XML DATA CONSTRAINT AND XINCAML

Jing Min Xu, Ying Nan Zuo, Shun Xiang Yang, Zhong Tian
IBM China Research Laboratory, 4/F, No.7, 5th Street, Shangdi, Beijing 100085, PRC

Keywords: XML Data Constraint, Constraint Specification Language, XincaML

Abstract: XML is becoming the de facto standard for data exchange. Because it brings structures and semantics to the contents, it is very important for applications to verify the validity of XML data before further processing. W3C XML Schema language can specify many of the constraints in XML data, but it lacks of the capability of expressing application specific inter-node constraints. Therefore XincaML (e**X**tensible **i**nter-**n**ode **c**onstr**a**int **M**arkup **L**anguage) is invented as a complement to XML Schema language to specify this kind of application constraints. XincaML is a descriptive inter-node constraint specification language. XincaML Processor is a reference Java implementation of the XincaML language parser and constraints checker. Developers can easily integrate the processor into their applications to handle inter-node constraints besides validating XML data against XML Schema. XincaML and the processor provide a common mechanism for applications to describe and process inter-node constraints, thus significantly eliminate the need and labor to hard-code the constraint handling in applications and speeds up the application development.

1 INTRODUCTION

More and more applications adopt XML [T. Bray, 2000] documents as method of data exchange. The exchanged XML documents convey not only the structure of the data but also application specific semantics of them. It is very important for applications, especially ones performing critical commerce transactions, to ensure the validity of the documents. Invalid data might lead to unexpected application behavior or even denial of service. Since XML data structures can be validated already, it is only natural to be able to validate more XML data constraints, e.g. relationships between elements or attributes belonging to different contexts, invariants over data models, limits over attribute values and so on.

W3C XML Schema [D. C. Fallside, 2001] [H. S. Thompson, 2001], as a typical grammar-based schema language [Dongwon Lee, 2000], has taken a big step in that direction. It allows users to specify various XML data constraint such as structures of XML documents and data types of elements or attributes. However, it is short of the ability to specify data constraints among elements or attributes located on different sub-branches of an XML document tree. We call this kind of constraint as inter-node constraint. Although such data relationships in an XML document are almost everywhere as part of application semantics, they can't be specified by current XML Schema language easily, if at all.

W3C Schema Working Group also noticed the inter-node constraint problem and considered adding co-constraints into W3C XML Schema as one of desiderata in the latest XML Schema requirements working draft [C. Campbell, 2003]. But we still think it is too rigid in that sense and diminishes the flexibility of XML. In addition, it may make the schema language too difficult to analyze.

In order to specify inter-node constraint, several constraint specification languages have been proposed. Schematron [Rick Jelliffe, 2002] is a typical one. It is a pattern-based language and more constraint-oriented than XML Schema [Dongwon Lee, 2000]. It is able to specify constraint patterns in XML documents based on the presence, names and value of elements and attributes along paths. Schematron completely relies upon XPath [J. Clark, 1999] and XSLT [James Clark, 1999] for defining context dependent rules. On one hand, it makes Schematron quite concise. On the other hand, it makes it more difficult for applications to analyze the structure and definition of constraints, let alone optimized code for efficient constraints handling. This is the main motivation for inventing XincaML (eXtensible inter-node constraint Markup Language).

Min Xu J., Nan Zuo Y., Xiang Yang S. and Tian Z. (2004).
XML DATA CONSTRAINT AND XINCAML.
In *Proceedings of the Sixth International Conference on Enterprise Information Systems*, pages 479-485
DOI: 10.5220/0002652604790485

Unlike XML Schema and Schematron, XincaML is designed as a constraint specification language rather than a schema language. Its constraint expressions are more descriptive and declarative than those of Schematron, so business rules that applications need to check can be mapped to XML data constraints more easily. XincaML concentrates on descriptively expressing inter-node constraints that XML Schema can not express. Hence, it is considered as a helpful supplement of XML Schema.

As a constraint specification language, XincaML focuses more on descriptiveness. It not only makes XincaML more like a natural language but also enables XML developers to write more optimized code for efficient constraint handling and to play with the constraint definition structure itself when needed. In addition, XincaML also gives users the flexibility of applying XPath to XincaML to the extent they like so that they can balance between a concise expression and a descriptive one.

A XincaML Processor reference implementation is already available for downloading from IBM Alphaworks [Ying Nan Zuo, 2002]. It is a Java package and provides APIs for constraints parsing and checking. Applications are able to concentrate on data processing by delegating the data validation work to the processor. The violation handling mechanism of the processor, which enables callbacks of the application specific code for violation handling, helps application developers create cleaner program logic.

In the rest of this paper, we'll first introduce the basic concepts of XML data constraint, and then discuss how XincaML expresses the inter-node constraints and its advantages. The reference implementation of XincaML Processor and several usage scenarios are also introduced so as to give a basic idea of how XML developers integrate XincaML into their applications. Some future works are presented in the end of the paper.

2 XML DATA CONSTRAINTS

Handling data constraints has been around for quite sometime. In a database, data constraints are mostly part of the database schema. The schema serves for two purposes. First, it describes the structure or type of the data; second, it describes certain constraints including assertion of the keys and inclusion dependencies. In general, all constraints on data can be divided into two groups-integrity constraints and data validity constraints. Integrity constraints (type constraints, path constraints etc.) describe semantic integrity of data. Data validity constraints describe conditions of validity of data. [Ekaterina Pavlova, 2000]

Semi-structured data is a generation of structured data in a sense, so it has integrity constraints and data validity constraints similar to those in structured data. XML data is usually treated as semi-structured data, thus the constraints in semi-structured data can mostly be applied to XML data. In practice, most of real-world logical constraints to data are very complex and not just pure integrity constraints or data validity constraints. It is impossible to make a complete taxonomy of all these constraints. But some kinds of constraints are most commonly used by lots of XML applications. It is more valuable to investigate these kinds of constraints.

In general, the commonly used XML data constraint can be classified as the following four categories:

i. **Containment structural constraint (structures)**: This kind of constraint describes the basic structure of XML documents such as element hierarchies, attributes of a element, inheritance for elements and attributes, cardinality of elements and so on.

ii. **Lexical structural constraint (data types)**: This kind of constraint describes data types and data formats in order to check the domain range of values of elements or attributes as well as ensure they follow certain formats.

iii. **Integrity constraint (identity constraint)**: This kind of constraint describes the reference relationship between elements or attributes like the key/foreign key mechanism in the relational database.

iv. **Inter-node constraint (co-constraint)**: This kind of constraint describes the presence/value dependencies between elements or attributes belonging to the same or different sub-branches of an XML document tree. It is usually the most fundamental part of data semantics.

XML Schema as of today has already covered the first three kinds of constraint, but it lacks of the capability of expressing the inter-node constraints in an XML document. XincaML is proposed to complement it. Before we go into detail about XincaML, let's take a closer look at the inter-node constraints.

First, a small piece of XML data is presented below serving as an example of XML data that have inter-node constraints.

```
<Contacts>
    <Person title="Mr">
        <Name> John Smith </Name>
        <Gender>Male </Gender>
    </Person>
    <Person title="Ms">
        <Name> Joan Smith </Name>
```

```
        <Gender>Female </Gender>
    </Person>
</Contacts>
```

At least one can find one inter-node constraint here that if the title of a person is “Mr”, then the gender of him must be “Male”. It’s hard to express this data constraint using XML Schema language.

In order to depict the details of inter-node constraint more clearly, let us simplify the XML data model first. In most cases, what applications care are the values and relationships of elements or attributes in an XML document. So, in XincaML, we only examine element nodes (and their values) and attribute nodes (and their values).

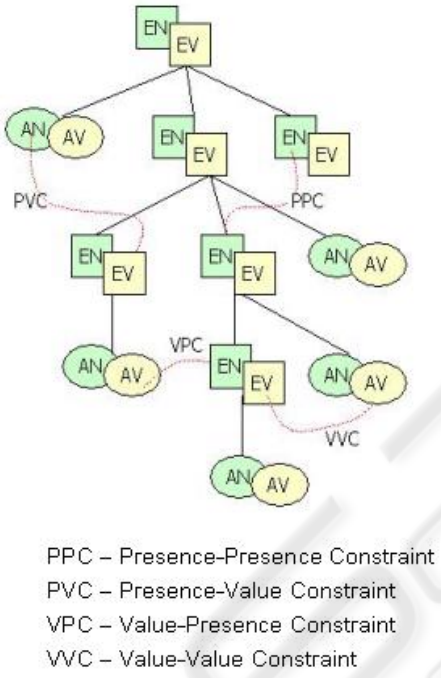

Figure 1: A Visual Representation of Four Types of Inter-node Constraints

The figure 1 provides a visual representation of four types of inter-node constraints in an XML document tree. The element node, attribute node, element value and attribute value are represented as EN, AN, EV and AV respectively.

The inter-node constraints are divided into the following four types:

i. **Presence-Presence Constraint (PPC)**: The presence of a set of nodes depends on the presence of another set of nodes.

ii. **Presence-Value Constraint (PVC)**: The presence of a set of nodes depends on the values of another set of nodes.

iii. **Value-Presence Constraint (VPC)**: The values of a set of nodes depend on the presence of another set of nodes.

iv. **Value-Value Constraint (VVC)**: The values of a set of nodes depend on the values of another set of nodes.

Since the nodes involved in an inter-node constraint do not necessarily have any direct structural relationships among them, it is difficult for a grammar-based schema language such as W3C XML Schema to express these constraints.

It is a general practice for application developers to validate data against structure definitions. They’d like to have as much application semantics checked as possible before the data is further processed. A common expression of the inter-node constraints enables application developers to relegate the constraint checking work to any third party tools so that they can focus on the application logic. XincaML is such a common mechanism for expressing the inter-node constraints.

3 XINCAML LANGUAGE AND PROCESSOR

First of all, XincaML is in XML syntax, which brings several benefits:

1) Users do not have to learn another proprietary syntax.

2) The expressed constraints can be readily applied to existing XML applications.

3) The expressed constraints can be stored in an XML storage system along with XML documents they are applied to.

4) The expressed constraints are extensible.

A XincaML document may contain one or more constraint declarations. The <constraint> element is used to declare a constraint. Multiple constraints can be grouped together with the element <constraints> as a constraint group. Two attributes are defined for the <constraint> element. One is the “name” attribute, which is used to uniquely identify a constraint within a constraint group. The other is the “context” attribute, which specifies the scope of a constraint. The context attribute is required to be expressed as an XPath path expression. For example, the following constraints are defined:

```
<xinca:constraints>
   <xinca:constraint name=”titleConstraint”
context=”/Contacts/Person”>
      … …
   </xinca:constraint>
   <xinca:constraint name=”otherConstraint”
context=”/Contacts/Other”>
      … …
```

```
  <xinca:constraint>
</xinca:constraints>
```

The constraint "titleConstraint" and "otherConstraint" are used to constrain the nodes located in the different sub-trees of an XML document: the "/Contacts/Person" sub-tree for the former and the "/Contacts/Other" sub-tree for the latter.

Each constraint is expressed as a rule, which usually contains three segments: <if>, <then> and <action>. The <if> and <then> segments are both required to state an assertion. If the assertion in the <if> segment holds true, the assertion in the <then> statement is also required to be true. If the assertion in the <then> statement is not true, the constraint is violated. However, if the assertion in the <if> segment holds false, the constraint will be ignored by the constraint checker. The <action> segment acts as a placeholder for applications to insert their particular constraint handling mechanism when the constraint is violated. XincaML defines the <message> element as a default violation handling method: reporting a message.

Each assertion states which nodes are present (presence assertion) or their values satisfy an expression (value assertion). The expression should be any expression which evaluation is a Boolean value such as a logical expression, a comparison expression or a logical expression. [Ying Nan Zuo, 2002] For example, the constraint "titleConstraint" can be expressed as follows:

```
<constraint name="titleGenderConstraint"
context="/Contacts/Person">
  <if>
    <assert>
      <node id="titleNode" location="@title"/>
      <satisfy>
        <eq>
          <stringValue ref="titleNode"/>
          <stringValue value="Mr"/>
        </eq>
      </satisfy>
    </assert>
  </if>
  <then>
    <assert>
      <node id="genderNode" location="Gender"/>
      <satisfy>
        <eq>
          <stringValue ref="genderNode"/>
          <stringValue value="Male"/>
        </eq>
      </satisfy>
    </assert>
  </then>
  <action>
    <message>
      If the title is "Mr." then the gender of the person
must be "Male".
    </message>
  </action>
</constraint>
```

This constraint specifies that if the title of a person is "Mr", then the gender of the person must be "Male". Thus, XincaML is quite straightforward due to its descriptiveness. However, the side effect is that it makes XincaML sort of verbose. In order to resolve this problem and fully leverage the functionality of XPath, XincaML allows expressing assertions or rules as XPath expressions. For examples, the "titleConstraint" can be expressed as follows:

```
<constraint name="titleGenderConstraint"
context="/Contacts/Person">
  <xpath exp="(@Title = 'Mr' and Gender = 'Male') or
@Title != 'Mr' " />
  <action>
    <message>
      If the title is "Mr." then the gender of the person
must be "Male".
    .</message>
  </action>
</constraint>
```

This feature gives users the flexibility of applying XPath expressions to XincaML to the extent they feel comfortable. But expressing assertions or rules as XPath expressions brings the following two disadvantages:

1) More devious for users to map business rules to XincaML constraints if they are not comfortable with XPath.

2) More difficult for applications to analyze the structure of XincaML constraints themselves.

So users have to balance the advantages of expressing assertions or rules as XPath expressions against the disadvantages based on application scenarios as well as XincaML processor they intend to use.

XincaML enhanced XML data validation is a multiple-stage process that begins with W3C XML Schema validation, followed by XincaML checking to handle the inter-node constraints. Like XML parsers, which relieve applications of validating XML documents based on predefined schemas, XincaML processors relieve applications of parsing and checking inter-node constraints expressed by XincaML.

A reference XincaML processor implementation is available from IBM Alphaworks. It contains two Java packages:

1) Constraint Parser, which parses XincaML constraints into a XincaML object tree so that

applications are able to navigate the tree to know the structures of each constraint.

2) Constraint Checker, which checks XincaML constraints against an XML document based on the constructed XincaML object tree.

The architecture of the XincaML Processor is depicted in the figure 2:

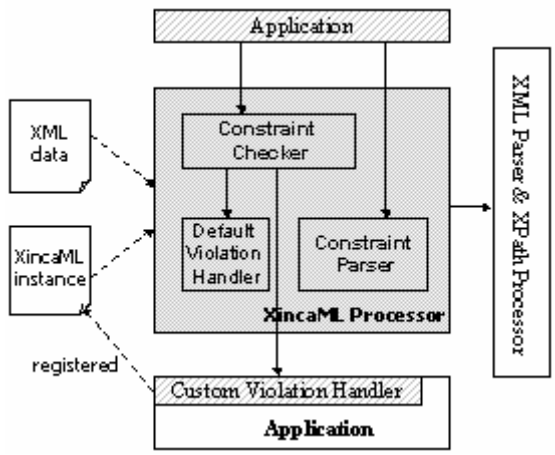

Figure 2: XincaML Processor Architecture

The XincaML Processor has a violation handling mechanism that allows application developers registering a violation handler for each constraint by specifying a <action> element to add custom application specific processing. For example:

```
<constraint name="titleGenderConstraint"
context="/Contacts/Person">
   ... ...
   <action>
      <handler>
           <java:class>xincaparser.SampleHandler
           </java:class>
           <java:method>showMessage</java:method>
      </handler>
   </action>
</constraint>
```

For the reference implementation, when a constraint violation occurs, the Constraint Checker will call the corresponding violation handler by use of Java reflection mechanism. The reference implementation also contains a default violation handler. An error message will be generated as return result if the checker does not find a registered violation handler.

In order to boost performance, the XincaML processor constructs an in-memory dependency graph that records which nodes are relevant to which constraints when parsing XincaML constraints. Therefore, it allows applications only checking constraints that are relevant to a specific node when its value changes. However, the dependency graph does not include the constraints that express assertions or rules as XPath expressions because XincaML does not reveal their structures.

4 USAGE SCENARIOS OF XINCAML

A typical use of XincaML processor is to use it as a guardian for XML applications. Any XML data must be validated first before they are fed into an application. If the processor tells the application that everything is ok, then the application goes on. Otherwise, the application should stop and deal with the data constraint violations through predefined violation handlers. The XincaML processor allows XincaML constraints to be embedded in **xs:annotation / xs:appinfo** W3C XML Schema elements, seamlessly combining W3C XML schema and XincaML. Thus it makes the XincaML enhanced XML Schema language compatible with conventional XML Schema validation, and allowing applications to put in additional validation and processing when appropriate.

In some cases, applications may not be willing to be stopped by the processors. They just want the processors to go through the input data, collect all the information about violated elements and attributes and then report to them. The applications may handle the constraint violations themselves later. XincaML Processor provides several violations reporting APIs [Ying Nan Zuo, 2002] to offer applications such kind of choices. With these APIs, an application can exactly locate the elements and attributes that violate the constraints.

In many visual modeling tools, a lot of application specific constraints (including syntax constraints and semantic constraints) exist among drawing elements. Usually, people use this kind of tools as thinking tools. They don't like to be interfered by such constraint details. It's better to deal with these constraints when the main body of the model is finished. XincaML can help a lot in this tools. For example, a state machine modeling tool which represents a state machine as an XML document, may use XincaML language to specify various constrains between states in a state machine and use XincaML processor to assist the well-form checking of the machine. Application specific constraints such as "There should be one and only one starting state in a state machine", "The transition target of a state must be an existing state in the state machine" and so on, are all mapped into XincaML constraint specification. After drawing a state machine using the modeling tool, a user can invoke the well-form checking function. The tool will

delegate the checking work to XincaML processor. Then, all constraints violations are reported through violations reporting APIs. The modeling tool can make use of these APIs to locate the violated states and highlight them to the user. Figure 3 illustrates the use of XincaML processor in this modeling tool.

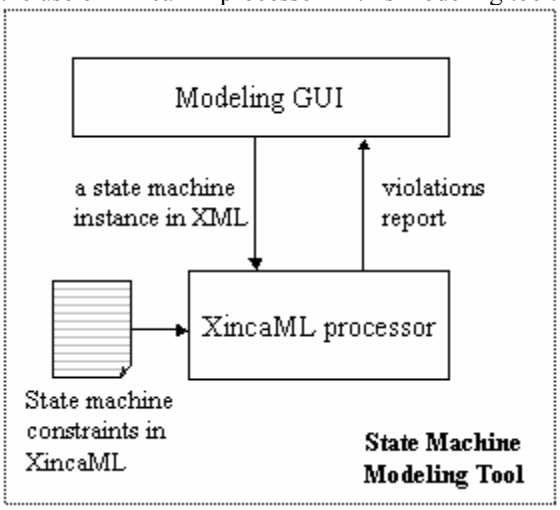

Figure 3: The Usage of XincaML Processor in State Machine Modeling Tool

In other cases, applications may want to know the detail syntax of each inter-node constraint so that they can generate their own codes to do the constraint checking. In these cases, only the XincaML language parser is needed. XincaML Processor Java package provides users a lot of classes, such as XincaParser class, XincaConstraint class, XincaAssert class, XincaLogicalExpression class and so on, to manipulate each component of a constraint definition. [Ying Nan Zuo, 2002]

To demonstrate this kind of usage scenario, we take D3From [Shun Xiang Yang, 2003] as an example. D3From is a dynamic Web form generator, which is used in an adaptive profiling framework for service provisioning [Shun Xiang Yang, 2003]. The framework provides flexible profiling mechanism to dynamically collect relevant information about service consumers according to diverse profiling requirements of different services in an e-Commerce platform. D3Form is used to generate different Web forms for different services according to their profiling requirements. The service profiling requirements are defined by use of XML Schema plus XincaML language. D3From takes the schema part as data and HTML form definitions, and takes XincaML part as constraints among the data elements that will be collected by the form. Figure 4 shows some details.

The HTML Form Generator is used to map XML schema into HTML form controls and apply XSL document on the generated form to control its appearance. The JavaScript Generator makes use of XincaML APIs to parse the XincaML constraint specifications and convert them to JavaScript codes. The generated scripts serve as inter-node constraint checker in client side when someone fills in the forms with data. Further more, if you specify the dependencies between form controls with PPC (Presence-Presence Constraint) or VPC (Value-Presence Constraint) style of constraints, the generated scripts are able to change the layout of the form based on users' input.

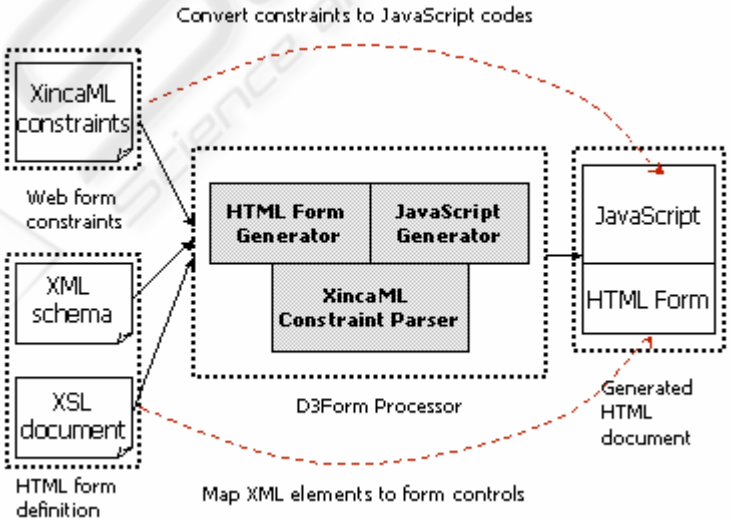

Figure 4: The using of XincaML in D3Form Processor

5 CONCLUSION AND FUTURE WORK

XincaML can describe some inter-node constraints that can't be expressed by the current XML schema language and therefore supplement it to capture application specific data constraints. The purpose of XincaML is to specify constraints that most often used by XML applications. For applications that require other arbitrary or complicated data constraints, they must perform their own additional validations.

XincaML Processor can validate XML documents against constraints specified in XincaML instances. The advantage of using XincaML in applications is that many of the constraints that previously had to be checked in applications can now be moved out of them and delegated to XincaML Processor. It can make users concentrate on the application processing logic itself and thereby create clearer application logic models.

Currently, we are thinking of adding some new features to XincaML. For example, supporting more logic expressions and constraints, checking of implication and conflicts among several constraints specified in one XincaML instance and applying constraints to multiple XML documents.

ACKNOWLEDGEMENT

We would like to thank Bob Schloss [rschloss@us.ibm.com], of IBM Watson Research Center, for his great contribution to the concept of XincaML.

REFERENCES

T. Bray, J. Paoli, C. M. Sperberg-McQueen and E. Maler, 2000. *Extensible Markup Language (XML) 1.0 (Second Edition).* W3C Recommendation, http://www.w3.org/TR/REC-xml, World Wide Web Consortium, Oct. 2000.

D. C. Fallside, 2001. *XML Schema Part 0: Primer.* W3C Recommendation, http://www.w3.org/TR/xmlschema-0, World Wide Web Consortium, May. 2001.

H. S. Thompson, D. Beech, M. Maloney and N. Mendelsohn, 2001. *XML Schema Part 1: Structures.* W3C Recommendation, http://www.w3.org/TR/xmlschema-1, World Wide Web Consortium, May. 2001.

Dongwon Lee, Wesley W. Chu, 2000. *Comparative Analysis of Six XML Schema Languages*. ACM SIGMOD Record, Vol. 29, No. 3, Sep. 2000.

C. Campbell, Ashok Malhotra and Priscilla Walmsley, 2003. *Requirements for XML Schema 1.1. W3C Working Draft,* http://www.w3.org/XML/Group/2002/07/xmlschema-1.1-current-reqs.html#N4000FC, Jan. 2003.

Rick Jelliffe, 2002. *The Schematron Assertion Language 1.5.* http://www.ascc.net/xml/resource/schematron/Schematron2000.html, Oct. 2002

J. Clark and S. DeRose, 1999. *XML Path Language (XPath) Version 1.0.* W3C Recommendation, http://www.w3.org/TR/xpath, World Wide Web Consortium, Nov. 1999.

James Clark, 1999. *XSL Transformations (XSLT) Version 1.0*. W3C Recommendation http://www.w3.org/TR/xslt, World Wide Web Consortium, Nov. 1999.

Zuo Ying Nan, Jing Min Xu and Shun Xiang Yang, 2002. *XincaML Technology*. IBM Alphaworks, http://www.alphaworks.ibm.com/tech/xincaml. Dec, 2002.

Ekaterina Pavlova , Igor Nekrestyanov and Boris Novikov, 2000. *Constraints for Semistructured Data*. Proc. of the Russian DL'2000, 214-219, Protvino, Russia, September 2000.

Shun Xiang Yang, Ying Nan Zuo , Jing Min Xu, Zhong Tian, 2003. *Adaptive Profiling Framework and System for Service Provisioning*. IEEE Conference on Electronic Commerce, USA 2003

C. Nentwich, W. Emmerich and A. Finkelstein, 2001. *Static Consistency Checking for Distributed Specifications.* Proceedings of the 16th International Conference on Automated Software Engineering (ASE), Coronado Island, CA, IEEE Computer Science Press, pages 115-124. November 2001.

Eric van der Vlist, 2001. *Comparing XML Schema Languages.* http://www.xml.com/pub/a/2001/12/12/schemacompare.html, Dec. 2001

Will Provost, 2002. *Beyond W3C XML Schema*. http://www.xml.com/pub/a/2002/04/10/beyondwxs.html, Apr. 2002.

M. H. Jacinto, G. R. Librelotto, J. C. Leite Ramalho and P. R. Henriques, 2002. *Constraint Specification Languages: comparing XCSL, Schematron and XML-Schemas.* XML EUROPE 2002, May 2002.

Ramalho, José C. and Henriques, Pedro R, 2001. *Constraining Content: Specification and Processing. XML* Europe 2001, International Congress Centrum (ICC), Berlin, Germany. 2001