Heuristic Approach to Manage Semantic Heterogeneity and Data Inconsistency in Enterprise Data Integration

Srinivasan Shanmugam and Lixin Tao
Pace University Seidenberg School of CSIS, White Plains, New York

Abstract—XML syntax and semantic validations are critical to the correct service transaction specification and service integration based on existing distributed and heterogeneous computing services. Current industry practice of XSLT-based Schematron validation may produce invalid results, and contributes a reusable XML validator component that supports sound integrated syntax/semantic validations and event-driven integration with its environment through public APIs. XML is a self-describing language, and data owners do not follow a standard in XML elements/attributes. i.e., data owners have freedom to define their own tags/attributes and nesting orders in XML, but, this inconsistency leads to constraint management inefficiency (redundant constraints/expensive reformulations). Hence the challenge, if constraints are specified to the concepts that can be applied to different XML syntax structure. This will impact the semantic validations flexibility. Heuristics to identify possible semantic heterogeneity between XML documents that have any syntactic difference and data inconsistency were also proposed. In this paper, we propose a heuristic-based mechanism to manage the data consistency and semantic heterogeneity that achieves data interoperability by ensuring more flexibility/efficiency in enterprise data integration.

Keywords-XML; OWL-Schematron; co-constraint; syntax validation; semantic validation; integrated validation; co-constraint; conceptual validation; semantic heterogeneity;

I. INTRODUCTION

Valid XML documents are critically important to services computing [7]. The service requests are often in form of XML documents. Web services, the basic communication technology for service access and new service integration based on the existing distributed and heterogeneous services, are based on XML dialects SOAP and WSDL [7].

The service consumer and provider must use the same XML dialect so they could understand each other. An XML dialect specifies the syntax of a class of XML (instance) documents including the supported tag names, element nesting, the supported attributes, and the basic element and attribute data types. DTD and XML Schema (XSD) are the standard schema languages to define XML dialects [2]. XML validating parsers [2], based on either the SAX or DOM framework, can be used to validate whether an XML instance document satisfies the syntax constraints specified in a DTD or XML Schema document.

But in services computing, there are many semantic constraints or co-constraints among the components of an XML instance document that cannot be specified by DTD or XML Schema. For example the value range of an element in an electronic medical record may depend on whether the record is for a male or female patient, and the sales tax rate in an e-commerce transaction depends on the state value for the transaction. Schematron [1] is a popular rule-based XML dialect that allows us to specify such co-constraints for a class of XML documents and then use a standard Schematron validator to validate the co-constraints without coding. Table 1 lists the common co-constraint types supported by Schematron and XSD [4][5].

Over the past decade, the standard implementation of the Schematron validator is to use a standard XSLT style sheet [6][3] to transform a Schematron document into a new validator XSLT style sheet, and then use the latter to validate the XML instance documents, as shown in the following Figure 1.

Table 1 Co-constraints Supported by Schematron and XSD

<table>
<thead>
<tr>
<th>Language Feature</th>
<th>Schematron</th>
<th>XSD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sibling content</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Sibling attribute values</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Mutual exclusion</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Element type from attribute presence</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Element type from attribute content</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Attribute type from element content</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Attribute value exclusion</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Abstract Patterns</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
The constraints are specified on top of hard coded XML tags and attributes names. Also, users tend to change the tag(s) names. Changing XML tag and attribute names mandates more constraint creation and inconsistencies as well as increases the asset maintaining the constraints. For example, while the transaction using the XML instance document, different Companies/Departments describe the same thing may use the different tag name.

In this research, our focus is to identify the syntaxes involved in XML (Dialects/Instance) documents that need to be categorized. Hence, each syntax is identified as a different category. This categorization can lead to different types mapping of element/attribute name to the ontology concepts. These also specify constraints in terms of the concepts instead of the element/attribute names in XML documents. Each different syntax type needs to do map only once, and don’t do transformation on XML documents. Also, in this research we are extending ISO Schematron to support semantic constraint specification at concept/knowledge level with concept inheritance, efficient notation for mapping element/attribute names to Ontology(OWL) concepts and properties, efficient for integrated syntax/semantic validation of ontology-based constraint specification.

Figure 1 Schematron Validation Using XSLT

II. SEPARATE SYNTAX AND SEMANTIC VALIDATIONS MAY NOT BE VALID

One important observation of the above XSLT-based implementation of Schematron validation is that it completely separates semantic validation from syntax validation. Now we use a simple counter example to prove that such separate validations may lead to invalid semantic validation results because the information in an XML instance document also includes those defined in the DTD or XML Schema syntax specifications.

Let the following excerpt be part of an XML instance document for e-commerce transactions, and it declares that a transaction has ID value “0120121” and amount “$225.45”.

```xml
<transacs>
  <trans>
    <trans_Id>0120121</trans_Id>
    <amount>225.45</amount>
  </trans>
  ……
</transacs>
```

Let the following excerpt be part of a DTD document for the XML dialect of the above e-commerce transaction, and it declares that a transacs element includes a sequence of one or more trans elements; each trans element includes elements trans_Id, amount, …, in the same order, and has an attribute pay_type that can take on value either “visa” or “master” with the default value being “visa”.

```xml
<!ELEMENT transacs (trans)+ >
<!ELEMENT trans (trans_Id, amount, …)>  
<!ATTLIST trans  pay_type (visa|master)  "visa"> 
……
```

The information in the above XML document actually includes the default value “visa” for attribute pay type specified in the DTD excerpt above. While this default value is available during syntax validation, it is not available to a Schematron validator if the semantic validation is separate from the syntax validation. Therefore the semantic validation will fail based on the XSLT-based validator. This counter example shows that in general semantic validation separated from syntax validation could be invalid.

III. INTEGRATED SYNTAX AND SEMANTIC VALIDATION THROUGH DOM AND XPATH- PURPOSE OF THE CO-CONSTRAINTS

In this research we integrate the syntax and semantic validations through a DOM tree [2] which is the output of the DOM-based syntax validation and the input of the XPath-based Schematron validation, as shown in Figure 2.
The DOM validating parser is first used to validate the XML document against its syntax specification in the DTD or XML Schema document, and all information in the XML and DTD/XSD documents is represented in the resulting DOM tree to the left. The same DOM validating parser is also used to validate the Schematron document against the Schematron’s XML Schema specification to ensure that the former is a valid semantic constraint specification and the resulting DOM tree to the right represents the Schematron document. Both of the two DOM trees are fed to our new XPath-based Schematron validator for semantic constraint validation.

Co-Constraint is a feature of Schematron, which is not possible in XSD and DTD. In general constraints are used in XML to enforce uniqueness of an element or an attribute. A co-constraint is a constraint between two or more values. A co-constraint can exist between data i.e. element-to-element, or element-to-attribute, or attribute-to-attribute. Also, a co-constraint can exist within a single XML document, or across multiple XML documents. The Table 1 is the high level representation of co-constraints, below Figure 3 shows the detailed and specific co-constraint types (20).

IV. CONCEPTUAL VALIDATOR

In addition to potentially invalid validation results, the XSLT-based Schematron implementation also has several additional drawbacks: (1) the validator result is for people to read thus the validator cannot be easily integrated with other system components; and (2) its functions are limited by the XSLT’s limitations and the latter was not designed for supporting semantic constraint validation. Previous integrated validator is designed as a reusable software component based on DOM Level 3 XPath [8]. It supports all key features of Schematron ISO [1] including abstract rules and abstract patterns, network integration through web services, and event-driven loose-coupling. Current conceptual validator is an extension of previous Integrated Validator. We generalize the concepts by introducing the ontology. The constraints are implemented in the ontology as base concept level, then specifying the constraint once in base concept level that can be supplying to the sub concepts. Since ontology supports the inheritance (is-a) concept, specification of constraints will not be duplicated. In this extension, we propose specifying assertion tags with ontology terms. Thus, with this specification, validations will not be based on tag names, which will be on the concepts. Most importantly, this research provides an open-source framework which serves as a test-bed for efficient and new conceptual validation. Below Figure 4 shows the detailed flow of conceptual validation flow.

![Figure 2 Integrated Syntax/Semantic Validation](image)

![Figure 3 Co-Constraint Types](image)

![Figure 4 Conceptual Validator](image)
indirectly implied in the original Schematron specification.

V. BLACK BOX TESTING

Blackbox is an approach that helps to test your cases with system which would expect to build. In order to accomplish the BlackBox approach, the following are required

- What would be the Input?
- What will be happening and handled by the solution suggested?
- What would we hypothesize as output?

Based on the above our requirements, below are the input/solution suggested and output

- Input – XML instance documents
- Syntactic/Semantic and Conceptual validation on the given XML document
- Output – Handling the Semantic Heterogeneity/ Data inconsistencies.

Below is the BlackBox approach in pictorial form

![BlackBox Process](image)

Figure 6 Balck Box Process

V. MANAGING SEMANTIC HETEROGENEITY AND DATA INCONSISTENCIES BY CONCEPTUAL VALIDATION

Semantic Heterogeneity occurs due to same concepts are referred by different terms (16). In general two major factors lead to Semantic Heterogeneity in XML documents i.e. inconsistency, ambiguity in XML documents. In the enterprise, representing same business data in two different formats by two different Line of Business (LOB) are common which causes the Semantic Heterogeneity. In medical realm, two hospitals will send patient information in two different formats to healthcare offices. The below example will explain that same concepts are referred by different terms (16).

For example: Patient(s) record from two hospitals PID tag in X1 xml document and SSN tag in X2 xml documents are notifying the Social Security Number of the patient.

<table>
<thead>
<tr>
<th>X1</th>
<th>X2</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Patient&gt;</td>
<td>&lt;Patient&gt;</td>
</tr>
<tr>
<td>……………</td>
<td>……………</td>
</tr>
<tr>
<td>&lt;PID&gt;123-45-6789&lt;/PID&gt;</td>
<td>&lt;SSN&gt;123-45-6789&lt;/SSN&gt;</td>
</tr>
<tr>
<td>……………</td>
<td>……………</td>
</tr>
<tr>
<td>&lt;/Patient&gt;</td>
<td>&lt;/Patient&gt;</td>
</tr>
</tbody>
</table>

Use of conceptual definitions allows us to perform semantic validation on the XML documents(X1/X2), regardless of the name of the tags and attributes. We no longer require a constraint check for exact tag in the document, but we can make use of the domain ontology to infer a semantic relationship between the concepts and terms (tags), consequently are that relationship to validate the tags in the document. Also, this helps to handle the ambiguity issues by generalizing the semantic relationships between document’s tag and the keywords [16, 17]. This allows us to create a minimal set of constraints, though the tags and attributes may change. Well tested robust validation of documents is the additional benefits. The below example depict advantages of conceptual validation and managing semantic heterogeneity and data inconsistencies using ontology [18].

Statement 1: “Physician Ann Taylor cares for patient Will Smith”.

The above statement can be written in two different syntaxes:

1. `<patient name='Will Smith'>
   <Physician>Ann Taylor</physician>
   </patient>`
2. `<physician name='Ann Taylor'>
   <Patient>Will Smith</patient>
   </physician>`

VI. COMPONENTS AND CONNECTIONS

OWL: Ontology is a dictionary, it is providing some terminology. Ontology will provide the bridge concept (terminology). Each Ontology describes the derived concepts. Ontology helps to improve the expressiveness semantic constraints. It helps to maintain uniformed semantic constraints. Basic use for ontologies will be referencing concepts to element /attribute name or element/attribute to concepts using the mapping files created by Syntax creators.
Schematron specifications (Semantic Constraints):
Schematron is an XML dialect to specify semantic or co-constraints on XML documents, just like DTD and XSD are for specifying syntax for XML dialects. Schematron writer can use terms in OWL to write rules. Constraints in Schematron rules using the concepts terms. Schematron writer can specify constraint using terminology defined in the ontology. Schematron writer can introduce artificial tags to specify the constraints in Schematron rules, which helps when dealing with more than one ontology and different syntaxes. Below Figure 10 is the Schematron constraint which has artificial tag ($Zip)

VII. HEURISTIC BASED METHOD
Methodology and Algorithm for Ontology Based Constraint Specification and Validation while current schema languages like Schematron define semantic constraints on XML tag and attribute names which carry limited semantic information, we introduce ontologies into Schematron ISO and specify constraints on well-defined concepts and support multi-tag representation of the same concept or single tag representing different concepts in different contexts, and constraint abstraction and reuse so constraints on a concept automatically apply to all of its derived concepts.

The below diagram is a subset of conceptual validator which describe the functionalities of mapper/Referencing Xpath to OWL and OWL concepts to Xpath
In any given mapping, following three main phases for any mapping process (19):
1. Mapping Discovery
2. Mapping Representation
3. Mapping Exploitation/Execution

We need to do this kind of details mapping, and come up with different meaning of different syntax for same data. When the syntax is correct then we map the mapping files to concepts. This mapping should be distributed in multiple files. Because two mapping files should belong to two different syntax Xml dialects. These two XML dialect’s Syntax creator is responsible for the mapping file content, i.e. /Customer/shipto/recipient/ should map to Addressee, when Addressee is concept in owl. Second Syntax creator is responsible to writing the second mapping file content, i.e. /Customer/billto/payee/ should map to Addressee. There is no centralizing file to provide to such a mapping to Addressee = /Customer/shipto/recipient/ and /Customer/billto/payee/. It is very complicate to enumerate all potential syntax element/attribute name that map to Addresssee by a one Syntax creator. This should be written piece by piece of the each Syntax creator.

Overview:
Business tends to change different syntax in XML document, though the information is same, due to their business requirements. We are using the Address information as the example which we mentioned in the previous chapter. Our Challenge is the Address XML documents have different syntax, but one general business validation would be Address XML content always should have city/state/zip. Our object is Schematron is used to specify semantic constraint, but, Semantic constraint is using xpath of specific Address syntax to define it. In this case, we have many different versions of schematron rules based on different syntax.

Example for the problem:
We consider Address based syntax1 is UC1_USAddressSchema1.xsd and syntax2 is UC1_USAddressSchema2.xsd using the current ISO-Schematron. When we need to validate the syntax1 (UC1_USAddressSchema1.xsd), we need to use a SchematronRules1 based syntax1 for the validation. When we need to validate the syntax2 (UC1_USAddressSchema2.xsd), we need to create another schematron rule, say Schematron2, based on syntax2 for the validation. That is the problem we don’t want to too many different schematron specifications based on different syntax. How to solve this problem?

Recommendations of not do (Inefficient)

a) Should not map the xml instance to the concepts
b) Should not transform xml instance document

Recommendations to use

c) map the xml dialect to the concepts

Example for the Approach:
Given two XML dialects, some of the attributes/element name in each XML dialect is different. Before the validation, each of the data owners will provide the mapping for attribute/element values in XML dialect to the shared concept in the ontologies.

We consider Address based syntax1 is UC1_USAddressSchema1.xsd and Address based syntax2 is UC1_USAddressSchema2.xsd. Syntax creator should create mapping file. Mapping file is basically for specific mapping of XML dialect’s element/attribute names to the concepts based on syntax1 and syntax2. We will discuss the validation for address based on syntax1. We want to write Schematron specification only once. In order to write Schematron once, we use concepts in the ontology to specify the constraints instead using xpath.

When we are validating for Address based syntax1, we need to check concept of the “ZIP” must be available in owl. If any instance documents are based on xml instance syntax1, use the mapping file for
syntax1 to transform the Schematron rules. Schematron rules contain the constraint based on concepts to Schematron constraints based on specific xpath defined based on syntax1.

The above doesn’t require changing validation engine. We use the existing validation engine to do the validation. If any future XML instances are using syntax1, then we can transform and use same mapper for Syntax1. How to map for syntax1?

Following are the approach of mapping algorithm

**Mapping Discovery**
For each concept, mapper’s left side is the xpath based on syntax1 and right side is the concepts. When the syntax1 is correct then we map the mapping files to concepts.

**Mapping Representation**
This mapping will be used to transform schematron rules. Schematron rules initially written based on concepts, and not on xpath. Because we need to generate Schematron rules for syntax1, hence we use syntax 1’s mapper to replace Schematron rule each of the concepts with xpath based on syntax1. After that process each xml instance based on syntax1 to use any schematron implementation using XSLT or integrated validator to do the validation.

**Mapping Exploitation/Execution**
Each XSLT, We will have a corresponding mapping file left side is will be xpath and right side concept. This mapping file use to transform Schematron rules from concept based Schematron constraints to xpath based Schematron constraints. Our object is Schematron rules once in concepts, then this concept based schematron rule will be transformed into each syntax. We can generate specific xpath based schematron rules beforehand, as soon as we know the existence of syntax, that can be generate d automatically corresponding Schematron rules then we can use any Schematron validator to do the validation. Below Figure 12 will depict the mapping associations

This kind of validation based on xpath, it has nothing to do with OWL. The validation is only on xpath. The purpose of OWL is it allows Schematron writer to specify the constraint at once based on concept in the owl. Each syntax owner/writer responsible and ability to map each of the concepts to the corresponding the xpath. When we validating for addressing based syntax1, we need to check concept of the “ZIP” must be available in owl.

**Mapping xpath to Concept:** Write a mapping file to map its element/attribute names to ontology concepts.

1. Read element and attributes from the XSD
2. Matching the elements/attributes to the Left (L1) values in the mapping file.
3. If it matches store the right (R1) side values
4. Look for matching values in OWL file Store the (R1) Values

**Converting concept to xpath:** Schematron writer can introduce artificial tags to specify the constraints in Schematron rules, which helps when we dealing with more than one ontology and different syntaxes. We need to use the mapping file for the Schematron constraint based on concepts (artificial tags) to specific xpath defined based on syntaxes.

1. Look at the rule in hand, identify the concept (artificial tags) for which the rule is written
2. Locate the concept in mapping file.
3. Find the corresponding element/attribute in the mapping file.
4. Substitute the element/attributes to the artificial tag in the Schematron rules

C2.7
VIII. Conclusion

Our system is limited and we can reconcile in data representation, and we deal with heterogeneity in data representation in these several forms. Change the order of Elements, change the data representation attributes to elements, we enumerated which we can do and we are not going to totally solve the problem. These problems which we described are simple but it is very challenging in enterprise data integration. It is required to work on how to address the different change in syntax. If it gives a satisfactory solution to one domain, we can apply this to other domains for few extra those are the longtime features. We need to categorize the different syntax changes how to reconcile the difference in different categories. Later, we need to work on the algorithm the notation level how to specify constraint on the medical and how to validate.

REFERENCES

[12] Isabel F. Cruz Huiyong Xiao, “The Role of Ontologies in Data Integration”,ADVIS Lab, UIC, pages 2-3
b. Benjamin Nguyen,“Transforming XML Schema to OWL Using Patterns”, University of Versailles st-Quentin &INRIA-Rocquencourt Project SMIS, France, 2011
[21] Using Ontologies in Enterprise Data Integration