Demonstration of XML Validation Framework using OASIS CAM approach

David Webber

OASIS CAM TC Chair,
630 Boston Road, Suite M-102
Billerica, MA 01821,
United States of America.
David Webber, drrwebber@acm.org

Abstract. A XML validation approach using OASIS Content Assembly Mechanism (CAM) templates is presented. The approach permits support of a wide array of complex message exchanges with singleton template patterns and sets of context rules. The approach with CAM templates simplify and externalize the validation rules while allowing a validation gateway to act as a pass-through on information that is not directly relevant. The demonstration uses an open source implementation component built using Eclipse and Java technology to deliver the needed validation services.

Keywords: XML, exchange, validation, XSD, schema, template, framework, OASIS, CAM, content assembly, CCTS.

1 Introduction

In today's complex information exchanges with XML and associated large XSD schema, coupled with an array of trading partners, it becomes a significant challenge to support and maintain accurate handling of all incoming transactions. Currently, XML schemas and DTDs provide the ability to validate, or verify, the structural content of a XML document. The associated business content validation rules cannot be adequately accommodated as part of XML schemas and hence use of OASIS CAM templates provides the ability to express extended contextual handling rules.

With the advent of industry specific exchange standards expressed as XML schemas the consumers and providers of information exchange services must comply with these schemas to be conformant and interoperable with industry partners. However, such industry specific schemas are loosely bound with minimal validations and can be used for only structural validation of the incoming XML.

The solution approach we present here is to implement the XML validation services based on the OASIS Content Assembly Mechanism (CAM) specification. The OASIS CAM template approach is based on a simple approach to XML content handling and validation that allows businesses to create common interchange models for their exchanges in XML. CAM templates support context-based rules, code-lists, and cross-field validations.

The solution includes CAM Studio (an Eclipse-based UI template editor) that is used to define the CAM template. Then the CAMV validation engine provides a set of open source Java APIs which are used to validate the XML with the specific compiled CAM templates for run-time deployment in production environments. The CAM Studio template editor supports adding custom XPath expressions to its generated templates. Writing rules with XPath expressions has proven to be an extremely powerful way of implementing the required XML handling and logic. These validations prevent errors when the data is received by applications or components that expect the data to be in a particular structure and comply with business content validation rules.

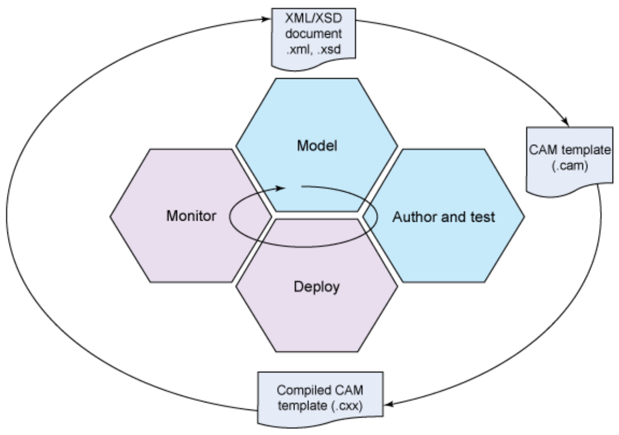
The demonstration will illustrate an exchange where two different exchange patterns are required based on the context of an exchange partner. Sample XML message instances show how dramatically different information exchange patterns can be handled with ease using the CAM template approach.

2 Implementation Approach

2.1 Validation Template Generation

Developing XML validation templates is discussed here. Figure 1 below shows the Model, Author and Test, Deploy, and Monitor stages in the life-cycle of developing the validation rules.

Figure 1. Validation rules life cycle



Source: IBM Developer Works - http://www.ibm.com/developerworks/java/library/x-camval

2.1.1 Model Stage

In this step the data entities and their data elements are identified along with their corresponding validation rules. The required XML exchange schema is designed; alternatively, the required elements are mapped to an existing industry standard schema. The CAM Studio editor provides the capability to import and tailor existing exchange schema this is discussed next stage.

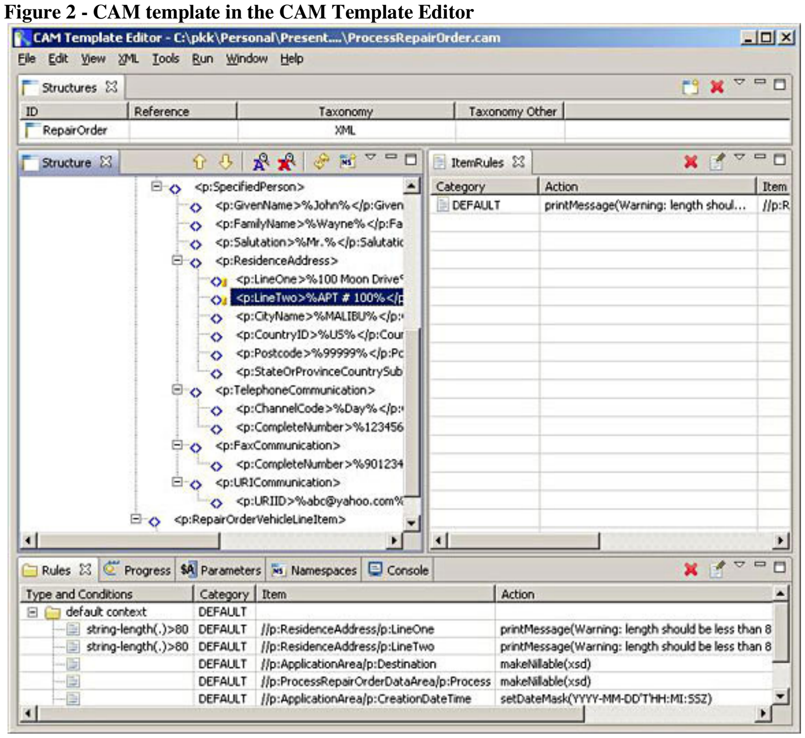
2.1.2 Author and Test Stage

CAM Templates are assembled or authored using the CAM Studio editor. There are the three possible editor options provided to create a CAM template:

1. Create from scratch or hand-crafted
2. Use an existing XML Schema
3. Use an existing XML instance

Once you create the CAM template, the next step is to review each required XML element and attribute and their applicable validation rules. A panel in the editor displays the rules for each template node. The Figure 2 here shows an example template structure in the CAM Template Editor.

Figure 2 - CAM template in the CAM Template Editor



CAM Template Editor - C:\pkk\Personal\Present....\ProcessRepairOrder.cam
File Edit View XML Tools Run Window Help
Structures
Structure
ItemRules
Rules
Progress
Parameters
Namespaces
Console

ID	Reference	Taxonomy	Taxonomy Other
RepairOrder		XML	

```
<p:SpecifiedPerson>
    <p:GivenName>%John%</p:Given
    <p:FamilyName>%Wayne%</p:Fa
    <p:Salutation>%Mr.%</p:Salutatic
    <p:ResidenceAddress>
        <p:LineOne>%100 Moon Drive
        <p:LineTwo>%APT # 100%</p
        <p:CityName>%MALIBU%</p:
        <p:CountryID>%US%</p:Cour
        <p:Postcode>%99999%</p:Pc
        <p:StateOrProvinceCountrySub
    <p:TelephoneCommunication>
        <p:ChannelCode>%Day%</p:
        <p:CompleteNumber>%123456
    <p:FaxCommunication>
        <p:CompleteNumber>%901234
    <p:URICommunication>
        <p:URIID>%abc@yahoo.com%
<p:RepairOrderVehicleLineItem>
```

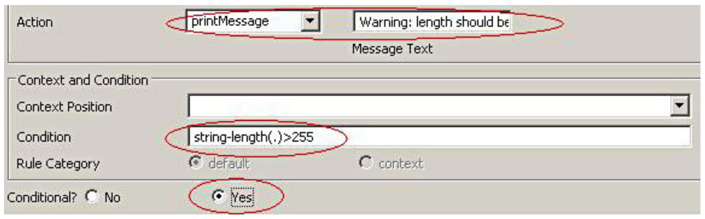
Category	Action	Item
DEFAULT	printMessage(Warning: length shoul...	//p:R

Type and Conditions	Category	Item	Action
default context	DEFAULT		
string-length(.)>80	DEFAULT	//p:ResidenceAddress/p:LineOne	printMessage(Warning: length should be less than 8
string-length(.)>80	DEFAULT	//p:ResidenceAddress/p:LineTwo	printMessage(Warning: length should be less than 8
	DEFAULT	//p:ApplicationArea/p:Destination	makeNillable(xsd)
	DEFAULT	//p:ProcessRepairOrderDataArea/p:Process	makeNillable(xsd)
	DEFAULT	//p:ApplicationArea/p:CreationDateTime	setDateMask(YYYY-MM-DD'T'HH:MI:SSZ)

While all such validation rules need not be binary in nature (that is, either pass or fail), CAM supports classifying validation failures as Warnings and Informational level reports. This feature comes in handy for scenarios where corrective action can be taken at the service provider-end, modifying the payload to make the message usable rather than rejecting the complete message. For example, a rule might require the length of a particular comment field to be within 255 characters; however, a request message should not be rejected when the length exceeds the maximum value, but a warning should be sent to the message consumer specifying that only the first 255 characters will be used from the comment.

As the warning is returned only if the length of specific element exceeds the specified length, this rule is specified as conditional and an XPath expression is created to perform the length check as depicted in Figure 3 screen capture of the CAM Studio Editor expression entry wizard tool:

Figure 3. How to configure a warning rule


Action: printMessage
Message Text: Warning: length should be
Context and Condition
Context Position:
Condition: string-length(.)>255
Rule Category: [✓ default] context
Conditional? No [✓ Yes]

Once definition of the rules is completed then the template can be deployed.

2.1.3 Deploy Stage

The CAM templates are compiled using the CAM Studio Editor before you use them with the application run-time CAMV engine. The compiled format is the condensed XML version of the original CAM template itself and is designed to optimize performance of the CAMV validation engine. To compile the CAM Template, select the menu option Tools **> Compile Template**. This will generate the .cxx file format of the template which will be used at run time.

The CAMV validation engine offers a simple, open-source Java API which can be used in any Java application to validate an input XML with the applicable CAM template. The code snippets in Listing 1 illustrate the usage of CAMV:

Listing 1. Usage of CAMV API

```
TemplateValidator tv = new TemplateValidator(templateDocument);
tv.setErrHandler(new ElementErrorHandler(tv));

boolean tvResult = tv.validate(ioReader);

if (tvResult){
        System.out.println("No errors, might be warnings.....");
}

List errList = tv.getErrors();
List warnList = tv.getWarnings();
```

You can cache CAMV templates into memory to perform repeated validations and not read the templates from the hard disk for each and every validation performed. This reduces the disk I/O and significantly improves the performance and throughput.

Next we consider the verification of the rules scenarios in the template.

2.1.4 Monitor Stage

During the monitor stage the performance of the templates is tracked and adjustments performed to fine tune handling of actual exchange message instances. By virtue of using the CAMV engine, you can now externalize all the validation checks and need not embed them inside code or otherwise implement rules using custom coding. During the monitoring cycle, you can meet the need for additional validations by simply updating the validation templates to add additional validations or remove existing ones. Then simply redeploy the compiled CAM template (.cxx files).

2.2 Further Template Considerations

The CAM template format is represented in XML itself (see Appendix A for example) and hence can be readily edited directly, as well as through using the CAM studio editor. This flexibility permits development of sets of CAM templates to match particular business application scenarios by cloning and adapting existing exchange templates.

3 Conclusion

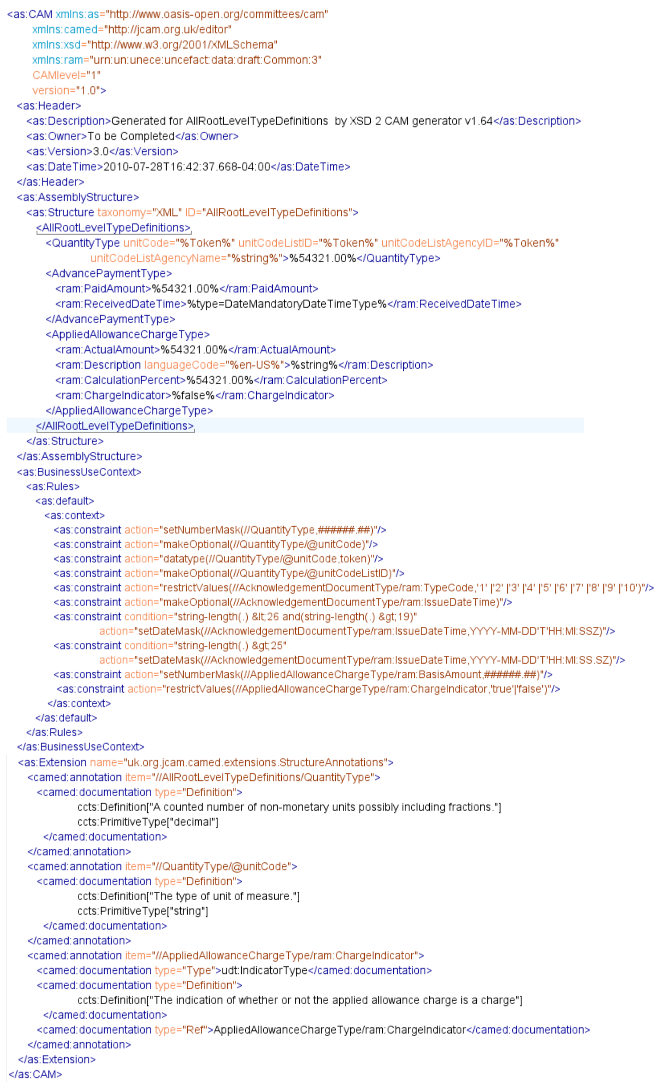
Using the CAM template XML validation framework approach, you can enforce validation checks consistently and then rapidly change rules to fine-tune message handling to match particular partner exchanges and content. By externalizing the validation rules, which in conventional deployment have been embedded deep inside the backend application code, you have much better control and management along with more predictable message handling. In addition the CAM rules templates can optionally be shared with partners to facilitate better content handling alignment across systems.

With a more adaptive and fault tolerant process, the template based validation framework is able to handle a wider variation in content and, hence, more easily support a broad set of interaction partners with reduced support and maintenance costs—which is the opposite of normal XML validation practice today.

The use of open source greatly facilitated collaboration on developing the solution and integrating the CAMV engine into the deployment environment.

Overall, this project demonstrated that innovative use of XML and dynamically configurable XML rule templates can provide a better, more stable, faster, and capable customer application experience than relying on static compiled code resources alone.

Appendix A – Example CAM template



```xml
<as:CAM xmlns:as="http://www.oasis-open.org/committees/cam"
        xmlns:camed="http://jcam.org.uk/editor"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:ram="urn:un:unece:uncefact:data:draft:Common:3"
        CAMlevel="1"
        version="1.0">
  <as:Header>
    <as:Description>Generated for AllRootLevelTypeDefinitions  by XSD 2 CAM generator v1.64</as:Description>
    <as:Owner>To be Completed</as:Owner>
    <as:Version>3.0</as:Version>
    <as:DateTime>2010-07-28T16:42:37.668-04:00</as:DateTime>
  </as:Header>
  <as:AssemblyStructure>
    <as:Structure taxonomy="XML" ID="AllRootLevelTypeDefinitions">
      <AllRootLevelTypeDefinitions>
        <QuantityType unitCode="%Token%" unitCodeListID="%Token%" unitCodeListAgencyID="%Token%"
                      unitCodeListAgencyName="%string%">%54321.00%</QuantityType>
        <AdvancePaymentType>
          <ram:PaidAmount>%54321.00%</ram:PaidAmount>
          <ram:ReceivedDateTime>%type=DateMandatoryDateTimeType%</ram:ReceivedDateTime>
        </AdvancePaymentType>
        <AppliedAllowanceChargeType>
          <ram:ActualAmount>%54321.00%</ram:ActualAmount>
          <ram:Description languageCode="%en-US%">%string%</ram:Description>
          <ram:CalculationPercent>%54321.00%</ram:CalculationPercent>
          <ram:ChargeIndicator>%false%</ram:ChargeIndicator>
        </AppliedAllowanceChargeType>
      </AllRootLevelTypeDefinitions>
    </as:Structure>
  </as:AssemblyStructure>
  <as:BusinessUseContext>
    <as:Rules>
      <as:default>
        <as:context>
          <as:constraint action="setNumberMask(//QuantityType,######.##)"/>
          <as:constraint action="makeOptional(//QuantityType/@unitCode)"/>
          <as:constraint action="datatype(//QuantityType/@unitCode,token)"/>
          <as:constraint action="makeOptional(//QuantityType/@unitCodeListID)"/>
          <as:constraint action="restrictValues(//AcknowledgementDocumentType/ram:TypeCode,'1' |'2' |'3' |'4' |'5' |'6' |'7' |'8' |'9' |'10')"/>
          <as:constraint action="makeOptional(//AcknowledgementDocumentType/ram:IssueDateTime)"/>
          <as:constraint condition="string-length(.) &lt;26 and(string-length(.) &gt;19)"
                    action="setDateMask(//AcknowledgementDocumentType/ram:IssueDateTime,YYYY-MM-DD'T'HH:MI:SSZ)"/>
          <as:constraint condition="string-length(.) &gt;25"
                    action="setDateMask(//AcknowledgementDocumentType/ram:IssueDateTime,YYYY-MM-DD'T'HH:MI:SS.SZ)"/>
          <as:constraint action="setNumberMask(//AppliedAllowanceChargeType/ram:BasisAmount,######.##)"/>
           <as:constraint action="restrictValues(//AppliedAllowanceChargeType/ram:ChargeIndicator,'true'|'false')"/>
        </as:context>
      </as:default>
    </as:Rules>
  </as:BusinessUseContext>
  <as:Extension name="uk.org.jcam.camed.extensions.StructureAnnotations">
    <camed:annotation item="//AllRootLevelTypeDefinitions/QuantityType">
      <camed:documentation type="Definition">
                ccts:Definition["A counted number of non-monetary units possibly including fractions."]
                ccts:PrimitiveType["decimal"]
       </camed:documentation>
    </camed:annotation>
    <camed:annotation item="//QuantityType/@unitCode">
      <camed:documentation type="Definition">
                ccts:Definition["The type of unit of measure."]
                ccts:PrimitiveType["string"]
       </camed:documentation>
    </camed:annotation>
    <camed:annotation item="//AppliedAllowanceChargeType/ram:ChargeIndicator">
      <camed:documentation type="Type">udt:IndicatorType</camed:documentation>
      <camed:documentation type="Definition">
                ccts:Definition["The indication of whether or not the applied allowance charge is a charge"]
       </camed:documentation>
      <camed:documentation type="Ref">AppliedAllowanceChargeType/ram:ChargeIndicator</camed:documentation>
    </camed:annotation>
  </as:Extension>
</as:CAM>
```

Fig. 4. XML syntax representation of CAM Template

Related Work

Schematron XML validation framework and ISO specification
http://www.schematron.com/resources.html
Reference Saxon XSLT engine implementation and Sourceforge project
http://saxon.sourceforge.net/

References

1. OASIS Content Assembly Mechanism (CAM) technical specification, version 1.1, published Boston, MA – http://docs.oasis-open.org/cam (2006).
2. OASIS SET TC work on automated component mapping – http://www.oasis-open.org/committees/set
3. Kathuria, P., Roberts, M.E., Webber, D.R.R. : XML Validation Framework using OASIS CAM (CAMV). In: IBM DeveloperWorks,
http:/www.ibm.com/developerworks/library/x-camval/index.html May (2010)

4. JCAM Engine with XML Editor / Validator: See information about the CAMV project at the SourceForge Web site.
5. The OASIS CAM Wiki: Visit a resource site for users and developers of CAM templates and CAM processors.
6. Meet CAM: A new XML validation technology (Brian M. Carey, developerWorks, September 2009): Read an introduction and overview of CAM.
7. Taking XML Validation to the Next Level: Introducing CAM: Read an article series representing CAM: The Missing Manual.