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

by
Jingkun Hu

Submitted in partial fulfillment
of the requirements for the degree of
Doctor of Professional Studies
in Computing

at

School of Computer Science and Information Systems

Pace University

December 2003

We hereby certify that this dissertation, submitted by Jingkun Hu, satisfies the dissertation requirements for the degree of *Doctor of Professional Studies in Computing* and has been approved.

_____-_____

Lixin Tao Date
Chairperson of Dissertation Committee

_____-_____

Fred Grossman Date
Dissertation Committee Member

_____-_____

Fran Gustavson Date
Dissertation Committee Member

School of Computer Science and Information Systems
Pace University 2003

Abstract

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

by
Jingkun Hu

Submitted in partial fulfillment
of the requirements for the degree of
Doctor of Professional Studies
in Computing

December 2003

The eXtensible Markup Language (XML) is rapidly becoming the industry standard format for exchanging information across the networks. The validity of XML documents is very important for users and/or applications that consume these documents. The data model of an XML document is usually defined in Document Type Definitions (DTDs) or XML Schemas that can then be used to validate their instances – XML documents. It is not enough to check the validity of XML documents using XML Schema or DTD because they cannot specify some non-structural constraints like relationships and consistencies.

Several researches have been conducted on how to express XML constraints and how to use them to validate XML documents. However, they do not have a comprehensive classification of XML constraints. Some of them are not flexible enough to define reusable XML constraint documents. Some of them are not convenient to design XML constraint documents. Also, none of them discusses the design automation of XML constraint documents. In this research, we conduct a critical review of the existing XML constraint languages, from which a comprehensive classification of XML constraints is made. We propose an improved XML constraint language – eXtensible Constraint Markup Language (XCML). It provides the syntax to define XML constraints that are either assertion-based or rule-based. We also propose a visual modeling of XCML constraints that is highly integrated with XML data modeling at the design phase. It automates the generation of XCML documents.

We have developed a reference implementation of the model-driven automation of XML Schema and XCML document generation. The generated XML Schemas are used for syntactic validation. The XCML document is transformed to an XSLT stylsheet by a reusable stylesheet we developed. The transformed stylesheet is then used for semantic validation. The result of this research can be used in various industry domains including

XML data management, data mining and data warehousing, and decision support systems.

Acknowledgements

Getting this dissertation research running and done is even more tough than life, especially with highly demands from work, school, and family.

I would like to thank my little daughter, Lily, for her patience. I kept telling her, "We'll go fishing after I finish my homework." Thanks to my wife, Li, for sacrifices and understanding. She has been taking care of all the housework and raising our children. Thanks to my old daughter, Xuejiao, for helping her mom taking care of housework. I would like to dedicate this to my parent and my wife's parent for their support.

This dissertation would not be complete without the guidance from my advisor, Dr. Tao, who has helped in many aspects of my thinking, writing, and publishing of the dissertation. He kept on pushing, driving, and encouraging me in getting this dissertation completed.

Thanks to the Committee Member, Dr. Grossman, for his suggestions and comments, tireless encouragement and support. Thanks to the Committee Member, Dr. Gustavson, for his suggestions and comments. Thanks to my classmates for sharing and helping each other. I've had the privilege to be part of a group of exceptional professionals.

I'd like to thank my boss, Dr. Routh, and ex-boss, Dr. Mankovich, for their support, my colleague, Dr. Lee for his comments and suggestions, and Philips Research USA for the opportunity and the challenge.

Table of Contents

List of Tables

List of Figures

List of Listings

Chapter 1

Introduction

Connectivity and interoperability have been the main obstacles for business-to-business (B2B) integration in heterogeneous environments because communication protocols, interfaces, and data representation were platform dependent. Several distributed computing architectures like DCOM, CORBA, and JINI in the computing industry have been trying to resolve the B2B system integration problem for more than a decade. The advent of the eXtensible Markup Language (XML) [1] opened new opportunities in addressing this problem.

XML, standardized by the World Wide Web Consortium (W3C) in February 1998, was originally intended to be the standard format for exchanging information across the networks. Since then it has gone far beyond information exchange because it is self-described, human and machine readable, extensible, flexible, and platform neutral. The W3C Web site carries dozens of XML related specifications in the areas from XML document processing to XML applications like VoiceXML [2] - the Voice Extensible Markup Language, Scalable Vector Graphics (SVG) [3] - a language for describing two-dimensional graphics in XML, and Web Services [4] - a suite of open architecture and XML technology for exchanging structured information in a decentralized, distributed, and platform independent environment. Based on XML, several domain markup languages have also been developed like Electronic Business XML (ebXML) for E-

Commerce [5], Web Ontology Language (OWL) for Ontology [6], Clinical Document Architecture (CDA) from Healthcare Level Seven [7], DocBook for document publishing [8], and XML Database (XMLDB) [9] for XML database management systems.

With the extensive use of XML in the various industries, the validation of XML data becomes more and more critical in areas like data integration, data warehousing, decision support, and protocol negotiation. The XML specification 1.0 describes two kinds of constraints on XML documents: well-formedness and validity constraints. Informally, the well-formedness constraints are 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; while validity constraints are those on document data structure, such as the accepted element types and/or element attributes, and the allowed nesting of these elements. The validity constraints are usually specified with a particular Document Type Definition (DTD) or XML Schema [10]. XML documents can be validated against their corresponding DTDs or Schemas. In practice, the validity constraints are not only on the XML document structure but also on the document content. XML Schema, which is more expressible than DTD, has limited expressiveness in specifying validity constraints especially those on the document content. As a result, these constraints cannot be validated using XML Schema. This research aims to complementing XML Schema in validating those constraints not expressible in XML Schemas by proposing an eXtensible Constraint Markup Language (XCML). We also put forward a model-driven approach to representing constraints on XML data models and automating the whole process of validating XML constraints using the eXtensible Stylesheet Language Transformations (XSLT) [11] technology.

1.1 Validation of XML Document Constraints

XML constraints are those restrictions to the structures, data types, data representations, inter-relationships among elements and/or attributes, and the content of XML documents. The DTD language, which was originally used for describing the structure of SGML documents, has traditionally been the most common and simple way of describing the structure of XML documents. However, DTD itself is not expressive enough to properly describe highly structured data. XML Schema, on the other hand, provides a much richer set of syntax for describing structures, data types, and constraints of structured data. It has become the most common method for specifying and validating XML documents. Consider this simple instance, *demo.xml*, the root element *SimpleDemo* contains two child elements: *UnitsInStock* and *ReorderFlag*. The definition of XML schema for *SimpleDemo* is given in *demo.xsd* shown in Listing 1-2.

Listing 1-1 XML instance, demo.xml

```
<?xml version="1.0"?>
<SimpleDemo xmlns:demo="http://www.jingkunhu.com/demo"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.jingkunhu.com/demo demo.xsd">
   <UnitsInStock>20</UnitsInStock>
   <ReorderFlag>false</ReorderFlag>
</SimpleDemo>
```

With XML Schema we can check the constraints:

- The root element *SimpleDemo* contains a sequence of elements, *UnitsInStock* followed by *ReorderFlag*;
- The element *UnitsInStock* contains an integer number; and,

- The element *ReorderFlag* contains a boolean value.

An XML Schema definition, *demo.xsd*, of the above constraints is:

Listing 1-2 XML Schema, demo.xsd

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
          targetNamespace="http://www.jingkunhu.com/demo"
          xmlns:demo="http://www.jingkunhu.com/demo"
          elementFormDefault="qualified">
   <xsd:element name="SimpleDemo">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="UnitsInStock" type="xsd:positiveInteger"/>
            <xsd:element name="ReorderFlag" type="xsd:boolean"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

However, XML Schema does not have the capability to express the following constraint:

Listing 1-3 Reorder flag constraint

- The value of the element *ReorderFlag* must be false if the value of the element *UnitsInStock* is greater than 10; otherwise,
- The value of the element *ReorderFlag* must be true.

This is a typical inter-relationship constraint that XML Schema cannot express. This constraint will be referred many times later.

In the XML community, the validation of XML documents is divided into two types: syntactical validation and semantic validation. Syntactic validation checks whether an XML document conforms to its XML schema in terms of document structures, data

types, and data representations. In the above example, any Schema-supported XML parsers can perform syntactical validation by checking the document (*demo.xml*) against its schema (*demo.xsd*). Semantic validation checks whether an XML instance meets the semantic constraints that XML Schemas cannot express. Some researchers call these constraints non-structural constraints. The typical ones are inter-relationship constraints among elements and/or attributes, domain-specific value constraints, and value consistency among multiple instances. The constraint in Listing 1-3 is an example of inter-relationship constraints.

1.2 The Challenges and the Driving Force

Syntactical validation is straightforward once XML Schemas are properly defined. Semantic validation is much more complicated than syntactical validation. The main challenges of doing semantic validation are:

- What are the semantic constraints?
- How to formally express semantic constraints?
- How to validate semantic constraints?
- Can the semantic constraints be considered in the design phase?
- How to leverage the existing XML core technologies?

It is still a very hot topic differentiate semantics from syntactics, specifically in describing the structures of XML documents. It is fair to say that XML Schema defines the syntax of XML documents. It is syntactically valid if an XML document conforms to its XML schema. On the other hand, XML Schema can define some value constraints like enumerations. It is also semantically valid to some extent when an XML document is

successfully validated against its XML schema. There is some overlap between syntactics and semantics in this specific field.

There are several ways to express semantic constraints such as using a programming language, defining another Schema language, leveraging XSLT/XPath, or something else. It is hard to decide which way is better. Another challenge is how to validate semantic constraints. Can we use the existing XML technologies for this task? So far, we have not seen how to automate the design and generation of various constraint documents.

Take the example in Section 1.1, the value of the element *ReorderFlag* tells whether an inventory system needs to reorder more units or not. This value depends upon the value of the element *UnitsInStock*. This inter-relationship constraint can be checked by a proprietary application. But such a legacy solution is error-prone, not reusable, and hard to maintain.

The extensive use of XML in the applications such as data integration, data warehousing, decision support, system configuration, and document publishing, plays an important role in demanding semantic validation on XML documents or fragments.

1.3 Related Researches

Three options to do semantic validation have been explored so far:

1. To supplement XML Schema with an XML constraint language,
2. To write program code to express additional constraints, and

3. To express additional constraints with an XSLT stylesheet or XPath [12] expressions.

Option 1 requires a constraint language to express additional constraints. It separates the semantic constraints from syntactic constraints. Option 2 brings in a number of disadvantages such as writing proprietary code, platform dependent, and not reusable. Option 3 removes the need to write proprietary code, but XSLT is a transformation language, not a constraint language. Several XML constraint languages have been proposed to express semantic constraints. Many researchers [13][14][15] prefer option 1.

Schematron, a pattern-based XML constraint language, can express a substantial number of semantic constraints, specifically assertion-based constraints. It is difficult to express rule-based constraints and dynamic constraints. XML Constraint Specification Language (XCSL) is another constraint language, but has not been used widely. It has the disadvantages similar to Schematron. The eXtensible Inter-Nodes Constraint Markup Language (XincaML) is yet another constraint language. It focuses on the inter-relationship constraints. It can express rule-based constraints and assertion-based constraints. It cannot express dynamic constraints. Another disadvantage is that it requires a proprietary application to perform validation because it does not leverage XSLT - a core XML technology. Xlinkit [16] is intended for the consistency check of elements among distributed XML documents. In addition, none of these approaches addresses modeling semantic constraints and automating the creation of different XML constraint documents.

1.4 Research Overview

1.4.1 Extensible Constraint Markup Language

In this research, we classify XML constraints into two main categories. They are assertion-based constraints and rule-based constraints. Assertion-based constraints are for the value format, range, pattern of an element/attribute, or the presence of an element/attribute. Rule-based constraints are for relationships among elements/attributes, e.g., the value or presence of an element/attribute depending upon the value or presence of one or more other elements/attributes. We come up with a new XML constraint language – XCML, or eXtensible Constraint Markup Language based on the existing XML constraint languages. It is much more expressive than the existing XML constraint languages by supporting dynamic constraints, inter-relationship constraints, and more. We compare XCML with other existing XML constraint languages.

1.4.2 Visual Modeling of XML Constraints

Model-driven design has been recognized as a good practice of software development process. The Model Driven Architecture (MDA) specification [17] from the Object Management Group (OMG) makes the Unified Modeling Language (UML) [18] [19] [20] [21] more than just a modeling language. In this research, we build a UML profile for XCML schema. Together with the Object Constraint Language (OCL) [19] [22] and a UML profile for XML Schema [23], it can model XML constraints and put them on XML data models. This approach allows the semantic constraints to be considered from the system design phase. It avoids miscommunication between software architects and

application developers. Furthermore, representing semantic constraints in UML models makes it possible to automate the generation of XML constraint documents.

1.4.3 Automatic Constraint Document Generation and XML Document Validation

Once such models are available, XML Metadata Interchange (XMI) [24] documents can be generated from UML models using an XMI toolkit. We implement a set of stylesheets to create XCML instance documents and XML Schemas from XMI documents. The XML Schemas are used for syntactic validation. We develop an XSLT stylesheet to generate the stylesheet from the XCML instance documents for semantic validation. This approach and the stylesheet modules can be reused in different applications.

1.5 Organization of Dissertation

Chapter 2 classifies XML constraints and investigates the existing XML constraint languages. Chapter 3 discusses the hypotheses and design goal of XCML and the schema design of XCML. A comprehensive comparison of the various XML constraint languages is performed. Chapter 4 introduces OCL and proposes a UML profile for XCML schema and illustrates how to model XML constraints. Chapter 5 demonstrates the implementation of automating the generation of XML Schemas and XCML instance documents. Chapter 6 describes the implementation of the XSLT template modules for generating XSLT stylesheets for the validation of XML documents against the XCML constraints. Finally, Chapter 7 gives the conclusions and future work.

Chapter 2

XML Constraint Languages

2.1 Classification of Constraints

XML constraints are the restrictions on the structures and/or the content of XML documents. The XML specification 1.0 describes two kinds of constraints on XML documents: *well-formedness* and *validity* constraints. Informally, the well-formedness constraints are 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, while validity constraints are the further constraints on document structure provided by a particular DTD. In reality, validity constraints already go far beyond the definition given in the XML specification. It can be non-structural type constraints. One of the known non-structural constraints is relationship-type constraint [15]. Consistency constraint is another type of non-structural constraints [16]. A general classification of XML constraints is discussed below:

1. Well-formedness constraints

   The well-formedness constraints are 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.

   Any XML parser can validate this kind of constraints.

2. Document structure constraints

This type of constraints specifies how an XML document is structured starting from the root of a document all the way to each individual sub element and/or attribute. This is usually defined using DTDs or XML Schemas.

The validation of such constraints can be done using any XML parser supporting XML Schema.

3. Data type/format constraints

This type of constraints is applied to the value of an attribute or a simple element. DTD does not have this capability, while XML Schema offers facilities for defining data types including primitive data types and derived data types. Aslo, XML Schema can further define the formats or representations of a value by specifying the length, pattern, etc.

4. Value constraints

The value of an element or attribute can be further constrained by specifying a fixed value or a value range. In most cases, the value constraints are static, which means that the value or value range of an element or attribute is fixed for all the instances of the same Schema document. This type of constraints is expressible in XML Schema. In some cases, these constraints are dynamic, which means that the value or value range of an element or attribute dynamically changes from case to case. This type of constraints is not expressible in XML Schema.

Value constraints can also be on more than one element that may join together by different operations, for example, the product of element *quantity* and element *priceUrice* must equal to the value of element *subtotal*. XML Schema cannot express this type of value constraints.

5. Presence constraints of elements and/or attributes

This type of constraints specifies the presence of an attribute or element and the number of occurrences of an element. Both DTD and XML Schema have this capability. The validation of such constraints can be done using any XML parsers. In some cases, the occurrences are also dynamic as value constraints. DTD and XML Schema cannot specify such dynamic occurrences.

Presence constraints can also be on more than one element that may join together by different operations, for example, the number of the occurrences of element *UnitsInStock* must equal to that of the element *product*. XML Schema cannot express this type of value constraints.

6. Inter-relationship constraints between elements and/or attributes

This is a kind of inter-relationship constraints defined in XincaML [15]. It is further categorized into the following subtypes:

a. Presence-Presence Constraint (PPC) - The presence of some elements or attributes depends on the presence of other elements or attributes.

b. Presence-Value Constraint (PVC) - The presence of some elements or attributes depends on the values of other elements or attributes.

c. Value-Presence Constraint (VPC) - The values of some elements or attributes depend on the presence of other elements or attributes.

d. Value-Value Constraint (VVC) - The values of some elements or attributes depend on or constrain the values of other elements or attributes.

This kind of constraints cannot be expressed using XML Schema. Therefore, they cannot be validated using XML Schemas.

7. Consistency constraints

This kind of constraints is applied to an element or attribute appearing in multiple XML documents to check whether all the occurrences have a same value. It is beyond the scope of XML parsers.

8. Other complex constraints

All the remaining constraints belong to this family of constraints.

We classify the above constraints into two types for the purpose of validation: assertion-based constraints and rule-based constraints. Categories 3, 4, and 5 belong to assertion-based constraints; and categories 6 and 7 belong to rule-based constraints. Category 8 is yet to be discovered and defined.

2.2 Constraint Examples

To illustrate a variety of constraints and ease the discussions later in this chapter and the following chapter, several constraint examples are given the following two sections.

The sample Employee profile schema similar to the example used in [15] is given in Listing 2-1.

Listing 2-1 Employee profile schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="employee" type="employeeType"/>
    <xs:complexType name="employeeType">
        <xs:sequence>
            <xs:element name="employeeID" type="xs:string"/>
            <xs:element name="firstName" type="xs:string"/>
            <xs:element name="lastName" type="xs:string"/>
            <xs:element name="email" type="xs:string"/>
            <xs:element name="telephoneNumber" type="xs:string"/>
            <xs:element name="address" type="addressType"/>
            <xs:element name="role" type="roleType"/>
            <xs:element name="department" type="departmentType" minOccurs="0"/>
            <xs:element name="yearsOfWork" type="xs:integer"/>
            <xs:element name="payroll" type="payrollType"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="addressType">
        <xs:sequence>
            <xs:element name="country" type="xs:string"/>
            <xs:element name="state" type="xs:string"/>
            <xs:element name="city" type="xs:string"/>
            <xs:element name="street" type="xs:string"/>
            <xs:element name="zipCode" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
    <xs:simpleType name="roleType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="secretary"/>
            <xs:enumeration value="manager"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:complexType name="departmentType">
        <xs:sequence>
            <xs:element name="departmentName" type="xs:string"/>
            <xs:element name="employeeID" type="xs:string" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="payrollType">
        <xs:sequence>
            <xs:element name="salary" type="xs:float"/>
            <xs:element name="bonus" type="xs:float"/>
            <xs:element name="taxRate" type="xs:float"/>
            <xs:element name="tax" type="xs:float"/>
            <xs:element name="netIncome" type="xs:float"/>
            <xs:element name="hasSavingFund" type="xs:boolean"/>
```

```
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

2.2.1 Examples of Assertion-Based Constraints

The following constraint examples are all on the instances of the above XML Schema.

2.2.1.1 Value Constraints

Take element *taxRate* under element *payroll* in Listing 2-1 as an example, a single value constraint can be specified as shown in Listing 2-2.

Listing 2-2 Single value constraint

- The Value of element *taxRate* must be 0.05.

In some cases, element *taxRate* may be one of given values as shown in Listing 2-3.

Listing 2-3 Multiple value constraint

- The Value of element *taxRate* must be one of the values 0.05, 0.06, or 0.08.

In some other cases, the value constraints are given in a range as shown in Listing 2-4.

Listing 2-4 Value range constraint

- The Value of element *taxRate* must be greater than 0.02 and less than 0.10.

The constraint to the value of element *taxRate* may change from state to state. In this case, the constraint is dynamic.

Listing 2-5 Dynamic value constraint

- The value constraint to element *taxRate* varies from state to state.

2.2.1.2 Presence Constraints

The similar constraints also apply to the presence or occurrence of elements, e.g., a constraint can be applied to the occurrences of element *EmployeeID* within *complextType departmentType*, as shown in Listing 2-6.

Listing 2-6 Occurrence constraint

- The number of occurrences of element *EmployeeID* is less than 25 (The total employee number must be less than 25 in a department).

2.2.1.3 Composite Constraints

Sometimes, assertion-based constraints apply to more than one elements or attributes, e.g., a constraint can be specified to several elements of payroll elements.

Listing 2-7 Composite constraint, Net income constraint

- The net income of an employee must equal the difference of the sum of salary and bonus after tax, i.e., netIncome = salary + bonus – tax.

2.2.2 Examples of Rule-Based Constraints

2.2.2.1 Simple Rule-Based Constraints

Rule-based constraints usually involve two or more elements or attributes. Simple rule-based constraints only specify a simple if-then rule, e.g., if an employee is a manager, then the managed department name must be given as shown in Listing 2-8.

Listing 2-8 Simple Managed department constraint

- If an employee is a manager, the managed department name must be given, i.e., if the value of element *role* is 'manager', then the occurrence of element *department* is 1.

2.2.2.2 Composite Rule-Based Constraints

A composite rule-based constraint is comprised of if-then-else rule, or each part has embedded if-then or if-then-else, or more than one elses, e.g., if we add an otherwise statement in the above example for further constraining element *department* as shown in Listing 2-9.

Listing 2-9 Composite Managed department constraint

- If an employee is a manager, the managed department name must be given, i.e., if the value of element *role* is 'manager', then the occurrence of element *department* is 1. Otherwise,
- the occurrence of element *department* is 0.

If the values or occurrences of elements vary from case to case, the rule-based constraints become dynamic.

2.3 Overview of Validating XML Documents

XML Schema is richer than DTD in expressing the structures, data types, data formats, and many more, which makes it dominant for validating XML documents. However, it is not powerful enough to express all kinds of constraints. There have been three options to extend XML Schema in expressing those constraints and validating XML documents so far:

1. to supplement XML Schema with another XML constraint language,
2. to write program code to express additional constraints, and
3. to express additional constraints with an XSLT/XPath stylesheet.

There are many other schema languages besides W3C XML Schema (WXS): Relax NG [25], Resource Description Framework (RDF) [26], Schematron [12], XCSL [14], XincaML [15], and xlinkit [16]. Each schema has its own capabilities and limitations. Relax NG is a schema language for XML simpler than WXS. RDF is a standard for describing the resources on the Web in XML. These two schema languages cannot supplement WXS to express additional constraints. Schematron, XCSL, XincaML, and Xlinkit are specifically for expressing constraints. They, often called XML constraint languages, can supplement WXS in expressing additional constraints to some extent. The advantage of the second option is that with a single programming language you can express all the additional constraints. But, it cannot leverage XSLT technology. Each of the constraint documents becomes a legacy application. In the third option, each application can create its own stylesheet to check constraints that are unique to the application. It does use XSLT/XPath, a core XML technology. However, the stylesheets

unique to the application are not reusable. It is a challenge to create complex stylesheet. Therefore, the first option is preferable.

2.4 W3C XML Schema

XML Schemas have functions similar to those of DTDs in defining the structures of an XML document, but are much more powerful in expressing data types, data formats, and object-oriented concepts. There are several versions of XML Schemas like Relax NG, RDF, and W3C XML Schema. Vlist [27] did a comprehensive comparison of Schema languages. We do not use Relax NG or RDF in this dissertation. The XML Schema in the rest of this dissertation means WXS, or W3C XML Schema.

WXS is developed by the W3C and standardized on May 2000. It has two parts: Structures and Datatypes. XML Schema: Structures specifies the XML Schema definition language, which offers facilities for describing the structure and constraining the contents of XML 1.0 documents, including those that exploit the XML Namespace facility. XML Schema: Datatypes is the second part of the specification of the XML Schema language. It defines facilities for defining datatypes to be used in XML Schemas as well as other XML specifications. The part one, which is represented in XML 1.0 and uses namespaces, substantially reconstructs and considerably extends the capabilities found in XML 1.0 DTDs. It depends on part two for data type definition.

WXS is a language to define the structures of an XML document, the data types and data formats of individual elements or attributes, and the occurrences of an element. It can also specify the value or value range of an element or attribute. However, it cannot express many non-structural or non-datatype related constraints. More specifically, it

cannot express dynamic value constraints, relationship constraints, and consistency constraints.

Let us reuse the example in Section 1.1. The *SimpleDemo* schema, *demo.xsd*, defines an element *SimpleDemo* that contains two child elements: *UnitsInStock* and *ReorderFlag*. The value type of element *UnitsInstock* is positive integer; and the value type of element *ReorderFlag* is boolean. A sample XML document, *demo.xml*, is a valid instance of the schema, *demo.xsd*. However, WXS does not have the capability to express the Reorder flag constraint of Listing 1-3, the value of *ReorderFlag* depending upon the value of *UnitsInStock*, a type of rule-based constraints. This is why several XML constraint languages such as Schematron, XCSL, and XincaML, have been proposed.

2.5 Schematron

The Schematron is a simple but powerful structural schema language. It differs in basic concept from other schema languages in that it is not based on grammars but on finding tree patterns in an XML document. This approach allows the representation of many kinds of structures inconvenient and difficult to specify in grammar-based schema languages. The Schematron allows developing and mixing two kinds of schemas: (1) *Report* elements allow you to diagnose which variant of a language you are dealing with. (2) *Assert* elements allow you to confirm that the document conforms to a particular schema. The Schematron is based on a simple action: First, find context nodes in the document (typically an element) based on XPath path criteria; Then, check to see whether some other XPath expressions are true, for each of those nodes. Free and open source

implementations of Schematron are available. The Schematron is trivially simple to implement on top of XSLT and to customize.

As mentioned above, the key elements *rule* and *assert* are defined in the schema. However, it is not clear how to define assertion-based constraints and rule-based constraints.

Strictly speaking, it is not a rule-based constraint language as it declares. Take the Reorder flag constraint in Listing 1-3 as an example, Schematron cannot specify such rule-based constraint in a straightforward way. A possible Schematron specification is shown in Listing 2-10. The attribute *@context* of the element *rule* embeds the condition, and two *rule* elements are needed to fulfill the task. Besides if the minimal value of element *UnitsInStock* changes from item to item, this inter-relationship constraint becomes dynamic. Schematron is not able to specify such constraints.

Listing 2-10 Schematron definition of Reorder flag constraint

```
<pattern name="Reorder flag constraint">
    <rule context="SimpleDemo[UnitsInStock &lt; 10]">
        <assert test="ReorderFlag='true'">ReorderFlag is not set to true.</assert>
    </rule>
    <rule context="SimpleDemo[UnitsInStock &ge; 10]">
        <assert test="ReorderFlag='false'">ReorderFlag is not set to false.</assert>
    </rule>
</pattern>
```

It is painful to manually construct such constraint documents.

2.6 XML Constraint Specification Language

Ramalho [28] proposed an XML Constraint Specification Language (XCSL) in his dissertation. A specification in XCSL is composed by one or more tuples. Each tuple has three parts: (1) Context Selector that selects the context where to enforce the constraint; (2) Context Condition that specifies the condition to be enforced; and (3) Action that defines the action to be triggered every time the condition does not meet.

XCSL does the similar job as Schematron but not widely used. It has roughly the same structures as Schematron. It cannot indicate whether it is an assertion type constraint or rule-based constraint. The XCSL definitions of the Reorder flag constraint are shown in Listing 2-11.

Listing 2-11 XCSL definition of Reorder flag constraint

```
<CONSTRAINT>
    <SELECTOR SELEXP=”SimpleDemo[UnitsInStock &lt; 10]”/>
    <CC>
        ReorderFlag=’true’
    </CC>
    <ACTION>
        <MESSAGE/>
    </ACTION>
</CONSTRAINT>
<CONSTRAINT>
    <SELECTOR SELEXP=”SimpleDemo[UnitsInStock &ge; 10]”/>
    <CC>
        ReorderFlag=’false’
    </CC>
    <ACTION>
        <MESSAGE/>
    </ACTION>
</CONSTRAINT>
```

Again, it is painful to manually construct such constraint documents.

2.7 XincaML

XincaML [15] is a markup language used to describe a set of rules that can express the presence or value dependencies amongst nodes located on different branches of an XML document. It aims to provide a mechanism for applications to express and validate XML data constraints. Similar to Schematron and XCSL, the user can construct XML data constraints in XincaML syntax, then the validation is automatically done with XincaML processor and XSLT processor.

XincaML separates the definition of the rule-based constraint from the assertion or check type constraint. An XincaML definition of Reorder flag constraint is shown in Listing 2-12.

Listing 2-12 XincaML definition of Reorder flag constraint

```
<constraint name="ReorderFlagConstraint1" context="/SimpleDemo">
    <if>
        <assert>
            <node id="UnitsInStock" location="UnitsInStock"/>
            <satisfy flag="true">
                <lt>
                    <numberValue ref="UnitsInStock"/>
                    <numberValue value="10"/>
                </lt>
            </satisfy>
        </assert>
    </if>
    <then>
        <assert>
            <node id="ReorderFlag" location="ReorderFlag"/>
            <satisfy flag="true">
                <eq>
                    <booleanValue ref="ReorderFlag"/>
                    <booleanValue value="true"/>
                </eq>
            </satisfy>
        </assert>
    </then>
    <action/>
</constraint>
<constraint name="ReorderFlagConstraint2" context="/SimpleDemo">
```

```
        <if>
            <assert>
                <node id="UnitsInStock" location="UnitsInStock"/>
                <satisfy flag="true">
                    <ge>
                        <numberValue ref="UnitsInStock"/>
                        <numberValue value="10"/>
                    </ge>
                </satisfy>
            </assert>
        </if>
        <then>
            <assert>
                <node id="ReorderFlag" location="ReorderFlag"/>
                <satisfy flag="false">
                    <eq>
                        <booleanValue ref="ReorderFlag"/>
                        <booleanValue value="false"/>
                    </eq>
                </satisfy>
            </assert>
        </then>
        <action/>
    </constraint>
```

As shown above, XincaML can specify constraints at very fine level especially for mathematical expressions. On the other hand, it makes the construction of XincaML documents complicated. It does not leverage XML core technology XSLT/XPath. It requires a proprietary application to validate XML documents against such constraints. XincaML does not support dynamic constraints either.

Again, it is painful to manually write such constraint documents.

2.8 XLinkit

Xlinkit [16] is a lightweight application service that provides rule-based link generation and checks the consistency of distributed web content. It is given a set of distributed XML resources and a set of potentially distributed rules that relate to the content of those

resources. It leverages the core technologies like XML, XPath, and Xlink. Xlinkit can be used as part of a consistency management scheme or in applications that require smart link generation, including portal construction and management of large document repositories.

2.9 Comparison of Existing Constraint Languages

The comparison of the existing constraint languages including WXS is summarized in Table 2-1.

Table 2-1 Comparison of Existing Constraint Languages

	Assertion-based constraint				Rule-based constraint		
	Value	**Presence**	**Composite**	**Dynamic**	**Simple**	**Composite**	**Dynamic**
WXS	Yes	Yes	No	No	No	No	No
Schematron	Yes	Yes	Yes	No	Yes	Partly	No
XincaML	Yes	Yes	Yes	No	Yes	Partly	No
XCSL	Yes	Yes	Yes	Yes	Yes	No	No

As shown above, WXS can deal with the simple assertion-based constraints. Through WXS, fixed or static value or occurrence constraints can be expressed. Schematron and XincaML can express static value or occurrence constraints, composite assertion-based constraints, and simple rule-based constraints. However, they do not support dynamic constraints. XCSL supports assertion-based constraints and simple rule-based constraint but not composite or dynamic constraints.

2.10 Design of XML Constraint Documents

As mentioned in the above sections, it is painful to manually create XML constraint documents especially when more constraints are applied and/or the constraints are more complex. XML constraint languages from Schematron to XincaML are tightly coupled with the XML Schema, the structure definition of XML documents. There have been several researches [23][29][32] in modeling XML Schemas using the Unified Modeling Language (UML) from the Object Management Group (OMG). No one has attempted to model XML constraints yet.

2.11 Summary

This chapter classifies various constraints on XML documents into two types: assertion-based constraints and rule-based constraints. Each type has its finer categories. As the major part of the chapter, the existing constraint languages are discussed in some detail. Schematron is not able to express dynamic constraints and difficult to express rule-based constraints. XCSL is not widely used as Schematron and has the similar drawbacks as Schematron has. XincaML can clearly define both assertion-based constraints and rule-based constraints. But it does not leverage the core XML technology, XPath, which makes the constraint document more complicated. XincaML also has the issue in representing dynamic constraints. In addition, no one has attempted to visually model XML constraints.

Chapter 3

Extensible Constraint Markup Language

As discussed in Chapter 2, there are two types of semantic constraints on XML documents: assertion-based constraints and rule-based constraints. The existing XML constraint languages [12][14][15] can more or less express these two types of constraints. However, each of them has its own drawbacks. Schematron and XCSL do not differentiate rule-based constraints from assertion-based constraints. They do not support composite rule-based constraints and dynamic value/occurrence constraints. This research develops a new XML constraint language – XCML, or eXtensible Constraint Markup Language. XCML provides a set of syntax elements to take advantage of the existing constraint languages and allows to express from simple to complex constraints.

3.1 Overview

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 XSLT/XPath. The XCML syntax is defined in an XCML schema specification.

3.2 Hypotheses

XCML provides a set of syntax elements to express contextual constraints over attributes/elements of XML documents. It does not supersede XML Schema but rather as a complement to semantically constrain XML documents. The following hypotheses are made:

a. XCML does not constrain the structure of XML documents.

b. XCML does not constrain the order of elements in XML documents.

c. XCML does not constrain the name of elements and/or attributes.

d. XCML must support namespaces.

e. XCML must support XPath 1.0 or above.

f. XCML should be extensible.

XCML documents can be either embedded within XML Schemas as annotations or as separate constraint documents.

3.3 XCML Design Goal

The goal of XCML is to leverage the core XML technologies to express the semantic constraints on XML documents. The XCML instance documents should be simple, easy to create, and easy to use to validate XML documents. It must support not only assertion-based constraints and simple rule-based constraints, like if-then, but also work with composites rules like if-then(s)-else(s). XCML has to support parameters in expressing dynamic constraints. It must support XPath 1.0 or above so that various expressions can

be processed by XPath supported processors. Also, such constraints should be expressible in XML data models.

3.4 XCML Syntax

Based on the above design goal and hypotheses, the syntax of XCML is designed in the form of W3C XML Schema, see its structure diagram in Figure 3-1.

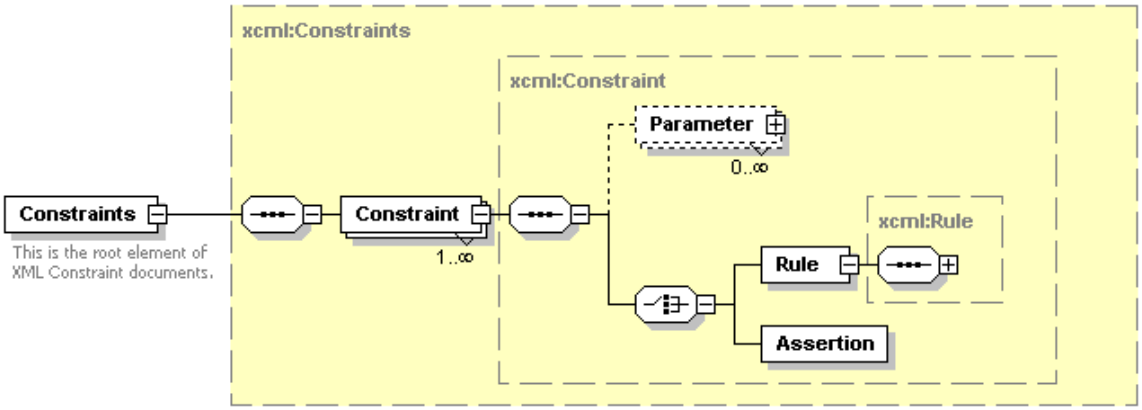

Figure 3-1 Structure diagram of XCML Schema

Constraints is the root element of XCML instance documents. *xcml:Constraints* indicated by the outmost dotted-line box is the type definition of element *Constraints*. An element *Constraints* have one or more *Constraint* elements. *xcml:Constraint* is the type definition of element *Constraint*. An element *Constraint* has zero or more *Parameter* elements and either a *Rule* element or an *Assertion* element. Attribute definitions are not shown in this figure. Section 3.4.1 through Section 3.4.10 describes this schema definition in detail.

3.4.1 Namespaces

The namespace of XCML Schema, *xmlns:xsd*, is *http://www.w3.org/2001/XMLSchema*. The target namespace, *xmlns:xcml*, is *http://www.csis.pace.edu/dps/xcml*. The XCML documents must conform to these namespaces. The Schema source is listed below:

Listing 3-1 XCML Namespace definition

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xcml="http://www.csis.pace.edu/dps/xcml"
targetNamespace="http://www.csis.pace.edu/dps/xcml"
```

3.4.2 Constraints - root element

The element *Constraints* is the root element of an XCML instance document. It is a type of *xcml:Constraints*. It contains one or more *Constraint* elements and zero or one *@name* attribute. It is used to specify one or more constraints on an XML document, see Section 3.4.3 for the definition of *xcml:Constraint.* The Schema source is listed below:

Listing 3-2 Definition of root element *Constraints*

```
<xsd:element name="Constraints" type="xcml:Constraints">
<xsd:complexType name="Constraints">
        <xsd:sequence>
            <xsd:element name="Constraint" type="xcml:Constraint" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string" use="optional"/>
</xsd:complexType>
```

3.4.3 Type Constraint

xcml:Constraint is an *xsd:complexType*. It contains zero or more <*Parameter*> elements, either a *Rule* element or an *Assertion* element. It has an attribute *@context* to specify the context to be constrained. It is used to specify the actual constraints that can be rule-based constraints or assertion-based constraints. These constraints can be static (fixed) values or dynamic (parameterized) values. The element *Parameter* is used to express the dynamic value constraints, see Section 3.4.4 for its definition. The actual value may be passed from a domain application. The Schema source is listed below:

Listing 3-3 Definition of complexType *Constraint*

```
<xsd:complexType name="Constraint">
    <xsd:choice maxOccurs="unbounded">
        <xsd:element name="Parameter" type="xcml:Parameter" minOccurs="0"
          maxOccurs="unbounded"/>
        <xsd:element name="Rule" type="xcml:Rule" maxOccurs="unbounded"/>
        <xsd:element name="Assertion" type="xcml:Assertion" maxOccurs="unbounded"/>
    </xsd:choice>
    <xsd:attribute name="context" type="xsd:string" use="required"/>
</xsd:complexType>
```

3.4.4 Type Parameter

xcml:Parameter is a type of *xsd:complexType*. It contains one <*name*> element and zero or one *defaultValue* element. The element *name* conveys the name of a parameter. It is a type of *xsd:NCName* (Non-colonized names). The element *defaultValue* conveys the default value of the parameter. It is a type of *xsd:string*. The Schema source is listed below:

Listing 3-4 Definition of complexType *Parameter*

```
<xsd:complexType name="Parameter">
    <xsd:annotation>
        <xsd:documentation>A parameter must have a name.</xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
        <xsd:element name="name" type="xsd:NCName"/>
        <xsd:element name="defaultValue" type="xsd:string" minOccurs="0"/>
    </xsd:sequence>
</xsd:complexType>
```

3.4.5 Type Rule

xcml:Rule is a type of *xsd:complexType*. It contains one *If* element (see Section 3.4.6) and zero or more *Else* elements. It is used to specify a rule-based constraint. If the constraint is a simple one, only the element *If* is present. If the constraint is a composite one, the element *Else* is present one or more times. The Schema source is listed below:

Listing 3-5 Definition of complexType *Rule*

```
<xsd:complexType name="RuleType">
    <xsd:sequence>
        <xsd:element name="If" type="xcml:If" />
        <xsd:element name="Else" type="xcml:Then" minOccurs="0"
          maxOccurs="unbounded"/>
    </xsd:sequence>
 </xsd:complexType>
```

3.4.6 Type If

xcml:If is a type of *xsd:complexType*. It contains one *Then* element and an attribute *@test*. The attribute *@test* conveys the conditions – the first part of the inter-relationship constraint. If such a condition is met, the constraint in the element *Then* is checked. The

condition in the attribute *@test* is expressed in one or more XPath expressions. The element *Then* conveys the second part of the inter-relationship constraint, see its definition in Section 3.4.7. The Schema source is listed below:

Listing 3-6 Definition of complexType *If*

```
<xsd:complexType name="If">
    <xsd:sequence>
        <xsd:element name="Then" type="xcml:Then"/>
    </xsd:sequence>
    <xsd:attribute name="test" type="xsd:string" use="required"/>
</xsd:complexType>
```

3.4.7 Type Then

xcml:Then is a type of *xsd:complexType*. It contains an attribute *@test* to check whether the XPath expression returns true (the expected return value). It may embed another *Rule* element to express a composite rule-based constraint. The element *Else* has the same type as the element *Then*. It is used for specifying composite rule-based constraints. The Schema source is listed below:

Listing 3-7 Definition of complexType *Then*

```
<xsd:complexType name="Then">
    <xsd:sequence minOccurs="0">
        <xsd:element name="If" type="xcml:If"/>
        <xsd:element name="Else" type="xcml:Then" minOccurs="0"
          maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="test" type="xsd:string" use="optional"/>
</xsd:complexType>
</xsd:schema>
```

3.4.8 Type Assertion

xcml:Assertion is a type of *xsd:complexType*. It contains only an attribute *@test* (see Section 3.4.9). It is used to specify an assertion-based constraint. The Schema source is listed below:

Listing 3-8 Definition of complexType *Assertion*

```
<xsd:complexType name="Assertion">
    <xsd:attribute name="test" type="xsd:string" use="required"/>
</xsd:complexType>
```

3.4.9 Attribute test

Attribute *@test* is a type of *xsd:string*. It has to be a valid XPath expression

3.4.10 Overall XCML Schema

The overall XCML syntax in W3C XML Schema definition is given in Appendix A. All the XCML instance documents must conform to this schema. It is used to guide the generation of XCML instance documents and validate the XCML instance documents.

3.5 Construction of XCML Documents

Once the XCML syntax is defined, an XCML document can be created to express the semantic constraints on an XML document that conforms to an XML Schema or DTD. It should support the namespaces used in the XCML schema. Also, the XPath expressions in the XCML document must conform to the namespaces used in the XML document to be validated.

The construction of XCML documents follows the steps below:

a. Identify the constraint types.
b. Check whether parameters/variables are needed.
c. Locate the context.
d. Create XPath expressions.
e. Validate the XCML document against the XCML schema.

For a simple application, it is trivial to manually generate XCML instance documents. For a large or complicated system, it will be very time consuming and error prone to do it manually. It is better to automate the XCML instance document generation. Any existing constraint languages have not addressed this challenging issue. We propose a mechanism of visually modeling XCML constraints in XML data models. The XCML documents can

be automatically generated together with XML schemas. Chapter 4 discusses the visual modeling of XCML constraints.

3.6 XCML Instance Documents

This section illustrates how XCML is used to express various XML constraints. We construct XCML document for each sample constraint listed in Section 2.2.

3.6.1 Assertion-based Constraints

The XCML definition of Listing 2-2 is constructed shown in Listing 3-9.

Listing 3-9 XCML definition of *taxRate value constraint*

```
<Constraint context="payroll">
    <Assertion test=”taxRate=0.06”/>
</Constraint>
```

The context of this constraint is the element *payroll*. The element *taxRate* is a child element of element *payroll*. It asserts the *taxRate* must be 0.05. The logical operators can be used with the test condition. The XCML definition of Listing 2-3 is such an example, see Listing 3-10. It would be better to use a parameter to represent such a value and let the application pass the given value through this parameter. The above XCML constraint then becomes another form:

Listing 3-10 The XCML definition of *value set Constraint of taxRate*

```
<Constraint context="payroll">
    <Assertion test=”taxRate=0.05 or taxRate=0.06 or taxRate=0.10”/>
```

```
</Constraint>
```

We can also define a variable or parameter to represent the given rates as shown in Listing 3-11 for the XCML definition of Listing 2-5.

Listing 3-11 XCML definition of dynamic value constraint

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

The XCML definition of a composite assertion-based constraint, net Income constraint in Listing 2-7 is created below:

Listing 3-12 XCML definition of netIncome constraint

```
<Constraint context="payroll">
     <Assertion test=”salary + bonus – tax = netIncome”/>
</Constraint>
```

3.6.2 Rule-based Constraints

Similarly, the XCML definitions for a simple rule-based constraint given in Listing 2-8 and composite rule-based constraint shown in Listing 2-9 are created shown in Listing 3-13 and Listing 3-14.

Listing 3-13 XCML definition of a simple rule-based constraint

```
<Constraint context="employee">
    <Rule>
        <If test="role='manager'">
            <Then test="count(department)=1"/>
        </If>
    </Rule>
</Constraint>
```

This is a simple rule-based constraint only with a simple elements *If* and *Then*. The next example has an extra element Else, which makes it a composite rule-based constraint.

Listing 3-14 XCML definition of a composite rule-based constraint

```
<Constraint context="employee">
    <Rule>
        <If test="role='manager'">
            <Then test="count(department)=1"/>
        </If>
        <Else test=”count(department)=0”/>
    </Rule>
</Constraint>
```

3.7 Comparison of XCML with Existing Constraint Languages

Schematron is the most popular XML constraint language among the existing ones. XincaML is recently proposed from IBM. XCSL is another constraint language but rarely known. This section compares XCML proposed in this dissertation with existing constraint languages as shown in Table 3-1.

Table 3-1 Comparison of different constraint languages

	Assertion-based constraint				Rule-based constraint		
	Value	**Presence**	**Composite**	**Dynamic**	**Simple**	**Composite**	**Dynamic**
Schematron	Yes	Yes	Yes	No	Yes	No	No
XincaML	Yes	Yes	Yes	No	Yes	No	No
XCSL	Yes	Yes	Yes	Yes	Yes	No	No
XCML	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Schematron, XincaML, and XCSL support static assertion-based constraints but not dynamic ones. They support simple rule-based constraints but not composite ones. XincaML does not leverage a core XML technology, XSLT/XPath. To illustrate their differences, various XML constraint definitions in each existing constraint language for an assertion-based constraint, Net income Constraint (Listing 2-7) and a rule-based constraint, Managed department Constraint (Listing 2-9) are constructed below.

3.7.1 Schematron Definition

Listing 3-15 shows the Schematron definition of Net income Constraint, an assertion-based constraint; and Listing 3-16 shows the Schematron definition of Managed department Constraint, a rule-based constraint. As seen, a rule of If-then-else is represented in two *rule* blocks. The rule condition is embedded inside the attribute *@context*. There is no obvious indication whether the constraint is a rule-based constraint or an assertion-based constraint.

Listing 3-15 Schematron definition of *Net income* Constraint

```
<pattern name="Net income assertion">
    <rule context="employee/payroll">
        <assert test="salary + bonus – tax = netIncome">netIncome is not correct.</assert>
```

```
        </rule>
    </pattern>
```

Listing 3-16 Schematron definition of *Managed department* Constraint

```
<pattern name="Managed department constraint">
        <rule context="employee[role='manager']">
            <assert test="count(department)=1">department is not present.</assert>
        </rule>
        <rule context=" employee[role!='manager']">
            <assert test=" count(department)=0">department is present.</assert>
        </rule>
    </pattern>
```

3.7.2 XincaML Definition

Listing 3-17 illustrates the XincaML definition of Net income Managed department Constraint, an assertion-based constraint; and Listing 3-18 illustrates the Schematron definition of Managed department Constraint, a rule-based constraint. XincaML does not leverage a core XML technology, XSLT/XPath, which makes the constraint definition much more complicated.

Listing 3-17 XincaML definition of *Net income Constraint*

```
<constraint name="Net income constraint" context="employee/payroll">
    <assert>
        <node id="netIncome" location="netIncome"/>
        <node id="salary" location="salary"/>
        <node id="bonus" location="bonus"/>
        <node id="tax" location="tax"/>
        <satisfy flag="true">
            <eq>
              <minus>
                <add>
                    <numberValue ref="salary"/>
                    <numberValue ref="bonus"/>
                </add>
                <numberValue ref="tax"/>
```

```
                </minus>
                <numberValue ref="netIncome"/>
            </eq>
        </satisfy>
    </assert>
    <action/>
</constraint>
```

Listing 3-18 XincaML definition of *Managed department Constraint*

```
<constraint name="Managed department Constraint" context="employee">
    <if>
        <assert>
            <node id="role" location="role"/>
            <satisfy flag="true">
                <eq>
                    <stringValue ref="role"/>
                    <stringValue value="manager"/>
                </eq>
            </satisfy>
        </assert>
    </if>
    <then>
        <assert>
            <node id="department" location="department"/>
            <present flag="true">
        </assert>
    </then>
    <action/>
</constraint>
<constraint name="ReorderFlagConstraint2" context="employee">
  <if>
        <assert>
            <node id="role" location="role"/>
            <satisfy flag="true">
                <ne>
                    <stringValue ref="role"/>
                    <stringValue value="manager"/>
                </ne>
            </satisfy>
        </assert>
    </if>
    <then>
        <assert>
            <node id="department" location="department"/>
            <present flag="false">
        </assert>
    </then>
    <action/>
</constraint>
```

3.7.3 XCSL Definition

Listing 3-19 illustrates the XCSL definition of Net income Managed department Constraint, an assertion-based constraint; and Listing 3-20 illustrates the XCSL definition of Managed department Constraint, a rule-based constraint. XCSL definition has structure similar to what Schematron has.

Listing 3-19 XCSL definition of *Net Income* Constraint

```
<CONSTRAINT>
        <SELECTOR SELEXP=”employee/payroll”/>
        <CC>
            salary + bonus – tax = netIncome
        </CC>
        <ACTION>
            <MESSAGE/>
        </ACTION>
</CONSTRAINT>
```

Listing 3-20 XCSL definition of *Managed department* Constraint

```
    <CONSTRAINT>
        <SELECTOR SELEXP=”employee[role=’manager’]”/>
        <CC>
            count(department)=1
        </CC>
        <ACTION>
            <MESSAGE/>
        </ACTION>
    </CONSTRAINT>
    <CONSTRAINT>
        <SELECTOR SELEXP=”employee[role!=’manager’]”/>
        <CC>
            count(department)=0
        </CC>
        <ACTION>
            <MESSAGE/>
        </ACTION>
    </CONSTRAINT>
```

It can be seen that the XCML definitions are more concise and clearer than the other three by comparing the above definitions with the XCML ones in Section 3.6. XCML leverages the core XML technology, XSLT/XPath. The XCML constraint documents are easier to create and understand. And more importantly, it can be written on the UML class models, which could lead to automatic generation of XCML constraint documents, see detail in Chapter 4.

3.8 Validation of XML Documents

As discussed in the previous chapter, the validation can be done at two levels: syntactic validation and semantic validation. Syntactic validation mainly checks whether the structure of an XML document conforms to its definition DTD or XML Schema Semantic validation mainly checks the non-structural constraints defined in an XCML document. Syntactic validation can be performed using a Schema-supported XML parser. If there are no syntactic validation errors, the semantic validation is the next step.

Once the XCML document is created, the semantic validation of an XML document can be realized in several ways. One is to transform the XCML document into an XSLT stylesheet and then run an XSLT processor to perform validation as Schematron does. The advantage of this approach is that it leverages the XSLT/XPath technology. The user does not need to do low level programming. Another approach is to write programming code to process the XCML document and perform validation. The disadvantage of this approach is that such code is proprietary. We use the first approach to do semantic validation, see detail in Chapter 5.

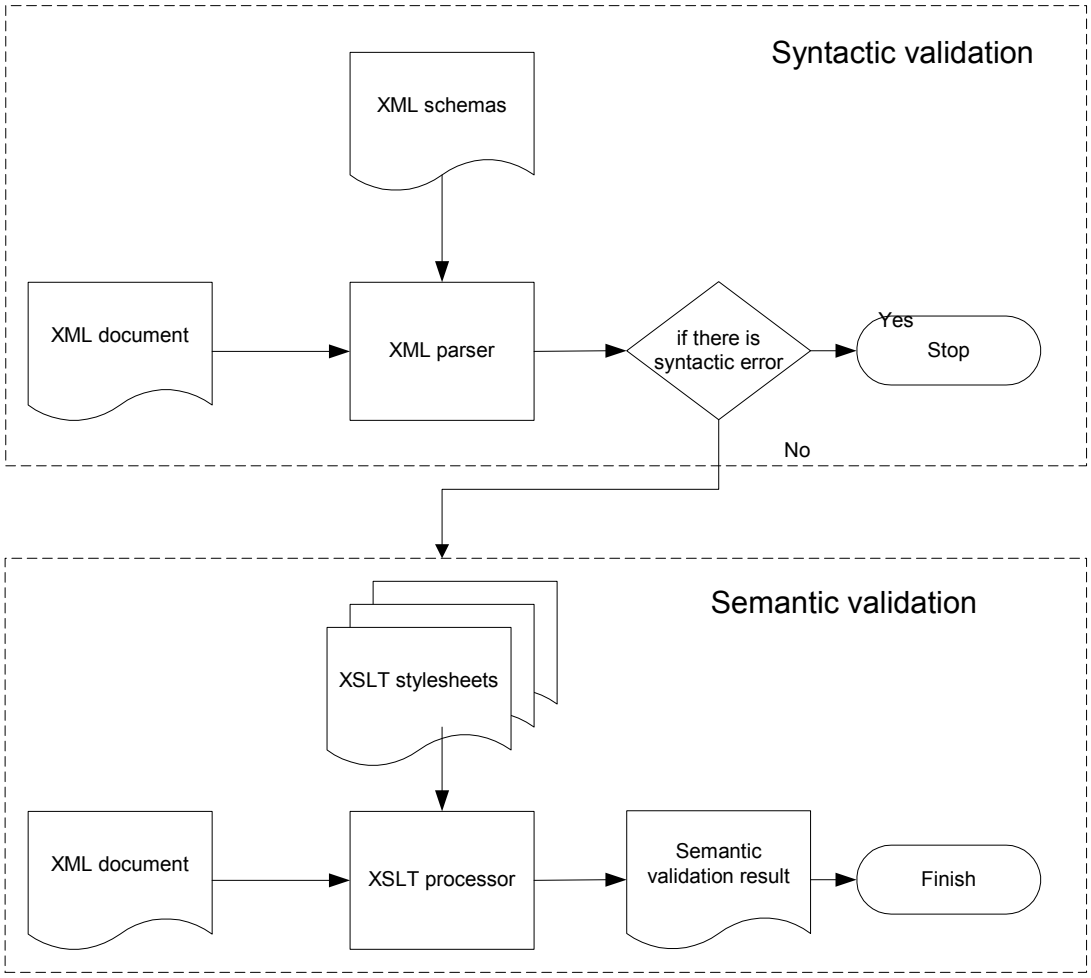

Figure 3-2 Workflow of XML document validation

3.9 Summary

This chapter discusses the design of XCML in detail. It provides the syntax to define both assertion-based constraints and rule-based constraints. XCML can express simple rule-based constraints and assertion-based constraints as the existing XML constraint languages. It can also express dynamic value constraints and composite rule-based constraints. It leverages the core XML technologies to simplify its design and

implementation. We compare XCML with the existing constraint languages in some detail. We also outline the construction of XCML document and how to perform semantic validation using XCML documents.

Chapter 4

Visual Modeling of XCML Constraints

As discussed in the previous chapters, the generation of XML constraint documents is a challenge. In this chapter, we propose a model-driven approach to automated this generation process. Visual modeling of XML constraints cannot be standalone since XML constraints are the limitations on elements and/or attributes of XML documents. It has to be done together with the visual modeling of XML data structures (so-called XML data modeling). As the second part of this research, we first build a UML profile for the XCML schema, and then apply such a profile together with an existing UML profile for W3C XML Schema to the conceptual information models. From such models, XCML instance documents and/or XML schemas can be automatically generated using the approach described in Chapter 5. XML schemas serve for syntactic validation; and XCML documents serve for semantic validation.

4.1 XML Data Modeling

There have been a number of approaches doing XML data modeling since XML was born. The graphical tree-based approach was the starting point and is still being used in modeling tools like XML Spy [29] and XML Authority [30] for simple XML applications. As more and more XML applications, especially Web Services, are used in B2B enterprise information integration, XML data modeling plays an important role in

the design and implementation of B2B systems. However, there is no such an environment specifically used for XML data modeling. The Unified Modeling Language (UML) from Object Management Group (OMG) has been well accepted as a software industry modeling language for software design since the OMG adopted the UML 1.1 specification [18] in November 1997. Although UML 1.1 was not originally created for data modeling, its newer versions UML 1.4 [19], MOF 1.4 [20], and recently standardized UML 2.0 [21] provide an extension mechanism that makes it possible to do data modeling by applying UML profiles. Several researchers [23][29][32] have proposed a similar approach, the three-level-design approach, to modeling XML applications. By this approach, the conceptual level is represented using standard UML class notation, the logical level is represented in UML class notation with XML Schema profile applied, and the physical level is represented by an XML Schema document.

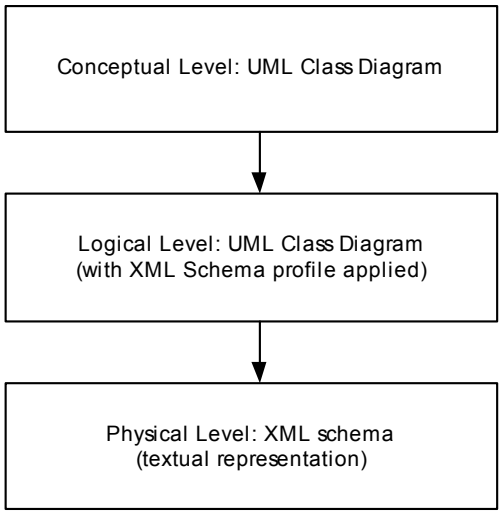

Figure 4-1 Three-level-design approach to XML data modeling

4.1.1 Conceptual Model

The conceptual model is a real world information model in the standard UML class notation. It normalizes the information entities of a system. Figure 4-2 shows a conceptual model of order reports of an Order Entry System. It has no concepts of XML Schema.

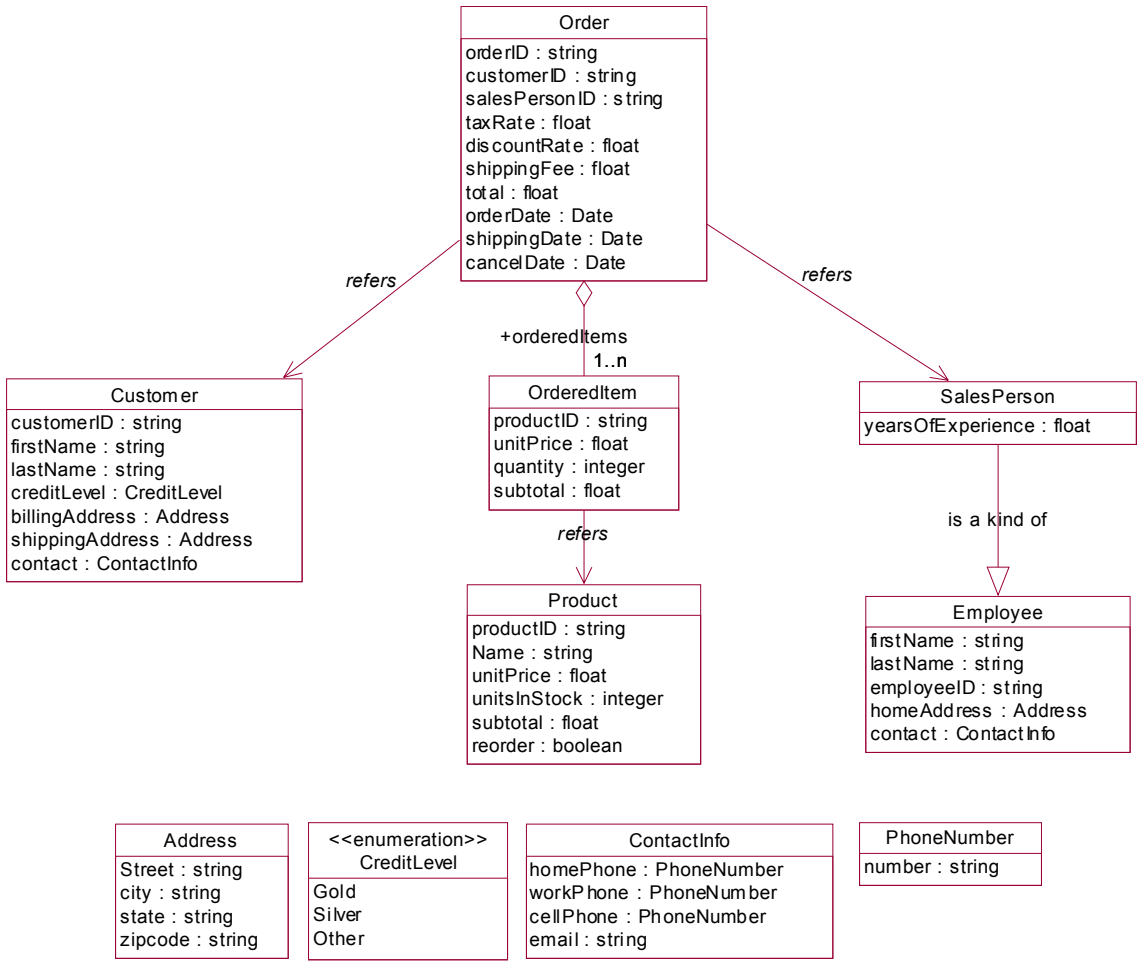

Figure 4-2 Conceptual model of Order documents of a sample Order Entry system

Class *Order* contains one or more objects of class *OrderedItems*. It also refers to classes *Customer* and *SalesPerson*. And class *OrderedItem* refers to class *Product*. There are several helper classes like *Address*, *CreditLevel*, and *ContactInfo*.

4.1.2 Logical Model

The logical model is the UML class diagram annotated with XML Schema vocabularies represented in a UML profile consisting of a set of UML stereotypes. Carlson [23] and Routledge et al [29] each built a UML profile for W3C XML Schema. Carlson's one is more popular than Routledge's because it is more intuitive, simpler, and easier to use, as presented in Figure 4-3.

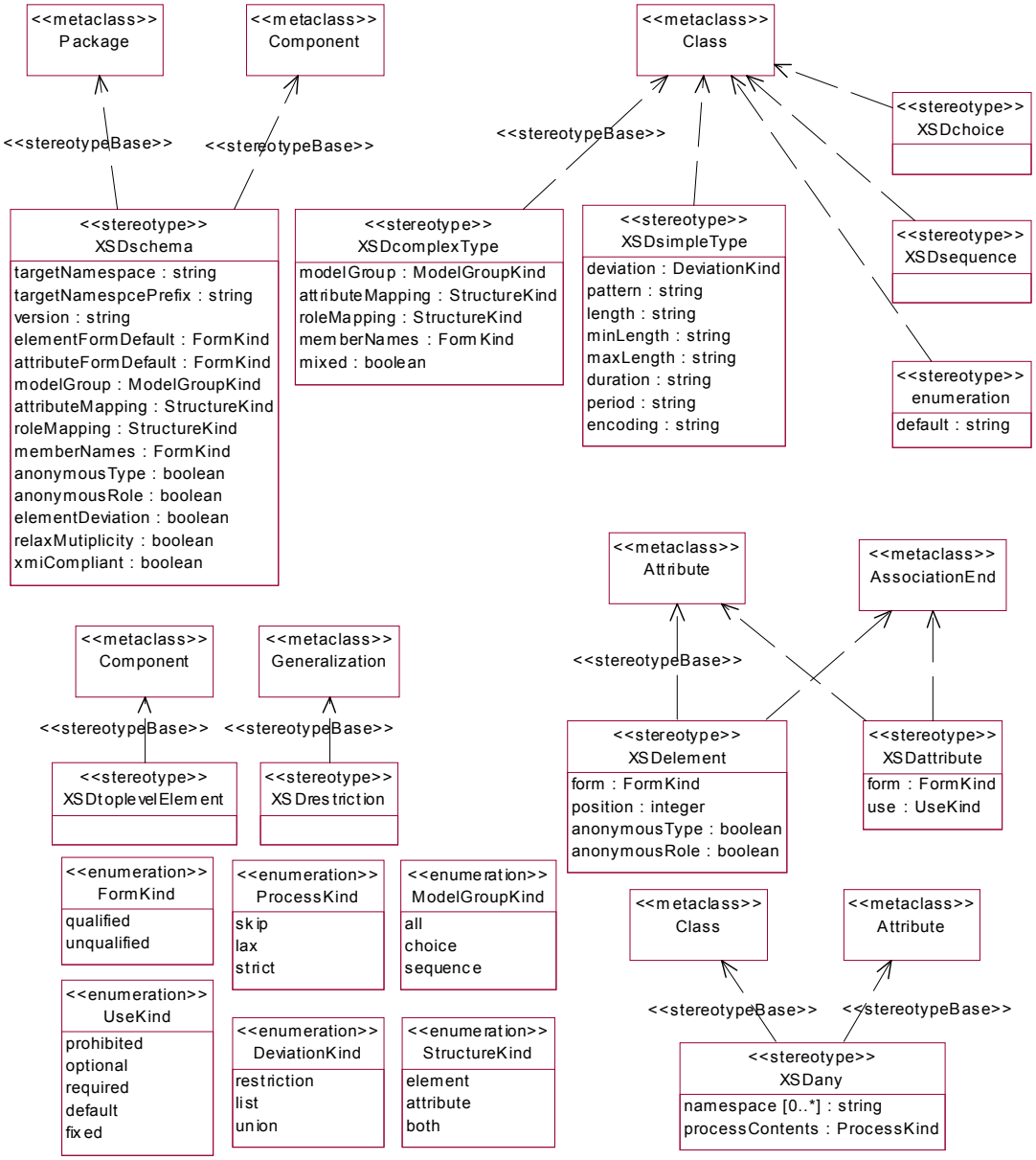

Figure 4-3 Carlson's UML profile for W3C XML Schema ([23], pp. 301-310)

This profile is built based on the core constructs of the XML Schema specification. The prefix 'XSD' (short for XML Schema Definition) is added to all stereotypes. Applying

these stereotypes to the conceptual model in Figure 4-2, we obtain the logical model in Figure 4-4.

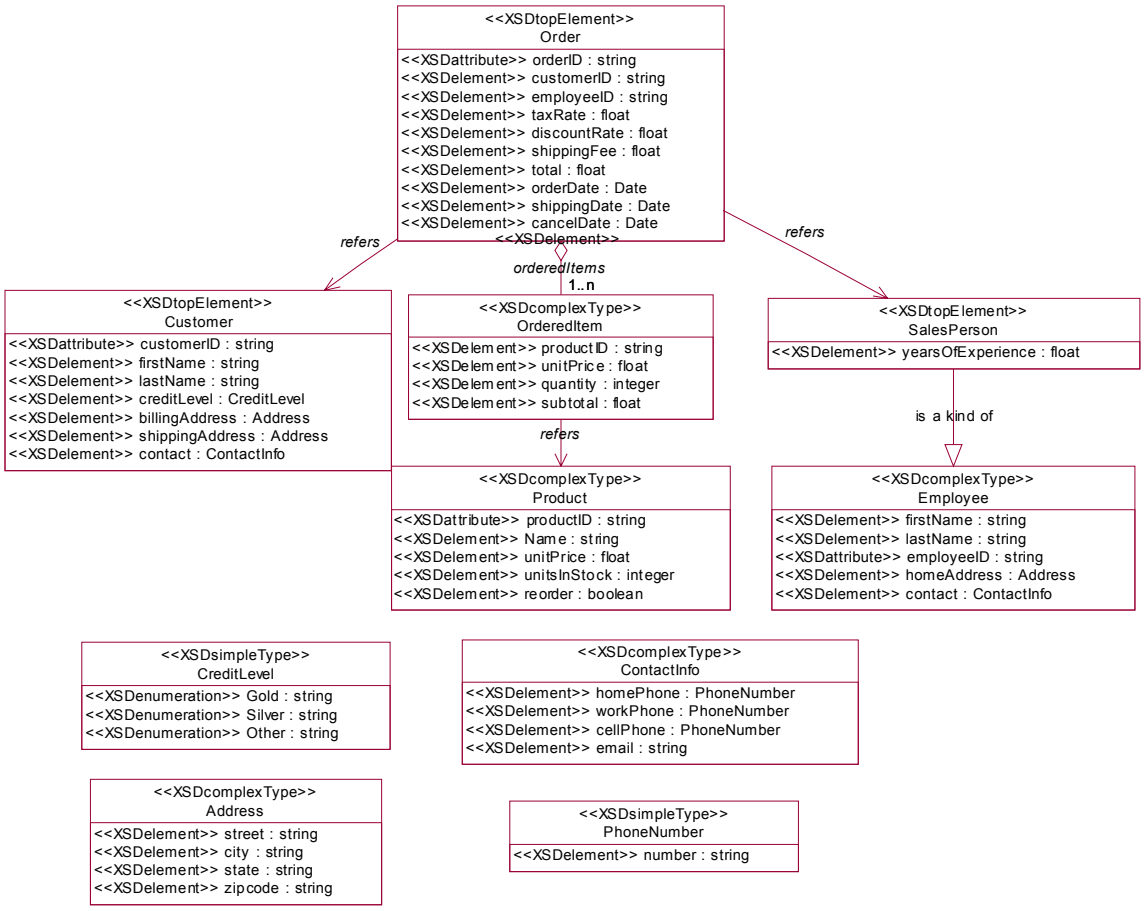

Figure 4-4 Logical model of Order documents of a sample Order Entry system

In the logical model, each class, each attribute of a class, and sometimes associations among classes are assigned XML Schema vocabularies. For example, stereotype *XSDtopElement* is assigned to class *Order*, stereotype *XSDelement* is assigned to attribute *total*, and stereotype *XSDelement* is assigned to the association between class *Order* and class *OrderedItem*.

4.1.3 Physical Model

The physical model is the textual representation of XML schema generated from the logical model following a set of mapping rules; see detail in 5.3. Listing 4-1 lists the XML schema generated from the logical model shown in Figure 4-4.

Listing 4-1 Physical model (XML schema) of Order documents of a sample Order Entry system

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.csis.pace.edu/dps/xcml"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:complexType name="Address">
        <xsd:sequence>
            <xsd:element name="street" type="xsd:string" />
            <xsd:element name="city" type="xsd:string" />
            <xsd:element name="state" type="xsd:string" />
            <xsd:element name="zipcode" type="xsd:string" />
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="ContactInfo">
        <xsd:sequence>
            <xsd:element name="homePhone" type="xcml:PhoneNumber" minOccurs="0"
              maxOccurs="5" />
            <xsd:element name="workPhone" type="xcml:PhoneNumber" maxOccurs="3" />
            <xsd:element name="cellPhone" type="xcml:PhoneNumber" minOccurs="0"
              maxOccurs="2" />
            <xsd:element name="email" type="xsd:string" maxOccurs="2" />
        </xsd:sequence>
    </xsd:complexType>
    <xsd:simpleType name="CreditLevel">
        <xsd:sequence />
    </xsd:simpleType>
    <xsd:element name="customer" type="xcml:Customer" />
    <xsd:complexType name="Customer">
        <xsd:sequence>
            <xsd:element name="firstName" type="xsd:string" />
            <xsd:element name="lastName" type="xsd:string" />
            <xsd:element name="creditLevel" type="xcml:CreditLevel" />
            <xsd:element name="billingAddress" type="xcml:Address" />
            <xsd:element name="shippingAddress" type="xcml:Address" />
            <xsd:element name="contact" type="xcml:ContactInfo" />
        </xsd:sequence>
        <xsd:attribute name="customerID" type="xsd:string" use="required" />
    </xsd:complexType>
    <xsd:complexType name="Employee">
        <xsd:sequence>
            <xsd:element name="firstName" type="xsd:string" />
```

```
            <xsd:element name="lastName" type="xsd:string" />
            <xsd:element name="homeAddress" type="xcml:Address" />
            <xsd:element name="contact" type="xcml:ContactInfo" />
        </xsd:sequence>
        <xsd:attribute name="employeeID" type="xsd:string" use="required" />
    </xsd:complexType>
    <xsd:element name="order" type="xcml:Order" />
    <xsd:complexType name="Order">
        <xsd:sequence>
            <xsd:element name="orderedItem" type="xcml:OrderedItem" maxOccurs="unbounded"
              />
            <xsd:element name="customerID" type="xsd:string" />
            <xsd:element name="employeeID" type="xsd:string" />
            <xsd:element name="taxRate" type="xsd:float" />
            <xsd:element name="discountRate" type="xsd:float" />
            <xsd:element name="shippingFee" type="xsd:float" />
            <xsd:element name="total" type="xsd:float" />
            <xsd:element name="orderDate" type="xsd:Date" />
            <xsd:element name="shippingDate" type="xsd:Date" />
            <xsd:element name="cancelDate" type="xsd:Date" minOccurs="0"
              maxOccurs="unbounded" />
        </xsd:sequence>
        <xsd:attribute name="orderID" type="xsd:string" use="required" />
    </xsd:complexType>
    <xsd:complexType name="OrderedItem">
        <xsd:sequence>
            <xsd:element name="productID" type="xsd:string" />
            <xsd:element name="unitPrice" type="xsd:float" />
            <xsd:element name="quantity" type="xsd:integer" />
            <xsd:element name="subtotal" type="xsd:float" />
        </xsd:sequence>
    </xsd:complexType>
    <xsd:simpleType name="PhoneNumber">
        <xsd:sequence>
            <xsd:element name="number" type="xsd:string" />
        </xsd:sequence>
    </xsd:simpleType>
    <xsd:complexType name="Product">
        <xsd:sequence>
            <xsd:element name="Name" type="xsd:string" />
            <xsd:element name="unitPrice" type="xsd:float" />
            <xsd:element name="UnitsInStock" type="xsd:integer" />
            <xsd:element name="ReorderFlag" type="xsd:boolean" />
        </xsd:sequence>
        <xsd:attribute name="productID" type="xsd:string" use="required" />
    </xsd:complexType>
    <xsd:element name="salesPerson" type="xcml:SalesPerson" />
    <xsd:complexType name="SalesPerson">
        <xsd:extension base="xsd:Employee">
            <xsd:sequence>
                <xsd:element name="yearsOfExperience" type="xsd:float" />
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexType>
</xsd:schema>
```

This schema can be used to guide the generation of order XML documents and validate them. However, the existing approaches to XML data modeling focus only on the mapping between UML and XML Schema. They have not considered the semantic constraints yet. The logical models from the previous approaches [23] [31][32] do not represent constraints, which could cause miscommunicaton between software architects and application developers. Furthermore, such models cannot guide and automate the semantic validation of XML documents. Is it possible to represent XML constraints over XML data models? The next section will give a positive answer.

4.2 Modeling XCML Constraints

The newer specification UML versions 1.4, provide a new notation language – Object Constraint Language (OCL). It allows software architects, designers, and software developers to write constraints over object models in the real world. Can UML/OCL be used to model XCML constraints? In this research, we model XCML constraints in the same design approach as we model XML data structures. The conceptual level is represented in a UML class diagram with the real world constraints represented in OCL. The logical level is represented in a UML class diagram annotated with an XML Schema profile and an XCML schema profile. Two UML profiles [23][31] for XML Schema are available. We choose Carlson's one for the XML Schema profile. We design a UML profile for XCML schema as part of the research. XML Schema and XCML documents represent the physical level.

4.2.1 OCL

The OCL [19][22] is a new notational language, a subset of the industry standard UML. It allows software architects, designers, and software developers to write constraints over object models in the real world. A constraint is a restriction on one or more values of an object-oriented model or system. There are three types of constraints defined in OCL: preconditions, postconditions, and invariants. Pre- and postconditions are defined for operations or behaviors in objects. Invariants are constraints expressed in a mathematical expression that must always be met by all instances of a class, type, or interface. Representing constraints over the object models convey a number of benefits such as better documentation, improved precision, and communication without misunderstanding. In UML 1.4 [19], a general Invariant constraint is expressed as shown in Listing 4-2.

Listing 4-2 General OCL constraint in UML 1.4

```
<<Invariant>>
Context: Class name (inv Constraint name)
-- some expression --
```

In UML 1.4, the stereotype Invariant is explicitly shown in an OCL expression. The context of an OCL expression within a UML model is specified through a so-called context declaration at the beginning of an OCL expression. If the constraint is shown in a diagram, with the proper stereotype and the dashed lines to connect to its contextual element, there is no need for an explicit context declaration in the test of the constraints.

The context declaration is optional. The actual context is usually a class. The constraint name is also optional. The basic types and operations for OCL 1.4 can be seen in [22].

Following the OCL syntax, we can put various invariant constraints on attributes, classes, and other types like associations. Figure 4-5 illustrates an example of putting a constraint on an attribute. And Figure 4-6 shows an example of putting an invariant constraint on an association.

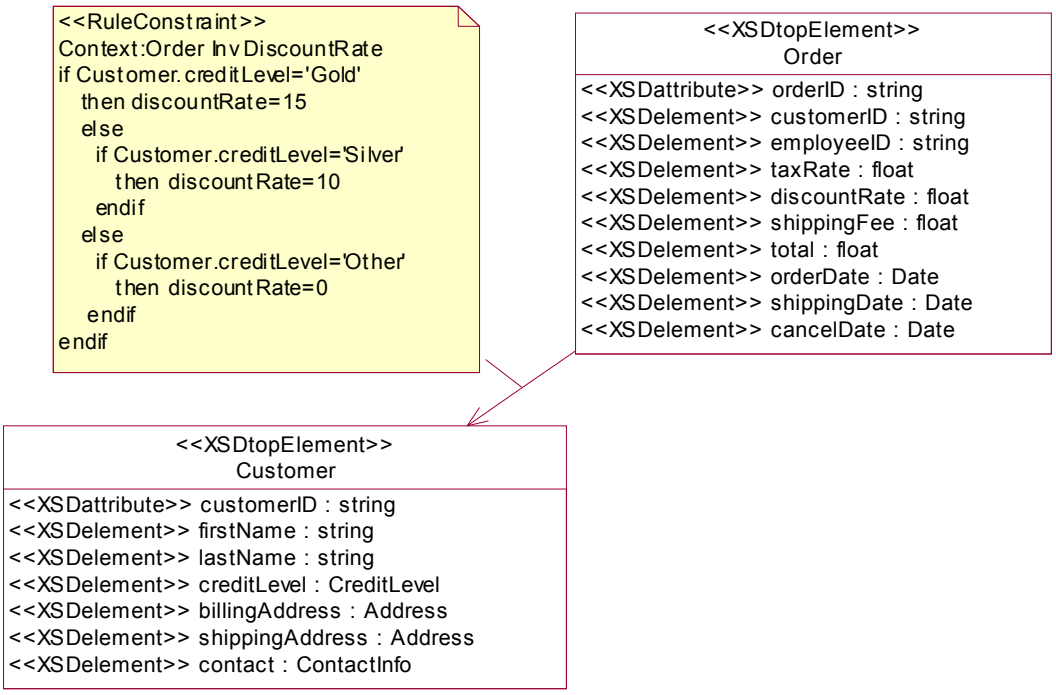

Figure 4-5 Constraint Discount Rate on attribute discountRate in class Order

The constraint *Discount Rate* is a value restriction on attribute *discountRate* of class *Order*. The actual value depends upon the value of attribute *creditLevel* (membership) of *Customer*.

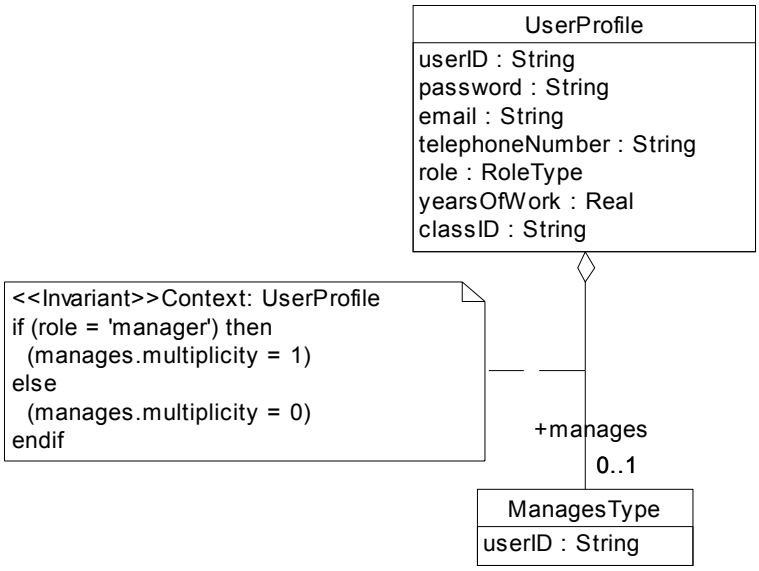

Figure 4-6 A sample of constraints on Associations

Figure 4-6 shows a sample of constraints on associations. It constrains that a *user* manages other users' account only if his *role* is "*manager*".

4.2.2 UML profile for XCML schema

In order to derive logical models from conceptual models, the domain specific vocabularies need to be put onto the models. UML profile, a UML extension mechanism, is used to represent those vocabularies. We choose Carlson's one for representing XML Schema vocabularies. We design a UML profile for XCML schema, as presented in Figure 4-7.

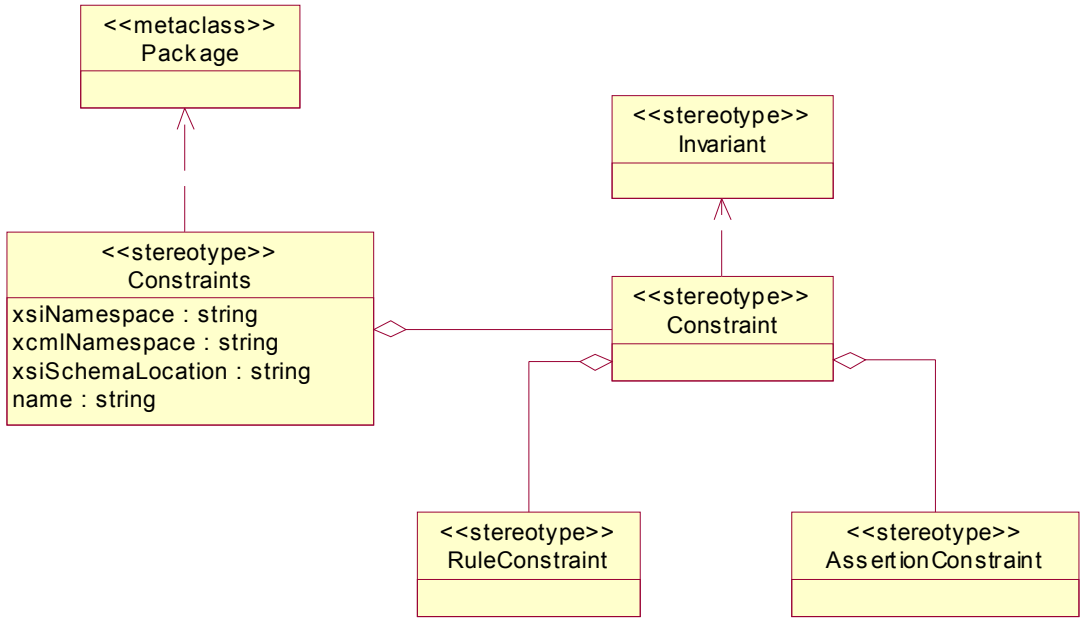

Figure 4-7 UML profile for XCML schema

Package is the standard UML metaclass. Invariant is a stereotype of constraints in OCL 1.4. *Constraints*, *Constraint*, *RuleConstraint*, and *AssertionConstraint* are the stereotypes extending UML/OCL to XCML schema.

4.2.2.1 Stereotype *Constraints*

Constraints is a stereotype with a base type of Package. In an XCML document, the root element Constraints constrains all the definitions for the namespaces of W3C XML Schema and XCML schema. If a UML package is assigned this stereotype, all the OCL constraints will be placed within one XCML document. Stereotype Constraints has four tagged values: *xsiNamespace*, *xcmlNamespace*, *xsiSchemaLocation*, and *name*.

- *xsiNamespace* is a URL representing the W3C XML Schema definition namespace. The default value is http://www.w3.org/2001/XMLSchema-instance.

- *xcmlNamespace* is a URL representing the XCML schema definition namespace. The default value is http://www.csis.pace.edu/dps/xcml.

- *xsiSchemaLocation* is the XCML schema location. The default value is http://www.csis.pace.edu/dps/xcml Constraints.xsd.

- *name* is the *Constraints* name.

4.2.2.2 Stereotype *Constraint*

Constraint is a stereotype with a base type of Invariant. It defines a container element of an XCML constraint. It has no tagged value. It must contain either a *Rule* element or an *Assertion* element.

4.2.2.3 Stereotype *RuleConstraint*

RuleConstraint is a stereotype with a base type of Invariant. It defines an element of a rule-based constraint. It has no tagged value. If an *Invariant* constraint is assigned with this stereotype, it must contain at least one *If* element and zero or more *Else* elements.

4.2.2.4 Stereotype *AssertionConstraint*

AssertionConstraint is a stereotype with a base type of *Invariant*. It defines an assertion-based constraint. It has no tagged value. If an *Invariant* constraint is assigned with this stereotype, it must contain one *Assertion* element.

4.2.3 Conceptual Model

The physical model at this point is represented in the standard UML class notation together with the constraints on a class, an attribute, or an association. We take the order information model for example and identify three constraints:

- Discount rate: the discount rate is fifteen percent if the customer's membership is *Gold*, ten percent for *Silver* member, and zero for all other members.

- Reorder flag: the reorder flag sets to true if the *unitsInStock* is less than 10; otherwise it resets to false.

- Subtotal: the *subtotal* must be equal to the product of *unitPrice* and *quantity*.

The constrained conceptual model is obtained after applying the above constraints to the conceptual model, as shown in Figure 4-8.

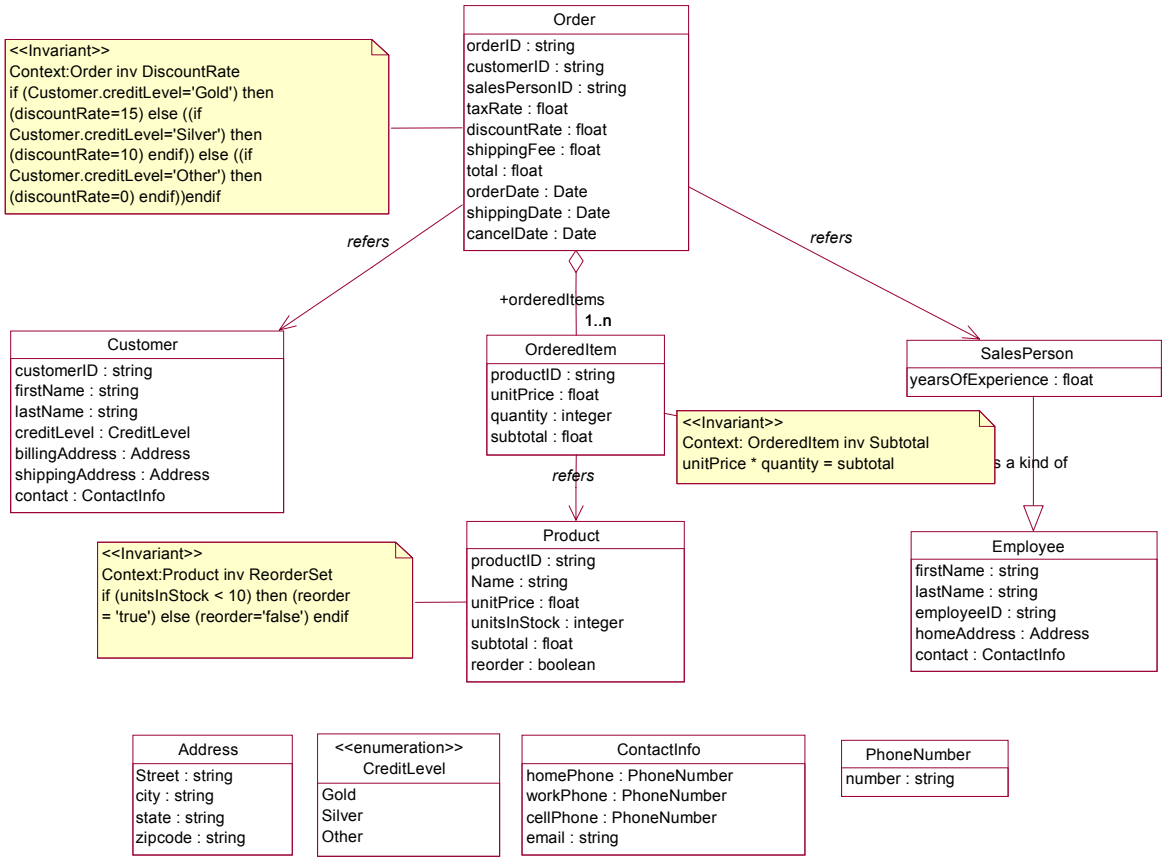

Figure 4-8 Constrained conceptual model of Order documents of a sample Order Entry System

This model represents not only the entities of the sample order system but also the constraints.

4.2.4 Logical Model

The conceptual model (Figure 4-8) does not indicate any XML data structure information. We apply the UML profile for XML Schema (Figure 4-3) and the UML profile for XCML schema (Figure 4-7) to the above conceptual model. The logical model that has XML Schema profile and XCML schema profile is presented in Figure 4-9.

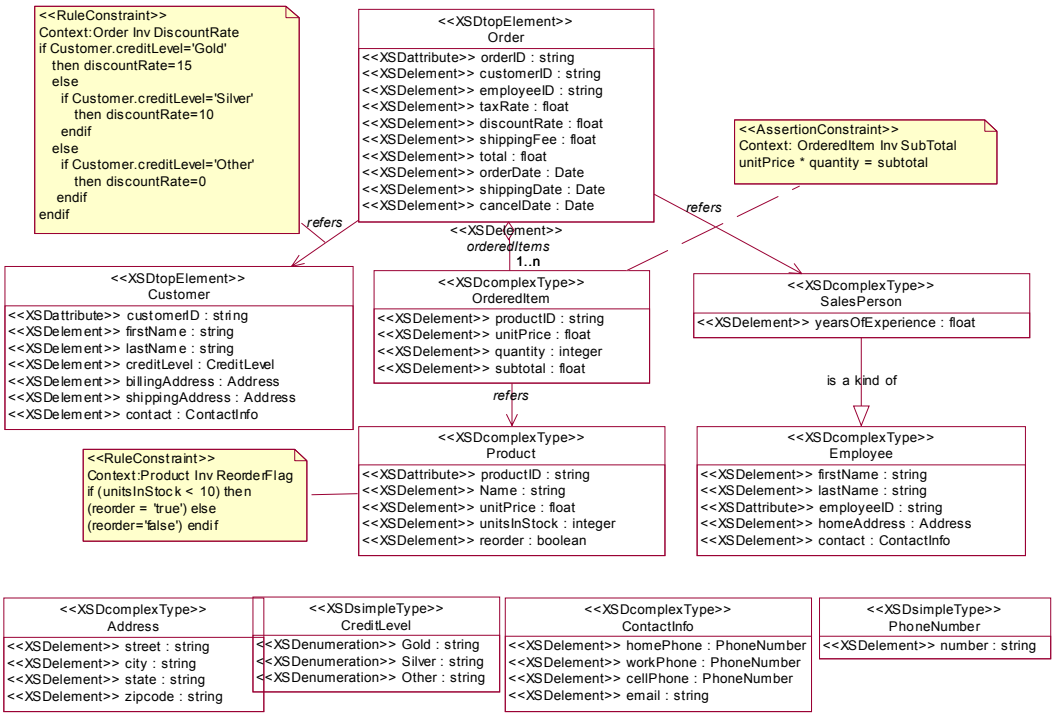

Figure 4-9 Constrained logical model of Order documents of a sample Order Entry system

This logical model is annotated with the XML Schema vocabularies and XCML schema concepts. Class *Order* is assigned stereotype *XSDtopElement*, which means that *Order*

will be mapped to the root element of an instance document for *Order*. *OrderID* is assigned stereotype *XSDattribute*, which means that *orderID* will be mapped to an attribute of the root element *Order*. In the same way, constraint *DiscountRate* is assigned stereotype *RuleConstraint*, which means that this constraint will be mapped to a *Rule* element within a *Constraint* element under the root element *Constraints*.

4.2.5 Physical Model

The physical model, the XML Schema and XCML instance documents, can then be generated from the constrained logical model above. The XCML instance for our example Order documents is presented in Listing 4-3. The implementation for this generation process will be discussed in Chapter 5.

Listing 4-3 XCML instance document

```
<?xml version="1.0" encoding="UTF-8"?>
<!--This document is automatically genereated through gen-ocl-doc.xsl!-->
<xcml:Constraints xmlns:xcml="http://www.csis.pace.edu/dps/xcml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.csis.pace.edu/dps/xcml Constraints.xsd" name="Order
Constraints">
    <Constraint context="Order">
        <Rule context="Product">
            <if test="unitsInStock &lt; 10">
                <then test="ReorderFlag = 'true'"/>
            </if>
            <else test="ReorderFlag='false'"/>
        </Rule>
    </Constraint>
    <Constraint context="Order">
        <Assertion context="OrderedItem">
            <test>unitPrice * quantity = subtotal</test>
        </Assertion>
    </Constraint>
    <Constraint context="Order">
        <Rule>
            <if test="Customer/creditLevel='Gold'">
                <then test="discountRate=15"/>
            </if>
            <else>
                <if test="Customer/creditLevel='Silver'">
```

```
                    <then test="discountRate=10"/>
                </if>
            </else>
            <else>
                <if test="Customer/creditLevel='Other'">
                    <then test="discountRate=0"/>
                </if>
            </else>
        </Rule>
    </Constraint>
</xcml:Constraints>
```

As discussed in Chapter 3, the generated XML schemas can be used to guide the generation of the Order instance documents and validate them syntactically. The generated XCML instance documents can be used to perform the semantic validation of the Order instance documents.

4.2.6 Issues in Modeling XCML Constraints

OCL together with the UML profile for XCML schema can model most of XCML constraints. However, UML 1.4 cannot represent parameterized constraints because OCL 1.4 can only specify static constraints on UML models. Most importantly, OCL (up to 1.4) has no metamodel, which makes it difficult to formally define the integration with the UML metamodel. OCL 2.0 submission version 1.6 defines a MOF 2.0 compliant metamodel for OCL. This metamodel defines the concepts and semantics of OCL and act as an abstract syntax for the language. Also, in this submission, several more stereotypes and key words are provides for specifying OCL expressions. However, the current UML modeling tools do not support UML 2.0 and OCL 2.0. Therefore, the UML models and OCL constraints only conforms to UML 1.4 in this research. We can explore to use OCL 2.0 support dynamic constraints, e.g., using key words *def* and *init* in a little bit different

way from its original definition in OCL 2.0. The key word *def* is used to declare a parameter/variable; and the key word *init* is used to assign an initial/default value of the parameter.

To illustrate how it works, an OCL constraint for the example in Listing 3-11 is shown in Listing 4-4.

Listing 4-4 Possible OCL 2.0 constraint of Tax rate constraint

```
context: Payroll inv Tax rate
def: rate
init: 0.05
taxRate = $rate
```

This Invariant constraint states that the value of *taxRate* attribute in the context *Payroll* is limited to the value conveyed by the parameter *rate* with a default value of 0.05.

Another issue is that even OCL 2.0 does not address how to OCL constraints work over inheritance [33].

These will be a very interesting research for the future.

4.3 Summary

This chapter extended the current state of XML data modeling. It introduced visual modeling of XML constraints. We designed a UML profile for XCML schema. Applying this UML profile and the UML profile for XML Schema, we can visually represent XML constraints over XML data models. The next chapter will describe the design and implementation of the generation process of XML schemas and XCML instance documents from the logical models.

Chapter 5

Generation of XML Schema and XCML Instance Documents

Once the logical model of a domain information model is available, the XML Schema and XCML instance documents can be generated. In this research, we propose to use XSLT and XMI technologies to accomplish this task. The major advantage of doing so is that both XMI and XSLT are open standards. Their toolkits are open source and freely available. We design and implement two toolkits. The first one is to generate W3C XML Schema instance documents from logical models. The second one is to generate XCML instance documents from logical models.

5.1 Common Mechanism

The XSLT specification 1.0 was originally designed for transforming from one XML document to another. It hides the details of the process of parsing XML documents behind the scene by using the high-level XSLT stylesheets. Such stylesheets are completely platform-independent like XML. The mechanism of this transformation is straightforward as shown in Figure 5-1. The XSLT processor is an implementation of XSLT specification 1.0, available in most of the known programming languages. The stylesheet is a pattern matching language implementing the transforming rules from the source, XML document A, to the target, XML document B. This mechanism is reusable for all the similar transformations.

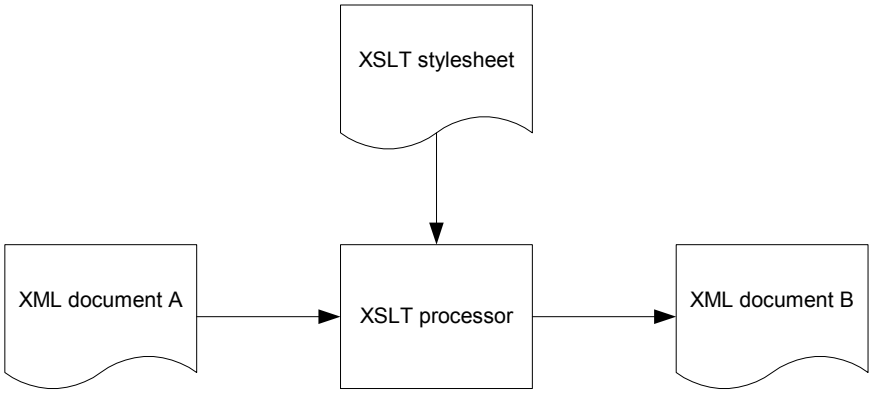

Figure 5-1 Common XSLT transformation approach

In this research, we use this approach for generating XML Schema and XCML instance documents from XMI documents. We also use it for the semantic validation of XML documents later in Chapter 6.

5.2 XMI Document Generation and Preprocessing

An XMI document is the XML representation of a UML model. It can be generated from a standalone XMI toolkit or an XMI plug-in component within a UML modeling environment such as Rational Rose from IBM Rational. The currently available XMI toolkits are implementations of XMI 1.1 [24]. No one has implemented the XML Schema production from XMI specification 1.0 [33] documents yet. We use the Rose 1.3 XMI 1.1 plug-in [35] from Unisys for the Rational Rose UML modeling tool [36] and the XMI toolkit 1.05 [37] from IBM as a standalone toolkit for XMI transformation. An XMI document is usually very big sine it contains all the information about a UML model. To simplify and speed up the transformation from XMI to XML Schema and XCML

documents, preprocessing the XMI document is needed. We use the XSLT approach (Figure 5-1) to preprocess the XMI document and extract the metadata for the generation of XML Schema and XCML instance documents.

5.2.1 XMI Document Generation

During this research, we use the Rational Rose Enterprise version 2002.05.20 for UML modeling. This version supports the Unisys Rose UML 1.3 XMI 1.1 plug-in. It can export a UML model to an XMI document. We also use IBM's XMI toolkit 1.05 as a standalone toolkit to generate XMI documents. Some UML tools like Poseidon even save UML models as XMI documents [43]. A block of an XMI document generated from the Unisys Rose UML 1.3 XMI 1.1 plug-in is shown in Listing 5-1.

Listing 5-1 A block of XMI document for a sample Order logical model

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- <!DOCTYPE XMI SYSTEM 'UMLX13-11.dtd' > -->
<XMI xmi.version="1.1" xmlns:UML="href://org.omg/UML/1.3" timestamp="Mon Jun 23 08:57:44 2003">
    <!-- etc -->
    <XMI.content>
        <!-- ==================== order-logical    [Model] ==================== -->
        <UML:Model xmi.id="G.0" name="order-logical" visibility="public" isSpecification="false"
          isRoot="false" isLeaf="false" isAbstract="false">
            <UML:Namespace.ownedElement>
                <!-- ======== order-logical::order    [Package] ============ -->
                <UML:Package xmi.id="S.173.0857.27.1" name="order" visibility="public"
                 isSpecification="false" isRoot="false" isLeaf="false" isAbstract="false" namespace="G.0">
                    <UML:Namespace.ownedElement>
                        <!-- etc -->
                        <!-- ============ order-logical::order::Address    [Class] ============= -->
                        <UML:Class xmi.id="S.173.0857.27.2" name="Address" visibility="public"
                          isSpecification="false" isRoot="true" isLeaf="true" isAbstract="false"
                          isActive="false" namespace="S.173.0857.27.1">
                            <UML:Classifier.feature>
                                <!-- === order-logical::order::Address.street    [Attribute] === -->
                                <UML:Attribute xmi.id="S.173.0857.27.3" name="street" visibility="private"
                                  isSpecification="false" ownerScope="instance" changeability="changeable"
                                  targetScope="instance" type="G.13">
                                    <UML:StructuralFeature.multiplicity>
                                        <UML:Multiplicity>
```

```
                    <UML:Multiplicity.range>
                        <UML:MultiplicityRange xmi.id="id.1741257.9" lower="1"
                          upper="1"/>
                    </UML:Multiplicity.range>
                </UML:Multiplicity>
            </UML:StructuralFeature.multiplicity>
            <UML:Attribute.initialValue>
                <UML:Expression language="" body=""/>
            </UML:Attribute.initialValue>
        </UML:Attribute>
        <!-- ===== order-logical::order::Address.city    [Attribute] ==== -->
        <UML:Attribute xmi.id="S.173.0857.27.4" name="city" visibility="private"
          isSpecification="false" ownerScope="instance" changeability="changeable"
          targetScope="instance" type="G.13">
            <UML:StructuralFeature.multiplicity>
                <UML:Multiplicity>
                    <UML:Multiplicity.range>
                        <UML:MultiplicityRange xmi.id="id.1741257.10" lower="1"
                          upper="1"/>
                    </UML:Multiplicity.range>
                </UML:Multiplicity>
            </UML:StructuralFeature.multiplicity>
            <UML:Attribute.initialValue>
                <UML:Expression language="" body=""/>
            </UML:Attribute.initialValue>
        </UML:Attribute>
        <!-- ==== order-logical::order::Address.state    [Attribute] ==== -->
        <UML:Attribute xmi.id="S.173.0857.27.5" name="state" visibility="private"
          isSpecification="false" ownerScope="instance" changeability="changeable"
          targetScope="instance" type="G.13">
            <UML:StructuralFeature.multiplicity>
                <UML:Multiplicity>
                    <UML:Multiplicity.range>
                        <UML:MultiplicityRange xmi.id="id.1741257.11" lower="1"
                          upper="1"/>
                    </UML:Multiplicity.range>
                </UML:Multiplicity>
            </UML:StructuralFeature.multiplicity>
            <UML:Attribute.initialValue>
                <UML:Expression language="" body=""/>
            </UML:Attribute.initialValue>
        </UML:Attribute>
        <!-- ==== order-logical::order::Address.zipcode    [Attribute] ==== -->
        <UML:Attribute xmi.id="S.173.0857.27.6" name="zipcode" visibility="private"
          isSpecification="false" ownerScope="instance" changeability="changeable"
          targetScope="instance" type="G.13">
            <UML:StructuralFeature.multiplicity>
                <UML:Multiplicity>
                    <UML:Multiplicity.range>
                        <UML:MultiplicityRange xmi.id="id.1741257.12" lower="1"
                          upper="1"/>
                    </UML:Multiplicity.range>
                </UML:Multiplicity>
            </UML:StructuralFeature.multiplicity>
            <UML:Attribute.initialValue>
```

```
                                        <UML:Expression language="" body=""/>
                                    </UML:Attribute.initialValue>
                                </UML:Attribute>
                            </UML:Classifier.feature>
                        </UML:Class>
                    <!-- etc -->
                </UML:Package>
                <!-- etc -->
            <UML:Namespace.ownedElement>
        </UML:Model>
    </XMI.content>
</XMI>
```

5.2.2 Preprocessing

Usually, the XMI documents are very big since they are the complete XML representation of a UML model. In order to efficiently and quickly generate XML Schema and XCML instance documents, we remove the graphical information from the XMI documents. This preprocessing is realized in XSLT technology (Figure 5-1). The source is an XMI document. The Pseudocode of the stylesheet for this preprocessing is illustrated in Listing 5-2. At this step, it only extracts the useful information such as information entities, data types, relationships, and stereotypes for the XML Schema and XCML instance document generation. No mappings are performed.

Listing 5-2 Pseudocode for preprocessing an XMI document

```
Read in XMI document

Template
    Match pattern: /
    Apply Template  Match pattern: Model
End Template

/* Template to process Model element */
Template
    Match pattern: Model
    Apply Template  Match pattern: Package
    Apply Template  Match pattern: DataType
```

```
End Template

/* Template to process Package element */
Template
    Match Package
    Apply Template  Match pattern: Class
    Apply Template  Match pattern: Association
    Apply Template  Match pattern: Comment
    Apply Template  Match pattern: Stereotype
End Template

/* Template to process DataType element */
Template
    Match DataType
    Extract DataType id
    Extract DataType name
End Template

/* Template to process Class element */
Template
    Match Class
    Extract Class name
    Extract Class id
    Apply Template  Match pattern: Generalization
    Apply Template  Match pattern: Attribute
End Template

/* Template to process Association element */
Template
    Match Association
    Extract Association name
    Extract Association id
    Apply Template  Match pattern: AssociationEnd
End Template

/* Template to process Comment element */
Template
    Match Comment
    If Comment contains "Invariant"
        Extract Comment content
    End If
End Template

/* Template to process Stereotype element */
Template
    Match Stereotype
    Extract Stereotype name
    Extract Stereotype id
    Extract ids that extend the current Stereotype
End Template

/* Template to process AssoicationEnd element */
Template
    Match AssociationEnd
    Extract AssociationEnd id
```

```
    Extract AssociationEnd name
    Extract aggreggation name
    Extract Associated Class ids
End Template
```

It starts from reading in an XMI document. It processes each Package element and each *DataType* element when an element *Model* is matched. For each *Package* container, it extracts *Class*es, *Association*s, *Comment*s, and *Stereotype*s. The reason to extract *Comment*s is that the constraints are embedded inside the element *Comment*. For each *Class*, it extracts the *Class name*, *Class id*, and *Attribute*s. The XSLT stylesheet implemented for this process is listed in Appendix B. A portion of the sample output is shown in Listing 5-3.

Listing 5-3 A portion of a sample output from preprocessing

```
<?xml version="1.0" encoding="UTF-8"?>
<Model>
    <Package name="order">
        <Class name="Address" id="S.173.0857.27.2">
            <Attribute id="S.173.0857.27.3">
                <name>street</name>
                <multiplicity>
                    <lower>1</lower>
                    <upper>1</upper>
                </multiplicity>
                <type>G.13</type>
            </Attribute>
            <Attribute id="S.173.0857.27.4">
                <name>city</name>
                <multiplicity>
                    <lower>1</lower>
                    <upper>1</upper>
                </multiplicity>
                <type>G.13</type>
            </Attribute>
            <Attribute id="S.173.0857.27.5">
                <name>state</name>
                <multiplicity>
                    <lower>1</lower>
```

```
                    <upper>1</upper>
                </multiplicity>
                <type>G.13</type>
            </Attribute>
            <Attribute id="S.173.0857.27.6">
                <name>zipcode</name>
                <multiplicity>
                    <lower>1</lower>
                    <upper>1</upper>
                </multiplicity>
                <type>G.13</type>
            </Attribute>
        </Class>
        <!-- etc -->
    </Package>
    <!-- etc -->
</Model>
```

Such an output will serve as the input source for generating both the XML Schema and XCML instance documents.

This process is specifically designed for the XMI output from Unisys Rose UML 1.3 XMI 1.1 plug-in component. The process is roughly the same if the input source is from different XMI toolkits.

5.3 XML Schema Generation

In this section, we focus on automating the generation of XML Schemas from a logical model that has W3C XML Schema vocabularies embedded. Routledge et al [29] proposed a method of designing XML Schemas using the three-level design approach. But their implementation has not been available yet. Carlson [23] proposed a similar approach of creating XML Schemas from UML class models. A commercial tool built from such an approach is available but not widely used. The OMG adopted the XMI production for XML Schema specification 1.0. This specification does specify the rules

for generating XML Schemas from object models but aims at exchanging object models in XML among a wide variety of objects: analysis (UML), software (Java, C++), components (e.g., EJB, IDL, CORBA Component Model), and databases. In this dissertation, we develop an XSLT application for generating XML Schemas from UML class models, more precisely, logical models (UML class models decorated with XML Schema vocabularies). The workflow of this generation reuses the approach shown in Figure 5-1. The XML source is replaced with the preprocessed XMI document. The implementation of this stylesheet is discussed below.

5.3.1 Mapping from Logical Models to XML Schemas

Carlson [23] proposed a set of criteria for W3C XML Schema generation from UML models. Based on Carlson's criteria, we develop the mapping rules for W3C XML Schema generation shown below.

Table 5-1 Mapping rules for W3C XML Schema generation

Mapping rule	Description
Namspace	The namespace from UML model is taken as the targetNamespace. The XML Schema namespace is http://www.w3.org/2001/XMLSchema by default. An entire model, or each package, or each class with a stereotype of <<XSDelement>> is represented in a single schema with the user's choice.
Element name uniqueness	Prefix each UML class with the package name followed by a '.' (for example, Package1.ClassA) if more than one package is contained in the model. Prefix each UML attribute and association role with the class name followed by a '.'.
Packages	A simple XML container element is generated for each package.
Classes	Generate a complexType definition when a UML class is assigned with a stereotype of <<XSDcomplexType>>

	or <<XSDtopElement>>. Generate a simpleType definition when a UML class is assigned with a stereotype of <<XSDsimpleType>>.
Elements or attributes	Generate an XML element for each UML attribute or association role with the assigned stereotype of <<XSDelement>>. Generate an XML attribute for each UML attribute or association role with the assigned stereotype of <<XSDattribute>>.
Multiplicity constraints	The values of minOccurs and maxOccurs attributes take the assigned values in the model.
Inheritance	When a UML class inherits from a single superclass, then generate a complexType that extends the superclass as the base type. When a UML class inherits multiple superclasses, then revert to copy-down inheritance to reproduce all superclass attribute and association roles following the mapping rule for Elements or Attributes.
Element order	The all container definition is used when the UML attributes are unordered. The sequence container definition is used when the UML attributes are ordered.
Datatypes	A Schema built-in data type is assigned to an attribute or element when its corresponding UML attribute has a UML primitive data type. A mapping from UML built-in data types to XML Schema built-in data types is seen in Table 5-2.

Table 5-2 Mapping from UML primitive data type to XML Schema built-in data type

UML primitive data type	**XML Schema built-in data type**
Integer	integer or int
Real	double or float
Boolean	boolean
String	string

5.3.2 Stylesheet Implementation

To implement the mapping rules defined in the above section, we develop five main anonymous XSLT templates, two named templates, and one helper template.

Figure 5-2 shows their relationships.

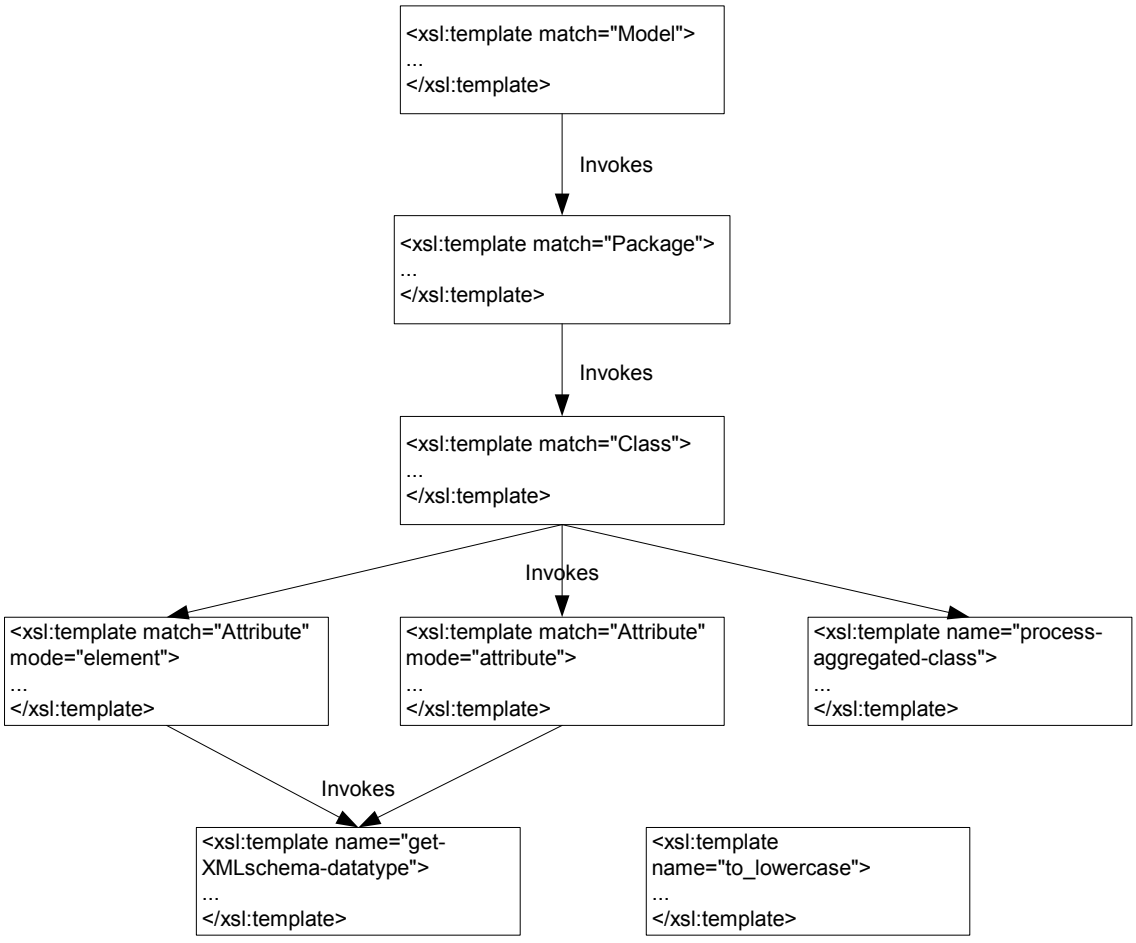

Figure 5-2 Relationships of various templates

Listing 5-4 illustrates the pseudocode of the XSLT stylesheet for generating XML Schemas from a preprocessed XMI document.

Listing 5-4 Pseudocode of XSLT Stylesheet for XML Schema Generation

```
Read in preprocessed XMI document

Template
    Match pattern: /
    Generate schema element container
        Generate namespaces
        Apply Template  Match pattern: Model
    End Generate
End Template

/* Template to process Model element */
Template
    Match pattern: Model
    For each Package element
        Apply Template  Match pattern: Package
    End For
End Template

/* Template to process Package element */
Template
    Match pattern: Package
    For each Class element
        Apply Template  Match pattern: Class
    End For
End Template

/* Template to process Class element */
Template
    Match pattern: Class
    Define classId  /* as the value of the attribute id of the Class */
    Define stereotype /* as the name of the Stereotype assigned to the Class */
    Choose
        When
            Test stereotype='XSDtopElement'
            Generate the top element
            Generate a complexType /* for the current Class */
                Apply Template  Match pattern: *       Mode: complexType
            End Generate
        End When
        When
            Test stereotype='XSDcomplexType'
            Generate a complexType /* for the current Class */
                Apply Template  Match pattern: *       Mode: complexType
            End Generate
        End When
        Otherwise
            Error message: wrong stereotype
        End Otherwise
    End Choose
End Template
```

```
/* Template to generate a complexType for a Class element */
Template
    Match pattern: *       Mode: complexType
    Choose
        When
            Test Generalization /* the current class has super classes */
            Generate an extension element /*with the base type of the super class */
            Apply Template  Match pattern: *       Mode: elementsAndAttributes
        End When
        Otherwise
            Apply Template  Match pattern: *       Mode: elementsAndAttributes
        End Otherwise
    End Choose
End Template

/* Template to generate a simpleType for a Class element, usually as datatype */
Template
    Match pattern: *       Mode: simpleType
    If ( Generalization )
        Generate an extension or restriction element /*with the base type of the super class */
        Generate a facet element /* from the current Class element */
    Else
        Generate primitive datatype
    End If
End Template

/* Template to generate elements and attributes of the Class
Template
    Match pattern: *       Mode: elementsAndAttributes
    If ( ordering='orderred' )
        Generate sequence element
            Apply Template  Name: get-associated-classes
            Apply Template  Match pattern: Attribute      Mode: element
        End Generate
    Else If ( ordering='unorderred' )
        Generate all element
            Apply Template  Name: get-associated-classes Parameter: subclassId
            Apply Template  Match pattern: Attribute      Mode: element
        End Generate
    End If
    Apply Template  Match pattern: Attribute      Mode: attribute
End Template

/* Template to process Attribute with stereotype of XSDelement */
Template
    Match pattern:Attribute      Mode: element
    Generate xsd:element element
        Generate name attribute
        Generate type attribute
        Generate minOccus attribute
        Generate maxOccus attribute
    End Generate
End Template

/* Template to process Attribute with stereotype of XSDattribute */
```

```
Template
    Match pattern:Attribute      Mode: attribute
    Generate xsd:attribute element
        Generate name attribute
        Generate type attribute
        Generate use attribute
    End Generate
End Template

/* Template to process associated Classes */
Template
    Name: get-associated-classes
    In: subclassId
    Apply Template  Match pattern:
//Association[AssociationEnd[classId=subclassId]/aggreggation='aggreggated']
End Template

/* Template to generate elements for associated classes */
Template
    Match pattern: //Association[AssociationEnd[classId=subclassId]/aggreggation='aggreggate']
    Generate xsd:element element
        Generate name attribute
        Generate type attribute
        Generate minOccus attribute
        Generate maxOccus attribute
    End Generate
End Template
```

The XSLT stylesheet is implemented in XSLT/XPath specification 1.0, see Apendix C for detail. Listing 5-5 shows the sample XML schema of a UML class *Address* in the sample Order Entry system.

Listing 5-5 A portion of generated XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.csis.pace.edu/dps/jhu"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:jhu="http://www.csis.pace.edu/dps/jhu">
    <xsd:complexType name="Address">
        <xsd:sequence>
            <xsd:element name="street" type="xsd:string"/>
            <xsd:element name="city" type="xsd:string"/>
            <xsd:element name="state" type="xsd:string"/>
            <xsd:element name="zipcode" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
    <!-- etc -->
 </xsd:schema>
```

5.4 XCML Document Generation

No one has attempted to generate XML constraint documents from UML models yet. The XCML proposed in this research provides the syntax to allow XML constraints represented in UML class models using OCL together with a UML profile for XCML schema. The XCML constraint definition is embedded in the XMI documents of the model. Thus, we use the XSLT approach shown in Figure 5-1 to generate XCML instance documents from the preprocessed XMI documents.

5.4.1 Mapping Rules

An OCL statement of a constraint in an XMI document is represented as a long string; see an example below:

```
<Constraint>&lt;&lt;RuleConstraint&gt;&gt; Context:Product Inv ReorderFlag if (UnitsInStock &lt;
10) then (ReorderFlag = 'true') else (ReorderFlag='false') endif</Constraint>
```

A mapping from OCL to XCML is demonstrated in Figure 5-3.

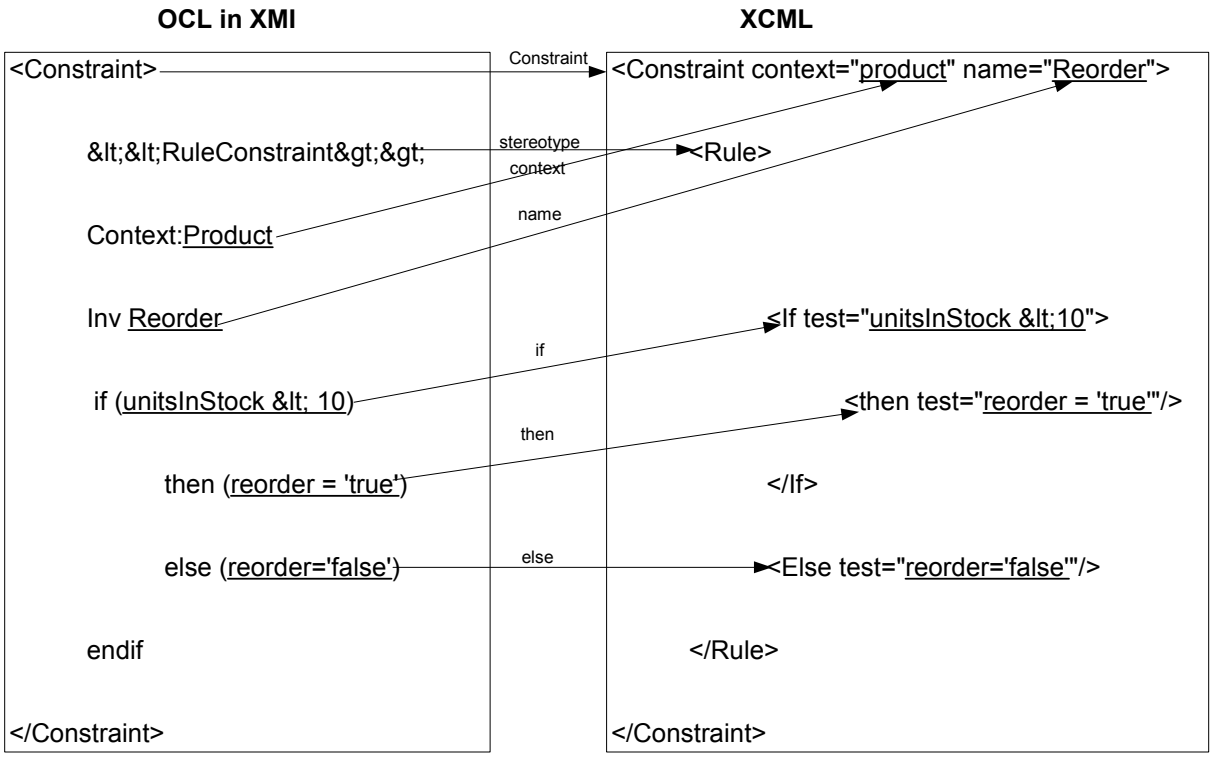

Figure 5-3 OCL to XCML mapping

It illustrates the mapping from an OCL statement in XMI to an XCML constraint fragment. An element *Constraint* in OCL is mapped to an element container *Constraint* in XCML. If its stereotype is *<<RuleConstraint>>*, an element container *Rule* is generated. The content of *Context* is assigned to the value of the attribute *@context*. The content of *Inv* is assigned to the value of the attribute *@name*. The content of *if* is mapped to the value of the attribute *@test* of the element *If*, and so on. Table 5-3 lists the mapping rules from OCL to XCML.

Table 5-3 Mapping rules from OCL to XCML

Mapping rule	Description
Namspace	The namespace from UML model is taken as the targetNamespace. The XML Schema instance namespace is http://www.w3.org/2001/XMLSchema-Instance by default.
Packages	An entire model or each package or each class with a stereotype of XSDtopElement is represented in a single XCML document with the root of Constraints.
Constraint	Generate Constraint element container for each OCL constraint.
RuleConstraint	Generate Rule element container for an OCL constraint with a stereotype of RuleConstraint.
If-then-else	Generate an If element container having an attribute of test and a child element of Then. The Then element has either an attribute of test or another If element container. Generate Else element that contains either an attribute of test or another If element container.
AssertionConstraint	Generate an Assertion element having an attribute of test.
Elements	Keep the element name.
Attributes	Add a '@' before the element name if its stereotype is XSDattribute.
Parameters	Generate Parameter elements.
.	Replace '.' With '/'

5.4.2 Stylesheet Implementation

We implement the mapping rules (Table 5-3) using XSLT stylesheet based on the XSLT 1.0 specification. Listing 5-6 lists the pseudocode of this implementation. Two template modules are developed. One is an anonymous template to identify the XCML constraint type of an OCL constraint and then construct an XCML element block. This template will invoke the other template to parse the OCL statement and tokenize it into finer units according to their XCML definition.

Listing 5-6 Pseudocode of XSLT stylesheet for generating XCML instance documents

```
Read in preprocessed XMI document

Template
    Match pattern: /
    Generate Constraints element container
        Apply Template  Match pattern: Model/Package/Constraint
    End Generate
End Template

/* Template to process Constraint element */
Template
    Match pattern: Constraint
    Generate Constraint element container
        Create context attribute
        Define ConstraintType
        If (ConstraintType = 'Assertion' )
            Generate Assertion element container
                Create test attribute
            End Generate
        Else If (ConstraintType = 'Rule' )
            Generate Rule element container
                Generate If element container
                    Create test attribute
                End Generate
                Generate Then element container
                    If (isComposite)
                        Apply Template Name: rule-tokenizer     Param: logicStatement
                    Else
                        Create test attribute
                End Generate
                If (isComposite)
                    Generate Else element container
                        If (isComposite)
                            Apply Template Name: rule-tokenizer     Param: logicStatement
                        Else
                            Create test attribute
                    End Generate
                End If
            End Generate
        End If
    End Generate
End Template

/* Template to recursively process the Rule constraint */
Template
    Name: rule-tokenizer    Param: logicStatement
    Generate If element container
        Create test attribute
        Generate Then element container
            If (isComposite)
                Apply Template Name: rule-tokenizer
            Else
                Create test attribute
```

```
            End If
        End Generate
        If (isComposite)
            Generate Else element container
                If (isComposite)
                    Apply Template Name: rule-tokenizer     Param: logicStatement
                Else
                    Create test attribute
            End Generate
        End If
    End Template
```

The source code of the implemented stylesheet is listed in Appendix D. This process may be simplified when OCL constraints are expressed in a more formal way, e.g., XML representation of OCL proposed by Ramalho [38]. The XCML instance document for the sample Order Entry system is shown in Listing 5-7.

Listing 5-7 Sample XCML document

```
<?xml version="1.0" encoding="UTF-8"?>
<!--This document is automatically genereated through gen-ocl-doc.xsl!-->
<xcml:Constraints xmlns:xcml="http://www.csis.pace.edu/dps/xcml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.csis.pace.edu/dps/xcml Constraints.xsd" name="Order
Constraints">
    <Constraint context="product">
        <Rule>
            <If test="UnitsInStock &lt; 10">
                <Then test="ReorderFlag = 'true'"/>
            </If>
            <Else test="ReorderFlag='false'"/>
        </Rule>
    </Constraint>
    <Constraint context="order/orderedItem">
        <Assertion test="unitPrice * quantity = subtotal"/>
    </Constraint>
    <Constraint context="order">
        <Parameter>
            <name>customer</ name >
            <defaultValue>C0001</ defaultValue >
        </ Parameter >
        <Rule>
            <If test="document($customer)//creditLevel='Gold'">
                <Then test="discountRate=15"/>
            </If>
```

```
            <Else>
                <If test="document($customer)/creditLevel='Silver'">
                    <Then test="discountRate=10"/>
                </If>
            </Else>
            <Else>
                <If test="document($customer)/creditLevel='Other'">
                    <Then test="discountRate=0"/>
                </If>
            </Else>
        </Rule>
    </Constraint>
</xcml:Constraints>
```

5.5 Summary

This chapter discusses the process of generating XML Schemas and XCML instance documents from UML logical models. A common mechanism of using XSLT to transform one XML document to another is reused for this process. The XSLT stylesheet implementation for each process is described in detail. Also, this chapter briefly illustrates some sample output of XML Schema and XCML instance document from the implementation. Next chapter will discuss the validation of XML documents against XML Schema and XCML instance documents.

Chapter 6

XML Document Validation

As discussed in Chapters 1, 2 and 3, the syntactic validation of an XML document is straightforward once its XML Schema is available. But the semantic validation of an XML document is much more complicated. In this research, we propose to use XCML to define XML constraints. This chapter focuses on how to perform the semantic validation of an XML document against its XCML instance document.

6.1 Workflow of XML Document Validation

The workflow of validating XML documents is shown in Figure 6-1. 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.

The syntactic validation checks whether the structures and data types of the XML document conform to the XML Schemas using any XML parser supporting XML Schema. The XML Schemas can be an output from the toolkit developed in Chapter 5. It can also be obtained from other applications or manual generation. The semantic validation checks whether the content of the XML document makes sense to specific applications. The XSLT stylesheets in this case are the XSLT interpretation of an XCML instance document – XML constraints. Section 6.3.1 discusses the generation of XSLT stylesheets from XCML instance documents.

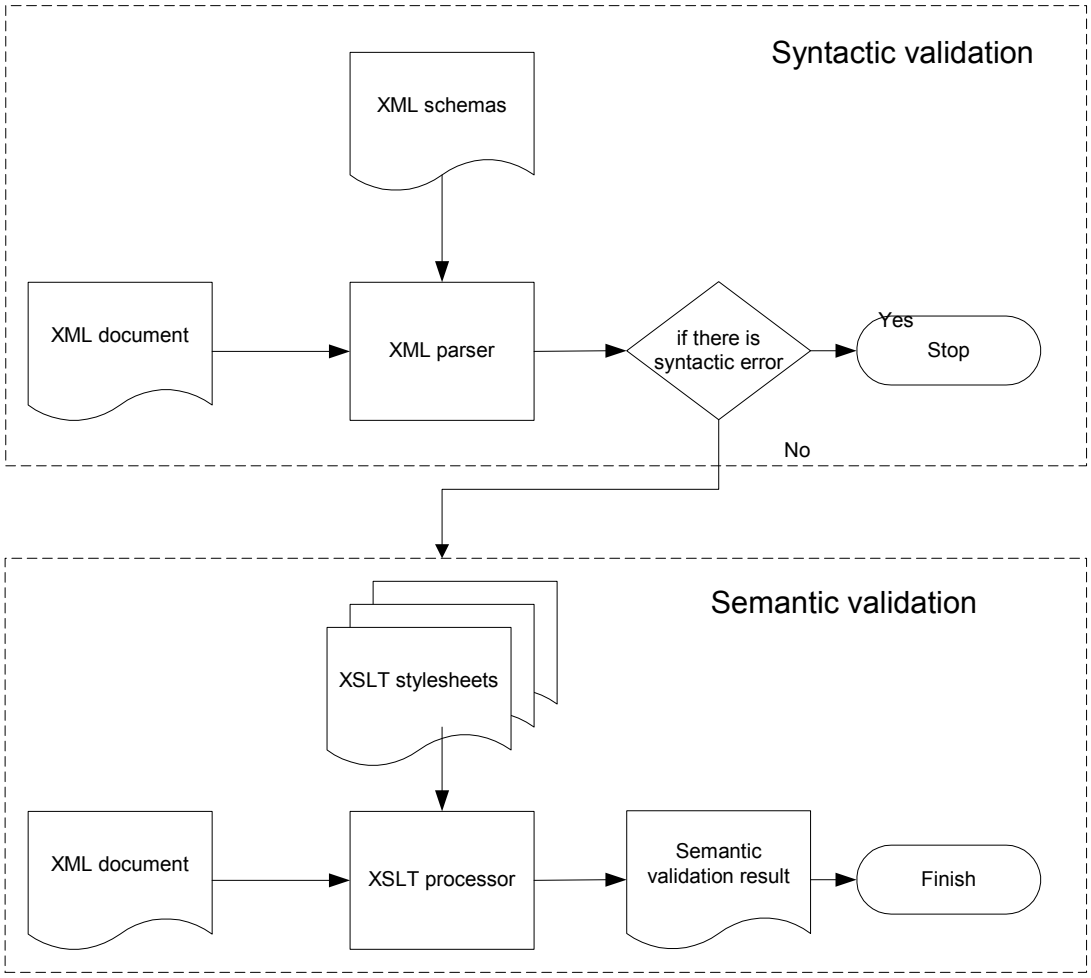

Figure 6-1 Workflow of XML document validation

6.2 Syntactic Validation

The syntactic validation doesn't require any coding work. It just performs a check of a XML document against its XML Schemas while an XML parser reads the document. The parser is freely available in most of the known programming languages. We have used Microsoft's latest XML parser 4.0 [39], Apache's Xalan 2.5.1 [40], Saxon 6.5 [41], and Saxon 7.6.5 [42] for test purpose.

6.3 Semantic Validation

6.3.1 Generating XSLT Stylesheets from XCML Documents

As discussed in Section 3.8 and Section 6.1, in order to perform semantic validation on an XML document using XSLT technology, an XSLT stylesheet that transforms an XCML document to another XSLT stylesheets is needed. In this section, we develop a reusable XSLT stylesheet that will be used to process the XCML documents and generate XSLT stylesheets for semantic validation.

6.3.1.1 Rules for Generating XSLT Templates from XCML Instance Documents

The rules for generating XSLT templates from XCML documents are straightforward. Figure 6-2 demonstrates the generation process for a sample XCML element *Constraint*.

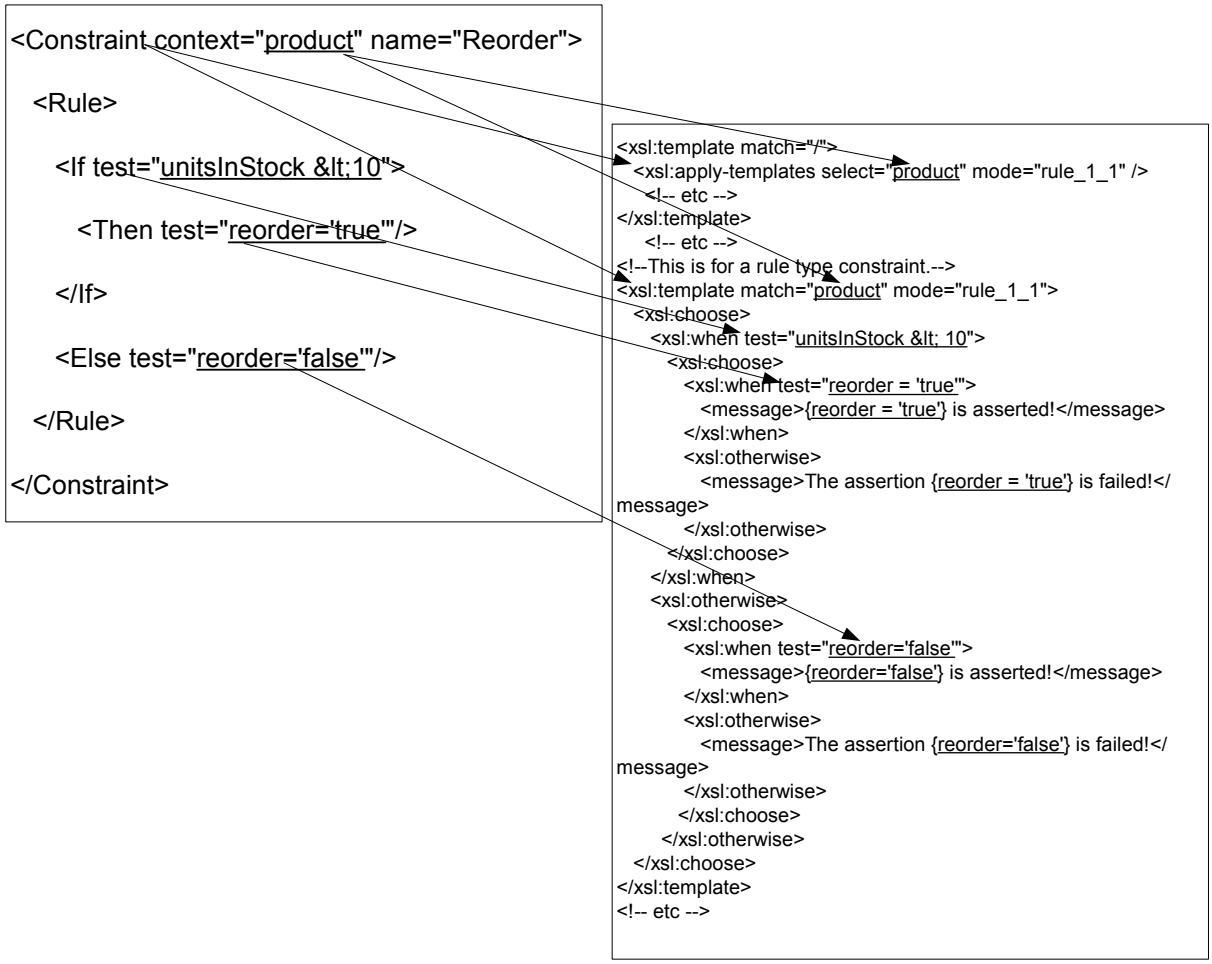

Figure 6-2 XCML to XSLT mapping

The template caller is embedded within the root template <*template match*="/"> ...</*template*>. The content of the element *Context* is taken as the matching pattern of the template caller and callee. The detail rules are given below:

1. The namespace of the XSLT is http://www.w3.org/1999/XSL/Transform by default. The targetNamespace of the XML document to be validated is taken if there is one.

2. The root template <xsl:template match="/">… </xsl:stylesheet> is generated for the root element *Constraint*.
3. An anonymous template caller <xsl:apply-templates select="…" mode="…" /> within the current template and the template <xsl:template match="…" mode="…">…</xsl:template> are generated for each element *Constraint*.
4. The value of attribute *@Context* is taken as the pattern of the attribute *@select* of the template caller and the pattern of the attribute *@match* of the callee.
5. The mode is used to uniquely identify the pair of caller and callee. The value of this attribute is decided by the position of the current element *Constraint* postfixed with a string of '*constraint_*'.
6. A simple <xsl:choose> clause with one <xsl:when> and <xsl:otherwise> is created for each element *Assertion*. The expression of the attribute *@test* in <xsl:when> directly takes the value of the attribute *@test* in the element *Assertion*.
7. A composite <xsl:choose> clause with one <xsl:when> and one <xsl:otherwise> is created for each element *Rule*. The <xsl:when> and <xsl:otherwise> each also contains a simple <xsl:choose> clause like Item 6 above if the element *Rule* defines a simple *If-Then-Else* constraint.
8. If there are any parameters in an element *Constraint*, the template caller passing the value to the parameter declared in the callee.

6.3.1.2 Stylesheet Implementation

We develop an XSLT stylesheet based on the rules specified in 6.3.1.1 in the XSLT specification 1.0. Listing 6-1 shows the pseudocode of XSLT stylesheet for generating XSLT stylesheet from an XCML document.

Listing 6-1 Pseudocode of XSLT stylesheet for XSLT stylesheet generation

```
Read in preprocessed XCML document

/* Matches the document root, creates the stylesheet container,      */
/* and invokes Template (match xcml:Constraints).                    */
Template
    Match pattern: /
    Generate xsl:stylesheet element container
        Apply Template      Match pattern:  Parameter (at all levels)
        Apply Template      Match pattern: xcml:Constraints
    End Generate
End Template

/* Matches all the Parameter elements, generate variable names,    */
/* and assigns values to them.                                       */
Template
    Match pattern: Parameter (at all levels)
    Create xsl:variable element
        Create attribute name
        Assign value to this parameter
    End Create
End Template

/* Matches xcml:Constraints element, creates the stylesheet container,          */
/* calls a Template (indicated by Mode: select) to create template invoker */
/* and calls a Template (indicated by Mode: match) to create the invokee.       */
Template
    Match pattern: xcml:Constraints
    Generate xsl:template element container
        Create match attribute
        Create a root element validation_result
            Apply Template      Match pattern: Constraint   Mode:select
        End Create
        Apply Template      Match pattern: Constraint   Mode:match
    End Generate
End Template

/* Template to create template invokers */
Template
    Match pattern: Constraint   Mode: select
    Apply Template      Match pattern: Rule     Mode:select
        In Parameter: parent_context, position
```

```
    End Apply
    Apply Template      Match pattern: Assertion        Mode:select
        In Parameter: parent_context, position
    End Apply
End Template

/* Template to create template invokees */
Template
    Match pattern: Constraint   Mode: match
    Apply Template      Match pattern: Rule     Mode:match
        In Parameter: parent_context, position
    End Apply
    Apply Template      Match pattern: Assertion        Mode:match
        In Parameter: parent_context, position
    End Apply
End Template

/* Template to create template invoker for Rule pattern */
Template
    Match pattern: Rule     Mode: select
    Define Parameter:parent_context, position
    Create xsl:apply-templates element
        Create attribute select
        Create attribute mode
    End Create
End Template

/* Template to create template invoker for Assertion pattern */
Template
    Match pattern: Assertion        Mode: select
    Define Parameter:parent_context, position
    Create xsl:apply-templates element
        Create attribute select
        Create attribute mode
    End Create
End Template

/* Template to create template invokee for Rule pattern */
Template
    Match pattern: Rule     Mode: match
    Define Parameter:parent_context, position
    Define variable: local_context
    Create xsl:templates element
        Create attribute match
        Create attribute mode
        If (count(Else)=0)
            Apply Template      Match pattern: If     Mode:if-then-simple
        Else
            Create xsl:choose element
                Apply Template      Match pattern: If     Mode: if-then-composite
                If (count(Else/If)=0)
                    Apply Template      Match pattern: Else     Mode: simple
                Else
                    Apply Template      Match pattern: Else     Mode: composite
                End If
```

```
            End Create
        End If
    End Create
End Template

/* Template to process xcml:Constraints element */
Template
    Match pattern: If      Mode:if-then-simple
    Create xsl:if element
        Call Template    Name: evaluate-expression
    End Create
End Template

/* Template to process xcml:Constraints element */
Template
    Match pattern: If      Mode:if-then-composite
    Create xsl:when element
        If (count(Else/If)=0)
            Apply Template       Match pattern: Then     Mode: simple
        Else
            Apply Template       Match pattern: Then     Mode: composite

        End If
    End Create
End Template

/* Template to invoke Template evaluate-expression for evaluating XPath  expressions of simple
Then or Else cases.*/
/* It can be invoked by two invokers */
/* with the Mode of simple. */
Template
    Match pattern: *       Mode: simple
    Call Template    Name: evaluate-expression
End Template

/* Template to process the composite Then or Else cases. It can be invoked by two invokers with
the Mode of composite. */
Template
    Match pattern: *       Mode: composite
    If (count(Else)=0)
        Apply Template  Match pattern: If Mode: if-then-simple
    Else
        Create xsl:choose element
            Apply Template  Match pattern: If Mode: if-then-composite
            Create xsl:otherwise element
                If (count(Else/If)=0)
                    Apply Template       Match pattern: Else       Mode: simple
                Else
                    Apply Template       Match pattern: Else       Mode: composite
                End If
            End Create
        End Create
    End If
End Template
```

```
/* Template to create template invokee for Assertion pattern */
Template
    Match pattern: Assertion        Mode: match
    Define parameter:parent_context, position
    Create xsl:template element
        Create attribute match
        Create attribute mode
        Call Template    Name: evaluate-expression
    End Create
End Template

/* Template to to create a block of XSLT source code */
/* for evaluating XPath  expressions*/
Template
    Name: evaluate-expression
    Create xsl:choose element
        Create xsl:when element
            Create attribute test
            Create element message reporting that the test case is true
        End Create
        Create xsl:otherwise element
            Create element message reporting that the test case is false
        End Create
    End Create
End Template
```

The implemented XSLT stylesheet is listed in Appendix E. Listing 6-2 shows a sample template generated from the above stylesheet.

Listing 6-2 Sample XSLT stylesheet generated by XSLT

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:output indent="yes" />
    <xsl:template match="/">
        <validation_result>
            <!-- etc -->
            <xsl:apply-templates select="product" mode="constraint_2" />
            <!-- etc -->
        </validation_result>
    </xsl:template>
    <!-- etc -->
    <xsl:template match="product" mode="constraint_2">
        <xsl:if test="UnitsInStock &lt; 10">
            <xsl:choose>
                <xsl:when test="ReorderFlag = 'true'">
                    <message>{ReorderFlag = 'true'} is asserted!</message>
                </xsl:when>
```

```
                <xsl:otherwise>
                    <message>The assertion {ReorderFlag = 'true'} is failed!</message>
                </xsl:otherwise>
            </xsl:choose>
        </xsl:if>
    </xsl:template>
    <!-- etc -->
</xsl:stylesheet>
```

6.4 Validation Tests

Up till now, we have developed a whole framework from the XCML syntax, visual modeling of XCML constraints, automation of the XML Schema and XCML document generation, to the XSLT stylesheets for semantic validation. In this section, we illustrate three test scenarios. The first one is the sample application used throughout this dissertation. We basically put each separate piece together to give a full view from modeling the XML constraints to validating XML documents against these constraints. The second one aims to show how XCML solves a sample problem used by the existing solutions. The last one demonstrates how XCML can represent some XML constraints that the existing solutions cannot do.

6.4.1 Order Report Validation

In this example, the constraints are considered at the system design phase. We write the various constraints on the UML class models in OCL (Figure 4-9). Applying the UML profile of XCML schema and the UML profile for XML Schema, the logical model with XML Schema vocabularies and XCML vocabularies is obtained (Figure 4-10). From this model, we generate an XML schema (Figure 4-5) and an XCML instance document (Figure 4-11) as well as the XSLT stylesheet from this XCML document for semantic

validation. We perform the syntactic validation and semantic validation using the approach outlined in Figure 6-1. The sample order report is given in Listing 6-3; a customer profile in Listing 6-4; and a product inventory in Listing 6-5. The Reorder constraint is shown in Listing 6-6. The order constraints are shown in Listing 6-7.

Listing 6-3 A sample order report, order.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file generated by XMLSPY v5 rel. 4 U (http://www.xmlspy.com)-->
<jhu:order xmlns:jhu="http://www.csis.pace.edu/dps/jhu"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.csis.pace.edu/dps/jhu order.xsd" orderID="order-0001-2003">
    <customerID>C0001</customerID>
    <employeeID>E0012</employeeID>
    <taxRate>6</taxRate>
    <discountRate>15</discountRate>
    <shippingFee>5</shippingFee>
    <total>9555.10</total>
    <orderDate>2003-08-13</orderDate>
    <shippingDate>2003-08-20</shippingDate>
    <orderItem>
        <productID>P0001</productID>
        <unitPrice>10.00</unitPrice>
        <quantity>10</quantity>
        <subtotal>100.00</subtotal>
    </orderItem>
    <orderItem>
        <productID>P0010</productID>
        <unitPrice>100.00</unitPrice>
        <quantity>100</quantity>
        <subtotal>10000.00</subtotal>
    </orderItem>
</jhu:order>
```

This is a sample order XML document. It is syntactically valid in terms of document structures and data types. However, with the XML Schema it cannot check if the discountRate of 15 is valid or not. It cannot check if the orderDate is before the shippingDate.

Listing 6-4 A sample customer document, customer-C0001.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<jhu:customer xmlns:jhu="http://www.csis.pace.edu/dps/jhu"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.csis.pace.edu/dps/jhu
C:\2003\dps2003\umlmodel\order.schema.xsd" customerID="C0001">
    <firstName>John</firstName>
    <lastName>Smith</lastName>
    <creditLevel>Gold</creditLevel>
    <billingAddress>
        <street>125 First Ave</street>
        <city>New York City</city>
        <state>New York</state>
        <zipcode>10026</zipcode>
    </billingAddress>
    <shippingAddress>
        <street>125 First Ave</street>
        <city>New York City</city>
        <state>New York</state>
        <zipcode>10026</zipcode>
    </shippingAddress>
    <contact>
        <homePhone>718-123-4567</homePhone>
        <workPhone>202-543-9876</workPhone>
        <cellPhone>718-678-1234</cellPhone>
        <email>johnsmith@johnsmith.com</email>
    </contact>
</jhu:customer>
```

This is the XML document for a specific customer C0001. It records the contact information and the membership level.

Listing 6-5 Product information, product-P0001.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file generated by XMLSPY v5 rel. 4 U (http://www.xmlspy.com)-->
<jhu:product xmlns:jhu="http://www.csis.pace.edu/dps/jhu"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.csis.pace.edu/dps/jhu
C:\2003\dps2003\umlmodel\order.schema.xsd" productID="P0001">
    <Name>Sample product</Name>
    <unitPrice>10.00</unitPrice>
    <UnitsInStock>20</UnitsInStock>
```

```
    <ReorderFlag>false</ReorderFlag>
</jhu:product>
```

This is a document about the inventory of the product *P0001*. Again, it is valid syntactically. But the XML schema cannot check whether the value of the element ReorderFlag is correct or not since it depends upon the value of the element *UnitsInStock*.

Listing 6-6 XCML document of Reorder constraint, reorder-xcml.xml

```
    <Constraint context="product">
        <Rule>
            <If test="UnitsInStock &lt; 10">
                <Then test="ReorderFlag = 'true'"/>
            </If>
            <Else test="ReorderFlag = 'true'"/>
        </Rule>
    </Constraint>
```

This is the XCML definition of the Reorder constraint. If the value of the element *UnitsInStock* is less than 10, the element *reorder* sets to true. Otherwise, it sets to false.

Listing 6-7 XCML document of order constraints, order-xcml.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<xcml:Constraints name="Order Constraints" xmlns:xcml="http://www.csis.pace.edu/dps/xcml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.csis.pace.edu/dps/xcml Constraints.xsd">
    <Constraint context="order">
        <Assertion test="orderDate > shippingDate"/>
    </Constraint>
    <Constraint context="order/orderItem">
        <Assertion test="unitPrice * quantity = subtotal"/>
    </Constraint>
    <Constraint context="order">
        <Rule>
            <If test="document('customer-C0001.xml')//creditLevel='Gold'">
```

```
                <Then test="discountRate=15"/>
            </If>
            <Else>
                <If test="document('customer-C0001.xml')//creditLevel='Silver'">
                    <Then test="discountRate=10"/>
                </If>
            </Else>
            <Else>
                <If test="document('customer-C0001.xml')//creditLevel='Other'">
                    <Then test="discountRate=0"/>
                </If>
            </Else>
        </Rule>
    </Constraint>
</xcml:Constraints>
```

This is the XCML definition of the shippingDate constraint, the Subtotal constraint, and the discountRate constraint generated from the XMI document.

Listing 6-8 Semantic validation result of Reorder constraint

```
<?xml version="1.0" encoding="UTF-8"?>
<validation_result>
    <UnitsInStock>20</UnitsInStock>
    <message>{ReorderFlag = 'false'} is asserted!</message>
</validation_result>
```

Listing 6-9 Semantic validation result of order constraints

```
<?xml version="1.0" encoding="UTF-8"?>
<validation_result xmlns:jhu="http://www.csis.pace.edu/dps/jhu">
    <message>{orderDate} is before {shippingDate} is asserted!</message>
    <message>{unitPrice * quantity = subtotal} is asserted!</message>
    <message>{unitPrice * quantity = subtotal} is not asserted!</message>
    <message>The assertion {discountRate=15} is failed!</message>
</validation_result>
```

As illustrated above, this validation process starts from the system design phase and ends at the final validation. The required XML Schemas for syntactic validation and the XSLT stylesheet for semantic validation are automatically generated from the logical model. The whole process is completely model-driven. At this point, we use a batch file to execute the four XSLT processings as a separate application. It can also be built as a plug-in component to a UML modeling toolkit or a similar environment. It can significantly reduce the development cycle of XML applications by improving the communication between software architects and application developers, precisely modeling the constraints, and model-driven automation of code generation that avoids creating error-prone source code.

6.4.2 User Profile Validation

This example has been used for illustrating how XincaML works. It is about validating user profiles. The schema document is given in Listing 6-10.

Listing 6-10 an XML Schema of user profile

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="userProfile" type="userProfileType"/>
    <xs:complexType name="userProfileType">
        <xs:sequence>
            <xs:element name="userID" type="xs:string"/>
            <xs:element name="password" type="xs:string"/>
            <xs:element name="email" type="xs:string"/>
            <xs:element name="telephoneNumber" type="xs:string"/>
            <xs:element name="address" type="AddressType"/>
            <xs:element name="role" type="RoleType"/>
            <xs:element name="manages" type="ManagesType"/>
            <xs:element name="yearsOfWork" type="xs:integer"/>
            <xs:element name="payroll" type="PayrollType"/>
        </xs:sequence>
        <xs:attribute name="classID" type="xs:string" fixed="234321"/>
    </xs:complexType>
    <xs:complexType name="AddressType">
```

```
        <xs:sequence>
            <xs:element name="country" type="xs:string"/>
            <xs:element name="state" type="xs:string"/>
            <xs:element name="city" type="xs:string"/>
            <xs:element name="street" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
    <xs:simpleType name="RoleType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="secretary"/>
            <xs:enumeration value="manager"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:complexType name="ManagesType">
        <xs:sequence>
            <xs:element name="userID" type="xs:string" minOccurs="0"
              maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="PayrollType">
        <xs:sequence>
            <xs:element name="salary" type="xs:float"/>
            <xs:element name="bonus" type="xs:float"/>
            <xs:element name="tax" type="xs:float"/>
            <xs:element name="netIncome" type="xs:float"/>
            <xs:element name="hasSavingFund" type="xs:boolean"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Three constraints are going to be specified:

- If the value of a user's role does not equal "manager", then <manages> element must not be present.
- If a user has worked for more than five years in this company, then he/she can get an additional saving fund in his/her payroll.
- The two constraints above are both rule style constraints. This one is an assertion style constraint which specifies that the net income of a user must equal the difference of the sum of salary and bonus with tax, i.e. netIncome = salary + bonus - tax.

  Their XCML definition is shown in Listing 6-11.

Listing 6-11 XCML definition of the user profile constraints

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

XCML is more concise and clearer. It leverages the XSLT/XPath – XML core technology. The XCML constraint documents are easier to be created and understood. And most importantly, it can be written on the UML class models, which automates the generation of XCML constraint documents.

A sample user profile is given in Listing 6-12.

Listing 6-12 Sample user profile

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- <userProfile> -->
<userProfile>
    <userID>zuozuo</userID>
    <password>password</password>
```

```
    <email>zuozuo@cn.ibm.com</email>
    <telephoneNumber>86-10-62986677</telephoneNumber>
    <address>
        <country>China</country>
        <state>Beijing</state>
        <city>Beijing</city>
        <street>ShangDi 5th Street</street>
    </address>
    <role>manager</role>
    <yearsOfWork>3</yearsOfWork>
    <payroll>
        <salary>10000</salary>
        <bonus>2000</bonus>
        <tax>2000</tax>
        <netIncome>10000</netIncome>
        <hasSavingFund>true</hasSavingFund>
    </payroll>
</userProfile>
```

It is validated against the XML Schema shown in Listing 6-10. The semantic validation result is shown in Listing 6-13.

Listing 6-13 Semantic validation result of userprofile.xml against the three constraints

```
<?xml version="1.0" encoding="UTF-8"?>
<validation_result>
    <message>{payroll/hasSavingFund = 'true'} is asserted!</message>
    <message>{salary + bonus - tax = netIncome} is asserted!</message>
    <message>{count(manages) = 1} is asserted!</message>
</validation_result>
```

6.5 Summary

This chapter discusses the workflow of validating XML documents. We describe the implementation of the XSLT stylesheet for generating XSLT stylesheets from XCML instance documents for semantic validation. The implemented XSLT stylesheet is

reusable for all the XCML instance documents. We demonstrate how to validate XML documents using XSLT technology.

Chapter 7

Conclusions and Future work

The validity of XML documents in reality means that such documents are valid syntactically and semantically. We discussed that syntactic validation can only gurranttee that an XML document satisfies its grammer. Semantic validation is often needed to check whether documents make sense to applications. We conducted a review of the current solutions of semantically validating XML documents. We proposed a new XML constraint language – XCML, or extensible constraint language. A comprehensive comparison has shown that XCML is much more powerful than the existing XML constraint languages in expressing various constraints. More importantly, the UML profile for XCML schema developed in this research allows XML constraints to be represented in XML data models.

7.1 Major Contributions

7.1.1 Extensible Constraint Markup Language

In this research, we classified XML constraints into two main categories: assertion-based constraints and rule-based constraints. We came up with an XML constraint language - XCML based on reviewing their strongths and weaknesses of the existing XML constraint languages. It is much more expressive than the existing XML constraint

languages by supporting dynamic constraints, composite rule-based constraints, and more.

7.1.2 Visual Modeling of XML Constraints

We built a UML profile for XCML schema. Together with OCL, or Object Constraint Language, and a UML profile for XML Schema [23], XML constraints can be modeled and put on XML data models. This approach allows semantic constraints to be considered from the system design phase. It avoids miscommunication between software architects and application developers. Furthermore, representing semantic constraints in UML models makes it possible to automate the generation of XML constraint documents.

7.1.3 XCML Document Generation and XML Document Validation

We use a common and reusable XSLT approach for the generation of XML Schemas and XCML instance documents from XMI documents and the semantic validation of XML documents. We implemented a set of stylesheets for this approach. The stylesheet modules can be reused in different applications.

7.2 Future Work

It is for sure that the validity of semantic constraints of XML documents is and will be playing more and more important role in data mining, data integration, decision support, document publishing, etc. XML is still relatively new. The technologies in processing, manipulating, and managing XML documents are not mature. For example, XPath 2.0 and XSLT 2.0 are not standardized yet. XML constraint languages are still in the research stage. Also, UML 2.0 suites are adopted by the OMG and vendors are upgrading heir tools to support UML 2.0. The following are some related research topics for the future.

7.2.1 Syntactic Validation vs. Semantic Validation

It sounds obvious that there is no confusion between syntactic validation and semantic validation. In reality, specifically, in XML domain, XML tags may convey some meaning ful concepts. Also, XML Schemas, or the structure definition of XML documents, can specify some values of XML elements or attributes. It means that a valid document against its XML schema is semantically valid to some extent. On the other hand, semantic validation can check the tag definitions of XML documents. There is some interleaving between syntactic validation and semantic validation. Is it necessary to explore how to differentiate them and how to separate them?

7.2.2 XML Constraints vs. XPath 2.0

The current solutions to semantic validation only consider XPath 1.0 if they leverage XPath. XPath 2.0 provides many more capabilities than XPath 1.0 does. How does XPath 2.0 in the future affect the current XML constraint languages is another research topic.

7.2.3 Mapping of XML Schema and UML

Although there have been several researches in mappings between XML Schema vocabularies and UML notations, some outstanding issues still exist.

7.2.4 OCL and XML Constraint Languages

In this research, very few UML modeling tools support UML 2.0 and OCL 2.0. The implementation of generating XML Schema and XCML instance documents from XMI documents is mainly based on UML 1.4. The mapping from UML 2.0/OCL 2.0 to XCML instance documents needs further research.

7.2.5 XML Schemas vs. XML Constraint Documents

In our research, we used XCML instance documents to express XML constraints, and these XCML instance documents are separate from XML Schema instance documents. In Schematron, the constraint documents can be embedded within XML Schemas. Which way is more efficient and easy to use for validating XML documents is also a future research topic.

7.2.6 Reference Implementaiton

The implementation in this research is just an initial prototype; it will be very significant to develop a reference implementation or an open source toolkit, specifically when UML 2.0 and OCL 2.0 become popular and XPath 2.0 and XSLT 2.0 become standards.

References

[1] World Wide Web Consortium (W3C), "Extensible Markup Language (XML) 1.0," W3C Recommendation, February 1998. Available at: http://www.w3.org/TR/1998/REC-xml-19980210.

[2] World Wide Web Consortium (W3C), "Voice Extensible Markup Language (VoiceXML) Version 2.0," W3C Recommendation, February 2003. Available at: http://www.w3.org/TR/2003/CR-voicexml20-20030220.

[3] World Wide Web Consortium (W3C), "Scalable Vector Graphics (SVG) 1.1 Specification," W3C Recommendation, January 2003. Available at: http://www.w3.org/TR/SVG11.

[4] World Wide Web Consortium (W3C), *Web Services Suites*, Web site: http://www.w3.org/2002/ws/#drafts.

[5] Organization for the Advancement of Structured Information Standards (OASIS), "Electronic Business XML Initiative (ebXML)," Technology Report, January 2003. Available at: http://xml.coverpages.org/ebXML.html.

[6] World Wide Web Consortium (W3C), "OWL Web Ontology Language Overview". W3C Candidate Recommendation, August 2003. Available at: http://www.w3.org/TR/owl-features.

[7] Healthcare Level Seven (HL7) Standard Organization, "The Clinical Document Architecture Level 1". Available at: http://www.hl7.org.

[8] W. Norman and M. Leonard, *Docbook: The Definitive Guide*, O'Reilly & Associates, Inc. October 1999.

[9] XML:DB Initiative for XML Databases, Web site: http://www.xmldb.org.

[10] World Wide Web Consortium (W3C), "W3C XML Schema 1.0," W3C Recommendations, May 2001. Available at: http://www.w3.org/XML/Schema#dev.

[11] World Wide Web Consortium (W3C), "XSL Transformations (XSLT) Version 1.0," W3C Recommendations, November 1999. Available at: http://www.w3.org/TR/xslt.

[12] World Wide Web Consortium, "XPath 1.0," W3C Recommendation, November 1999. Available at: http://www.w3.org/TR/xpath.

[13] L. Dodds, "Schematron: Validating XML Using XSLT," Proceedings of XSLT UK Conference, 2001, Keble College, Oxford, England.

[14] M. H. Jacinto, G. R. Librelotto, J. C. L. Ramalho, and P. R. Henriques, "Constraint Specification Languages: Comparing XCSL, Schematron and XML-Schemas," XML Europe 2002, Barcelona, Spain.

[15] IBM alphaWorks, eXtensible Inter-Nodes Constraint Mark-up Language (XincaML), December 2002. Available at: http://www.alphaworks.ibm.com/tech/xincaml.

[16] C. Nentwich, L. Capra, W. Emmerich, A. Finkelstein, "Xlinkit: A Consistency Checking and Smart Link Service," Research Note RN/00/66, University College London, Dept. of Computer Science, 2000.

[17] Object Management Group, Model-Driven Architecture specification 1.0, 2002. Available at: http://www.omg.org/uml.

[18] Object Management Group (OMG), Unified Modeling Language specification 1.1, 1997. Available at: http://www.omg.org.

[19] Object Management Group, Unified Modeling Language specification 1.4, Sept 2001. Available at: http://cgi.omg.org/docs/formal/01-09-67.pdf.

[20] Object Management Group, Metadata Object Facility specification 1.4, April 2002. Available at: http://cgi.omg.org/docs/formal/02-04-03.pdf.

[21] Object Management Group, Unified Modeling Language specification 2.0 suite, 2003. Available at: http://www.omg.org/uml.

[22] W. Jos and K. Anneke. *The Object Constraint Language: Precise Modeling with UML*, Addison-Wesley publisher, 1999.

[23] D. Carlson, *Modeling XML Applications with UML*, Addison-Wesley Publisher, 2001.

[24] Object Management Group, "XML Metadata Interchange specification 1.1," October 1999. Available at: http://www.omg.org/uml.

[25] Organization for the Advancement of Structured Information Standards (OASIS), "Relax NG Schema Specification," November 2001. Available at: http://www.oasis-open.org/committees/relax-ng/spec-20011203.html.

[26] World Wide Web Consortium, "Resource Description Framework Suites," Available at: http://www.w3.org/RDF/#specs.

[27] E. V. der Vlist, "Comparing XML Schema Languages," published on http://www.XML.com. Available at: http://www.xml.com/pub/a/2001/12/12/schemacompare.html.

[28] J. C. L. Ramalho, "Constraining Content: Specification and Processing," XML Europe 2001, Berlin, Germany.

[29] Altova, Inc., XMLSpy 2004: XML development environment, 2003. Web site: http://www.xmlspy.com.

[30] TIBCO Software Inc., XML Authority tool, 2003. Web site: http://www.extensibility.com.

[31] N. Routledge, L. Bird, and A. Goodchild, "UML and XML Schema," Database Technologies 2002, Thirteenth Australasian Database Conference (ADC2002), Monash University, Melbourne, Victoria, January/February 2002.

[32] W. Provost, "UML For W3C XML Schema Design," XML.com. Accessed on August 2002. http://www.xml.com/pub/a/2002/08/07/wxs_uml.html.

[33] T. Beale, "OCL 2.0 Review Based on Initial Submission 1.6 - 6 Jan 2003," draft review, Deep Thought Informatics, April 2003. Available at: http://www.deepthought.com.au/it/ocl_review.html. Accessed on Sept 2003.

[34] Object Management Group, XML Metadata Interchange Production of XML Schema specification 1.0, June 2001. Available: http://cgi.omg.org/docs/ad/01-06-12.pdf.

[35] Unisys Software Corp, Unisys Rose UML 1.3.4 plug-in. Accessed in April 2003. http://www.unisys.com.

[36] Rational Software Corporation, Rational Rose Enterprise, version 2002.05.20. 2002.

[37] IBM, XMI toolkit 1.05. Accessed in July 1999. http://www.alphaworks.ibm.com/tech/xmitoolkit.

[38] F. Ramalho, J. Robin, and R. Barros, "XOCL – an XML Language for Specifying Logical Constraints in Object Oriented Models," Journal of Universal Computer Science, Vol. 9, no.8 (2003), pp. 956-969.

[39] Microsoft Corp., MSXML 4.0 Service Pack 2 (Microsoft XML Core Services), June 2003. Available at: http://www.microsoft.com/downloads.

[40] Apache Software Foundation, Xalan XSLT Processor version 2.5.1, March 2003. Available at: http://xml.apache.org/xalan-j/index.html.

[41] SourceForge, XML processing tool suite version 6.5.3. Available at: http://saxon.sourceforge.net/saxon6.5.3/index.html.

[42] SourceForge, XML processing tool suite version 7.6.5. Available at: http://saxon.sourceforge.net/saxon7.6.5/index.html.

[43] Gentalware AG Poseidon for UML: UML tool, Sept 2003. Web site: http://www.gentleware.com.

Glossary

CDA	Clinical Document Architecture
DTD	Document Type Definition
ebXML	Electronic Business XML
MDA	Model-Driven Architecture
OCL	Object Constraint Language
OMG	Object management Group
SGML	Standard Generalized Markup Language
SVG	Scalable Vector Graphics
UML	Unified Modeling Language
W3C	World Wide Web Consortium
XMI	XML Metadata Interchange
XincaML	eXtensible Inter-node Constraint Markup Language
XML	eXtensible Markup Langauge
XPath	XML Path Language
XSLT	eXtensible Stylesheet Language Transformations
XCML	eXtensible Constraint Markup Language
XCSL	XML Constraint Specification Language

Appendix A

XCML Schema Definition

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.csis.pace.edu/dps/xcml"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xcml="http://www.csis.pace.edu/dps/xcml"
elementFormDefault="unqualified" attributeFormDefault="unqualified">
    <xsd:element name="Constraints" type="xcml:Constraints">
    </xsd:element>
    <!-- ********************************************************************************************
    *                                                                                                *
    *                                                                                                *
    *                    This is the definition of Constraints type.                                 *
    *                                                                                                *
    *                                                                                                *
    *********************************************************************************************-->
    <xsd:complexType name="Constraints">
        <xsd:annotation>
            <xsd:documentation>
                It contains one or more constraint elements.
            </xsd:documentation>
            <xsd:documentation>
                It may have a name for this constraints.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="Constraint" type="xcml:Constraint" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string" use="optional"/>
    </xsd:complexType>
    <!-- ********************************************************************************************
    *                                                                                                *
    *                                                                                                *
    *                    This is the definition of Parameter type.                                   *
    *                                                                                                *
    *                                                                                                *
    *********************************************************************************************-->
    <xsd:complexType name="Parameter">
        <xsd:annotation>
            <xsd:documentation>
                A parameter must have a name.
            </xsd:documentation>
            <xsd:documentation>
                A parameter may have a defaultValue.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
```

```
            <xsd:element name="name" type="xsd:string"/>
            <xsd:element name="defaultValue" type="xsd:string" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
    <!-- ********************************************************************************************
    *                                                                                                *
    *                                                                                                *
    *                    This is the definition of Constraint type.                                  *
    *                                                                                                *
    *                                                                                                *
    *********************************************************************************************-->
    <xsd:complexType name="Constraint">
        <xsd:annotation>
            <xsd:documentation>
                Constraint may have one or more Parameters used by individual constraints.
                </xsd:documentation>
            <xsd:documentation>
                It has either a Rule element or an Assertion element.
            </xsd:documentation>
            <xsd:documentation>
                A constraint should be on a specific context.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="Parameter" type="xcml:Parameter" minOccurs="0"
maxOccurs="unbounded"/>
            <xsd:choice>
                <xsd:element name="Rule" type="xcml:Rule"/>
                <xsd:element name="Assertion" type="xcml:Assertion"/>
            </xsd:choice>
        </xsd:sequence>
        <xsd:attribute name="context" type="xsd:string" use="required"/>
    </xsd:complexType>
    <!-- ********************************************************************************************
    *                                                                                                *
    *                                                                                                *
    *                    This is the definition of Rule type.                                        *
    *                                                                                                *
    *                                                                                                *
    *********************************************************************************************-->
    <xsd:complexType name="Rule">
        <xsd:annotation>
            <xsd:documentation>
                A Rule type is kind-of if-then or if-then-else constraint.
            </xsd:documentation>
            <xsd:documentation>
                Else is optional.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="If" type="xcml:If"/>
            <xsd:element name="Else" type="xcml:Then" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <!-- ********************************************************************************************
```

```
    *                                                                                                   *
    *                                                                                                   *
    *                    This is the definition of Assertion type.                                      *
    *                                                                                                   *
    *                                                                                                   *
    ****************************************************************************************************-->
    <xsd:complexType name="Assertion">
        <xsd:annotation>
            <xsd:documentation>
                Assertion only has an attribute test.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:attribute name="test" type="xsd:string" use="required"/>
    </xsd:complexType>
    <!-- ****************************************************************************************************
    *                                                                                                   *
    *                                                                                                   *
    *                    This is the definition of If type.                                             *
    *                                                                                                   *
    *                                                                                                   *
    ****************************************************************************************************-->
    <xsd:complexType name="If">
        <xsd:annotation>
            <xsd:documentation>
                If must have an attribute test which holds an XPath expression.
            </xsd:documentation>
            <xsd:documentation>
                It also has a mandatory element Then.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="Then" type="xcml:Then"/>
        </xsd:sequence>
        <xsd:attribute name="test" type="xsd:string" use="required"/>
    </xsd:complexType>
    <!-- ****************************************************************************************************
    *                                                                                                   *
    *                                                                                                   *
    *                    This is the defintion of Then type. It also aplies to else type.               *
    *                                                                                                   *
    *                                                                                                   *
    ****************************************************************************************************-->
    <xsd:complexType name="Then">
        <xsd:annotation>
            <xsd:documentation>
                Then either contains another if-then or if-then-else or a simple test.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence minOccurs="0">
            <xsd:element name="If" type="xcml:If"/>
            <xsd:element name="Else" type="xcml:Then" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="test" type="xsd:string" use="optional"/>
    </xsd:complexType>
</xsd:schema>
```

Appendix B

Stylesheet for Preprocessing XMI Documents

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:UML="href://org.omg/UML/1.3"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
    <!--                                    -->
    <!-- File Name: postprocess.xslt -->
    <!-- Description: This stylesheet extracts the model information for XML generating Schema
and XCML instance document. -->
    <!-- Date created: June 2003 -->
    <!-- Date modified: August 2003 -->
    <!--                                    -->
    <xsl:template match="/">
        <xsl:apply-templates select="XMI"/>
    </xsl:template>
    <xsl:template match="XMI">
        <xsl:apply-templates select="XMI.content"/>
    </xsl:template>
    <xsl:template match="XMI.content">
        <xsl:apply-templates select="UML:Model"/>
    </xsl:template>
    <!--  ************************************************************
    *        Processing Model                                        *
    *                                                                *
    ********************************************************************* -->
    <xsl:template match="UML:Model">
        <xsl:element name="Model">
            <!-- Extract Packages -->
            <xsl:apply-templates select="UML:Namespace.ownedElement/UML:Package"/>
            <!-- Extract datatypes -->
            <xsl:apply-templates select="UML:Namespace.ownedElement/UML:DataType"/>
        </xsl:element>
    </xsl:template>
    <!--  ************************************************************
    *        Processing Packages                                       *
    *                                                                                  *
    ********************************************************************* -->
    <xsl:template match="UML:Package">
        <xsl:element name="Package">
            <xsl:attribute name="name"><xsl:value-of select="@name"/></xsl:attribute>
            <!-- Extract classes -->
            <xsl:apply-templates select="UML:Namespace.ownedElement/UML:Class"/>
            <!-- Extract associations -->
            <xsl:apply-templates select="UML:Namespace.ownedElement/UML:Association"/>
```

```
        <!-- Extract comments -->
        <xsl:apply-templates select="UML:Namespace.ownedElement/UML:Comment"/>
        <!-- Extract stereotypes -->
        <xsl:apply-templates select="UML:Namespace.ownedElement/UML:Stereotype"/>
    </xsl:element>
</xsl:template>
<!--  *************************************************************
*       Processing Classes                                        *
*                                                                 *
********************************************************************* -->
<xsl:template match="UML:Class">
    <xsl:element name="Class">
        <xsl:attribute name="name"><xsl:value-of select="@name"/></xsl:attribute>
        <xsl:attribute name="id"><xsl:value-of select="@xmi.id"/></xsl:attribute>
        <xsl:apply-templates select="UML:Namespace.ownedElement/UML:Generalization"/>
        <xsl:apply-templates select="UML:Classifier.feature/UML:Attribute"/>
    </xsl:element>
</xsl:template>
<!--  *************************************************************
*       Processing Generalizations                                *
*                                                                 *
********************************************************************* -->
<xsl:template match="UML:Generalization">
    <xsl:element name="Generalization">
        <xsl:attribute name="id"><xsl:value-of select="@xmi.id"/></xsl:attribute>
        <xsl:element name="parent">
            <xsl:attribute name="id"><xsl:value-of select="@parent"/></xsl:attribute>
        </xsl:element>
        <xsl:element name="child">
            <xsl:attribute name="id"><xsl:value-of select="@child"/></xsl:attribute>
        </xsl:element>
    </xsl:element>
</xsl:template>
<!--  *************************************************************
*       Processing Attributes                                     *
*                                                                 *
********************************************************************* -->
<xsl:template match="UML:Attribute">
    <xsl:element name="Attribute">
        <xsl:attribute name="id"><xsl:value-of select="@xmi.id"/></xsl:attribute>
        <xsl:element name="name">
            <xsl:value-of select="@name"/>
        </xsl:element>
        <xsl:element name="multiplicity">
            <xsl:element name="lower">
                <xsl:value-of select=".//UML:MultiplicityRange/@lower"/>
            </xsl:element>
            <xsl:variable name="value" select=".//UML:MultiplicityRange/@upper"/>
            <xsl:if test="$value='-1'">
                <xsl:element name="upper">n</xsl:element>
            </xsl:if>
            <xsl:if test="$value!='-1'">
                <xsl:element name="upper">
                    <xsl:value-of select="$value"/>
                </xsl:element>
```

```
                </xsl:if>
            </xsl:element>
            <xsl:element name="type">
                <xsl:value-of select="@type"/>
            </xsl:element>
        </xsl:element>
    </xsl:template>
    <!--  *************************************************************
    *        Processing Comments - constraints                  *
    *                                                                                                  *
    ********************************************************************** -->
    <xsl:template match="UML:Comment">
        <xsl:if test="contains(UML:ModelElement.name,'Inv')">
            <xsl:element name="Constraint">
                <xsl:copy-of select="translate(normalize-
space(UML:ModelElement.name),'&#x000d;&#x000a;',')"/>
            </xsl:element>
        </xsl:if>
    </xsl:template>
    <!--  *************************************************************
    *        Processing Stereotypes                                        *
    *                                                                                                  *
    ********************************************************************** -->
    <xsl:template match="UML:Stereotype">
        <xsl:element name="Stereotype">
            <xsl:attribute name="id"><xsl:value-of select="@xmi.id"/></xsl:attribute>
            <xsl:element name="name">
                <xsl:value-of select="@name"/>
            </xsl:element>
            <xsl:choose>
                <xsl:when test="contains(normalize-space(@extendedElement),' ')">
                    <xsl:call-template name="tokenizer">
                        <xsl:with-param name="ids" select="normalize-
space(@extendedElement)"/>
                    </xsl:call-template>
                </xsl:when>
                <xsl:otherwise>
                    <xsl:element name="extendedElement">
                        <xsl:value-of select="normalize-space(@extendedElement)"/>
                    </xsl:element>
                </xsl:otherwise>
            </xsl:choose>
        </xsl:element>
    </xsl:template>
    <xsl:template match="@extendedElement">
        <xsl:value-of select="."/>
    </xsl:template>
    <xsl:template name="tokenizer">
        <xsl:param name="ids"/>
        <xsl:choose>
            <xsl:when test="contains($ids,' ')">
                <xsl:variable name="id" select="substring-before($ids,' ')"/>
                <xsl:element name="extendedElement">
                    <xsl:value-of select="$id"/>
                </xsl:element>
```

```
            </xsl:when>
            <xsl:otherwise>
                <xsl:element name="extendedElement">
                    <xsl:value-of select="$ids"/>
                </xsl:element>
            </xsl:otherwise>
        </xsl:choose>
        <xsl:variable name="id" select="substring-before($ids,' ')"/>
        <xsl:if test="contains($ids,' ')">
            <xsl:call-template name="tokenizer">
                <xsl:with-param name="ids" select="normalize-space(substring-after($ids,$id))"/>
            </xsl:call-template>
        </xsl:if>
    </xsl:template>
    <!--  *************************************************************
    *       Processing Datatypes                                        *
    *                                                                   *
    ********************************************************************* -->
    <xsl:template match="UML:DataType">
        <xsl:element name="Datatype">
            <xsl:attribute name="id"><xsl:value-of select="@xmi.id"/></xsl:attribute>
            <xsl:element name="name">
                <xsl:value-of select="@name"/>
            </xsl:element>
        </xsl:element>
    </xsl:template>
    <!--  *************************************************************
    *       Processing Associations                                     *
    *                                                                   *
    ********************************************************************* -->
    <xsl:template match="UML:Association">
        <xsl:element name="Association">
            <xsl:attribute name="id"><xsl:value-of select="@xmi.id"/></xsl:attribute>
            <xsl:element name="name">
                <xsl:value-of select="@name"/>
            </xsl:element>
            <xsl:apply-templates select="UML:Association.connection"/>
        </xsl:element>
    </xsl:template>
    <xsl:template match="UML:Association.connection">
        <xsl:apply-templates select="UML:AssociationEnd"/>
    </xsl:template>
    <xsl:template match="UML:AssociationEnd">
        <xsl:element name="AssociationEnd">
            <xsl:attribute name="id"><xsl:value-of select="@xmi.id"/></xsl:attribute>
            <xsl:element name="name">
                <xsl:value-of select="@name"/>
            </xsl:element>
            <xsl:element name="ordering">
                <xsl:value-of select="@ordering"/>
            </xsl:element>
            <xsl:element name="aggregation">
                <xsl:value-of select="@aggregation"/>
            </xsl:element>
            <xsl:element name="classId">
```

```
                <xsl:value-of select="@type"/>
            </xsl:element>
        </xsl:element>
        <xsl:element name="multiplicity">
            <xsl:element name="lower">
                <xsl:value-of select=".//UML:MultiplicityRange/@lower"/>
            </xsl:element>
            <xsl:variable name="value" select=".//UML:MultiplicityRange/@upper"/>
            <xsl:if test="$value='-1'">
                <xsl:element name="upper">n</xsl:element>
            </xsl:if>
            <xsl:if test="$value!='-1'">
                <xsl:element name="upper">
                    <xsl:value-of select="$value"/>
                </xsl:element>
            </xsl:if>
        </xsl:element>
    </xsl:template>
</xsl:stylesheet>
```

Appendix C

XSLT Stylesheet for XML Schema Generation

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:UML="href://org.omg/UML/1.3"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:jhu="http://www.csis.pace.edu/dps/jhu">
    <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
    <xsl:variable name="type-prefix">jhu:</xsl:variable>
    <xsl:variable name="w3c-prefix">xsd:</xsl:variable>
    <!--  ****************************************************************
    *        Processing Model                                        *
    *                                                                *
    **************************************************************** -->
    <xsl:template match="/">
        <xsl:element name="xsd:schema">
            <xsl:attribute
name="targetNamespace">http://www.csis.pace.edu/dps/jhu</xsl:attribute>
            <xsl:apply-templates select="Model"/>
        </xsl:element>
    </xsl:template>
    <!--  ****************************************************************
    *        Processing Model                                        *
    *                                                                *
    **************************************************************** -->
    <xsl:template match="Model">
        <xsl:apply-templates select="Package"/>
    </xsl:template>
    <!--  ****************************************************************
    *        Processing Packages                                        *
    *                                                                *
    **************************************************************** -->
    <xsl:template match="Package">
        <xsl:apply-templates select="Class"/>
    </xsl:template>
    <!--  ****************************************************************
    *        Processing Classes                                      *
    *                                                                *
    **************************************************************** -->
    <xsl:template match="Class">
        <xsl:variable name="classId" select="@id"/>
        <xsl:variable name="stereotype"
select="../Stereotype[extendedElement=$classId]/name"/>
        <xsl:if test="$stereotype='XSDcomplexType' or $stereotype='XSDsimpleType'">
            <xsl:element name="{concat($w3c-prefix,substring-after($stereotype,'XSD'))}">
                <!-- Create a complexType for each class. -->
```

```
                <xsl:attribute name="name"><xsl:value-of select="@name"/></xsl:attribute>
                <xsl:choose>
                    <xsl:when test="not(Generalization)">
                        <xsl:element name="xsd:sequence">
                            <!-- Check if there are any aggregate classes in this class. If so, insert an
element in this sequence. -->
                            <xsl:call-template name="get-aggregated-classes"/>
                            <xsl:apply-templates select="Attribute" mode="element"/>
                        </xsl:element>
                        <xsl:apply-templates select="Attribute" mode="attribute"/>
                    </xsl:when>
                    <!-- This type inherits other type -->
                    <xsl:otherwise>
                        <xsl:variable name="parent">
                            <xsl:value-of select="Generalization/parent/@id"/>
                        </xsl:variable>
                        <xsl:element name="xsd:extension">
                            <xsl:attribute name="base"><xsl:value-of select="concat($w3c-
prefix,//Class[@id=$parent]/@name)"/></xsl:attribute>
                            <xsl:element name="xsd:sequence">
                                <!-- Check if there are any aggregate classes in this class. If so, insert
an element in this sequence. -->
                                <xsl:call-template name="get-aggregated-classes">
                                    <xsl:with-param name="classId" select="@id"/>
                                </xsl:call-template>
                                <xsl:apply-templates select="Attribute" mode="element"/>
                            </xsl:element>
                            <xsl:apply-templates select="Attribute" mode="attribute"/>
                        </xsl:element>
                    </xsl:otherwise>
                </xsl:choose>
            </xsl:element>
        </xsl:if>
        <xsl:if test="$stereotype='XSDtopElement'">
            <xsl:element name="xsd:element">
                <xsl:variable name="name">
                    <xsl:value-of select="@name"/>
                </xsl:variable>
                <xsl:attribute name="name">
                    <xsl:call-template name="toLowercase">
                        <xsl:with-param name="text" select="$name"/>
                    </xsl:call-template>
                </xsl:attribute>
                <xsl:attribute name="type"><xsl:value-of select="concat($type-
prefix,@name)"/></xsl:attribute>
            </xsl:element>
            <xsl:element name="xsd:complexType">
                <!-- Create a complexType for each class. -->
                <xsl:attribute name="name"><xsl:value-of select="@name"/></xsl:attribute>
                <xsl:choose>
                    <xsl:when test="not(Generalization)">
                        <xsl:element name="xsd:sequence">
                            <!-- Check if there are any aggregate classes in this class. If so, insert an
element in this sequence. -->
                            <xsl:call-template name="get-aggregated-classes">
```

```
                                <xsl:with-param name="classId" select="@id"/>
                            </xsl:call-template>
                            <xsl:apply-templates select="Attribute" mode="element"/>
                        </xsl:element>
                        <xsl:apply-templates select="Attribute" mode="attribute"/>
                    </xsl:when>
                    <!-- This type inherits other type -->
                    <xsl:otherwise>
                        <xsl:variable name="parent">
                            <xsl:value-of select="Generalization/parent/@id"/>
                        </xsl:variable>
                        <xsl:element name="xsd:extension">
                            <xsl:attribute name="base"><xsl:value-of select="concat($w3c-
prefix,//Class[@id=$parent]/@name)"/></xsl:attribute>
                            <xsl:element name="xsd:sequence">
                                <!-- Check if there are any aggregate classes in this class. If so, insert
an element in this sequence. -->
                                <xsl:call-template name="get-aggregated-classes">
                                    <xsl:with-param name="classId" select="@id"/>
                                </xsl:call-template>
                                <xsl:apply-templates select="Attribute" mode="element"/>
                            </xsl:element>
                            <xsl:apply-templates select="Attribute" mode="attribute"/>
                        </xsl:element>
                    </xsl:otherwise>
                </xsl:choose>
            </xsl:element>
        </xsl:if>
    </xsl:template>
    <!--  ***************************************************************
    *        Processing Attributes - generate XSD elements
    *
    *                                                                 *
    ********************************************************************* -->
    <xsl:template match="Attribute" mode="element">
        <xsl:variable name="attributeId" select="@id"/>
        <xsl:if test="//Stereotype[extendedElement=$attributeId]/name='XSDelement'">
            <xsl:element name="xsd:element">
                <xsl:attribute name="name"><xsl:value-of select="name"/></xsl:attribute>
                <xsl:variable name="type" select="type"/>
                <xsl:if test="//Datatype/@id=$type">
                    <xsl:attribute name="type"><xsl:value-of select="concat($w3c-
prefix,//Datatype[@id=$type]/name)"/></xsl:attribute>
                </xsl:if>
                <xsl:if test="//Class/@id=$type">
                    <xsl:attribute name="type"><xsl:value-of select="concat($type-
prefix,//Class[@id=$type]/@name)"/></xsl:attribute>
                </xsl:if>
                <xsl:if test="multiplicity/lower!='1'">
                    <xsl:attribute name="minOccurs"><xsl:value-of
select="multiplicity/lower"/></xsl:attribute>
                </xsl:if>
                <xsl:if test="multiplicity/upper!='1' and multiplicity/upper!='n'">
                    <xsl:attribute name="maxOccurs"><xsl:value-of
select="multiplicity/upper"/></xsl:attribute>
```

```
                </xsl:if>
                <xsl:if test="multiplicity/upper='n'">
                    <xsl:attribute name="maxOccurs">unbounded</xsl:attribute>
                </xsl:if>
            </xsl:element>
        </xsl:if>
    </xsl:template>
    <!--  *************************************************************
    *        Processing Attributes - generate XSD attributes
        *
    *                                                                   *
    ********************************************************************* -->
    <xsl:template match="Attribute" mode="attribute">
        <xsl:variable name="attributeId" select="@id"/>
        <xsl:if test="//Stereotype[extendedElement=$attributeId]/name='XSDattribute'">
            <xsl:element name="xsd:attribute">
                <xsl:attribute name="name"><xsl:value-of select="name"/></xsl:attribute>
                <xsl:variable name="type" select="type"/>
                <xsl:attribute name="type"><xsl:value-of select="concat($w3c-
prefix,//Datatype[@id=$type]/name)"/></xsl:attribute>
                <xsl:attribute name="use"><xsl:if test="multiplicity/lower='0'">optional</xsl:if><xsl:if
test="multiplicity/lower='1'">required</xsl:if></xsl:attribute>
            </xsl:element>
        </xsl:if>
    </xsl:template>
    <!--  *************************************************************
    *        Processing aggregation     - generate XSD elements from aggregation *
    *                                                                   *
    ********************************************************************* -->
    <xsl:template name="get-aggregated-classes">
        <xsl:param name="classId"/>
        <xsl:apply-templates
select="//Association[AssociationEnd[classId=$classId]/aggregation='aggregate']"/>
    </xsl:template>
    <xsl:template match="Association">
        <xsl:if test="AssociationEnd[aggregation!='aggregate']/name!=''">
            <xsl:element name="xsd:element">
                <xsl:attribute name="name"><xsl:value-of
select="AssociationEnd[aggregation!='aggregate']/name"/></xsl:attribute>
            </xsl:element>
        </xsl:if>
        <xsl:if test="AssociationEnd[aggregation!='aggregate']/name=''">
            <xsl:element name="xsd:element">
                <xsl:variable name="classId">
                    <xsl:value-of select="AssociationEnd[aggregation!='aggregate']/classId"/>
                </xsl:variable>
                <xsl:variable name="name">
                    <xsl:value-of select="//Class[@id=$classId]/@name"/>
                </xsl:variable>
                <xsl:attribute name="name">
                    <xsl:call-template name="toLowercase">
                        <xsl:with-param name="text" select="$name"/>
                    </xsl:call-template>
                </xsl:attribute>
                <xsl:attribute name="type"><xsl:value-of select="concat($type-
```

```
prefix,//Class[@id=$classId]/@name)"/></xsl:attribute>
                <xsl:if test="multiplicity/lower!='1'">
                    <xsl:attribute name="minOccurs"><xsl:value-of
select="multiplicity/lower"/></xsl:attribute>
                </xsl:if>
                <xsl:if test="multiplicity/upper!='1' and multiplicity/upper!='n'">
                    <xsl:attribute name="maxOccurs"><xsl:value-of
select="multiplicity/upper"/></xsl:attribute>
                </xsl:if>
                <xsl:if test="multiplicity/upper='n'">
                    <xsl:attribute name="maxOccurs">unbounded</xsl:attribute>
                </xsl:if>
            </xsl:element>
        </xsl:if>
    </xsl:template>
    <!--  ********************************************************************
    *        Transform the first letter of a word/text from upper  *
    *        to lower case. It can be extended to any number    *
    *        of letters easily.                                 *
    *                                                           *
    *                                                           *
    ********************************************************************* -->
    <xsl:template name="toLowercase">
        <xsl:param name="text"/>
        <xsl:variable name="uppercase" select="'ABCDEFGHIJKLMNOPQRSTUVWXYZ'"/>
        <xsl:variable name="lowercase" select="'abcdefghijklmnopqrstuvwxyz'"/>
        <xsl:value-of
select="concat(translate(substring($text,1,1),$uppercase,$lowercase),substring($text,2,string-
length($text)-1))"/>
    </xsl:template>
</xsl:stylesheet>
```

Appendix D

XSLT Stylesheet for XCML Document Generation

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:jhu="http://www.csis.pace.edu/jhu"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
    <xsl:variable name="prefix">jhu:</xsl:variable>
    <xsl:variable name="namespace">"http://www.csis.pace.edu/jhu"</xsl:variable>
    <xsl:template match="/">
        <xsl:comment>This document is automatically genereated through gen-ocl-
doc.xsl!</xsl:comment>
        <xsl:element name="{concat($prefix,'Constraints')}">
            <xsl:attribute name="xsi:schemaLocation">http://www.csis.pace.edu/jhu
Constraints.xsd</xsl:attribute>
            <xsl:attribute name="Name"><xsl:value-of select="concat(Model/Package/@name,'
constraints')"/></xsl:attribute>
            <xsl:apply-templates select="Model/Package/Constraint"/>
        </xsl:element>
    </xsl:template>
    <xsl:template match="Constraint">
        <xsl:element name="Constraint">
            <xsl:variable name="OCLstatement">
                <xsl:copy-of select="normalize-space(.)"/>
            </xsl:variable>
            <xsl:variable name="constraintType" select="translate(normalize-space(substring-
before($OCLstatement,'Inv')),'&lt;&lt;','')"/>
            <xsl:attribute name="context">
                <xsl:choose>
                    <xsl:when test="not(contains($OCLstatement,'Context:'))">
                        <xsl:value-of select="concat('.','')"/>
                    </xsl:when>
                    <xsl:otherwise>
                        <xsl:variable name="context-value" select="normalize-space(substring-
after($OCLstatement,'Context:'))"/>
                        <xsl:choose>
                            <xsl:when test="contains($context-value,'if')">
                                <xsl:value-of select="translate(normalize-space(substring-
before($context-value,'if')),'.','/')"/>
                            </xsl:when>
                            <xsl:otherwise>
                                <xsl:value-of select="translate(normalize-space(substring-
before($context-value,' ')),'.','/')"/>
                            </xsl:otherwise>
                        </xsl:choose>
```

```
                    </xsl:otherwise>
                </xsl:choose>
            </xsl:attribute>
            <xsl:element name="{$constraintType}">
                <xsl:variable name="logicalStatement" select="substring-
after($OCLstatement,'Context:')"/>
                <xsl:choose>
                    <xsl:when test="$constraintType='Assertion'">
                        <xsl:attribute name="test">
                            <xsl:value-of select="substring-after(normalize-space($logicalStatement),'
')"/>
                        </xsl:attribute>
                    </xsl:when>
                    <xsl:when test="$constraintType='Rule'">
                        <xsl:element name="if">
                            <xsl:variable name="ifStatement" select="translate(normalize-
space(substring-before(substring-after($logicalStatement,'if'),'then')),'()','')"/>
                            <xsl:attribute name="test">
                                <xsl:choose>
                                    <xsl:when test="contains($ifStatement,'multiplicity')">
                                        <xsl:value-of select="concat('count(',translate(substring-
before($ifStatement,'.multiplicity'),'.','/'),')',substring-after($ifStatement,'multiplicity'))"/>
                                    </xsl:when>
                                    <xsl:otherwise>
                                        <xsl:value-of select="translate($ifStatement,'.','/')"/>
                                    </xsl:otherwise>
                                </xsl:choose>
                            </xsl:attribute>
                            <xsl:element name="then">
                                <xsl:variable name="thenStatement">
                                    <xsl:choose>
                                        <xsl:when test="contains($logicalStatement,'else')">
                                            <xsl:value-of select="translate(normalize-space(substring-
before(substring-after($logicalStatement,'then'),'else')),'()','')"/>
                                        </xsl:when>
                                        <xsl:otherwise>
                                            <xsl:value-of select="translate(normalize-space(substring-
after($logicalStatement,'then')),'()','')"/>
                                        </xsl:otherwise>
                                    </xsl:choose>
                                </xsl:variable>
                                <xsl:choose>
                                    <xsl:when test="not(contains($thenStatement,'if'))">
                                        <xsl:attribute name="test">
                                            <xsl:choose>
                                                <xsl:when test="contains($thenStatement,'multiplicity')">
                                                    <xsl:value-of
select="concat('count(',translate(substring-before
    ($thenStatement,'.multiplicity'),'.','/'),')',substring-after($thenStatement,'multiplicity'))"/>
                                                </xsl:when>
                                                <xsl:otherwise>
                                                    <xsl:value-of
select="translate($thenStatement,'.','/')"/>
                                                </xsl:otherwise>
                                            </xsl:choose>
```

```
                                    </xsl:attribute>
                                </xsl:when>
                                <xsl:otherwise>
                                    <xsl:call-template name="rule_tokenizer">
                                        <xsl:with-param name="logicalStatement"
select="$thenStatement"/>
                                    </xsl:call-template>
                                </xsl:otherwise>
                            </xsl:choose>
                        </xsl:element>
                    </xsl:element>
                    <xsl:if test="contains($logicalStatement,'else')">
                        <xsl:element name="else">
                            <xsl:variable name="elseStatement" select="translate(normalize-
space(substring-before(substring-after($logicalStatement,'else'),'endif')),'()',')"/>
                            <xsl:choose>
                                <xsl:when test="not(contains($elseStatement,'if'))">
                                    <xsl:attribute name="test">
                                        <xsl:choose>
                                            <xsl:when test="contains($elseStatement,'multiplicity')">
                                                <xsl:value-of
select="concat('count(',translate(substring-before($elseStatement,'.multiplicity'),'.','/'),')',substring-
after($elseStatement,'multiplicity'))"/>
                                            </xsl:when>
                                            <xsl:otherwise>
                                                <xsl:value-of
select="translate($elseStatement,'.','/')"/>
                                            </xsl:otherwise>
                                        </xsl:choose>
                                    </xsl:attribute>
                                </xsl:when>
                                <xsl:otherwise>
                                    <xsl:call-template name="rule_tokenizer">
                                        <xsl:with-param name="logicalStatement"
select="normalize-space(substring-after($logicalStatement,'else'))"/>
                                    </xsl:call-template>
                                </xsl:otherwise>
                            </xsl:choose>
                        </xsl:element>
                    </xsl:if>
                </xsl:when>
                <xsl:otherwise>
                    <message>Invalid constraint type!</message>
                </xsl:otherwise>
            </xsl:choose>
        </xsl:element>
    </xsl:element>
</xsl:template>
<xsl:template name="rule_tokenizer">
    <xsl:param name="logicalStatement"/>
    <xsl:element name="if">
        <xsl:variable name="ifStatement" select="translate(normalize-space(substring-
before(substring-after($logicalStatement,'if'),'then')),'()',')"/>
        <xsl:attribute name="test">
            <xsl:choose>
```

```
                    <xsl:when test="contains($ifStatement,'multiplicity')">
                        <xsl:value-of select="concat('count(',translate(substring-
before($ifStatement,'.multiplicity'),'.','/'),')',substring-after($ifStatement,'multiplicity'))"/>
                    </xsl:when>
                    <xsl:otherwise>
                        <xsl:value-of select="translate($ifStatement,'.','/')"/>
                    </xsl:otherwise>
                </xsl:choose>
            </xsl:attribute>
            <xsl:element name="then">
                <xsl:variable name="thenStatement" select="translate(substring-before(normalize-
space(substring-after($logicalStatement,'then')),')'),'()','')"/>
                <xsl:choose>
                    <xsl:when test="not(contains($thenStatement,'if'))">
                        <xsl:attribute name="test">
                            <xsl:choose>
                                <xsl:when test="contains($thenStatement,'multiplicity')">
                                    <xsl:value-of select="concat('count(',translate(substring-
before($thenStatement,'.multiplicity'),'.','/'),')',substring-after($thenStatement,'multiplicity'))"/>
                                </xsl:when>
                                <xsl:otherwise>
                                    <xsl:value-of select="translate($thenStatement,'.','/')"/>
                                </xsl:otherwise>
                            </xsl:choose>
                        </xsl:attribute>
                    </xsl:when>
                    <xsl:otherwise>
                        <xsl:call-template name="rule_tokenizer">
                            <xsl:with-param name="logicalStatement" select="normalize-
space(substring-after($logicalStatement,'else'))"/>
                        </xsl:call-template>
                    </xsl:otherwise>
                </xsl:choose>
            </xsl:element>
        </xsl:element>
        <xsl:if test="contains($logicalStatement,'else')">
            <xsl:element name="else">
                <xsl:variable name="elseStatement" select="normalize-space(substring-
before(substring-after($logicalStatement,'else'),'endif'))"/>
                <xsl:choose>
                    <xsl:when test="not(contains($elseStatement,'if'))">
                        <xsl:attribute name="test">
                            <xsl:choose>
                                <xsl:when test="contains($elseStatement,'multiplicity')">
                                    <xsl:value-of select="concat('count(',translate(substring-
before($elseStatement,'.multiplicity'),'.','/'),')',substring-after($elseStatement,'multiplicity'))"/>
                                </xsl:when>
                                <xsl:otherwise>
                                    <xsl:value-of select="translate($elseStatement,'.()','/')"/>
                                </xsl:otherwise>
                            </xsl:choose>
                        </xsl:attribute>
                    </xsl:when>
                    <xsl:otherwise>
                        <xsl:call-template name="rule_tokenizer">
```

```
                        <xsl:with-param name="logicalStatement" select="$elseStatement"/>
                    </xsl:call-template>
                </xsl:otherwise>
            </xsl:choose>
        </xsl:element>
    </xsl:if>
  </xsl:template>
</xsl:stylesheet>
```

Appendix E

XSLT Stylesheet for XSLT Stylesheet Generation

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:jhu="http://www.csis.pace.edu/jhu">
    <xsl:output method="xml" indent="yes"/>
    <!-- create the root element of the stylesheet -->
    <xsl:template match="/">
        <xsl:element name="xsl:stylesheet">
            <xsl:attribute name="version">1.0</xsl:attribute>
            <xsl:element name="xsl:output">
                <xsl:attribute name="indent">yes</xsl:attribute>
            </xsl:element>
            <xsl:apply-templates select="jhu:Constraints"/>
        </xsl:element>
    </xsl:template>
    <xsl:template match="jhu:Constraints">
        <xsl:element name="xsl:template">
            <xsl:attribute name="match">/</xsl:attribute>
            <xsl:element name="validation_result">
                <xsl:apply-templates select="Constraint" mode="select"/>
            </xsl:element>
        </xsl:element>
        <xsl:apply-templates select="Constraint" mode="match"/>
    </xsl:template>
    <xsl:template match="Constraint" mode="select">
        <xsl:apply-templates select="Rule" mode="select">
            <xsl:with-param name="parent_context" select="@context"/>
            <xsl:with-param name="position" select="position()"/>
        </xsl:apply-templates>
        <xsl:apply-templates select="Assertion" mode="select">
            <xsl:with-param name="parent_context" select="@context"/>
            <xsl:with-param name="position" select="position()"/>
        </xsl:apply-templates>
    </xsl:template>
    <xsl:template match="Constraint" mode="match">
        <xsl:apply-templates select="Rule" mode="match">
            <xsl:with-param name="parent_context" select="@context"/>
            <xsl:with-param name="position" select="position()"/>
        </xsl:apply-templates>
        <xsl:apply-templates select="Assertion" mode="match">
            <xsl:with-param name="parent_context" select="@context"/>
            <xsl:with-param name="position" select="position()"/>
        </xsl:apply-templates>
    </xsl:template>
```

```
    <xsl:template match="Rule" mode="select">
        <xsl:param name="parent_context"/>
        <xsl:param name="position"/>
        <xsl:comment>This is a rule type constraint.</xsl:comment>
        <xsl:variable name="local_context">
            <xsl:value-of select="@context"/>
            <xsl:if test="not(@context)">.</xsl:if>
        </xsl:variable>
        <xsl:element name="xsl:apply-templates">
            <xsl:if test="$local_context='.'">
                <xsl:attribute name="select"><xsl:value-of
select="$parent_context"/></xsl:attribute>
            </xsl:if>
            <xsl:if test="$local_context!='.'">
                <xsl:attribute name="select"><xsl:value-of
select="concat($parent_context,'/',$local_context)"/></xsl:attribute>
            </xsl:if>
            <xsl:attribute name="mode"><xsl:value-of
select="concat('constraint_',$position)"/></xsl:attribute>
        </xsl:element>
    </xsl:template>
    <xsl:template match="Assertion" mode="select">
        <xsl:param name="parent_context"/>
        <xsl:param name="position"/>
        <xsl:comment>This is an assertion type constraint. </xsl:comment>
        <xsl:variable name="local_context">
            <xsl:value-of select="@context"/>
            <xsl:if test="not(@context)">.</xsl:if>
        </xsl:variable>
        <xsl:element name="xsl:apply-templates">
            <xsl:if test="$local_context='.'">
                <xsl:attribute name="select"><xsl:value-of
select="$parent_context"/></xsl:attribute>
            </xsl:if>
            <xsl:if test="$local_context!='.'">
                <xsl:attribute name="select"><xsl:value-of
select="concat($parent_context,'/',$local_context)"/></xsl:attribute>
            </xsl:if>
            <xsl:attribute name="mode"><xsl:value-of
select="concat('constraint_',$position)"/></xsl:attribute>
        </xsl:element>
    </xsl:template>
    <xsl:template match="Rule" mode="match">
        <xsl:param name="parent_context"/>
        <xsl:param name="position"/>
        <xsl:comment>This is a rule type constraint.</xsl:comment>
        <xsl:variable name="local_context">
            <xsl:value-of select="@context"/>
            <xsl:if test="not(@context)">.</xsl:if>
        </xsl:variable>
        <xsl:element name="xsl:template">
            <xsl:if test="$local_context='.'">
                <xsl:attribute name="match"><xsl:value-of
select="$parent_context"/></xsl:attribute>
            </xsl:if>
```

```
            <xsl:if test="$local_context!='.'">
                <xsl:attribute name="match"><xsl:value-of
select="concat($parent_context,'/',$local_context)"/></xsl:attribute>
            </xsl:if>
            <xsl:attribute name="mode"><xsl:value-of
select="concat('constraint_',$position)"/></xsl:attribute>
            <xsl:choose>
                <xsl:when test="count(else)=0">
                    <xsl:apply-templates select="if" mode="if-then"/>
                </xsl:when>
                <xsl:when test="count(else)=1">
                    <xsl:element name="xsl:choose">
                        <xsl:apply-templates select="if" mode="if-then-else"/>
                        <xsl:element name="xsl:otherwise">
                            <xsl:choose>
                                <xsl:when test="count(else/if)=0">
                                    <xsl:apply-templates select="else" mode="simple"/>
                                </xsl:when>
                                <xsl:otherwise>
                                    <xsl:apply-templates select="else" mode="complex"/>
                                </xsl:otherwise>
                            </xsl:choose>
                        </xsl:element>
                    </xsl:element>
                </xsl:when>
                <xsl:otherwise>
                    <xsl:element name="xsl:choose">
                        <xsl:apply-templates select="if" mode="if-then-else"/>
                        <xsl:choose>
                            <xsl:when test="count(else/if)=0">
                                <xsl:element name="xsl:when">
                                    <xsl:apply-templates select="else" mode="simple"/>
                                </xsl:element>
                            </xsl:when>
                            <xsl:otherwise>
                                <xsl:apply-templates select="else" mode="complex"/>
                            </xsl:otherwise>
                        </xsl:choose>
                    </xsl:element>
                </xsl:otherwise>
            </xsl:choose>
        </xsl:element>
    </xsl:template>
    <xsl:template match="if" mode="if-then">
        <xsl:element name="xsl:if">
            <xsl:attribute name="test"><xsl:value-of select="@test"/></xsl:attribute>
            <xsl:choose>
                <xsl:when test="count(then/if)=0">
                    <xsl:apply-templates select="then" mode="simple"/>
                </xsl:when>
                <xsl:otherwise>
                    <xsl:apply-templates select="then" mode="complex"/>
                </xsl:otherwise>
            </xsl:choose>
        </xsl:element>
```

```
</xsl:template>
<xsl:template match="if" mode="if-then-else">
    <xsl:element name="xsl:when">
        <xsl:attribute name="test"><xsl:value-of select="@test"/></xsl:attribute>
        <xsl:choose>
            <xsl:when test="count(then/if)=0">
                <xsl:apply-templates select="then" mode="simple"/>
            </xsl:when>
            <xsl:otherwise>
                <xsl:apply-templates select="then" mode="complex"/>
            </xsl:otherwise>
        </xsl:choose>
    </xsl:element>
</xsl:template>
<xsl:template match="then" mode="simple">
    <xsl:call-template name="evaluate-expression"/>
</xsl:template>
<xsl:template match="then" mode="complex">
    <xsl:choose>
        <xsl:when test="count(else)=0">
            <xsl:apply-templates select="if" mode="if-then"/>
        </xsl:when>
        <xsl:when test="count(else)!=0">
            <xsl:element name="xsl:choose">
                <xsl:apply-templates select="if" mode="if-then-else"/>
                <xsl:choose>
                    <xsl:when test="count(else/if)=0">
                        <xsl:apply-templates select="else" mode="simple"/>
                    </xsl:when>
                    <xsl:when test="count(else/if)!=0">
                        <xsl:apply-templates select="else" mode="complex"/>
                    </xsl:when>
                </xsl:choose>
            </xsl:element>
        </xsl:when>
    </xsl:choose>
</xsl:template>
<xsl:template match="else" mode="simple">
    <xsl:call-template name="evaluate-expression"/>
</xsl:template>
<xsl:template match="else" mode="complex">
    <xsl:element name="xsl:choose">
        <xsl:apply-templates select="if" mode="if-then-else"/>
        <xsl:choose>
            <xsl:when test="count(else/if)=0">
                <xsl:apply-templates select="else" mode="simple"/>
            </xsl:when>
            <xsl:otherwise>
                <xsl:apply-templates select="else" mode="complex"/>
            </xsl:otherwise>
        </xsl:choose>
    </xsl:element>
</xsl:template>
<xsl:template name="evaluate-expression">
    <xsl:element name="xsl:choose">
```

```
            <xsl:element name="xsl:when">
                <xsl:attribute name="test"><xsl:value-of select="@test"/></xsl:attribute>
                <xsl:element name="message">{<xsl:value-of select="@test"/>} is
asserted!</xsl:element>
            </xsl:element>
            <xsl:element name="xsl:otherwise">
                <xsl:element name="message">The assertion {<xsl:value-of select="@test"/>} is
failed!</xsl:element>
            </xsl:element>
        </xsl:element>
    </xsl:template>
    <xsl:template match="Assertion" mode="match">
        <xsl:param name="parent_context"/>
        <xsl:param name="position"/>
        <xsl:comment>This is an assertion type constraint. </xsl:comment>
        <xsl:variable name="local_context">
            <xsl:value-of select="@context"/>
            <xsl:if test="not(@context)">.</xsl:if>
        </xsl:variable>
        <xsl:element name="xsl:template">
            <xsl:if test="$local_context='.'">
                <xsl:attribute name="match"><xsl:value-of
select="$parent_context"/></xsl:attribute>
            </xsl:if>
            <xsl:if test="$local_context!='.'">
                <xsl:attribute name="match"><xsl:value-of
select="concat($parent_context,'/',$local_context)"/></xsl:attribute>
            </xsl:if>
            <xsl:attribute name="mode"><xsl:value-of
select="concat('constraint_',$position)"/></xsl:attribute>
            <xsl:element name="xsl:choose">
                <xsl:element name="xsl:when">
                    <xsl:attribute name="test"><xsl:value-of select="@test"/></xsl:attribute>
                    <xsl:element name="message">{<xsl:value-of select="@test"/>} is
asserted!</xsl:element>
                </xsl:element>
                <xsl:element name="xsl:otherwise">
                    <xsl:element name="message">{<xsl:value-of select="test"/>} is not
asserted!</xsl:element>
                </xsl:element>
            </xsl:element>
        </xsl:element>
    </xsl:template>
</xsl:stylesheet>
```