily adapted by their members for their own business context. Allied with this is assembling the transactions from a set of industry standard information core components to ensure uniformity of definition and understanding. Then being able to use context to control how those core components behave for a given business process scenario and role.

For XSD schema which has no context mechanisms this is problematic and so the precise interchange implementation detail is instead left open ended and non-deterministic. This leads to the situation where each participant in an exchange interprets aspects of the schema use slightly differently and creates different and incompatible XML as a consequence. This is compounded by XSD syntax nuances such as namespaces, null (empty) constructs and lack of cross field edits

[1] OASIS CAM V1.1 standard - http://wiki.oasis-open.org/cam

(where entries in one part of the schema specifically restrict content that can occur elsewhere). Additionally in business exchanges heavy use of code list values predominate and again the use of context and related versioning mechanisms are needed to determine exact allowed sets of values for partner's exchanges.

These costly issues arise during the most critical time of an implementation rollout of the standard, sometimes months after the XSD schema has past scrutiny of levels of sign-off and agreements. A process which uses CAM addresses these issues upfront and when change is least costly; thus is an essential risk mitigation tool for business integrators as well as those involved in the document standardization process. OASIS team work reviewing XSD schemas over the past six months has shown that using the CAM technology and techniques have detected schema issues that in some cases have been dormant and undetected for years. Utilizing CAM is an emerging best practice that has already demonstrated significant cost savings and reduced project delivery timelines. How does CAM achieve this compared to the W3C XSD schema approach?

The CAM approach is to provide a structure instance directly as a template so that implementers can see unequivocally what is expected for an exchange, and then a declarative rules section to augment this with XPath rules that explicitly determine the content use patterns. In addition CAM templates are context aware so that dynamically the template can be adjusted to match specific patterns required by a business process. Context can be supplied externally by parameters passed into the template process, or derived by rules within the template, or computed at runtime from content found in the XML instance.

Summarizing this support of interoperability OASIS CAM templates add:

- Lightweight template structure representation that can be directly visually inspected
- Context mechanisms at template, structure and content levels
- Set of structure manipulation functions that are XPath driven and declarative
- Content assembly using domain specific core components patterns (want lists and libraries)
- Flexible content model visual data masks for specifying date, time, and number formats
- Support for direct versioning techniques for content values and structure components

This emphasis on simple lighter weight syntax is now paying off as tools are developed to exploit the capabilities that CAM templates open up and provide more consistent optimized and interoperable XML transaction exchanges. This later aspect we will now explore in more detail next.

Extending the semantic manipulation options

Having simple light weight syntax has other dramatic consequences beyond the obvious ones of making the exchange model more accessible to human inspection. It also makes the syntax much easier to program using W3C XSLT transformations. Ironically there is a dearth of XSLT tools that manipulate and exploit the semantic information contained in W3C XSD schema simply because of the extended complexity of the XSD schema specification and local variations in how people actually use it to represent different information model scenarios.

After 4 months of concerted programming development work our team has built XSLT tools that are able to ingest a W3C XSD schema and automatically create the equivalent OASIS CAM template from that. Complete with structure example, XPath rules and information content validations that replicate what the original XSD schema is doing. Quite apart from the simple

software achievement this then opens up the whole semantic information model previously buried inside the XSD syntax to further XSLT processing, this time using the simpler CAM template syntax as the input to the XSLT.

This base CAM template syntax consists of three succinct template sections for header, structure and rules definitions. This provides very clear delineation of the purpose of each section. Unlike with the XSD schema approach where the entire definition consists of collections of structure fragments interwoven with content and type definitions that are linked together using type, reference or base attribute key names and namespaces as complex and simple types that must be traversed by software to infer the resulting structure.

What would previously have taken weeks to program in XSLT against the schema syntax can be achieved in a matter of a few days or less using the equivalent CAM template format rendering. *We were thus able to then rapidly develop a complete suite of XSLT tools that can parse the CAM template format and provide crucial new capabilities in support of interoperability.*

An interoperability tool suite

While commercial schema editor tools provide a selection of tools for software developers these are closed solutions that do not offer the flexibility that XSLT scripting enables. Also since XSD syntax has no context mechanism it is an "all or nothing" situation for implementers particularly when dealing with complex schemas. With using XSLT in tandem with CAM templates this situation is transformed. Developers can use the CAM XSLT tool[2] suite to easily select and restrict the structure template to only the specific content they desire for their business solution. Then they can create rich example XML test case instances with actual content values either directly provided via a data hinting mechanism or automatically generated based on type definitions. Plus the tools allow generation of both valid and invalid examples – essential for interoperability test suites between partners.

They can then also generate a matching XSD schema sub-set exported from the CAM template that restricts the exchange model to only what they require. This is essential for particular large industry standard XSD schemas (these schemas are often so large that software development tooling or WSDL web service tools cannot reliably process them).

Then various styles of familiar tabular documentation are supported that are more suited to review by business analysts to confirm the content and rules being applied to the XML exchanges.

Of course this can also be done theoretically using XSD schema but the fact this is not readily available already speaks to the complexity, challenges and level of effort needed for doing this using XSD syntax.

A further example of such challenges is shown by the interoperability validation reporting provided for CAM templates, shown in Figure 1 here. While this provides useful statistics about your business exchange document it also reports on potential interoperability issues.

[2] The CAM XSLT toolkit is available via the tool option menus in the Eclipse jCAM editor tool available from http://www.jcam.org.uk or standalone from http://wiki.oasis-open.org/cam tools page links.

The information is grouped into header information for the statistics, then warnings and errors relating to the CAM template syntax itself followed by the analysis of the associated XSD Schema Generation and possible interoperability issues that occur from that.

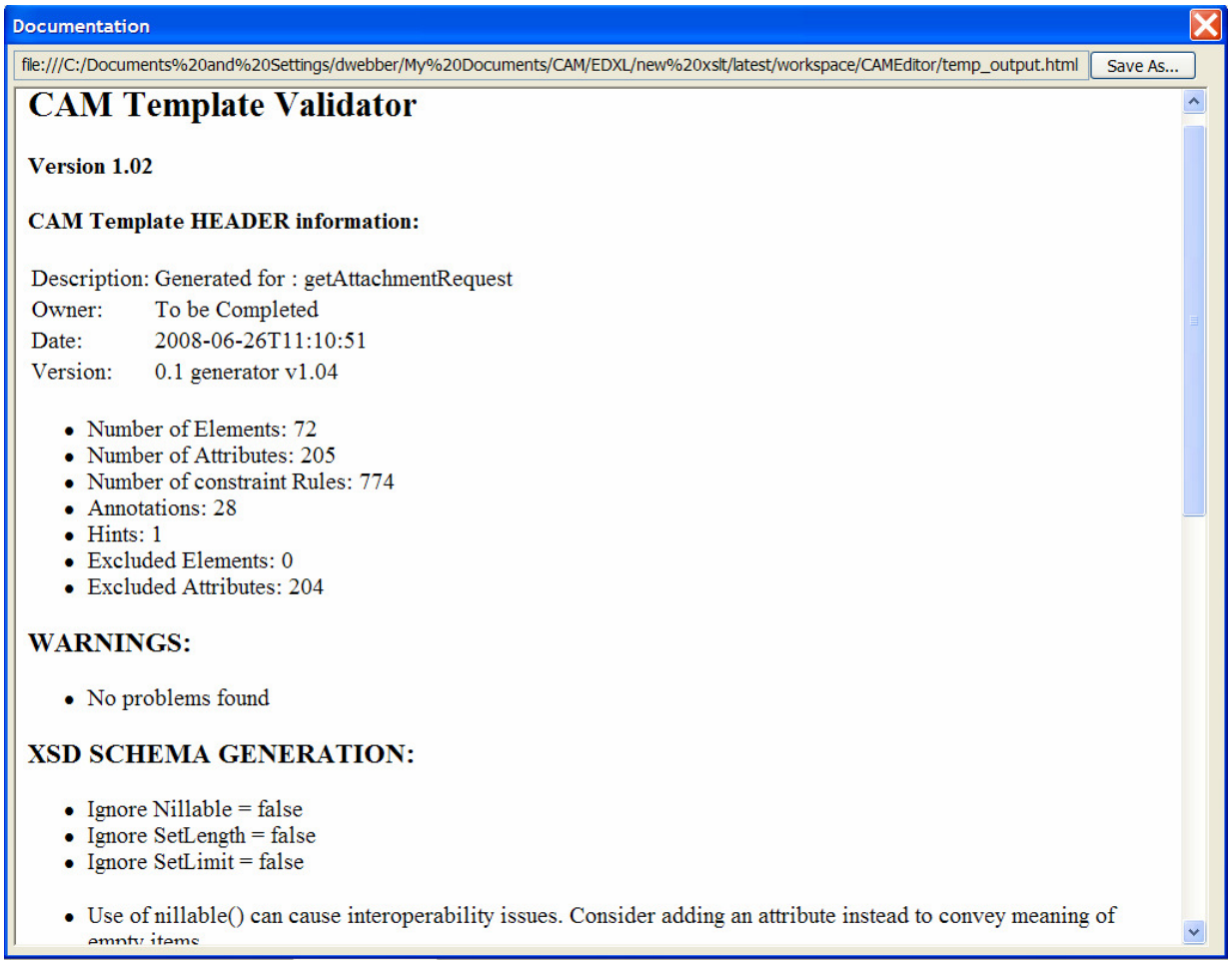

Figure 1 – CAM template validation and interoperability reporting

The report is created by an XSLT script that parses the CAM template and looks for specific conditions. As shown here Nillable, SetLength and SetLimit conditions are all tested for (each check can be optionally ignored if preferred) and in the example here use of nillable was detected in the schema. The error message offers guidance on resolving this be replacing an indeterminate use of a nillable assertion by a concrete attribute with specific values that indicate the exact reason for the use pattern occurring (e.g. instead of nillable, use enumerations for 'unknown', 'not provided', 'not available', 'incomplete data', 'pending' and so on). The error message will identify the exact locations in the structure that this applies to.

With XSD schema many of these conditions and potential issues remain hidden inside the XSD syntax and only emerge later, often only after interfaces have been developed and thus causing expensive re-work and software changes. Also because these validations are implemented as an XSLT script developers can choose to extend these rules and crosschecks themselves to enforce their own standards for exchange development. Next we consider specific some examples of business use challenges with current XSD schema syntax.

W3C XSD Schema Syntax Considerations

In this section we highlight just a couple of examples of existing challenges with W3C syntax and then compare that to the equivalent logic expressed in OASIS CAM template syntax.

First consider the challenge of a fairly common business information pattern where there is a choice needed between two child items within a parent structure node, and then a rule validation that requires that if one child item is present then the other is optional.

To achieve this in W3C schema syntax requires the following syntax fragment (Figure 2) here.

```
<xs:complexType name="ResponsibleOfficerStructure">
    <xs:sequence>
        <xs:choice>
            <xs:sequence>
                <xs:element name="Responsibility" type="xs:token"/>
                <xs:element name="Name" type="PersonNameStructure" minOccurs="0"/>
            </xs:sequence>
            <xs:element name="Name" type="PersonNameStructure"/>
        </xs:choice>
        <xs:element name="Contact" type="ContactDetailsStructure" minOccurs="0"/>
        <xs:any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="Id" type="xs:NMTOKEN" use="optional"/>
</xs:complexType>
```

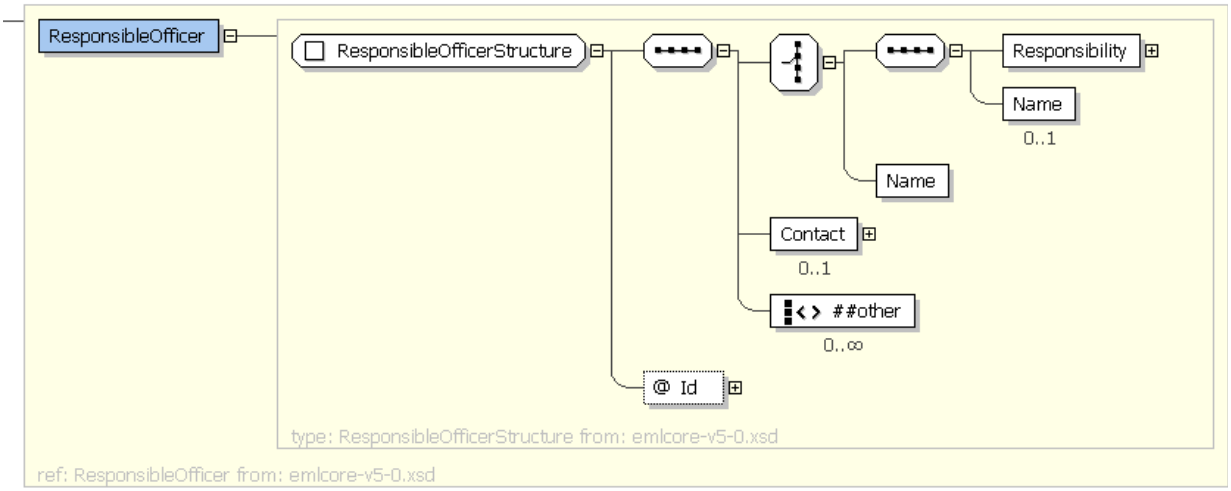

Figure 2 – choice of two items with optional item – XSD syntax fragment and its visual representation

Notice that the one element item "Name" definition of type "PersonNameStructure" is repeated even though it only in fact occurs once in an XML document built from this definition. Quite apart from the obvious potentially confusing aspect of this, it requires that schema developers know about the particularly syntax nuance. Finding and distilling this syntax knowledge from the actual W3C specification is definitely non-trivial. To understand what is really going on here, we need to see the actual XML structure instance that matches the schema structure definition. This is the approach that CAM templates employ.

The equivalent structure syntax definition in CAM template notation is shown here (Figures 3a, 3b, 3c).

```
<ResponsibleOfficer Id="%xs:NMTOKEN%">
  <Responsibility>%Token%</Responsibility>
  <Name PartyType="%xs:anyType%" Code="%xs:anyType%">
  <Contact DisplayOrder="%1%">
</ResponsibleOfficer>
```

Figure 3a – choice of two items with optional item – CAM template excerpt structure definition

It is clear from this that Name and Responsibility are singular children of the ResponsibleOfficer element. Notice also the use of content hinting via the %value% notation that simply allows documenting of typical values and types throughout the structure.

The associated choice rule to indicate that the Responsibility and Name are linked as a paired choice is shown in Figure 3b along with the data type rules for each of the structure elements here.

```
<as:constraint
  action="setChoice(//ResponsibleOfficer/*[contains(name(),'Responsibility') or contains(name(),'Name')])"
/>
<as:constraint action="makeOptional(//ResponsibleOfficer/@Id)" />
<as:constraint action="makeOptional(//Name)" />
<as:constraint action="datatype(//ResponsibleOfficer/@Id,NMTOKEN)" />
<as:constraint action="datatype(//Responsibility,token)" />
<as:constraint action="datatype(//Name,anyAttribute)" />
<as:constraint action="datatype(//ResponsibleOfficer/Name,anyAttribute)" />
```

Figure 3b – choice of two items with optional item – CAM template excerpt of rules

Visually this can then be simply displayed in a hierarchically tree structure viewer as:

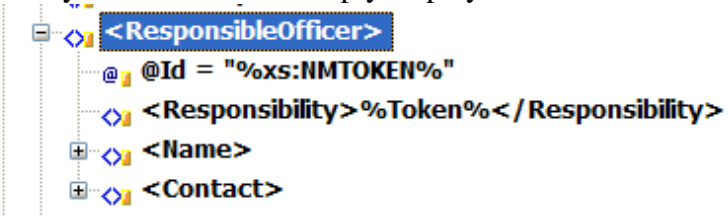

Figure 3c – tree structure – visual representation

Of particular note is the ability here to quickly make context rule driven changes by using constraint rule statements in the CAM model that are difficult and complex in the XSD approach. Each constraint rule can have an optional condition assertion (Figure 3d) that controls when the associated action occurs.

```
<as:constraint condition="exists(//ResponsibleOfficer/Name)"
               action="makeOptional(//ResponsibleOfficer/@Id)"/>
```

Figure 3d – context rule example controlling structure use pattern

Next we consider the need to express content models such as for a decimal number. In XSD syntax shown in Figure 4 again we see that XSD has mechanisms that make versioning and context changes problematic because structure and definition usage are mixed in together.

```
<xs:element name="Rate">
 <xs:simpleType>
  <xs:restriction base="xs:decimal">
   <xs:fractionDigits value="2" />
   <xs:totalDigits value="6"/>
   <xs:minInclusive value="1"/>
   <xs:maxInclusive value="9999"></xs:maxInclusive>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

Figure 4 – decimal item with value range – XSD syntax excerpt

This also makes it difficult for XSLT parsing of the XSD syntax itself since XSLT works recursively stepping through the XML content whereas XSD syntax requires multiple passes to interpret how all the various pieces of markup relate structurally to each other, e.g. in Figure 4 we would need to know where the element "Rate" is to being referenced from and how the base type namespace "xs:" of decimal is declared.

The equivalent CAM template syntax is shown in Figure 5 where the rules are entirely separate but reference back to the overall structure using an XPath reference expression. The content model is defined using a visual data mask (#4.##) for specifying the number format.

```
<as:constraint action="setNumberMask(//CategoryRate/Rate,#4.##)"/>
<as:constraint action="setNumberRange(//CategoryRate/Rate,1-9999)"/>
```

Figure 5 – decimal item with value range – CAM template excerpt

Because CAM allows you to also use context driven mechanisms you can then easily extend this as shown in Figure 6 to select different data mask patterns based on business partner exchange details.

```
<as:context condition="//CategoryRate='AA'">
 <as:constraint action="setNumberMask(//CategoryRate/Rate,#4.##)"/>
 <as:constraint action="setNumberRange(//CategoryRate/Rate,1-9999)"/>
</as:context>
```

Figure 6 – decimal item with context – CAM template excerpt

These two examples have been chosen to illustrate the differences between the XSD approach and the CAM template methods. This is not intended to be a tutorial in either nor to supplement the actual base specification details. Next we consider the intersections between XSD schema syntax and CAM templates.

XSD Schema V1.1 enhancements and CAM

The question arises of how compatible XSD syntax is with CAM syntax? Are there things in XSD that CAM does not support and vice versa? With the new release of XSD the W3C is introducing two new major features that provide better round trip compatibility between the two technologies. The most significant is the use of XPath syntax and assertions. The W3C has learned from both the work on CAM and also Schematron that use of XPath provides significant advantages. By providing XPath assertion support the W3C has reduced the gap that previously existed in this area. So now XPath assertions that were only possible before in CAM can also be exported to XSD schema, and of course the reverse mechanism applies. However the way that schema implements support for XPath is different to the CAM method and so completely automatic ingestion from XSD and the reverse generation to subset schema may still require manual determinations in some circumstances. Of particular concern is that the XPath approach in W3C schema may well be a double edged sword in that it solves some issues but creates others especially relating to interoperability of business exchanges and the predictability of how partner systems will handle and apply such XPath rules. This may well lead implementers to adopt guidelines that exclude certain techniques with the W3C approach and by having CAM templates as the interoperability intermediary agent one can avoid such potential risks.

XSD assert syntax

```
<xsd:complexType name="article">
   <xsd:assert test=
      "if (exists(affiliation))
      then exists(author)
      else true()"/>
</xsd:complexType>
```

CAM rule

```
<as:constraint
   condition="exists(//article/affliation)"
   action="makeMandatory(//article/author)"/>
```

Figure 7 – Example of XSD XPath mechanism compared to the CAM approach

The other important distinction you can see in Figure 7 is that the CAM syntax references the structure context directly, while the W3C XSD approach is implied when the complexType definition is referenced. The largest gap however is that in the CAM XPath you may reference parameters and values that are not part of the XML instance, such as global variables passed into the CAM template processor and defined in the template header section. This allows contextual control over the processing not related to explicit values in the XML transaction instance.

The second change in XSD V1.1 is that the W3C is also implementing conditional type assignment. Again this was previously something that only CAM supported, so now you can implement this is both syntaxes. The most obvious place this is needed is in different handling of date formats and numeric values or in things like telephone number formats. Figure 6 above already shows an example using CAM assertions to control the content typing. Again context can be externally supplied (such as versioning) whereas in the W3C approach this has to be part of the actual XML instance referenced. Context mechanisms are discussed further in the next section.

It has never been a design goal of CAM however to be 100% compatible with all aspects of XSD schema. The two technologies have different audiences and hence there are features in XSD that business exchanges either will never need or should never use because of interoperability concerns. That said there is a high level of equivalence, probably exceeding 95%, between the two syntaxes. Where they diverge is in the level of modeling information theory support and for abstract concepts and inheritance. *It should be noted that CAM is focused on the direct rendering of the actual XML instances permitted and their interoperability rather than the schema metadata and model semantics associated with them.*

From the perspective of round trip syntax manipulation from XSD schema into CAM template and back to XSD schema this could be accomplished by retaining XSD schema syntax fragments as node annotations within the CAM template and then having specialized logic to re-constitute the original XSD syntax. This has not been implemented currently however in the interoperability tool suite. Instead the XSD sub-set generation from CAM is designed to create simple direct XSD schema syntax constructs. This is deliberately minimalistic and intended to support the use of software tooling that requires such sparser use of schema syntax in order to optimize exchange handling performance.

There remains however the significant ability in CAM to use context mechanisms and the related capability to directly exclude or insert structure members and components that do not have exact equivalents in XSD schema. This also impacts on the whole area of versioning of structure components and their definitions. The context mechanisms are discussed next.

CAM context mechanism techniques

As shown in Figure 6 above using the CAM context apparatus allows for matching of business process scenarios to precise information exchange patterns and/or code values. This is particularly important in today's business process automations where explicit behaviour and versions are needed depending on the applications role and use patterns.

Context mechanisms in CAM allow control over not only the content model but also the structure items and their associated choices. This is essential for interoperability in being able to capture and understand the specific details of partners information exchange needs.

In addition to XPath driven context conditional techniques CAM also provides for the declaration of global parameter values in the template header section. This allows context to be set externally and passed into the template which is particularly important for aligning exchange behaviour to specific business processing handling scenarios. This agility and deterministic control is how the use of CAM templates augments the base structural definitions possible with XSD schema syntax alone. In classic EDI terms the CAM template is allowing you to express the implementation convention (IC) for partner exchanges. However unlike the EDI IC which is static documentation only, the CAM template is dynamic and able to be applied at runtime as scripting technology programmatically.

In addition as previously noted in the interoperability tool suite section the CAM template syntax supports the creation of realistic sample test data by using XSLT scripting. Extending this is straightforward through the flexibility of the open source XSLT scripting approach that allows for direct tailoring to support specific project needs. Whereas the non-deterministic nature of XSD schema means that tools that output XML instance examples from XSD syntax cannot

discern exact use patterns and hence can only produce random and less useful examples automatically.

The next section looks at the support for the SOA concepts of core components and content assembly along with versioning.

Library-based core components and structures

Of particular importance in the OASIS CAM approach is the support for the concept of a library of industry domain core components that provide a common information model and practice that can be used to assemble business transactions from pre-existing standard components. The W3C XSD schema approach eschews this in favor of a localized model where all aspects are directly defined within the XSD only[3]. In addition this localized model lacks direct versioning support within XSD relying instead on indirect namespace mechanisms. Also versioning really requires mechanisms that can operate at all levels including down to the attribute level within elements and on code list values (XSD enumerations) as well.

As an illustration of the use of library concepts and content assembly with CAM templates consider the US government National Information Exchange Model (NIEM) work (http://www.NIEM.gov). Here a vocabulary of common components is provided that are then used to assemble together XSD schema definitions. In the CAM approach the assembly avoids the complexity associated with XSD schema syntax and the need for manual creation of subset, constraint, exchange and extension schema definitions as shown in Figure 8.

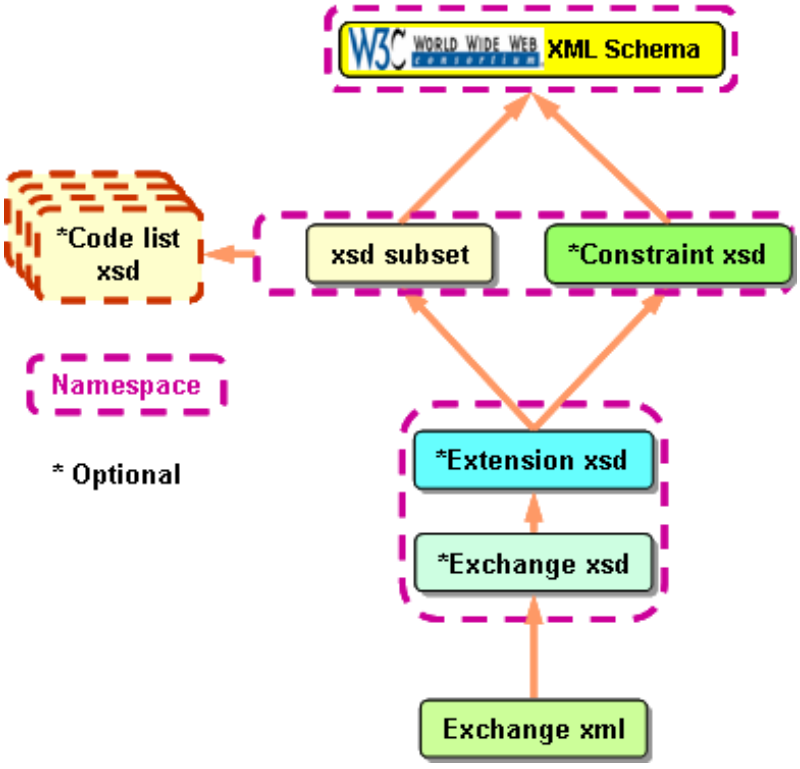

Figure 8 – NIEM assembly approach using XSD schema syntax

[3] XLink was developed partly in response to this but offers very different usage and creates complications of its own for implementers.

Instead the CAM template method allows a natural approach where logical blocks of components[4], or individual components as needed, are inserted directly into the structure hierarchy. Then the associated constraint and content model facet details can be referenced automatically from the library definitions and inserted into the accompanying template rules section. The lightweight nature of CAM template syntax makes this cut and paste approach possible.

For code list values CAM templates provide the lookup() function that can reference external include files and named code lists. Therefore implementers can apply context selection rules to determine the precise set of values based on business process, exchange content details and partner context.

Once the desired structure is complete the associated XSLT scripts can be applied to the CAM template to automatically generate equivalent static XSD schema syntax structure definitions (keying off namespace prefixes to determine and create schema imports and global definitions).

This is particularly useful for web service exchange definitions where they require a XSD schema be associated with a particular action. Often using the complete industry defined standard XSD schema is too non-deterministic (everything is marked as optional content) to allow consistent interoperable exchange patterns to be deduced, or the industry schema is simply too large for the software tools to work with. What is required is a "want list" selection from the original main schema of only the parts needed for the particular web service application. The CAM template XSLT tools provide all these capabilities to allow sets of XSD schema to be derived from a master template definition.

This then leads into the use of context to implement versioning needs. By declaring a global parameter then referencing this from XPath rules you can directly control structure and content. The CAM syntax includes excludeTree(), excludeElement() and excludeAttribute() functions that allow direct pruning of the structure template. Conversely you can select or add elements and choice items. This provides rule driven control over the structure and allows dynamic modification of both the structure and the associated information content model. In addition the CAM template allows named structures to be defined. Therefore a template may contain more than one structure and contextually select the one required.

Using these techniques implementers may tailor by version and by partner the explicit interchange patterns needed. The CAM template allows you to derive from this, by applying a XSLT script, one or more XSD schema definitions based on your business process and partner requirements.

[4] In core component parlance Business Information Entities (BIE) pre-defined sets of related parts such as address or order line item.

Semantic Extensions Work

Recently the OASIS Semantic Support for Electronic Business Document Interoperability technical committee (OASIS SET TC) has started work on specifying semantic mechanisms for interoperability among core component based electronic business document standards. The CAM template approach has been identified as one such component that can contribute to this new work. As we saw in the section of library definitions the CAM template approach provides a very natural assembly from core components into a complete structure.

Additional semantic support can make this task more automated by finding matching core components more easily. For example simple key word searches into a library may return dozens of matches and present challenges for modelers to know which one to then select for their specific needs. The hope is that OASIS SET work will facilitate the selection of the correct core components when doing such initial transaction assembly.

As an illustration of the capabilities here an automated matching between schemas and the NIEM library definitions has been built using XSLT tools and CAM templates. The NIEM library definitions are exported from the Access database containing them into a simple XML structure. The XSLT tool then steps through the CAM template definitions and produces a report of those matches found into a XML output file. That file can then be opened directly as a spreadsheet and shared accordingly to document the correspondence between the original schema and the library definitions. These tools and examples are available from the CAM wiki site – http://wiki.oasis-open.org/cam as open source.

As industry groups adopt sets of core components from complimentary work (such as the OASIS CIQ TC work on defining address, name and organization definitions) it becomes more important to be able to align use patterns correctly across implementations.

Another such example is the OASIS Genericode TC work that provides semantic definitions for code list values. Many standard ISO and UN/CEFACT codes are available in Genericode semantic formats (http://docs.oasis-open.org/ubl). These can be quickly converted to the compact CAM lookup function format by applying a XSLT script (available from the CAM wiki resource site and as menu option in the jCAM editor implementation – http://www.jcam.org.uk).

Summary

XML document usage consists of traditional publication uses for manual visual information purposes such as magazines, books, manuals, news feeds, multimedia and other online web content and then also machine generated XML business information exchange documents intended for automated processing by software. From the business information exchange perspective the W3C schema syntax was missing several important components and characteristics to support interoperability between partners systems. To address the business exchange area specifically the OASIS Content Assembly Mechanism (CAM) work developed.

Now today we are seeing that communities of business users particularly understand how this is relevant to their emerging needs and particularly for SOA applications with extended use patterns across multiple participants systems. A simple schema may suffice for limited interchanges between a handful of partners or for defining an internal interface where both ends are managed by the in-house development staff. With today's expanding global needs the risk is that each participant in an exchange interprets aspects of the schema use slightly differently and creates different and incompatible XML exchanges as a consequence.

Such interoperability challenges directly affect the ability to manage extended interactions and business processes across diverse partners systems especially when scaling out to a national or global context. Therefore for industry communities particularly the need is to define what those interoperable exchanges are assembled from a set of industry standard information core components and then express those templates in a succinct way that can be readily adapted by their members for their own business context.

For XSD schema which has no context mechanisms this is problematic and so the precise interchange implementation detail is left open ended and non-deterministic.

We have covered the details of the techniques and capabilities that using CAM templates provide in solving these challenges and particularly how developers can take advantage of the XSLT tool suite that is available to work from existing XSD schemas and rapidly develop better interoperable business information exchanges between exchange partners systems using CAM templates.

The XSLT tools are able to naturally exploit the semantic power and simplicity of the CAM template approach in ways that are not possible when attempting to apply XSLT to XSD schema syntax. These XSLT tools provide:

- want list driven selections from existing XSD schemas
- building context aware structure templates and use patterns from the original XSD schema
- interoperability validations and crosschecks
- automatically generating XSD schema subsets for use with web service interfaces
- creating a test suite of realistic XML instances with both pass and fail conditions
- tabular documentation formats of structure and rules usage for business user verification
- automated cross-referencing to libraries of core components

The CAM template work is part of a range of technical committees within the OASIS standards community that is providing additional semantics tools and technologies to extend and augment what is possible with W3C XSD schema for better interoperability between business information exchanges.

Implementers can now explore all these options with their own work by reviewing the tutorial materials and downloading the open source tools available from both the OASIS CAM wiki site and the jCAM Java tool SourceForge implementation site. These are all listed in the resources section below.

Resources

OASIS CAM specification – http://docs.oasis-open.org/cam

OASIS CAM wiki developers site – http://wiki.oasis-open.org/cam

OASIS CAM developers support list – cam-dev@lists.oasis-open.org

SourceForge camprocessor implementation site – http://www.jcam.org.uk

Tutorial on using camprocessor tool with XSD schema – http://www.drrw.net/CAM/XSD%20and%20jCAM%20tutorial.pdf

NIEM resource site – http://www.niem.gov

W3C XSD specification work – http://www.w3c.org

Michael Kay – Saxonica – Discussion of XSD Schema V1.1 new features http://assets.expectnation.com/15/event/3/Will%20XML%20Schema%201_1%20solve%20the%20problem_%20Presentation%201.pdf

Acknowledgements

Appreciation and thanks to the following for providing review comments and feedback:

Bruce Peat, Stephen Green, Alan Kotok, and Martin Roberts.