Visual Modeling of XML Constraints Based on a New Extensible Constraint Markup Language

Jingkun Hu and Lixin Tao, *Senior Member, IEEE*

***Abstract*— With the mature of e-business on the Internet, the *eXtensible Markup Language (XML)* is rapidly becoming the industry standard for business-to-business (B2B) data integration. While *Document Type Definitions (DTDs)* and *XML Schemas* can be used to specify and validate syntactic constraints on XML documents, currently there are no effective languages or tools for specifying and validating semantic constraints, in particular dynamic ones, on XML documents.**

We conduct a critical review of the existing XML constraint languages and classify the types of semantic constraints based on their forms. We propose a new XML constraint language, *eXtensible Constraint Markup Language (XCML)*, which is more expressive than the current constraint languages by better supporting the specification of dynamic and inter-relationship constraints. *Unified Modeling Language (UML)* and *Object Constraint Language (OCL)* are adopted to support visual specification and automatic generation of XCML and XML Schema instance documents, which are further used by our reusable XSLT stylesheets to support both semantic and syntactic XML document validation.

The technologies proposed can be used in e-business data integration, XML data management, data warehousing, and decision support systems for various industry domains.

***Index Terms*—XML constraints, semantic validation, semantic constraint modeling, XML Schema, OCL, XSLT, XMI**

I. Introduction

Behind the success of e-business on the Internet is the ever increasing demand for business-to-business (B2B) enterprise system integration. The data processing systems of different companies need to communicate with each other to share data, pass business transactions, and hierarchically integrate finer-grain services into coarser ones. Data integration is becoming critical for communicating parties to have a common language and understand each other's data.

The *eXtensible Markup Language (XML)* [3], standardized by the *World Wide Web Consortium (W3C)* in February 1998, is self-describing, human and machine readable, extensible, flexible, and platform neutral. XML has become the standard format for exchanging information across the networks. To achieve the goal of data integration, the communicating parties need to agree on an XML dialect for their particular business domain and needs. This dialect is usually defined in a *Document Type Definition (DTD)* or *XML Schema* [8] document, which defines the syntax and data types to which all of its instance XML documents must conform. The data producer system will generate XML data according to their DTD or Schema definition. The data consumer system can use an XML validating parser to verify the syntax of the incoming data before passing them to its data processing system.

While syntax validation is important in preventing erroneous data from disrupting the data consumer system, it cannot verify the equally important non-structural semantic constraints on XML data. In reality, the value or presence of an element or attribute may depend on the value or presence of another element(s) and/or attribute(s); and the value scope of an element or attribute may vary for different document instances and be decided by system environment. A grammatically validated XML document does not guarantee to be meaningful. Even though XML Schema is much more powerful than DTD, it cannot be used to specify non-structural constraints. We need an extensible, expressive, platform-neutral, and domain-independent way of specifying semantic constraints on XML documents.

Another challenge for data integration is the specification of complex constraints on business data models. While in theory we can use a text editor to specify such constraints in a particular constraint specification language, the complexities of real-world business data structures could make such constraint specifications cryptic and error-prone. Ideally we could specify such constraints at a more abstract data model level so the human users and domain experts can visually help represent and verify the constraints, and the constraint documents could be derived from such models mechanically.

The third challenge is about constraint validation. XML validating parsers cannot use the constraint documents to validate non-structural constraints. Hard coding such constraints into a program is not attractive, since such a program may not truthfully implement the constraints, is not flexible for system modifications or extensions, and cannot be reused. Mature XML technologies should be used to provide a generic framework for automatic constraint validation

This paper first provides a critical review of the existing approaches for specifying semantic constraints on XML documents, and classifies the commonly used semantic

Manuscript received February 4, 2006. This work was supported in part by the Pace University CDNY grant 2006-2007.

Dr. Jingkun Hu is Senior Software Engineer with Philips Medical Systems, Milpitas, CA 95035 USA (email: jingkun.hu@philips.com).

Dr. Lixin Tao is with Pace University, Pleasantville, NY 10570 USA (e-mail: ltao@pace.edu).

constraints into a few categories according to their forms. A new more expressive XML-based *eXtensibe Constraint Markup Language (XCML)* is proposed to specify various semantic constraints including dynamic and inter-relationship constraints. *Unified Modeling Language (UML)* [19] and *Object Constraint Language (OCL)* [13] are used to support visual specification and automatic generation of XCML and XML Schema instance documents, thus greatly reducing the complexity in designing complex XML data structures with extensive semantic constraints. Reusable *XSLT stylesheets* [5] are designed to transform the XCML and XML Schema instance documents for an XML data model into model-specific stylesheets that can implement both semantic and syntactic XML document validation with an XSLT processor.

The technologies proposed in this paper can also be used in XML data management, data warehousing, and decision support systems for various industry domains.

II. Classification and Specification of XML Constraints

While XML syntactic constraints specify the static structure of a type of XML documents, an XML semantic constraint imposes static/dynamic limitations to value/presence (occurrence) of the elements/attributes of a type of XML documents.

An XML instance document exists in its system environment and its element/attribute values are usually cross-referenced in multiple documents. If an XML semantic constraint depends upon its environment, we say it is dynamic; otherwise we say it is static. A dynamic constraint may impose different limitations on an element or attribute for different instance documents defined by the same Schema.

A constraint can be expressed in the form of an assertion (true/false statement) or a conditional rule (if-then) with embedded assertions. While in theory the constraints could be all expressed as assertions, rule-based constraints allow for more natural and concise specification of many types of constraints.

For an assertion-based constraint, we call it simple or composite depending on whether it involves one element/attribute or more.

For a rule-based constraint, we call it simple if it is of an if-then structure; or composite if it contains an else-clause or nested rule-based constraints.

We can classify both syntactic and semantic constraints on XML documents that commonly appear in the literature into one of the following categories:

1. Well-formedness constraints: those imposed by the definition of XML itself such as the rules for the use of the < and > characters and the rules for proper nesting of elements.
2. Document structure constraints: how an XML document is structured starting from the root of a document all the way to each individual sub element and/or attribute.
3. Data type/format constraints: those applied to the value of an attribute or a simple element.
4. Value constraints on elements or attributes: the value (range) of an element/attribute that cannot be specified by a DTD or XML Schema document; such constraints could be either static or dynamic.
5. Presence constraints of attributes and/or elements: the presence of an attribute or element and the number of occurrences of an element, which could be either static or dynamic.
6. Inter-relationship constraints between elements and/or attributes: the presence or value of an element/attribute depends on the presence or value of another element/attribute.
7. Consistency constraints: corresponding elements/attributes in multiple documents have consistent values.

The above categories 1 through 3 are for syntactic constraints, and categories 4 through 7 are for semantic constraints. Constraints in categories 1 through 3 can be specified by DTD or Schema documents and validated with an XML validating parser. Constraints in categories 4 and 5 are usually more natural to be specified with assertions, and their static ones can also be specified using XML Schema. Constraints in categories 6 and 7 are usually more natural to be specified with conditional rules.

While XML Schema is richer than DTD in expressing the structures, data types, and data formats, it is not powerful enough to express semantic constraints. There have been three options to extend XML Schema in expressing semantic constraints [25]:

- to supplement XML Schema with another XML constraint language,
- to write program code to express semantic constraints, and
- to express semantic constraints with an XSLT/XPath stylesheet.

The advantage of the second option is that with a single programming language you can express all the semantic constraints. But, it cannot leverage XSLT technology. Each of the constraint documents becomes a legacy application. In the third option, each application creates its own stylesheet to specify and check constraints that are unique to the application. However, these stylesheets are not human-oriented and not reusable. It is also a challenge to create complex stylesheets. Therefore, the first option is preferable.

The major XML constraint languages in the literature are Schematron [7], XML Constraint Specification Language (XCSL) [12], XincaML [10], and xlinkit [15]. Schematron, a pattern-based XML constraint language, can express a substantial number of semantic constraints, specifically assertion-based constraints. It is the most popular XML constraint language among the existing ones. But it is difficult

to express rule-based constraints and dynamic constraints. XCSL has not been used widely and has the disadvantages similar to Schematron. XincaML, recently proposed by IBM, focuses on the inter-relationship constraints. It cannot express dynamic constraints and requires a proprietary application to perform validation because it does not leverage XSLT, a core XML technology. Xlinkit is intended for the consistency check of elements among distributed XML documents and beyond the scope of this paper.

In the next section we introduce a new XML constraint language, *XCML (eXtensible Constraint Markup Language)*. XCML provides a set of syntax elements to express both static and dynamic semantic constraints in their either simple or composite forms. Table 1 compares the expressiveness of XML Schema, Schematron, XincaML, XCSL, and XCML.

Table 1: Expressiveness Comparison of Constraint Languages

Language	Assertion-based constraint				Rule-based constraint			
	simple		composite		simple		composite	
	stat	dyn	stat	dyn	stat	dyn	stat	dyn
XML Schema	Yes	No	No	No	No	No	No	No
Schematron	Yes	No	Yes	No	Yes	No	No	No
XincaML	Yes	No	Yes	No	Yes	No	Yes	No
XCSL	Yes	Yes	Yes	Yes	Yes	No	No	No
XCML	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**

III. EXTENSIBLE CONSTRAINT MARKUP LANGUAGE XCML

The existing constraint languages cannot express certain constraints including dynamic value/occurrence constraints and composite rule-based constraints. We propose a new XML constraint language – XCML. It is an XML based markup language. It leverages the core XML technologies including XML Schema and XPath. The XCML syntax is defined in an XML Schema document. XCML instance documents can be either embedded within XML Schemas as annotations or as separate constraint documents.

The XCML instance documents are simple, concise, easy to create, and easy to use to validate XML documents. It supports not only assertion-based constraints and simple rule-based constraints, like if-then, but also composite rule-based constraints like nested if-then-else. XCML supports parameters for expressing dynamic constraints. It supports XPath 1.0 [6] or above so that various expressions can be processed by XSLT processors. XCML also supports the visual specification of constraints on XML data models, as shown in Section 4.

A. XCML Syntax

The XCML syntax is defined in an XML Schema document. An XCML document contains a single top-level element *Constraints*, which contains a sequence of one or more *Constraint* elements. A Constraint element must specify its scope through its *context* attribute. It starts with an optional sequence of *Parameter* elements, each specifying the name, type, and optional default value of a parameter for passing in an external environment value. The main body of a Constraint element is either a *Rule* element or an *Assertion* element. A Rule element is basically a sequence of *If* element, *Then* element, and an optional *Else* element. An If element allows for the specification of an assertion as the value of its test attribute. A Then element or an Else element allows for the specification of either an assertion as the value of its test attribute, or a nested if-then-(else) structure.

B. XCML Instance Document Samples

In this section we provide simple examples to demonstrate that XCML can be used to specify constraints that some of the other constraint languages cannot, as summarized in Table 1.

- *Example for assertion-based constraints which are simple and dynamic.* This example declares that in the context of element “employee”, the value of “taxRate” must be equal to the value of parameter “rate”, which is dynamically set by the system environment.

```
<Constraint context="employee">
  <Parameter>
      <name>rate</name>
      <type>decimal</type>
      <defaultValue>0.07</defaultValue>
   </Parameter>
   <Assertion test=”taxRate=$rate”/>
</Constraint>
```

- *Example for assertion-based constraints which are composite and dynamic*. This example declares that in the context of element “employee”, the value of “tax” must be equal to the value of element “netIncome” multiplied with the value of parameter “rate”, which is dynamically set by the system environment.

```
<Constraint context="employee">
  <Parameter>
      <name>rate</name>
      <type>decimal</type>
      <defaultValue>0.07</defaultValue>
   </Parameter>
   <Assertion test=”tax=netIncome*$rate"/>
</Constraint>
```

- *Example for rule-based constraints which are simple and*

dynamic. This example declares that in the context of element "employee", if the value of "netIncome" is less than or equal to the value of parameter "level", then the value of "taxRate" should be 0.05.

```
<Constraint context="employee">
  <Parameter>
    <name>level</name>
    <type>decimal</type>
  </Parameter>
  <If test="netIncome<=$level"/>
  <Then test="taxRate=0.05"/>
</Constraint>
```

- *Example for rule-based constraints which are composite and static*. This example declares that in the context of element "employee", if the value of "netIncome" is less than or equal to $50,000, then the value of "taxRate" should be 0.05; otherwise if the value of "netIncome" is less than or equal to $100,000, then the value of "taxRate" should be 0.07; otherwise the value of "taxRate" should be 0.1.

```
<Constraint context="employee">
  <If test="netIncome<=50000"/>
  <Then test="taxRate=0.05"/>
  <Else>
    <If test="netIncome<=100000"/>
    <Then test="taxRate=0.07"/>
    <Else test="taxRate=0.1"/>
  </Else>
</Constraint>
```

- *Example for rule-based constraints which are composite and dynamic*. This example declares that in the context of element "employee", if the value of "netIncome" is less than or equal to the value of parameter "level1", then the value of "taxRate" should be 0.05; otherwise if the value of "netIncome" is less than or equal to the value of parameter "level2", then the value of "taxRate" should be 0.07; otherwise the value of "taxRate" should be 0.1.

```
<Constraint context="employee">
  <Parameter>
    <name>level1</name>
    <type>decimal</type>
  </Parameter>
  <Parameter>
    <name>level2</name>
    <type>decimal</type>
  </Parameter>
  <If test="netIncome<=$level1"/>
  <Then test="taxRate=0.05"/>
  <Else>
    <If test="netIncome<=$level2"/>
    <Then test="taxRate=0.07"/>
    <Else test="taxRate=0.1"/>
  </Else>
</Constraint>
```

Table 1 summarizes the expressiveness of four XML constraint languages Schematron, XincaML, XCSL, and XCML based on our classification of semantic constraint forms.

IV. Visual Modeling of XML Semantic Constraints

The generation of XML constraint documents for real-world complex XML documents is a challenging topic. Even though XCML syntax supports more natural specification of many semantic constraints, XCML documents are still system-oriented and not easy for communicating with domain experts.

We propose a model-driven approach to automate the XCML document generation process. Our approach is based on visual modeling of XML data structures (XML data modeling) and the three-level-design approach (conceptual, logical, and physical levels) for generating XML Schema documents [4][21][22][23].

Our approach starts with a UML class diagram representing the visual modeling of an XML data structure. The invariant structure of *Object Constraint Language (OCL)* [13] is used to specify semantic constraints associated with classes, attributes, or associations. The output is our *constrained conceptual model*, which can facilitate communications between domain experts/users and data modelers. Our *constrained logical model* is obtained from the constrained conceptual model after we annotating its classes, attributes and associations with stereotypes from Carlson's UML profile for XML Schema [4] and our UML profile for XCML Schema, the latter is described in Figure 1.

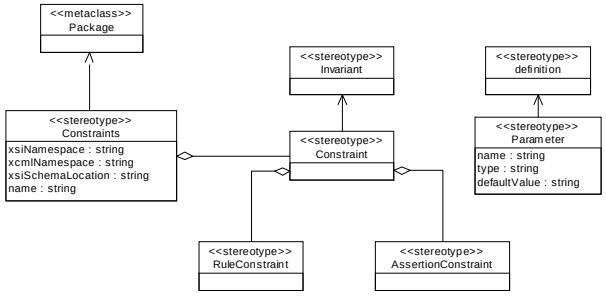

Figure 1: UML profile for XCML Schema

Our *physical models* are XML Schema and XCML instance documents derived from our constrained logical models. We

use *XML Metadata Interchange (XMI)* [20][24][20] and *XSLT* [5] technologies to accomplish this task. The major advantage of doing so is that both XMI and XSLT are open standards and their toolkits are open source and freely available. We designed and implemented three reusable sets of XSLT stylesheets [9]. The workflow of this process is shown in Figure 2. A constrained logic model is first written in an XMI (a kind of XML) file. The first XSLT stylesheet was used to extract information related to syntax and constraints out of the XMI file with the help of an XSLT processor. The extracted partial XMI document is further processed by the same XSLT processor, once to derive XML Schema instance document according to our second XSLT stylesheet, and the second time to derive XCML instance document according to our third XSLT stylesheet.

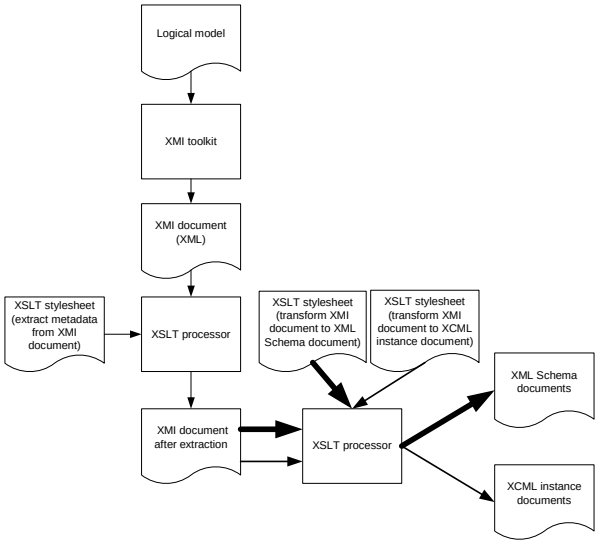

Figure 2: Workflow of deriving XML Schema and XCML instance documents

Now we present a concrete example for an *Employee* profile. Figure 3 shows the constrained conceptual model in which three semantic constraints are specified with OCL invariants: (1) an employee has savings fund if and only if he/she has worked for five years in the company; (2) an employee's net income should be equal to his/her salary plus his/her bonus minus his/her tax; (3) an employee will manage one or more departments if and only if he/she is a manager.

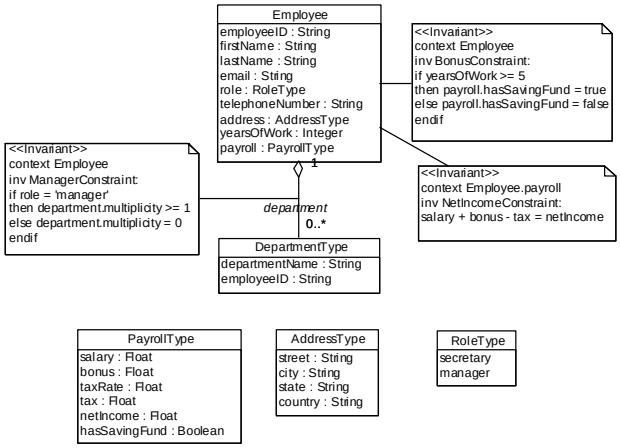

Figure 3: Constrained conceptual model of *Employee* profile

Figure 4 shows the constrained logical model for the Employee profile.

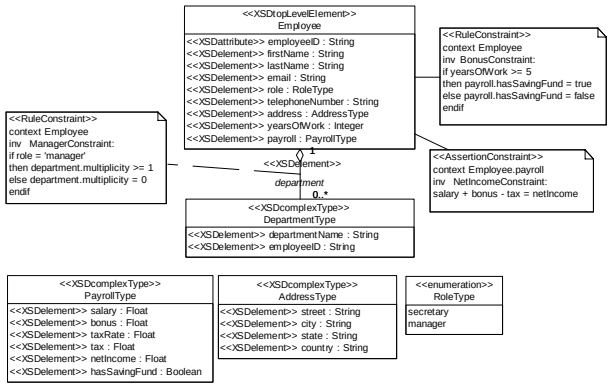

Figure 4: Constrained logical model of *Employee* profile

Listing 1 shows the XCML instance document derived from the constrained logical model for the Employee profile.

Listing 1: XCML instance document for *Employee* profile

```
<?xml version="1.0" encoding="UTF-8"?>
<xcml:Constraints
    xmlns:xcml="http://www.csis.pace.edu/dps/xcml">
    <Constraint context="employee">
        <Rule>
            <If test="role='manager'"/>
            <Then test="count(department) >= 1"/>
            <Else test="count(department) = 0"/>
        </Rule>
    </Constraint>
    <Constraint context="employee">
        <Rule>
            <If test="yearsOfWork = 5"/>
            <then test="payroll/hasSavingFund = 'true'"/>
            <Else test="payroll/hasSavingFund = 'false'"/>
        </Rule>
    </Constraint>
    <Constraint context="employee/payroll">
        <Assert test="salary + bonus - tax = netIncome"/>
    </Constraint>
</xcml:Constraints>
```

V. XSLT-Based XML Constraint Validation

While the syntactic validation of an XML document is straightforward once its XML Schema is available, the semantic validation of an XML document is much more complicated. This section focuses on how to perform the semantic validation of an XML document against its XCML instance document.

The workflow of validating XML documents is shown in Figure 5. The syntactic validation against XML Schemas is executed in the first step. If there are any syntactic errors, the validation process stops. Otherwise, the semantic validation is performed.

A reusable XSLT stylesheet [9] is written to convert an XCML instance document into a model-specific XSLT stylesheet, with the help of an XSLT processor. The latter stylesheet is, in turn, used to semantically validate the XML instance documents, with the help of an XSLT processor, to see whether their contents make sense to the particular application.

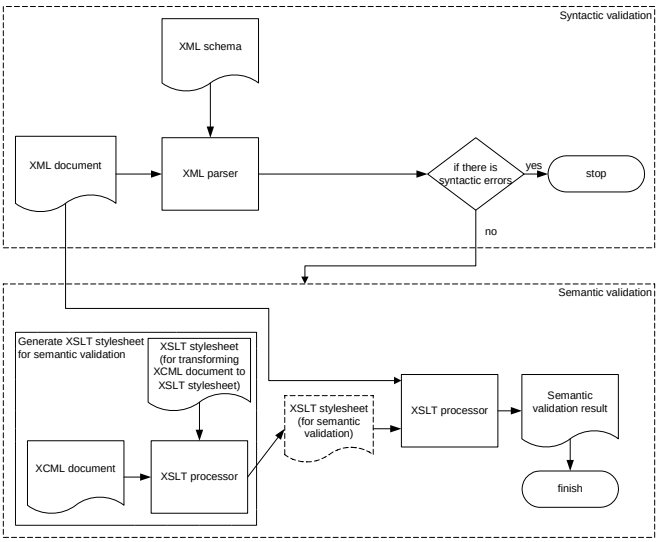

Figure 5 Workflow of XML document validation

VI. Conclusion

A complete framework has been proposed for XML semantic constraint specification, modeling, document generation, and validation, all based on public domain technologies XML, XML Schema, UML, OCL, XSLT, and XPath. Its potential applications include system data integration, XML data management, data warehousing, and decision support systems for various industry domains like e-commerce.

References

[1] Altova, Inc., "XMLSpy 2004: XML development environment," 2004, http://www.xmlspy.com (valid by February 20, 2006)

[2] T. Beale, "OCL 2.0 Review Based on Initial Submission 1.6 - 6 Jan 2003," draft review. *Deep Thought Informatics*, April 4, 2003, http://www.deepthought.com.au/it/ocl_review.html (valid by February 20, 2006)

[3] T. Bray, et al., "Extensible Markup Language (XML) 1.0," *World Wide Web Consortium (W3C) Recommendation*, 1998, http://www.w3.org/TR/1998/REC-xml-19980210 (valid by February 20, 2006)

[4] D. Carlson, *Modeling XML Applications with UML: Practical e-Business Applications*, Addison-Wesley, 2001.

[5] J. Clark, "XSL Transformations (XSLT) 1.0," *W3C Recommendations*, 1999, http://www.w3.org/TR/xslt (valid by February 20, 2006)

[6] J. Clark and S. DeRose, "XML Path 1.0 (XPath)," *W3C Recommendation*, 1999, http://www.w3.org/TR/xpath (valid by February 20, 2006)

[7] L. Dodds, "Schematron: Validating XML Using XSLT," *Proceedings of XSLT UK Conference*, Keble College, Oxford, England, 2001, http://www.ldodds.com/papers/schematron_xsltuk.html (valid by February 20, 2006)

[8] D. Fallside, "XML Schema Part 0: Primer," *W3C Recommendations*, 2001. http://www.w3.org/TR/xmlschema-0 (valid by February 20, 2006)

[9] J. Hu, "Visual Modeling of XML Constraints Based on a New Extensible Constraint Markup Language," *doctoral dissertation*, CSIS, Pace University, New York, 2003.

[10] IBM Corporation, "eXtensible Inter-Nodes Constraint Mark-up Language (XincaML)," 2002, http://www.alphaworks.ibm.com/tech/xincaml (valid by February 20, 2006)

[11] IBM Corporation, "XMI toolkit 1.05," 1999, http://www.alphaworks.ibm.com/tech/xmitoolkit (valid by February 20, 2006)

[12] M. H. Jacinto et al., “Constraint Specification Languages: Comparing XCSL, Schematron and XML-Schemas,” *XML Europe 2002*, Barcelona, Spain, 2002
[13] W. Jos and K. Anneke, *Object Constraint Language: Precise Modeling with UML*, Addison-Wesley, 1998.
[14] Microsoft Corp., *MSXML 4.0 Service Pack 2 (Microsoft XML Core Services)*, 2003, http://www.microsoft.com/downloads, (valid by February 20, 2006)
[15] C. Nentwich et al., “Xlinkit: A Consistency Checking and Smart Link Service,” *Research Note RN/00/66*, University College London, Dept. of Computer Science, 2000.
[16] OMG, *XML Metadata Interchange Production of XML Schema Specification 1.0, 2001*, http://www.omg.org/docs/formal/03-05-02.pdf (valid by February 20, 2006)
[17] OMG, *Unified Modeling Language Semantics v1.1*, 1997, http://www.omg.org/docs/ad/97-08-04.pdf (valid by February 20, 2004)
[18] OMG, *OMG Unified Modeling Language Specification 1.4*, 2001, http://www.omg.org/docs/formal/01-09-67.pdf (valid by February 20, 2006)
[19] OMG, *Unified Modeling Language Specification 2.0 Suite*, 2003, http://www.omg.org/uml (valid by February 20, 2006)
[20] OMG, *XML Metadata Interchange Specification 1.1*, 1999, http://www.omg.org/docs/formal/00-11-02.pdf (valid by February 20, 2006)
[21] W. Provost, *UML for W3C XML Schema Design*, 2002, http://www.xml.com/pub/a/2002/08/07/wxs_uml.html (valid by February 20, 2006)
[22] J. C. L. Ramalho, “Constraining Content: Specification and Processing,” *XML Europe 2001*, Berlin, Germany, 2001.
[23] N. Routledge et al., “UML and XML Schema,” *Thirteenth Australasian Database Conference (ADC2002)*, Monash University, Melbourne, Victoria, 2002.
[24] Unisys Software Corp, “Unisys Rose UML 1.3.6 plug-in,” 2003, http://www-106.ibm.com/developerworks/rational/library/content/03July/2500/2834/Rose/rational_rose.html, (valid by February 20, 2006)
[25] XML-Dev List Group, “Extending XML Schemas,” *XML Schemas: Best Practices*, http://www.xfront.com/BestPracticesHomepage.html (valid by February 20, 2006)

Jingkun Hu is a senior software engineer at Philips Medical Systems - Nuclear Medicine SPECT, Milpitas, CA. Part of this paper is based on his dissertation research work. He received his Doctorate degree in Computing from Pace University in 2004.

Lixin Tao is a tenured full professor of computer science with Computer Science Department of Pace University. He obtained his Ph.D. degree in computer science from University of Pennsylvania in 1988. His current research interests include Internet computing, XML and Web services, distributed software component technologies, and combinatorial optimization. He is a senior member of IEEE.