University of Twente
department of computer science

Mapping UML Domain Models onto XML Schemas using UML Profiles

M.Sc. Telematics Final Project

Linda Ariani Gunawan

Graduation Committee:
Prof. Dr. Ir. Mehmet Aksit
Dr. Ir. Klaas van den Berg
Ivan Kurtev Ivanov, M.Sc.

Software Engineering Group
Department of Computer Science
University of Twente
The Netherlands

June 2003

Abstract

XML (eXtensible Markup Language) is expected to be a standard method of data interchange format for the Internet. XML Schema, which describes the structure of XML data, plays an important role in validation of data. One suitable way to design an XML Schema is by using a UML domain model. Therefore, a mechanism to map UML models onto XML Schema is needed.

A UML Model can be mapped into several valid XML Schemas. In other words, there are alternative mappings. The purpose of this project is to identify those alternatives, find a way to represent the chosen alternatives, and transform the UML model together with the chosen alternatives into XML Schema.

In this project, the alternatives are identified and organized as two steps of the mapping process. The first step is from UML model into an XML Schema skeleton. The second step is refinement XML Schema skeleton into an actual Schema, for which again there are alternatives.

Alternatives can be represented using a UML Profile, an extension mechanism of UML. This profile and the model will be transformed into XML Schema. In this project, the transformation is based on XMI serialization and XSLT. XMI serialization converts UML model and Profile into an XML document. This XML document is then transformed into XML Schema using XSLT stylesheet.

The mapping mechanism is implemented using the tools: Rational Rose, Unisys Rational Tool and XML Spy. The XSLT files also created. As a case study, Telecommunication Service Access and Subscription (TSAS) UML model is used. This shows the applicability of the approach and that the implementation generates the required schema.

It is concluded that the project has identified alternative mappings from UML model onto XML Schema. It is shown that UML Profiles can be used as a method to represent the alternatives. Furthermore, the mechanism of transformation using XMI serialization and XSLT is feasible as shown in a case study.

Abbreviations

DTD	: Document Type Definition
HTML	: HyperText Markup Language
ISO	: International Organization for Standarization
MOF	: Meta Object Facility
OMG	: Object Modeling Group
RFP	: Request For Proposal
SGML	: Standard Generalized Markup Language
TSAS	: Telecommunication Service Access and Subscription
UML	: Unified Modeling Language
W3C	: World Wide Web Consortium
XMI	: XML Metadata Interchange
XML	: eXtensible Markup Language
XPath	: XML Path Language
XSLT	: eXtensible Sytlesheet Language Transformation

Table of Content

1 Introduction

This chapter contains an introduction to the project. First, the background of the project is explained in section 1.1. Following that, problem statement is formulated in section 1.2. Section 1.3 contains the objectives of this project. Then, solution approach is explained in section 1.4. The last section of this chapter contains an overview of this report.

1.1 Background

XML (eXtensible Markup Language) is a technology to describe data in a structured and semi-structured manner. This technology can be used to store data, exchange data, or create other languages. Moreover, XML is expected to be a standard method of data interchange format over the Internet. In order the data interchange to be successful, the structure of XML document (what element, attribute, etc should an XML document contains) must be preserved. The structure is described in XML Schema. It is needed in the sender and receiver of data interchange to ensure the validity of the data. Thus, XML Schema plays an important role in data interchange over the Internet.

An XML document structure is described using a schema definition language. XML Schema is a text-based schema. Creating text-based XML Schema manually is error prone, especially for complex schemas. Moreover, it is not convenient. Nowadays, graphical modeling is preferable. There are several tree-based graphical XML Schema editors, for example XML Spy, Biztalk editor. On one side, these editors have an advantage, i.e. graphical view. But, on the other hand, they also have a week point, i.e. their limitation to a strict hierarchical view.

Another way to design XML Schema is based on conceptual models of the application domain which uses XML. Conceptual models give a good understanding of the application domain [7]. A conceptual model can be expressed as a UML model (UML is the Unified Modeling Language).

UML is a standard graphical modeling language from OMG (Object Modeling Group) to specify, visualize, construct, and document a system. This language is widely used. UML is a flexible and powerful modeling language. It provides an extension mechanism to handle domain specific applications.

Thus, XML Schemas may be derived from UML models. In order to use this, a method to transform a UML Model into XML Schema is needed.

1.2 Problem Statement

The goal of this project is to transform a UML model into XML Schema. The problem is how this transformation is done. This problem is depicted in the figure 1.1.

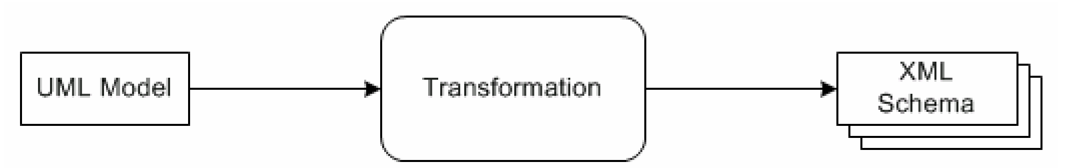


Figure 1.1 Transforming UML Model into XML Schema

As seen in the figure above, a UML model may be transformed into several valid XML Schemas. There are alternatives or choices. This is the case because one UML Model component may be transformed into several XML Schema constructs. The resulting XML Schemas may be used to validate XML documents. The production of these XML documents is out of the scope of this project.

1.3 Objectives

The objectives of this project based on the problem statement above are:

1. To identify the alternative mappings from UML Model onto XML Schema. Moreover, to choose among the alternatives based on constraints and other considerations.
2. To represent the alternative mappings of UML Model onto XML Schema.
3. To design a mechanism to do the transformation and to implement it.

1.4 Solution Approach

To solve the problem stated above, the following approach is used.

1. Literature study

   Articles, books, etc. about UML, XML, and the mappings are studied in order to obtain necessary theoretical background.

2. Analysis

   At this step, investigation on the alternative mappings is done and organized systematically. Moreover, techniques to express the mapping alternatives are also analyzed.

3. Design

   A Mechanism to perform the mapping of UML Model onto XML Schemas with knowledge of alternative mappings is designed.

4. Implementation

   At this last step, the design is implemented using standard technologies. Furthermore, a case study is taken to show how the design and implementation solve the problem.

1.5 Outline

This report is structured as follows.

Chapter 2 contains the basic concepts for this project, i.e. XML, UML, Modeling XML applications with UML, and XMI.

Chapter 3 contains mapping from UML model into XML schema. A discussion about the relation of the mapping to the OMG four-layer modeling architecture is given. Moreover, two steps of alternatives generation are discussed. The second step of alternative generation for each UML component is explained in detail.

Chapter 4 contains the representation of alternatives generation and how the transformation is done.

Chapter 5 contains the implementation of the solution described in the previous chapters. Tools that are needed are explained. How to add UML Profile in a class diagram in the tools and XMI serialization are also presented.

Chapter 6 contains a case study. It is a UML class diagram about Telecommunication Services Access and Subscription. UML Profile is added to this diagram. Then the class diagram together with the profile is transformed into an XML Schema using the implementation in chapter 5.

Chapter 7 contains the summary of the project, the comparison with existing work, and the suggestion to future research.

2 Concepts

This chapter contains the basics concepts for this project. In section 2.1 and 2.2 basic concept of XML and UML are introduced respectively. Modeling XML application with UML by David Carlson is presented in section 2.3. In the next section, XMI is explained. Last section gives a summary of this chapter.

2.1 XML (eXtensible Markup Language)

eXtensible Markup Language (XML) is a Meta language which is designed to describe data in a structured manner. This is a subset of SGML (Standard Generalized Markup Language). SGML is a meta language that creates markup language. It is a standard from ISO (International Organization for Standarization) number 8879:1986.

W3C (World Wide Web Consortium) organization defines the specification of XML version 1.0 [16]. According to the specification, an XML document contains of elements and attributes. It has a tree structure with one element as the root element. The more detail syntax of XML is explained below.

2.1.1 XML Syntax

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- comment -->


<company>
    <name>ABC</name>
    <address>
        <street>ABCstreet 5</street>
        <city>ABCcity</city>
        <postcode>12345</postcode>
    </address>
    <employees>
        <employee id="e1">
            <name>aaa</name>
        </employee>
        <employee id="e2">
            <name>bbb</name>
        </employee>
    </employees>
</company>
```

The syntax rules of XML are best described using an example. Above is an example of an XML document. This is an XML document about a company with name, address, and employees.

The syntax rules of XML are: [16]

- An XML document may contain an XML declaration
  XML declaration is used to identify the version of XML and the encoding type. An XML declaration starts with <? and ends with ?>. The first line of the example above is an XML declaration.
- An XML document may contain XML comments
  An XML comment is always starts with <!-- and end with -->. XML comments will not be parsed (see part 2.1.2 about parsing). The second line of the example above is an XML comment.
- An XML document contains one or more elements
  Element is everything between its start tag and its end tag. Tags are keywords contained in a pair of angle braces (< >). Tag <name> is a start tag and </name> is an end tag. In the example above there are 11 elements. An example of an element is <name>ABC</name>.

  An element may contain other elements. The relations between those elements are parent and child. Elements that have the same parent are called siblings. In the example above, <address> is the parent element of three child elements, i.e. <street>, <city>, and <postcode>. <street>, <city> and <postcode> are siblings.
- All XML elements must be properly nested
  <a><b>c</a></b> is an error. Element b must end before element a.
- An XML document must contain exactly one root element
  In the example above, <company> … </company> is the root element.
- An XML element may have attributes
  The attributes are in pairs of name and value. It is located in the start tag. In the example above the first employee element has an attribute id, which has value 1.

2.1.2 XML Validation

An XML document can be processed by a software program called XML parser or XML processor. XML parser parses or reads XML document, checks its syntax and validates it. This introduces two important terms concerning XML document, i.e. well-formed and valid.

- well-formed

  An XML document is well-formed if it conforms to XML syntax rules.
- valid

  An XML document is valid if it conforms to schemas, such as a DTD (Document Type Definition) or a XML Schema. DTD and XML Schema define the structure of an XML document. They specify what elements, attributes, etc. are allowed. DTD and XML Schema will be explained below.

2.1.3 DTD and XML Schema

As mentioned above, DTD (Document Type Definition) and XML Schema define the structure of an XML document. A DTD of the XML document in the example above is as follows.

```
<?xml version="1.0" encoding="UTF-8"?>

<!ELEMENT company (name, address, employees)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT address (street, city, postcode)>
<!ELEMENT street (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT postcode (#PCDATA)>
<!ELEMENT employees (employee+)>
<!ATTLIST employee id ID #REQUIRED>
<!ELEMENT employee (name)>
```

The DTD above shows that the root element of the XML document is <company>. The root element must have exactly three elements, subsequently element `<name>`, `<address>`, and `<employees>`. Element `<address>` contains other elements, i.e. `<street>`, `<city>`, and `<postcode>`. Element `<employees>` contains one or more `<employee>` elements, as indicated with + sign. This sign is occurrence indicator. There are other indicators, i.e. * means the element appears zero or more times, and ? means the element appears zero or one time. The `<employee>` element contains element `<name>`. This element must have one attribute id. Elements `<name>`, `<street>`, `<city>`, `<postcode>` is

PCDATA, a parsable character data (character data that can be parsed by XML parser).

DTD's have several limitations. It is not an XML document. Thus, it is difficult to be manipulated, e.g. transformed into other documents. Also, in DTD we cannot specify that an element can only contain certain character. For example, in the example of XML document above element postcode should only contain numbers. DTD cannot express this constraint. XML Schema overcomes these limitations.

XML Schema that will be used in this project is the one recommended by W3C [19] [20]. An XML Schema of the XML document in the example above is as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xs:element name="company" type="company"/>
    <xs:complexType name="company">
        <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="address" type="address"/>
            <xs:element name="employees" type="employees"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="address">
        <xs:sequence>
            <xs:element name="street" type="xs:string"/>
            <xs:element name="city" type="xs:string"/>
            <xs:element name="postcode" type="xs:integer"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="employees">
        <xs:sequence>
            <xs:element name="employee" type="employee" minOccurs="1" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="employee">
        <xs:sequence>
            <xs:element name="name" type="xs:string"/>
        </xs:sequence>
        <xs:attribute name="id" type="xs:string"/>
    </xs:complexType>
</xs:schema>
```

The structure of the XML document in this XML Schema is similar to the document's DTD. One difference is in the postcode element. In DTD this element may contain character, but in XML Schema, it can only contain an integer.

2.1.4 XML Namespaces

In XML Element names are user-defined. This may cause name conflicts. Some documents use the same element names, but they describe different things. This problem is solved using XML namespaces. A namespace is declared using `xmlns` attribute in the start tag of an element. Below is XML document with namespace.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- comment -->

<company xmlns="aaa:bbb:ccc:x"
           xmlns:c = "aaa:bbb:ccc:y">
    <c:name>ABC</c:name>
    <c:address>
        <c:street>ABCstreet 5</c:street>
        <c:city>ABCcity</c:city>
        <c:postcode>12345</c:postcode>
    </c:address>
    <employees>
        <employee id="e1">
            <name>aaa</name>
        </employee>
        <employee id="e2">
            <name>bbb</name>
        </employee>
    </employees>
</company>
```

In the example above, there are two namespace names. The first one is the default namespace, i.e. `aaa:bbb:ccc:x`. The second one is `aaa:bbb:ccc:y`. This second namespace is bounded to namespace prefix `c`. This namespace prefix associates element `<name>`, `<address>`, `<street>`, `<city>` and `<postcode>` with the second namespace.

2.1.5 XPath and XSLT

XPath (XML Path Language) is a W3C recommendation [18]. It is a language for expression that locates a part of an XML document. This language is used by XSLT (eXtensible Sytlesheet Language Transformation). In XPath, an XML document is modeled as a tree. The nodes of the tree are parts of XML document. Thus, XPath will traverse the tree to find the specific node. An example of XPath expression that selects the name of the first employee of above XML document is `/company/employees/employee[@id='e1']/name`.

XSLT (eXtensible Stylesheet Language Transformation) is also a recommendation from W3C [21]. It is a language to transform an XML document to another document. The resulting document may be another XML document, plain text, HTML, or other text-based document. This language uses XPath to select nodes from the XML document. An example of XSLT that transforms the XML document in section 2.1.1 into HTML is as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
    <xsl:template match="/company">
        <html>
            <head>
                <title>Company <xsl:value-of select="name"/></title>
            </head>
            <body bgcolor="white">
                <h1>Company <xsl:value-of select="name"/></h1>
                <br/>
                <h3>Contact Address:</h3>
                <xsl:value-of select="address/street"/>
                <br/>
                <xsl:value-of select="address/city"/>-<xsl:value-of select="address/postcode"/>
                <br/>
                <h3>Employees </h3>
                <table border="1">
                    <thead>
                        <tr>
                            <th>Id Number</th>
                            <th>Name</th>
                        </tr>
                    </thead>
                    <tbody>
                        <xsl:for-each select="employees/employee">
                        <tr>
                            <th><xsl:value-of select="@id"/></th>
                            <th><xsl:value-of select="name"/></th>
                        </tr>
                        </xsl:for-each>
                    </tbody>
                </table>
            </body>
        </html>
    </xsl:template>
</xsl:stylesheet>
```

2.2 UML (Unified Modeling Language)

Unified Modeling Language (UML) is a graphical language for modeling software systems. In this project, a subset of this language will be used to express a conceptual model.

2.2.1 UML Class Diagram

In UML, there are nine types of diagrams used to visualize a system from different perspectives. They are class diagram, object diagram, use case diagram, sequence diagram, collaboration diagram, statechart diagram, activity diagram, component diagram, and deployment diagram. Only class diagrams will be used in this project.

A class diagram contains a set of classes, interfaces, collaborations and their relationships. Figure 2.1 shows an example of a class diagram. This diagram is a modified version of *subscription information model of Telecommunication Service Access and Subscription (TSAS)* [13]. This model will be used in the case study (chapter 6). The components of class diagram that will be used in this project are described below.

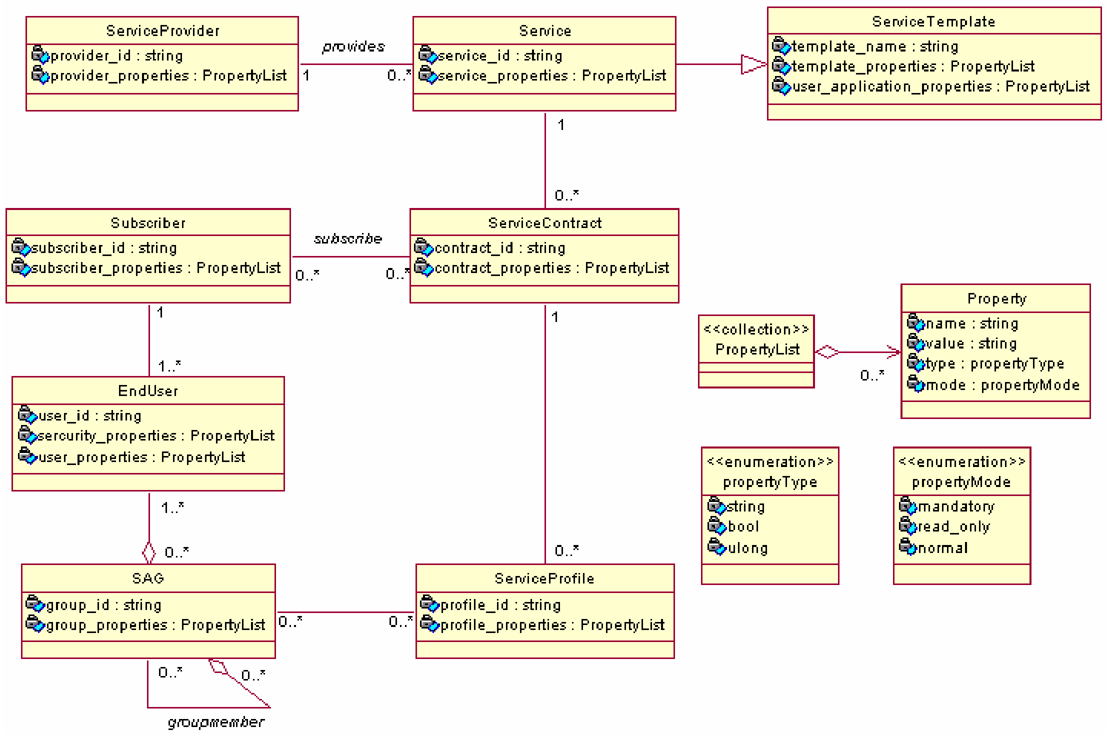


Figure 2.1 UML class diagram

2.2.1.1 Class

A class is a description of a set of objects that share the same attributes, operation, relationships, and semantics. In UML, it is represented as a rectangle. An example of a class is depicted in the figure 2.2 below.

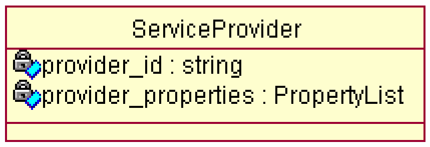


Figure 2.2 A UML class

A UML class has a name, attributes and operations. Operation will not be used in this project because operation specifies the behavior of the class (Behavior is not expressed in an XML Schema but usually in programming languages such as Java, C++). A Class has two properties as follows.

- Name

  A class must have a unique name. The name is textual string. In the example, the name of the class is `ServiceProvider`.

- Attribute

  A class may have any number of attributes or none. Attribute describes the property of the class. An attribute can have multiplicity, type, and initial value. The syntax for it is: `attribute's_name [multiplicity] [:type] [= initial_value]`. Multiplicity may be omitted, if it is exactly one. Attribute's type can be a primitive data type (such as string, integer, Boolean) or it can be user-defined one.

  Class `ServiceProvider` has two attributes, i.e. `provider_id` which has type `string` and `provider_properties` which has type `PropertyList`. Attribute `address[1..*]:string` has type string and multiplicity one or more. Attribute `code: integer = 1` has type integer and initial value 1.

2.2.1.2 Relationship

Relationship shows the collaboration between classes in UML. An example of relationship is depicted in figure 2.3.

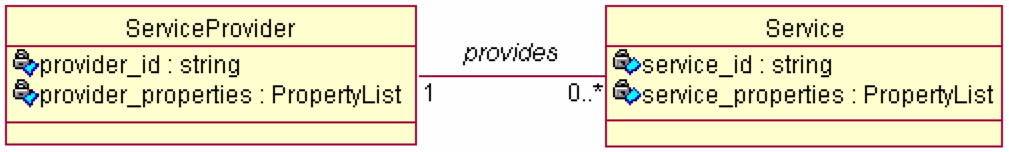


Figure 2.3 Relationship

There are four types of relationships in UML, i.e. association, generalization, dependency, and realization. Only the first two will be used in this project.

- Association

  Association specifies a bidirectional connection between classes in the model. It is depicted as a solid line. It is possible to have one directional connection between classes by placing an arrow head. In figure 2.3, the line represents a connection between class `Provider` and class `Service`. There are several properties related to association, i.e.:

  - Name

    An association may have a name that describes the relationship. In the example, the name of the association is `provides`.

  - Role

    Role shows the role of a class in the association. It is represented at the end of an association line near the class that has the role. In figure 2.3, there is no role name in the association.

  - Multiplicity

    Multiplicity shows how many instances of a class may be connected across an instance of an association. It is expressed as a range of value. For example, one or more (1..*), zero or more (0..*), exactly one (1), zero or one (0..1), etc.

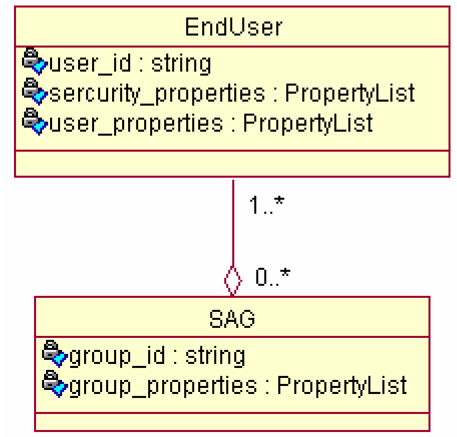

EndUser
user_id : string
sercurity_properties : PropertyList
user_properties : PropertyList
1..*
0..*
SAG
group_id : string
group_properties : PropertyList

Figure 2.4 Aggregation

  - Aggregation

    Aggregation is a special type of association, which represents “has-a” relationship between classes. In UML class diagram it is shown as a line with a diamond end. Figure 2.4 shows an example of aggregation. The aggregation means SAG has one or more EndUser.

- Generalization

  Generalization is a relationship between a general class (super or parent class) with a more specific class (sub or child class). A child class inherits the attributes and operations of the parent. This relationship is represented as a solid directed line with a large open arrowhead. Figure 2.5 shows an example of generalization with specialized class Service as the child class while ServiceTemplate class as the parent class.

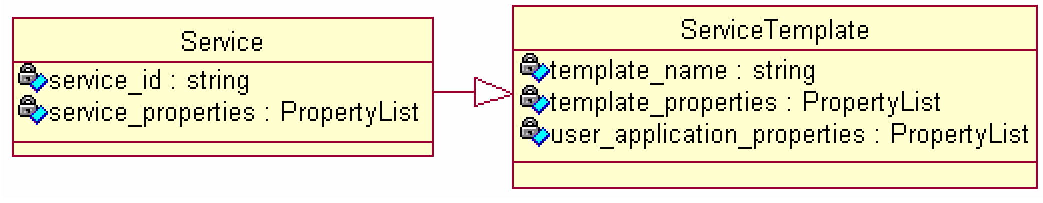

Service
service_id : string
service_properties : PropertyList
ServiceTemplate
template_name : string
template_properties : PropertyList
user_application_properties : PropertyList

Figure 2.5 Generalization

2.2.2 UML Profile

Profile is a specification that specializes one or several standard metamodels [2]. UML Profile is profile for UML. UML Profile is used to customize UML to meet specific needs. For example when transforming a UML class into XML Schema, it is required to specify that some attributes of the class are generated into child elements and some other into attributes. UML Profile can be used for this.

UML profile uses UML extension mechanisms to extend the standard language in controlled way. The figure 2.6 shows an example of UML extensibility mechanism. There are three extension mechanisms, i.e.:

- Stereotype

  A stereotype extends the vocabulary of UML by creating a new building block with its own special properties and semantics. It is represented as a name enclosed by guillemets (<<name>>) and placed before a class name or an attribute name or an association. From the figure above, attribute provider_id has a stereotype XSDattribute and provider_properties has a stereotype XSDelement.

- Tagged Value

  Tagged value extends the properties of a UML element (class, generalization, association, etc) by creating new information in the element's specification. It is represented as a name/value string enclosed with the braces: `{tagName=value}`. It is attached below a name of the element.

- Constraint

  Constraint extends the semantics of an UML element by adding new rules or modifying existing ones. It is represented as a string enclosed with the braces: `{string}`.

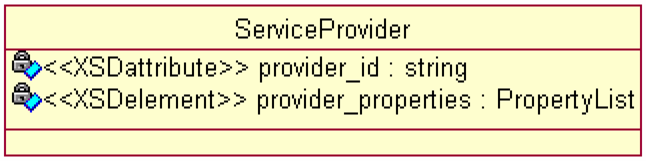

ServiceProvider
<<XSDattribute>> provider_id : string
<<XSDelement>> provider_properties : PropertyList

Figure 2.6 UML Extension mechanisms

2.3 Modeling XML Applications with UML

This section contains a description about Carlson's technique of modeling XML applications with UML. This is taken from Carlson's book [3]. The book describes a mechanism for mapping UML to Schemas.

2.3.1 Common Principles

Carlson introduces common principles of schema generation. These principles define the rules mapping of UML Model into a schema. He categorized the mapping rules into nine groups. They are as follows:

- Namespace mapping

  defines the mapping of namespace, one namespace for an entire UML model or a separate namespace for each UML package.

- Element name uniqueness

  Using namespace mapping and the generated schema language, this criterion ensures the uniqueness of generated XML element names.

- Elements or attributes

  defines that a UML attribute of a class should be mapped into an XML element or XML attribute.

- Multiplicity constraints
  defines the mapping of multiplicity of a UML attribute and association role in UML.
- Generalization
  defines the mapping of inheritance of a UML class.
- Content Model
  defines the XML content model to be used to control the presence of elements and text (empty, textOnly, or elementOnly) and element group structures (sequence, choice, or all) in an XML element definition.
- Element Order
  defines the mapping of unordered UML attributes and association roles into XML element content models.
- Data types
  defines the mapping of UML data types to XML element and attribute data types.
- Linking
  This defines the mapping of UML association to XML links within and between documents.

2.3.2 Generating XML Schema

Carlson introduces a pre-standard prototype of mapping from a UML Model into an XML Schema. He defines rules of so called relaxed and strict XML Schema ([3], chapter 9)

Generating Relaxed XML Schema

General characteristics of relaxed XML Schema are:

- Minimum occurrence for all schema elements is zero
- Content model is unordered
- Choice between elements or attributes is not restricted when mapping UML to XML structure

The rules for generating relaxed XML Schema are:

- Namespace mapping
  An entire UML models is represented in a single schema. Any document instance is free to assign a namespace prefix for the elements.

A simple XML container element is generated for each package that may contain that package's classes and sub packages as child elements.

- Element name uniqueness

  Prefix each UML attribute and association role with the class name followed by a '.'.

  If it is not unique, add additional prefix of the UML package name that contain the class. Reapply this if necessary.

- Elements or attributes

  Generate an XML element for each UML attribute and each association role.

  Generate an XML attribute for each UML attribute having a primitive data type with maximum multiplicity equal to one.

  Generate an XML attribute for each association role that is not part of a composition association.

- Multiplicity constraints

  All element multiplicity has `minOccurs="0"`, and the `maxOccurs` constraint is set to the value specified in the UML model.

- Generalization

  Generalization from a single super class generates `complexType` with an extension child element whose base attribute is assigned to the super class name.

  For generalization from multiple super classes, use copy-down inheritance to reproduce all superclass' attributes and association roles in each subclass.

- Content model

  Each UML class is generated as a `complexType` definition using the `<all>` unordered model group.

- Element order

  Unordered content is allowed by `<all>`, but elements are still constrained by the UML multiplicity.

- Data types

  Each UML attribute is generated as an XML attribute declaration, where the XML attribute type is assigned equal to the corresponding UML attribute type.

  If the UML attribute type is not equal to a class defined in the current model, and is not one of the built-in data types from XML Schema, then define a `simpleType` derived from string whose name is equal to the UML type name.

- Linking

  Use ID and IDREF linking within a document.

  Use href attribute for linking to other resources.

Generating Strict XML Schema

General characteristics of a strict XML Schema are:

- Multiplicity is restricted according to UML model
- Content model is unordered
- Only XML elements are generated from UML class' attributes, not both elements and attributes

Most rules of generating strict XML Schema are the same as relaxed XML Schema, except the following:

- Elements or attributes

  Generate an XML element for each UML attribute and each association role.
- Multiplicity constraints

  The minOccurs and maxOccurs attributes on all element declarations are assigned values equal to those specified in the UML model.

2.3.3 UML Profile

Carlson defines an UML Profile for XML ([3] appendix C). It is summarized in the following table.

Table 2.1 UML Profile for XML

Stereotype	***UML construct***	***Tagged Values***

XSDschema	Package
	Component
		targetNamespace
		targetNamespacePrefix
		Version
		elementFormDefault(qualified|unqualified)
		attributeFormDefault(qualified|unqualified)
		modelGroup(all|sequence|choice)
		attributeMapping(element|attribute|both)
		roleMapping(element|attribute|both)
		memberNames(qualified|unqualified)
		anonymousType(true|false)
		anonlymousRole(true|false)
		elementDerivation(true|false)
		relaxedMultiplicity(true|false)
		xmiCompliant(true|false)

XSDcomplexType	Class	mixed(true|false)
		modelGroup(all|sequence|choice)
		attributeMapping(element|attribute|both)
		roleMapping(element|attribute|both)
		memberNames(qualified|unqualified)

XSDsimpleType	Class	Derivation(restriction|list|union) Pattern Length minLength maxLength Duration Period Encoding
Enumeration	Class	Default
XSDsequence	Class	
XSDchoice	Class	

SimpleXLink	Class	Role
		Arcrole
		show(new|replace|embed|other|none)
		actuate(onload|onRequest|other|none)

ExtendedXLink	Association	
XSDrestriction	Generalization	

XSDelement	Attribute
	AssociationEnd
		form(qualified|unqualified)
		Position
		anonymousType(true|false)
		anonymousRole(true|false)

XSDattribute	Attribute
	AssociationEnd
		form(qualified|unqualified)
		use(prohibited|optional|required|default|fixed)

XSDtopLevelElement	Component	
XSDany	Class Attribute	Namespace processContents(skip|lax|strict)

2.4 XMI (XML Metadata Interchange)

XMI is one of OMG (Object Management Group) Specification. The main purpose of XMI is to enable easy interchange of metadata between modeling tools (based on the OMG-UML) and metadata repositories (OMG-MOF based) in distributed heterogeneous environments. XMI integrates three key industry standards: [15]

1. XML – eXtensible Markup Language, a W3C standard
2. UML – Unified Modeling Language, an OMG modeling standard
3. MOF – Meta Object Facility, an OMG metamodeling and metadata repository standard (this is not used in this project)

Thus, using XMI, a UML class diagram made with a UML tool (for example Rational Rose) can be exchanged with another UML tool (for example ArgoUML). For this exchange, XMI uses XML as the interchange format.

OMG issues the XMI specification that specifies XML DTD production rules and XML document production rules. But up to now, the XMI production of XML Schema is still an RFP (Request For Proposal) document. XML document production rules is defined as a set of rules, which when applied to a model or model elements, produces an XML document [15].

2.5 Summary

This chapter has given basic concepts needed for the project. First, it explained about XML, including XML Schema and XSLT. After that, a subset of UML is described. Some components of UML Class Diagram that are used in this project are class, attribute, association, aggregation and generalization. Moreover, UML Profile which uses UML Extension Mechanisms is explained. The mechanisms are by using stereotype, tagged value, and constraint.

Next section gives a description about Carlson's technique of modeling XML applications with UML. He defines nine common principles and rules for relaxed and strict XML Schema generation. He also defines a UML Profile. Fourth section explains about XMI which will be used in the method to transform UML class diagram into XML Schema.

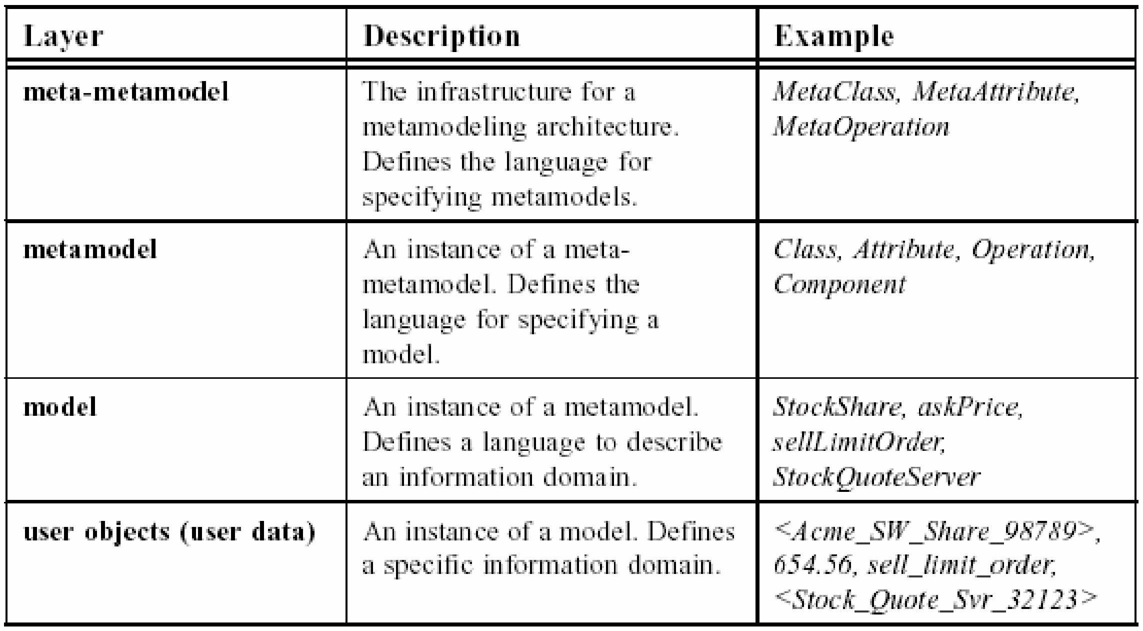
3 Mapping UML Model to XML Schema

This chapter contains the description about the mapping from UML model into XML Schema. Section 3.1 gives a description about four-layer modeling architecture and about this architecture related to the mapping. Section 3.2 introduces two steps of alternatives generation. Alternatives generation step 2 for UML Class and UML Attribute are discussed in Section 3.3. Then, section 3.4 and 3.5 also discuss about alternatives generation step 2 for UML Association and UML Generalization. The last section gives a summary of this chapter.

3.1 Four-Layer Metamodeling Architecture

The Four-Layer Metamodeling Architecture consists of four layers. Those layers are explained in the table 3.1 ([14] page 2-4).

Table 3.1 Four Layer Metamodeling Architecture



Layer	Description	Example
meta-metamodel	The infrastructure for a metamodeling architecture. Defines the language for specifying metamodels.	*MetaClass, MetaAttribute, MetaOperation*
metamodel	An instance of a meta-metamodel. Defines the language for specifying a model.	*Class, Attribute, Operation, Component*
model	An instance of a metamodel. Defines a language to describe an information domain.	*StockShare, askPrice, sellLimitOrder, StockQuoteServer*
user objects (user data)	An instance of a model. Defines a specific information domain.	*<Acme_SW_Share_98789>, 654.56, sell_limit_order, <Stock_Quote_Svr_32123>*

From the table, we conclude that one layer defines one layer lower and is an instance of one layer higher. For example, metamodel defines model, which is one layer below. It is also an instance of meta-metamodel, which is one layer higher. Terms, such as data, metadata, etc., are MOF (Meta Object Facitlity) terms. Those terms have correlation with the four layers in the table 3.1. This can be seen in the table 3.2 ([15] page 1-3).

Table 3.2 OMG Metadata Architecture

Meta-level	MOF terms	Examples
M3	meta-metamodel	The "MOF Model"
M2	meta-metadata metamodel	UML Metamodel, CWMI Metamodel(s), etc.
M1	metadata model	UML Models, Warehouse Schemas, etc.
M0	data	Modeled systems, Warehouse databases, etc.

The generation of XML Schema from a UML model related to four-layer modeling architecture can be depicted in the figure 3.1.

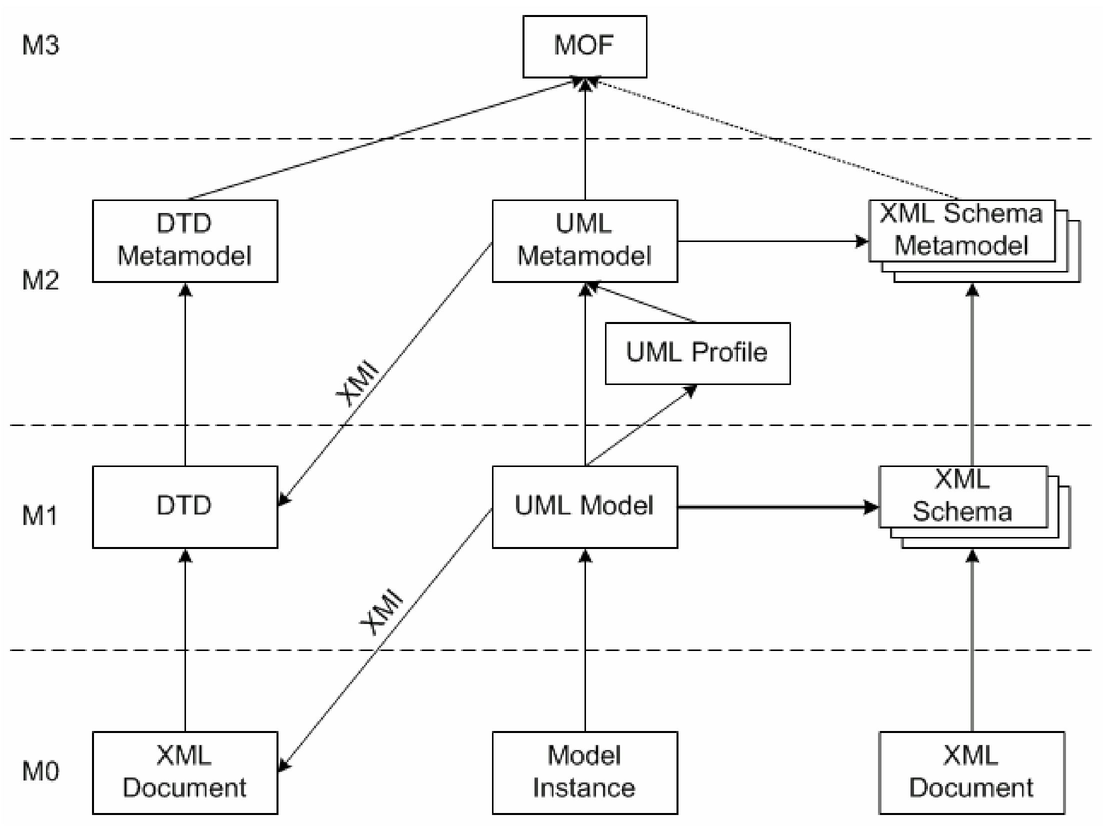


Figure 3.1 Mapping UML to XML Schema related to Four-Layer Modeling Architecture

As can be seen from the figure and also from table 3.2, MOF is in layer M3 (layer meta-metamodel). UML Metamodel, XML Schema Metamodel and DTD Metamodel are in layer M2 (layer metamodel). UML Profile is also in layer M2. In layer M1 (layer model), there are UML Model, XML Schema, and DTD. The last, in one layer M0 (layer data or object), there are Model instance and XML document.

UML Metamodel and DTD Metamodel are instances of MOF. XML Schema Metamodel currently is not defined as MOF instance (shown in dotted arrow). OMG has not standardized this. UML Model is an instance of UML Metamodel and Model instance is an instance of UML model. XML Schema is also instance of XML Schema Metamodel, so does DTD from DTD Metamodel. XML Document can be validated using XML Schema or DTD.

The generation from UML model into XML Schema is located in the layer M1. In the figure it is depicted as an arrow from UML model into XML Schema. This project uses UML Metamodel and XML Schema Metamodel in layer M2 to do the mapping. The two alternatives generation methods, i.e. Design Algebra and Feature Diagram are shown in the arrow from UML Metamodel to XML Schema Metamodel. The result of Feature Diagram is UML Profile which is also located in layer M2. XMI that is also used in implementation generates DTD in layer M1 from UML Metamodel in layer M2. It can also generate XML Document in layer M0 from UML Model in layer M1.

3.2 Two Steps of Alternatives Generation

As can be seen in figure 3.1, a UML Model can be mapped into several XML Schemas. This introduces alternatives. The alternatives can be identified from UML Metamodel and XML Schema Metamodel in layer M2. The identification of alternatives are organized in two steps, i.e. AG1 (Alternatives Generation Step 1) is called schema skeleton generation and AG2 (Alternatives Generation Step2) is called schema skeleton refinement. Those two steps will be explained further below. Figure 3.2 depicts the two steps of alternatives generation for mapping UML Model into XML Schema.

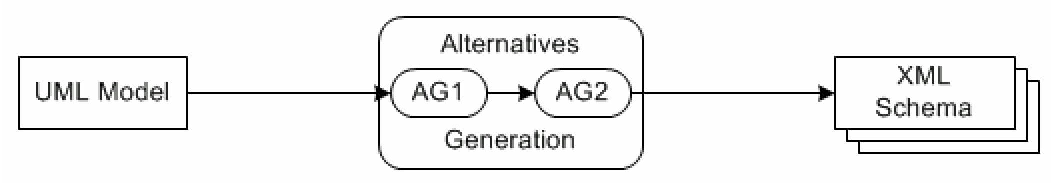


Figure 3.2 Two Steps of Alternatives Gene rations

The two steps of mapping of UML Model into XML Schema can also be expressed by using set and operation as follows.

M_{UML} ® XML Schema$_{skeleton}$ ® XML Schema

Where:
M_{UML} metamodel = {Class, Attribute, Generalization, Aggregation, Association}
XML Schema$_{skeleton}$ metamodel = (Component, Relation)
Component = {CT, ST, E, A, AG, MG}
Relation = {Der, Subst, Cont, Ref}

M_{UML} is a UML model. From UML metamodel, which defines UML model, we can identify several components. There are Class, Attribute, Generalization, Aggregation and Association. From XML Schema metamodel, which defines XML Schema, there are components and relation. The available components are Complex Type (CT), Simple Type (ST), Element (E), Attribute (A), Attribute Group (AG) and Model Group (MG). The Relations are Derivation (Der), Substitution (Subst), Containment (Cont), and Reference (Ref).

In the expression above, XML Schema skeleton is expressed in XML Schema constructs. An XML Schema construct has further alternatives, e.g. based on its properties. This more refined XML Schema construct is the required XML Schema. Thus, XML Schema is a refinement of XML Schema skeleton.

To illustrate the difference, consider a UML model component, UML Class. Suppose this UML Class is mapped into Complex Type. In this case, the M_{UML} is UML Class and the XML Schema$_{skeleton}$ is Complex Type. A Complex Type has several properties, one of them is optional property name. It may be named or anonymous. Suppose anonymous is selected. Thus, in this case, the XML Schema contains an anonymous Complex Type.

The two steps of mapping are explained below.

1. Schema Skeleton Generation ($M_{UML} \rightarrow$ XML Schema$_{skeleton}$)

   This step corresponds to AG1 in the figure 3.2. Alternatives in this step are the consequence of alternative mappings of a UML model component into several XML Schema components and relations. For example, a UML Class may be mapped into several components in XML Schema, such as Complex Type, Simple Type, Element, Attribute Group, and Model Group. Table 3.3 shows alternatives generation step 1 of a UML model component.

   A UML model consists of many UML model components. Because a UML component may be mapped into alternatives XML Schema components, a UML model may be mapped into many combinations of XML Schema components. This set of alternatives (called design space) is very huge. A method named *Design Algebra* is used to generate and evaluate alternative schemas [16]. This method evaluates alternative by using heuristic rules. The detail of this is out of the scope for this project. The result of applying Design Algebra is a set of alternative with valid combinations of XML Schema components.

Table 3.3 Schema Skeleton Generation

UML Component	***XML Schema Skeleton***
Class	Element
	Complex Type
Attribute	Attribute
	Attribute Group
	Element
	Model Group
Association	Element
	Complex Type
	Reference
	Containment
	Substitution
Aggregation	Treated just like ordinary association
Generalization	Derivation
	Containment
	Reference

2. Schema Skeleton Refinement (XML Schema$_{skeleton}$ $\rightarrow$ XML Schema)

   This step corresponds to AG2 in figure 3.2. In this step, XML Schema construct is refined further. For UML Class and Attribute, alternatives are identified based on the properties of Schema components. For Association and Aggregation, they are based on UML Metamodel. Alternatives are represented as UML Profile. The following section will explore about the alternatives, next chapter will address representation in UML Profile.

3.3 Schema Skeleton Refinement for UML Class and UML Attribute

As mentioned before, alternatives are identified based on XML Schema component's properties. These properties are included in the XML Schema specification and Schema Metamodel. Feature diagram will be used to capture the points of variability from these properties.

UML Class and Attribute can be mapped into XML Schema components, i.e. complex type, simple type, attribute, element, attribute group and model group. In this project, simple type is leaved out. It is rarely used in UML model and it has too many varieties. Before explaining each XML component, description about feature diagram will be given.

3.3.1 Feature Diagram

A feature diagram is a visual representation of concepts, features, and their relationship. A concept is an element or structure in the domain of interest. A feature is an important property of a concept. Relationship shows the relation between a concept and features, also among features.

A feature diagram forms a tree diagram that contains of nodes, directed edges, and edge decorations. The root node of the diagram is a concept. The remaining are features. Three types of features which will be used in this project are:

- Mandatory feature

  This feature is included in the description of a concept instance if and only if its parent is included. It is depicted as a node pointed to by an edge ending with a filled circle.

- Optional feature

  This feature may be included in the description of a concept instance if and only if its parent is included. Optional feature is depicted similarly as mandatory feature, except that the edge ending is an open circle.

- Alternative feature

  A concept or a feature may have one or more sets of alternative features. If the parent of a set of alternative feature is included in the description of a concept, then exactly one feature among the set is included. In another words, the nodes in the set of alternatives features are mutually exclusive. The nodes of this set are pointed to by edges connected by an arc.

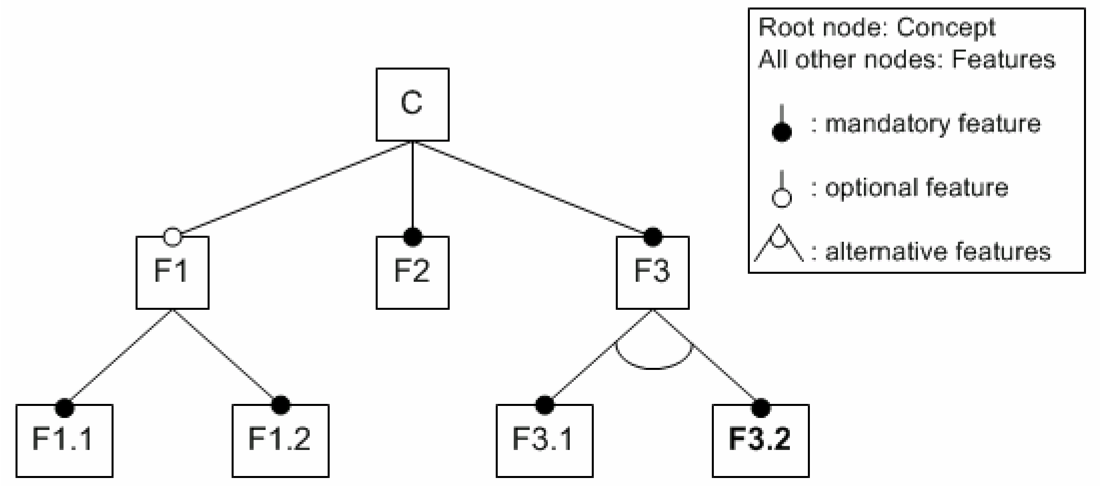


Figure 3.3 An example of feature diagram

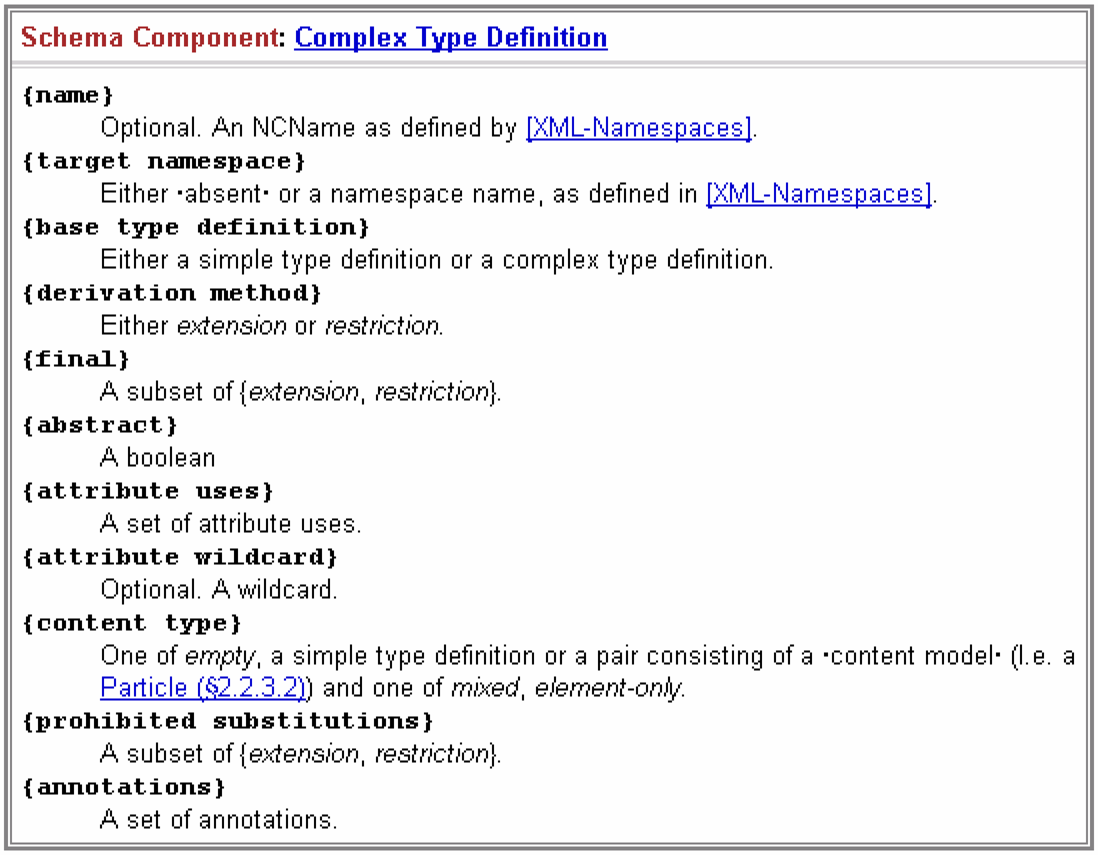
An example of feature diagram of concept C is depicted in the figure 3.3. C has three features, i.e. F1, F2, and F3. Features F1 and F3 may further be refined into sub features F1.1 and F1.2 for F1, F3.1 and F3.2 for F3. The features F2, F3, F1.1, and F1.2 are mandatory features (filled circles). F1 is optional feature (open circle). F3.1 and F3.2 are in a set of alternative features. In the figure, F3.2 is bold typed. This is mere a convention in this document to show that the feature is the default one among the alternatives.

Thus, we can interpret the feature diagram above as follows. Concept C has two direct mandatory features (F2 and F3) and one optional one (F1). If feature F1 is available, then it must have two sub features F1.1 and F1.2. Feature F3 must have sub feature either F3.1 or F3.2 but not both. If one does not mentioned whether to include F3.1 or F3.2 in the C definition, then the default one, F3.2, should be included.

Points of variability are identified in the optional feature and alternative feature. Thus, concept C has 2 points of variability. The first one is whether F1 is included in the description of concept C or not. The other one is in feature F3, a choice between F3.1 and F3.2.

3.3.2 Complex Type

A Complex Type definition schema component has properties. Those properties are summarized in below [18].



Schema Component: Complex Type Definition

{name}
Optional. An NCName as defined by [XML-Namespaces].

{target namespace}
Either ·absent· or a namespace name, as defined in [XML-Namespaces].

{base type definition}
Either a simple type definition or a complex type definition.

{derivation method}
Either *extension* or *restriction*.

{final}
A subset of {*extension*, *restriction*}.

{abstract}
A boolean

{attribute uses}
A set of attribute uses.

{attribute wildcard}
Optional. A wildcard.

{content type}
One of *empty*, a simple type definition or a pair consisting of a ·content model· (I.e. a Particle (§2.2.3.2)) and one of *mixed*, *element-only*.

{prohibited substitutions}
A subset of {*extension*, *restriction*}.

{annotations}
A set of annotations.

A feature diagram of Complex Type schema component is derived. It is shown in the figure below.

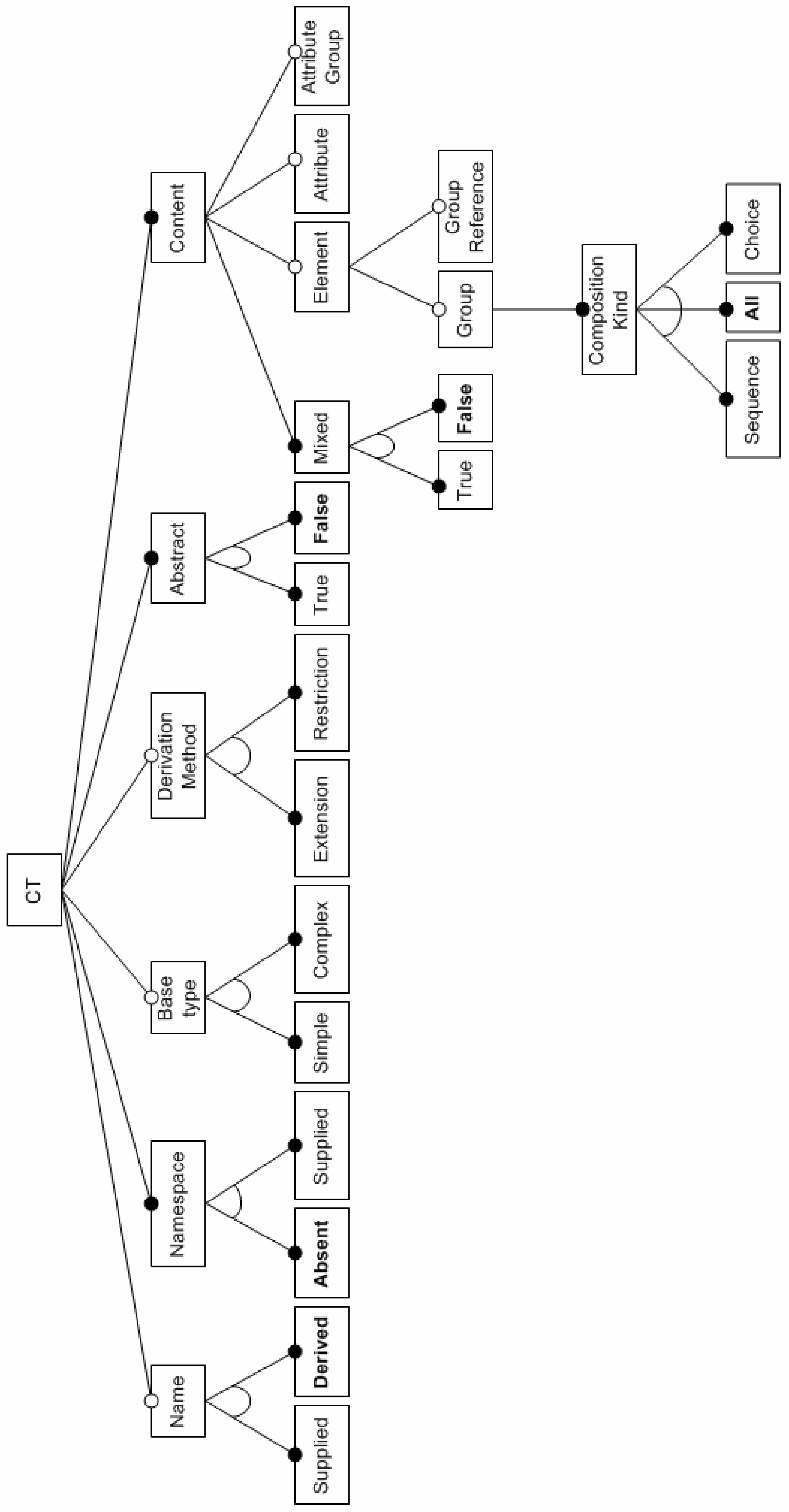


Figure 3.4 Feature Diagram: Complex Type

The feature diagram above is derived from the properties of a Complex Type and Schema Metamodel. Therefore, there are relations between the feature diagram with some important properties of a Complex Type, i.e.:

- Feature Name

  It corresponds to property {name}. This is an optional feature. If this feature is not available, then the complex type is called anonymous complex type. If there is, then it is called named complex type. The name of the complex type may be supplied by a user or derived from the model.
- Feature Namespace

  It corresponds to property {target namespace}. This feature may be absent, i.e. no target namespace, or the namespace maybe supplied.
- Feature Base type

  It corresponds to property {base type definition}. This is an optional feature. A base type may be a simple or complex type.
- Feature Derivation Method

  It corresponds to property {derivation method}. This is also an optional feature. A derivation may be a restriction type or an extension.
- Feature Abstract

  It corresponds to property {abstract}. A complex type may be abstract or not. If it is abstract, then it cannot be used in an instance document. In an element that corresponds to an abstract complex type, all instances of the element must use xsi:type to indicate a derived type that is not abstract.
- Feature Content

  It corresponds to property {content type}. This feature is also derived from the schema abstract data model. Mixed content complex type means that the complex type may contains text. A complex type may also contains elements, attributes, and attribute groups.

Some properties such as {final}, {attribute uses}, {attribute wildcard}, {prohibited substitution}, {annotation} are not used. These properties are used very rarely. Properties {final} and {prohibited substitution} have no direct relation with UML. In UML there is no method to disallow substitution.

There is a constraint in the property of Complex Type. If property Derivation is included in the definition Complex Type, then other property, namely Base Type must also be included. This constraint is not represented in the feature diagram. Feature diagram does not have symbol to represent constraint.

From the feature diagram, the following points of variability can be identified:

- Feature Name, Base type, and Derivation Method may be included in Complex Type concept definition.
- Feature Element, Group, Group Reference, Attribute, and Attribute group are optional. Thus, they may be included in concept definition.
- Feature Name: Supplied or Derived

  If feature name is available, then the name of a complex type may be supplied by the user or derived from the model.
- Feature Namespace: Absent or Supplied (default: Absent)

  A complex type may have target namespace which is supplied by the user. It also can have no target namespace.
- Feature Base type: Simple or Complex

  This feature is optional but must be available if the feature derivation method is also available. A base type may be either simple or complex.
- Feature Derivation Method: Extension or Restriction

  Optional derivation method of a complex type is either extension or restriction.
- Feature Abstract: True or False (default: false)

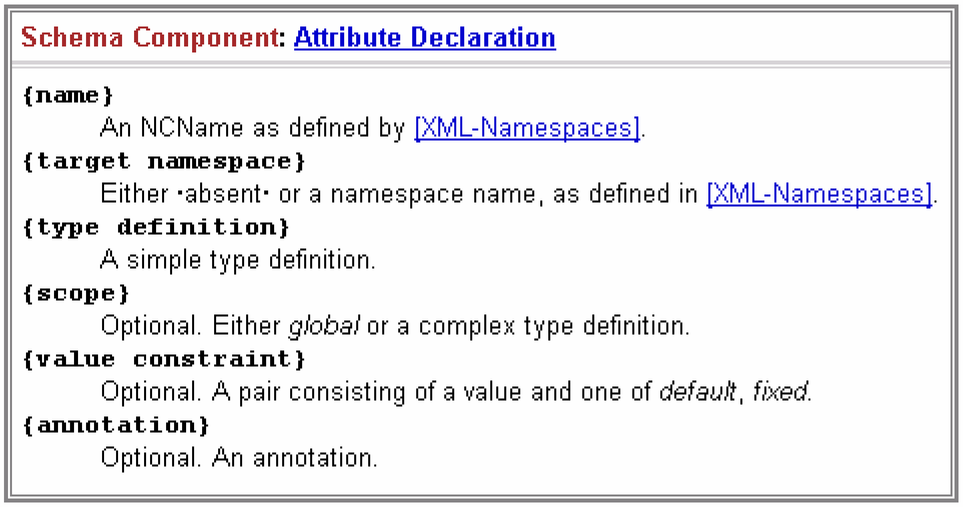
  A complex type may be abstract or not. If it is abstract, then a type derived from it is required.
- Feature Mixed: True or False (default: False)

  A complex type may contain text (mixed content) or only elements, group reference, attributes, and group of attributes.
- Feature Composition Kind: Sequence or All or Choice (default: All)

  The group of element in a complex type may have one of these composition kinds, i.e. sequence, all, or choice.

3.3.3 Attribute

An Attribute definition schema component has properties. Those properties are summarized in below [18].



Schema Component: Attribute Declaration

{name}
An NCName as defined by [XML-Namespaces].

{target namespace}
Either ·absent· or a namespace name, as defined in [XML-Namespaces].

{type definition}
A simple type definition.

{scope}
Optional. Either *global* or a complex type definition.

{value constraint}
Optional. A pair consisting of a value and one of *default*, *fixed*.

{annotation}
Optional. An annotation.

A feature diagram of attribute schema component is derived. It is shown in the figure below.

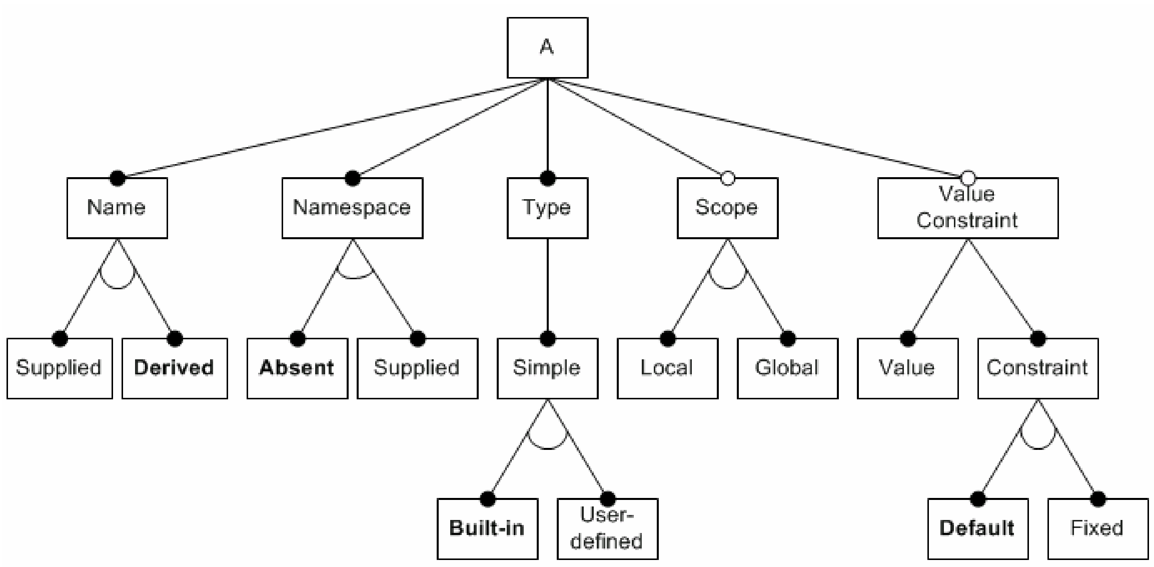


Figure 3.5 Feature Diagram: Attribute

The feature diagram above is derived from the properties of an Attribute. Therefore, there are relations between the feature diagram with some important properties of an Attribute, i.e.:

- Feature Name

  It corresponds to property {name}. An attribute must have a name. The name may be derived from the model or supplied by a user.

- Feature Namespace

  Same as complex type's namespace.
- Feature Type

  It corresponds to property {type definition}. An attribute must have a simple type. If the UML component has primitive data types, then the type of corresponding XML attribute should be built-in. In other cases, the type should be user-defined simple type. There is a possibility primitive data types of UML component are mapped into user-defined type of XML attribute.
- Feature Scope

  It corresponds to property {scope}. Feature Scope local means that the attribute is defined locally, within another XML component (e.g. XML complex type). Scope global means that the attribute is defined directly under schema element (<xs:schema>). The property is absent if the attribute is declared within attribute group definition.
- Feature Value Constraint

  It corresponds to property {value constraint}. This is the mapping of initial value of UML component (if specified). When the attribute does not appear in the instance document, feature default provides an attribute with a default value. If it appears and is different from default value, the default value will not override it. Feature fixed means that in the instance document the value of the attribute must be fixed as specified in the defined value.

From the feature diagram, the following points of variability can be identified:

- Feature Value Constraint and Scope may be included in Attribute concept definition
- Feature Name: Supplied or Derived (default: Derived)

  The name of an attribute may be supplied by the user or derived from the model.
- Feature Namespace: Absent or Supplied (default: Absent)

  Same as complex type's namespace.
- Feature Simple: Built-in or User-defined (default: Built-in)

  The simple type may be a built-in simple type or a user-defined one.

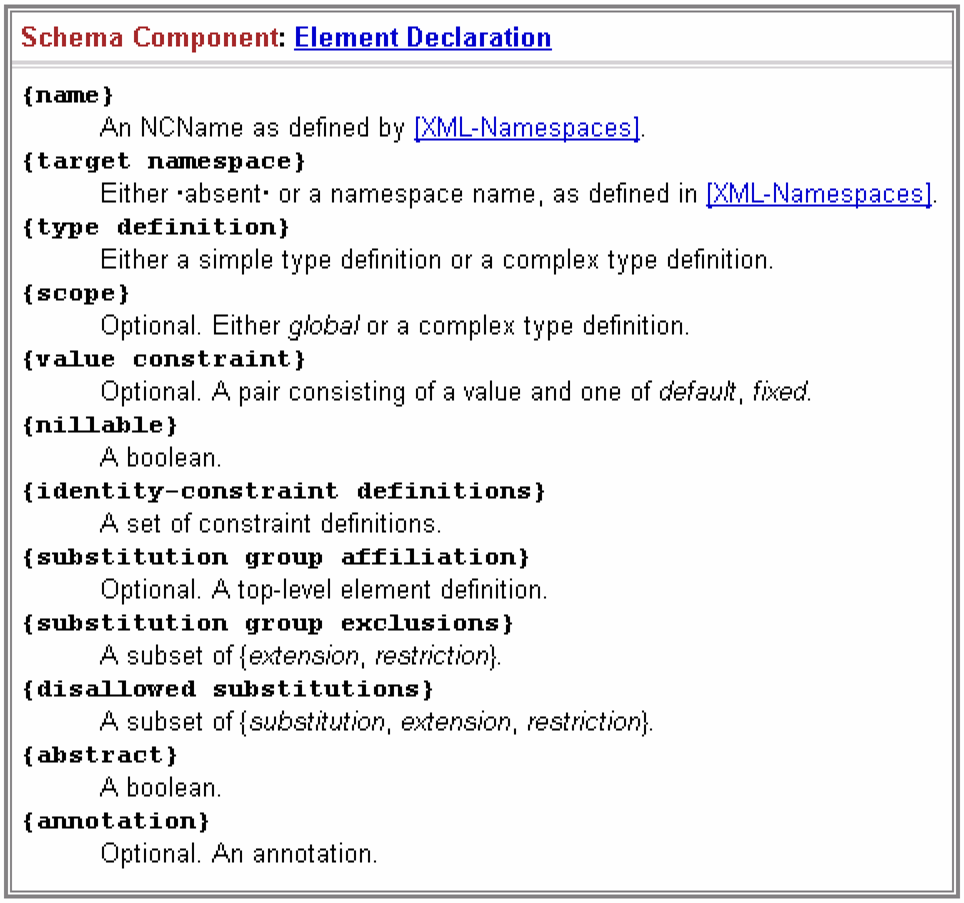
- Feature Scope: Local or Global

  The scope of the attribute is either local (within another XML Schema component) or global.
- Feature Constraint: Default or Fixed (default: Default)

  The constraint of an attribute is either a default value or a fixed value.

3.3.4 Element

An element declaration schema component has the following properties [18].



Schema Component: Element Declaration

{name}
An NCName as defined by [XML-Namespaces].
{target namespace}
Either ·absent· or a namespace name, as defined in [XML-Namespaces].
{type definition}
Either a simple type definition or a complex type definition.
{scope}
Optional. Either *global* or a complex type definition.
{value constraint}
Optional. A pair consisting of a value and one of *default*, *fixed*.
{nillable}
A boolean.
{identity-constraint definitions}
A set of constraint definitions.
{substitution group affiliation}
Optional. A top-level element definition.
{substitution group exclusions}
A subset of {*extension*, *restriction*}.
{disallowed substitutions}
A subset of {*substitution*, *extension*, *restriction*}.
{abstract}
A boolean.
{annotation}
Optional. An annotation.

A feature diagram of element schema component is derived. It is shown in the figure below.

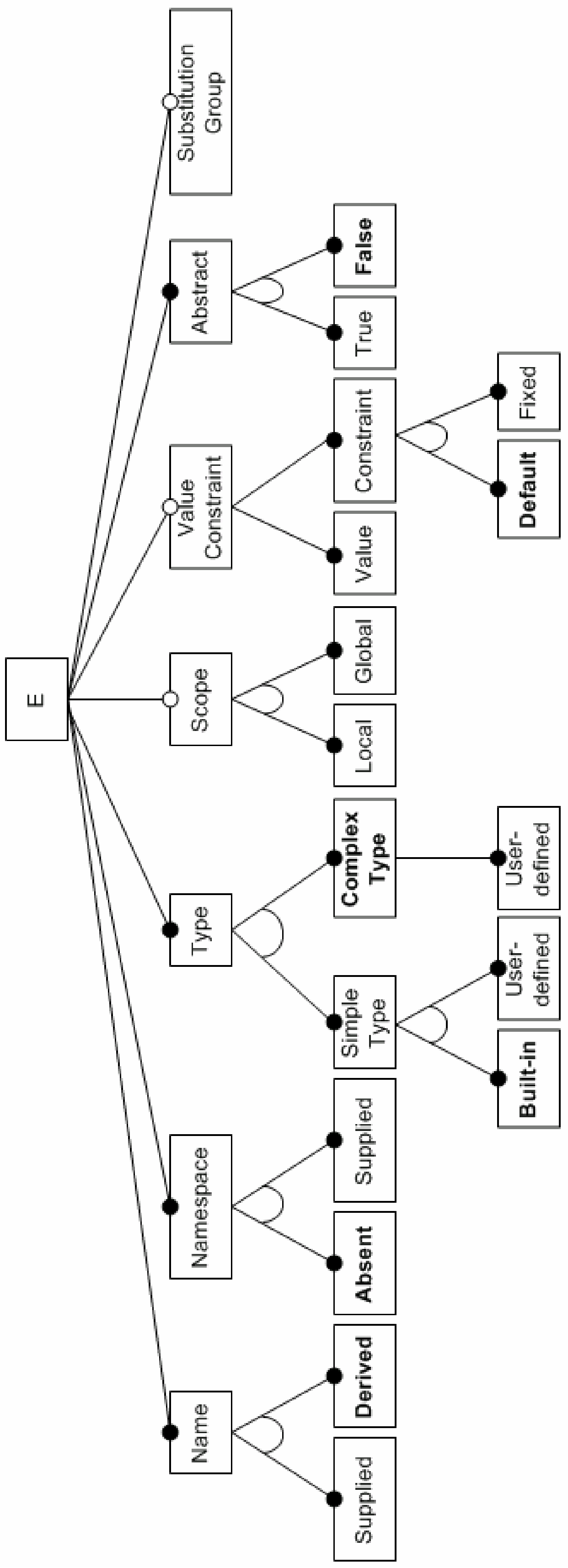


Figure 3.6 Feature Diagram: Element

The relations between element's properties with the feature diagram are as follows.

- Feature Name

  Same as attribute's name.
- Feature Namespace

  Same as complex type's namespace.
- Feature Type

  It corresponds to property {type definition}. An element may be a simple or complex type. A complex type always user defined. A simple type may be user-defined or built-in. If the UML component has primitive data types, then the type of corresponding XML element should be built-in.
- Feature Scope

  Same as attribute's scope.
- Feature Value Constraint

  It corresponds to property {value constraint}. This is the mapping of initial value of UML component (if specified). When the element appears in the instance document without any content, then default provides the default value. If it appears and is different from default value, the default value will not override it. If it does not appear at all, then the element will not be provided. Fixed means that in the instance document the element must be empty (then fixed behave as default) or the value of the element is as specified in the defined value.
- Feature Abstract

  It corresponds to property {abstract}. An element may be abstract or not. If it is abstract, then it cannot be used in an instance document. It can appear in content model where substitution is allowed.
- Feature Substitution Group

  It corresponds to property {substitution group affiliation}. An element may become a member of substitution group (if specified).

From the feature diagram, the following points of variability can be identified:

- Feature Value Constraint, Scope, and Substitution Group may be included in Element concept definition.
- Feature Name: Supplied or Derived (default: Derived)

  Same as attribute's name.

- Feature Namespace: Absent or Supplied (default: Absent)

  Same as complex type's namespace.
- Feature Simple Type: Built-in or User-defined (default: Built-in)

  Same as attribute's simple type.
- Feature Scope: Local or Global

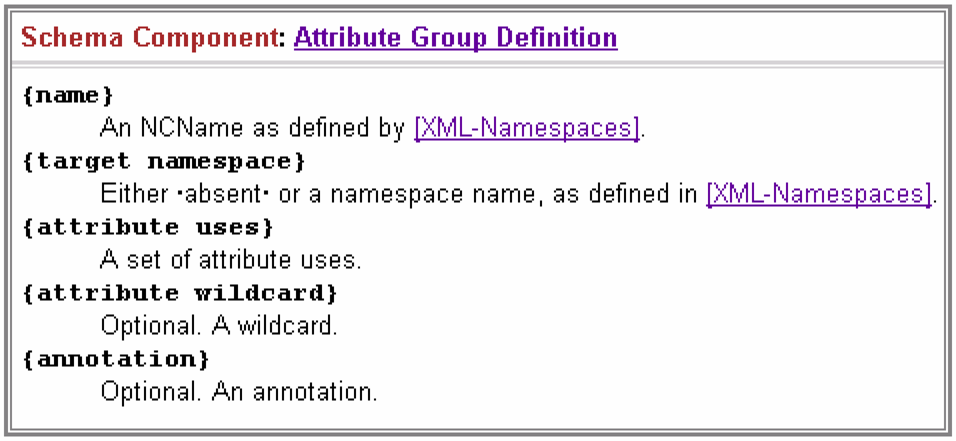
  Same as attribute's scope
- Feature Constraint: Default or Fixed (default: Default)

  The constraint of an element is either a default value or a fixed value.
- Feature Abstract: True or False (default: False)

  An element may be abstract or not. If it is abstract, then it cannot be used in instance document.

3.3.5 Attribute Group

An attribute group declaration schema component has the following properties [18].



Schema Component: Attribute Group Definition

{name}
An NCName as defined by [XML-Namespaces].
{target namespace}
Either ·absent· or a namespace name, as defined in [XML-Namespaces].
{attribute uses}
A set of attribute uses.
{attribute wildcard}
Optional. A wildcard.
{annotation}
Optional. An annotation.

A feature diagram of Attribute Group schema component is shown in the figure 3.5. The relations between that feature diagrams with the properties of an attribute group are:

- Feature Name

  Same as attribute's name.
- Feature Namespace

  Same as complex type's namespace.

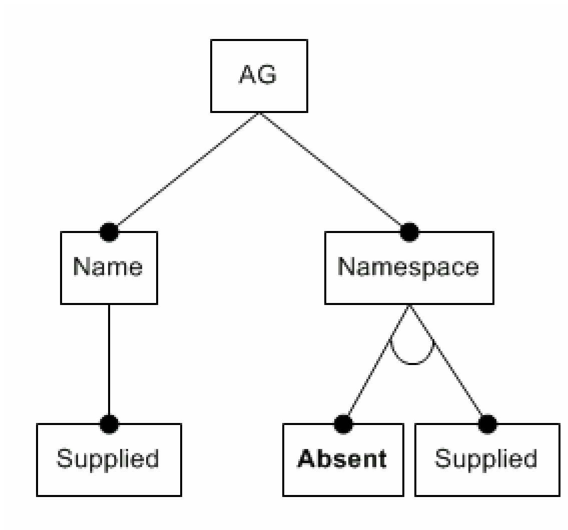


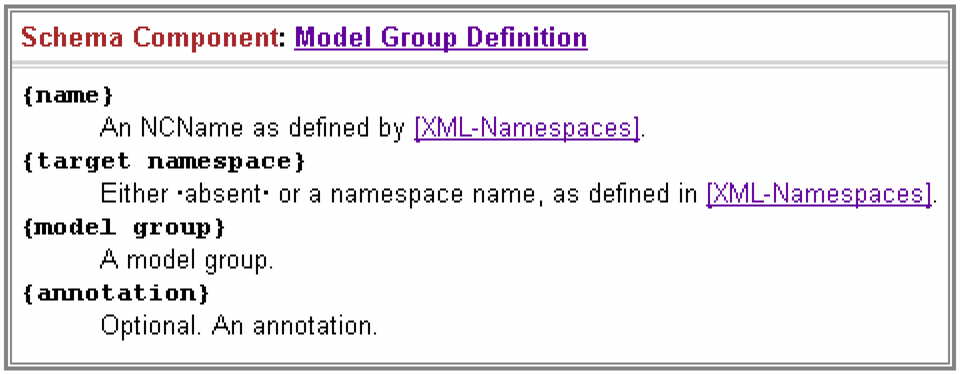
Figure 3.7 Feature Diagram: Attribute Group

Like Complex Type, other properties such as {attribute uses}, {attribute wildcard}, {annotation} are not used. These properties are used very rarely.

From the feature diagram, one point of variability can be identified. It is in the Namespace feature: Absent or Supplied (default: Absent).

3.3.6 Model Group

A model group declaration schema component has the following properties [18].



Schema Component: Model Group Definition

{name}
An NCName as defined by [XML-Namespaces].
{target namespace}
Either ·absent· or a namespace name, as defined in [XML-Namespaces].
{model group}
A model group.
{annotation}
Optional. An annotation.

A feature diagram of Model Group schema component is shown in the figure 3.6 below. The relation between that feature diagrams with the properties of a model group are:

- Feature Name

  Same as attribute's name.
- Feature Namespace

  Same as complex type's namespace.
- Feature Composition Kind

  It corresponds to property `{model group}`. A model group may have a composition kind sequence, all, or choice. All is the default one.

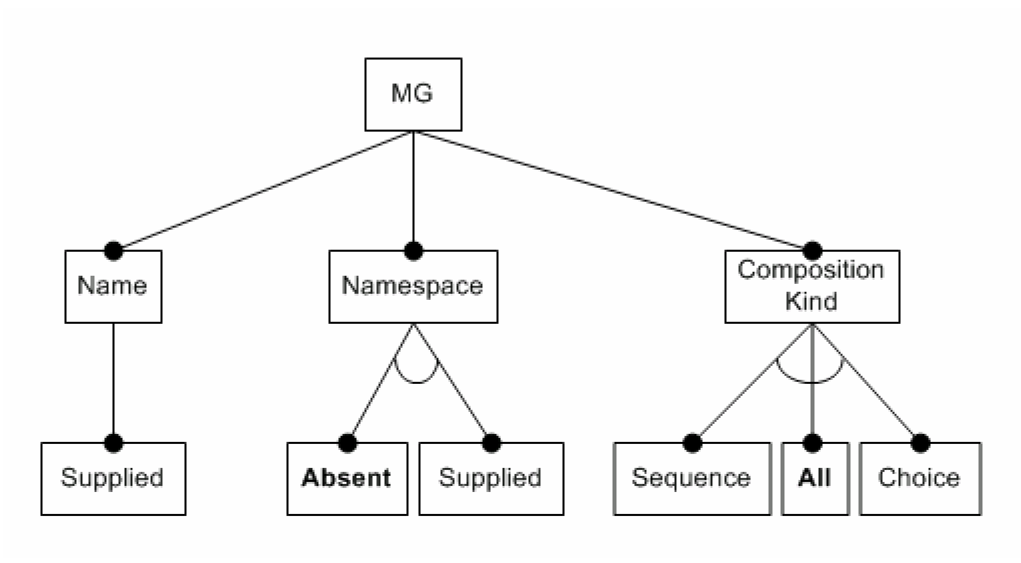


Figure 3.8 Feature Diagram: Model Group

Feature Model Group is not included in the feature diagram because this feature has no representation in UML.

From the feature diagram, two points of variability can be identified. They are:

- Feature Namespace: Absent or Supplied (default: Absent)

  Same as complex type's namespace.
- Feature Composition Kind: Sequence or All or Choice (default: All)

  Same as complex type's composition kind.

3.4 Schema Skeleton Refinement for UML Association / UML Aggregation

UML association defined in UML Metamodel is depicted in the following figure.

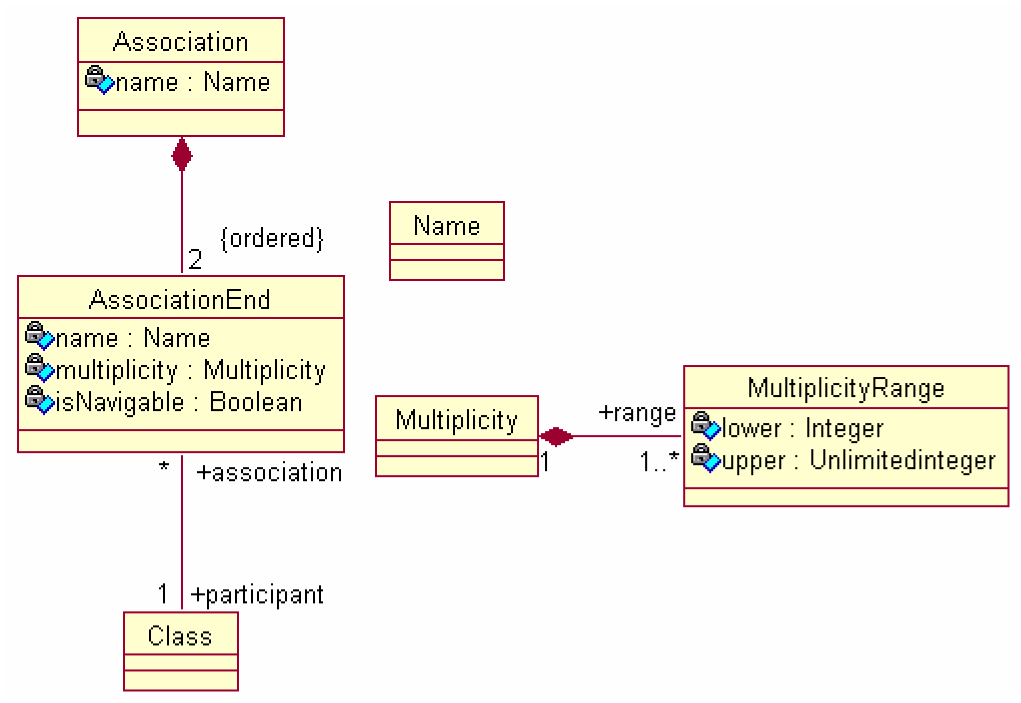


Figure 3.9 UML Metamodel: Association

As explained in chapter 2 in section about UML, only binary association (association that has exactly two association ends) will be considered. An association end may have several attributes, but here only three are considered, namely name (role name), multiplicity, and whether the association end is navigable or not.

A UML Association may be mapped into several XML Schema constructs, i.e. Element (E), Complex Type (CT), Reference (Ref), Containment (Cont), and Substitution (Subst). Mapping UML Association into Element, Complex Type, Reference and Containment will be discussed further. Mapping into Substitution is possible in XML Schema but then the class that is substituted by other class will never be used. Therefore, this mapping is excluded.

Alternatives for the mappings that are discussed are identified and refined. Here Design Algebra is used. After the alternatives are identified, those will be evaluated based on valid combination XML Schema construct and also constraint in XML Schema. Moreover, some alternatives are eliminated for simplicity reason.

3.4.1 Element

An association from UML Class C1 to UML Class C2 that is mapped into an XML Element is depicted in the following figure. Note that XSDelement is a stereotype. It means that the association will be mapped into an XML element. This will be explained in chapter 4.

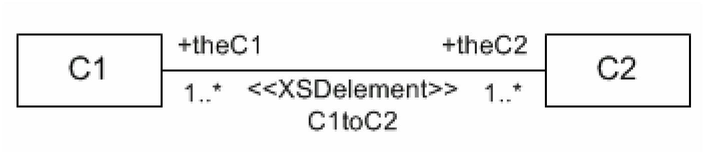


Figure 3.10 Mapping UML Association to XML Element

According to UML Metamodel, the association can be depicted as follows. Note that A is Association, AE is AssociationEnd, and [E] means that the Association will be mapped into an Element. Design Algebra is applied to this model.

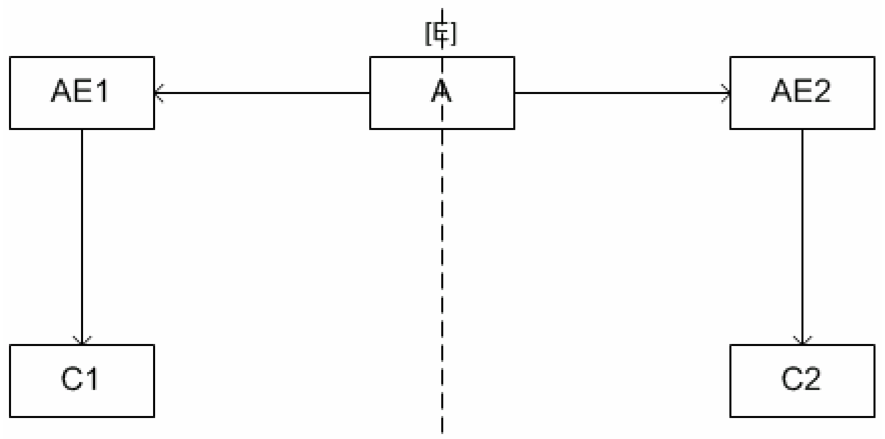


Figure 3.11 Mapping UML Association to XML Element Refined

From the figure, it is seen that the left side is similar to the right side using A as the axis. Thus, the alternatives mapping of the right part will be discussed and then it will be combined with the other part. Figure below shows all alternatives mapping of classes and associations of the right part of figure 3.9. Those alternatives are the alternatives for UML class and association (see section 3.2). The notation in the [] means the possible mapping of the component. For example AE2 may be mapped into Element, Complex Type or not mapped at all.

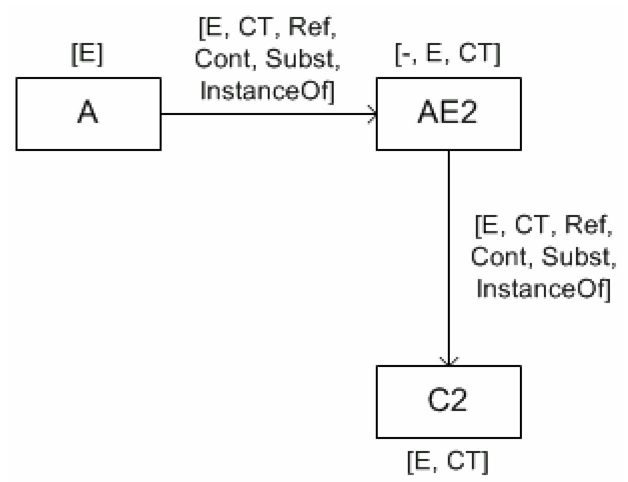

[E]
A
[E, CT, Ref, Cont, Subst, InstanceOf]
[-, E, CT]
AE2
[E, CT, Ref, Cont, Subst, InstanceOf]
C2
[E, CT]

Figure 3.12 Mapping to Element: Before Elimination

If all the combinations of the mapping in figure above are used, then there will be (1 x 6 x 3 x 6 x 2) possibilities or 216 possibilities. The numbers are the mapping possibilities for each component. One is from mapping A, Two from mapping C2, three from AE2 mapping, and six each for mapping A $\rightarrow$ AE2 and AE2. For both parts, there are 216^2 possibilities. Fortunately, we can select and eliminate possibilities. First, the elimination is based on the possible combination of components in XML. Then, a few mapping is also eliminated for simplicity.

Below are the rules or guides of elimination for one part of association:

- Mapping A $\rightarrow$ AE2 into XML element and complex type are eliminated. These mappings require further refinement of the association, i.e. apply the association from UML Metamodel in figure 3.9. In this project, the refinement stops at this level. Thus, these possibilities are eliminated.
- The same reason above applies to mapping AE2 $\rightarrow$ C2. Thus, mapping AE2 $\rightarrow$ C2 into element and complex type are eliminated.
- Mapping AE2 into CT is excluded. Association end needs to be instantiated in the XML document. The instantiation needs an element.
- As mentioned above, substitution is not addressed in this project. Thus, it is eliminated in mapping A $\rightarrow$ AE2 and AE2 $\rightarrow$ C2.
- Mapping A $\rightarrow$ AE2 into Reference is eliminated since it introduces indirect reference.
- Mapping A $\rightarrow$ AE2 into InstanceOf is also eliminated for simplicity reason. So is when AE2 is not mapped.

The result of elimination process is depicted in the following figure.

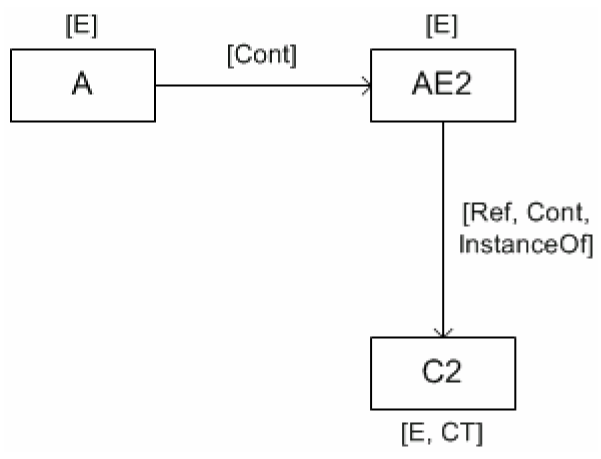


Figure 3.13 Mapping to Element: After Elimination

Further, there are several combinations that need to be eliminated. Those are the following.

- The combination: AE2 is mapped into E, AE2 $\rightarrow$ C2 is mapped into Cont, C2 is mapped into CT

  C2 is also defined outside this association, i.e. as an UML Class. Therefore, to avoid repeating mapping Class into CT, that combination is excluded.
- The combination: AE2 is mapped into E, AE2 $\rightarrow$ C2 is mapped into InstanceOf, C2 is mapped into E

  Class C2 is defined outside the association. If this is mapped into element with named complex type, then the combination above is valid. But it is not in the case if the complex type is anonymous. Thus, this combination is excluded.

As mentioned above, the mapping is only for one part. This part must be combined with the other part. There is one more combination to be excluded. The following table summarizes the possibilities for both parts after excluding the combinations above.

Table 3.4 Mapping UML Association to XML Element

C1	AE1 ® C1	AE1	A ® AE1	A	A ® AE2	AE2	AE2 ® C2	C2
E	Cont	E	Cont	E	Cont	E	Cont	E
E	Cont	E	Cont	E	Cont	E	InstanceOf	CT
E	Cont	E	Cont	E	Cont	E	Ref	E

E	Cont	E	Cont	E	Cont	E	Ref	CT
CT	InstanceOf	E	Cont	E	Cont	E	Cont	E
CT	InstanceOf	E	Cont	E	Cont	E	InstanceOf	CT
CT	InstanceOf	E	Cont	E	Cont	E	Ref	E
CT	InstanceOf	E	Cont	E	Cont	E	Ref	CT
E	Ref	E	Cont	E	Cont	E	Cont	E
E	Ref	E	Cont	E	Cont	E	InstanceOf	CT
E	Ref	E	Cont	E	Cont	E	Ref	E
E	Ref	E	Cont	E	Cont	E	Ref	CT
CT	Ref	E	Cont	E	Cont	E	Cont	E
CT	Ref	E	Cont	E	Cont	E	InstanceOf	CT
CT	Ref	E	Cont	E	Cont	E	Ref	E
CT	Ref	E	Cont	E	Cont	E	Ref	CT

As we can see from the table, the variability is in the mapping of association end to a class. It can be mapped into reference, containment of instance of.

3.4.2 Complex Type

This mapping is similar to mapping UML Association into XML Element in section 3.4.1 except that the association is mapped into a complex type instead of element. Thus, it is not discussed further.

3.4.3 Containment

For mapping UML association into containment, consider a unidirectional association between Class C1 and C2 as depicted in the following figure.

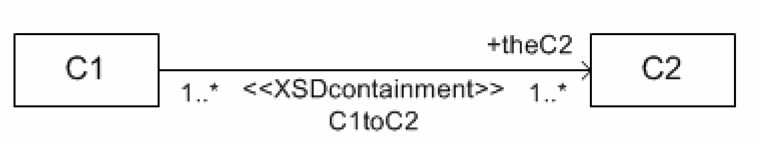


Figure 3.14 Mapping UML Association to XML Containment

According UML Metamodel for association, the mapping can also be depicted as in figure 3.15. From the figure, we can see that Class C1 which is mapped into either element (which has complex type) or complex type contains the mapping of Association End 2, which further relates to Class C2. Figure 3.15 shows the mapping possibilities after elimination process. Note that Association End 1 is not used in the mapping.

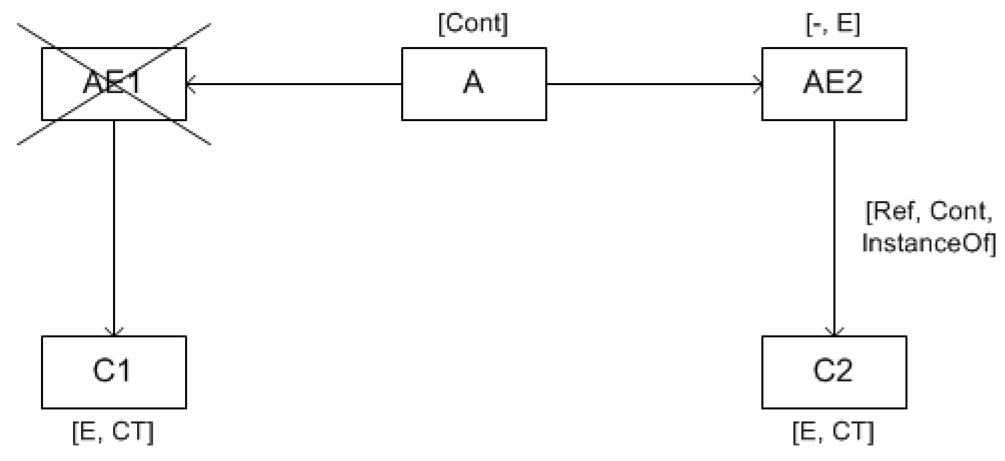


Figure 3.15 Mapping to Containment: C1 contains C2

The different combination of mapping for figure 3.15 is listed in the table 3.6. The third row of the table means that an XML element (the mapping of C1) will have another element (the mapping of association end) which a type of a complex type (the mapping of C2).

Table 3.6 Mapping to Containment: Unidirectional

C1	A	AE2	AE2 ® C2	C2
E	Cont	-	-	E
E	Cont	E	Cont	E
E	Cont	E	InstanceOf	CT
E	Cont	E	Ref	E
E	Cont	E	Ref	CT
CT	Cont	-	-	E
CT	Cont	E	Cont	E
CT	Cont	E	InstanceOf	CT
CT	Cont	E	Ref	E
CT	Cont	E	Ref	CT

If the direction of association is reversed, then the combination mapping is like the table above but changes in the index of C, A and AE. For bidirectional association, two unidirectional mappings must be comb ined. The result of this is listed in table below.

Table 3.7 Mappping UML Association to XML Containment

C1	AE1 ® C1	AE1	A	AE2	AE2 ® C2	C2
E	-	-	Cont	-	-	E
E	Cont	E	Cont	-	-	E
E	Ref	E	Cont	-	-	E
E	-	-	Cont	E	Cont	E

E	Cont	E	Cont	E	Cont	E
E	Ref	E	Cont	E	Cont	E
E	-	-	Cont	E	InstanceOf	CT
E	Cont	E	Cont	E	InstanceOf	CT
E	Ref	E	Cont	E	InstanceOf	CT
E	-	-	Cont	E	Ref	E
E	Cont	E	Cont	E	Ref	E
E	Ref	E	Cont	E	Ref	E
E	-	-	Cont	E	Ref	CT
E	Cont	E	Cont	E	Ref	CT
E	Ref	E	Cont	E	Ref	CT
CT	InstanceOf	E	Cont	-	-	E
CT	Ref	E	Cont	-	-	E
CT	InstanceOf	E	Cont	E	Cont	E
CT	Ref	E	Cont	E	Cont	E
CT	InstanceOf	E	Cont	E	InstanceOf	CT
CT	Ref	E	Cont	E	InstanceOf	CT
CT	InstanceOf	E	Cont	E	Ref	E
CT	Ref	E	Cont	E	Ref	E
CT	InstanceOf	E	Cont	E	Ref	CT
CT	Ref	E	Cont	E	Ref	CT

From the table, it is shown that there are two points of variability, i.e. in the mapping of association end to a class (which can be mapped into containment, reference and instance of) and the mapping of an association end (mapped into an element or it is not mapped).

3.4.4 Reference

There are several methods to use reference in XML, e.g. using xlink [17], key - keyref, and ID - IDREF/IDREFS. In this project ID - IDREF/IDREFS will be used, since it is the most commonly used.

ID – IDREF/IDREFS relates elements or complex type. In this project, the use of ID – IDREF/IDREFS method is as follows. One element/complex type (or more) has id as its attribute. This attribute is of type xs:ID. This is a simple type built in to XML Schema. This means that id holds the identification of the element. One element/complex type has an attribute idrefs. This attribute may be a type of xs:IDREF or xs:IDREFS. If the attribute is a xs:IDREF type, then this attribute contains one id that refers to an element/complex type. If it is xs:IDREFS, then it contains one or more ids. It means that it refers to one or more elements.

A unidirectional association between Class C1 and C2 which is mapped into reference is depicted in the figure 3.17.

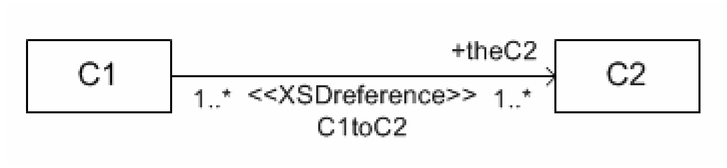


Figure 3.16 Mapping UML Association to XML Reference

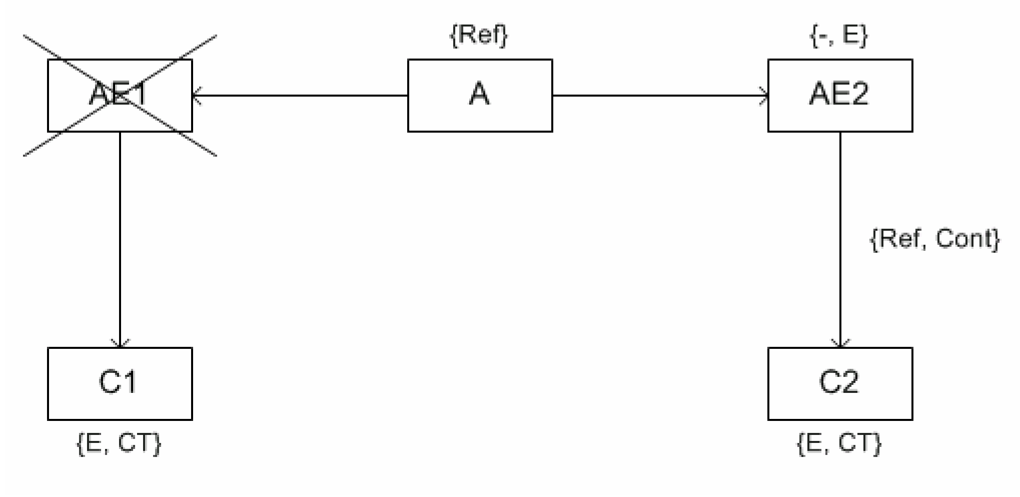


Figure 3.17 Mapping to Reference: C1 refers to C2

According to figure 3.7, the mapping can also be depicted as in figure 3.17. From the figure, Class C1 which is mapped into either element with complex type or only complex type refers to the mapping of Association End 2, which further relates to Class C2. That Figure shows the mapping possibilities after elimination process. Note that Association End 1 is not used in the mapping. If Association End 2 is mapped into nothing, then E or CT of C1 contains E or CT of C2 directly. The combination mapping for figure above is listed in the table below.

Table 3.8 Mapping to Reference: Unidirectional

C1	A	AE2	AE2 ® C2	C2
E	Ref	-	-	E
E	Ref	-	-	CT
E	Ref	E	Cont	E
E	Ref	E	InstanceOf	CT
E	Ref	E	Ref	E

E	Ref	E	Ref	CT
CT	Ref	-	-	E
CT	Ref	-	-	CT
CT	Ref	E	Cont	E
CT	Ref	E	InstanceOf	CT
CT	Ref	E	Ref	E
CT	Ref	E	Ref	CT

If the direction of association is reversed, then the mapping combination is like the table above but changes in the index of C, A and AE. For bidirectional association, two unidirectional mappings must be combined. The result of this is listed in table below.

Table 3.9 Mapping UML Association to XML Reference

C1	AE1 ® C1	AE1	A	AE2	AE2 ® C2	C2
E	-	-	Ref	-	-	E
E	Cont	E	Ref	-	-	E
E	Ref	E	Ref	-	-	E
E	-	-	Ref	-	-	CT
E	Cont	E	Ref	-	-	CT
E	Ref	E	Ref	-	-	CT
E	-	-	Ref	E	Cont	E
E	Cont	E	Ref	E	Cont	E
E	Ref	E	Ref	E	Cont	E
E	-	-	Ref	E	InstanceOf	CT
E	Cont	E	Ref	E	InstanceOf	CT
E	Ref	E	Ref	E	InstanceOf	CT
E	-	-	Ref	E	Ref	E
E	Cont	E	Ref	E	Ref	E
E	Ref	E	Ref	E	Ref	E
E	-	-	Ref	E	Ref	CT
E	Cont	E	Ref	E	Ref	CT
E	Ref	E	Ref	E	Ref	CT
CT	-	-	Ref	-	-	E
CT	InstanceOf	E	Ref	-	-	E
CT	Ref	E	Ref	-	-	E
CT	-	-	Ref	-	-	CT
CT	InstanceOf	E	Ref	-	-	CT
CT	Ref	E	Ref	-	-	CT
CT	-	-	Ref	E	Cont	E
CT	InstanceOf	E	Ref	E	Cont	E
CT	Ref	E	Ref	E	Cont	E
CT	-	-	Ref	E	InstanceOf	CT
CT	InstanceOf	E	Ref	E	InstanceOf	CT
CT	Ref	E	Ref	E	InstanceOf	CT
CT	-	-	Ref	E	Ref	E
CT	InstanceOf	E	Ref	E	Ref	E

CT	Ref	E	Ref	E	Ref	E
CT	-	-	Ref	E	Ref	CT
CT	InstanceOf	E	Ref	E	Ref	CT
CT	Ref	E	Ref	E	Ref	CT

The points of variability for this case are the same as containment above.

3.5 Schema Skeleton for UML Generalization

For UML Generalization, three possibilities of mapping will be discussed. There are Derivation, Containment, and Reference. These possibilities are similar to what have been explained above. Here, those mapping will be shortly described.

Mapping UML Generalization to Derivation means that the child class is a derivation of its parent class. This mapping is using feature Derivation Method and Base Type from XML Complex Type. (See page 3-6) Feature Derivation Method has two sub features, i.e. Extension and Restriction. Feature Base Type also has two sub features, i.e. Simple and Complex. Those features must be included in the child class. They are the alternative for mapping UML Generalization to Derivation.

Mapping UML Generalization to Containment or called copy-down inheritance means that the attributes of parent class is included as the attributes of child class.

Mapping UML Generalization to Reference is similar to mapping UML Association to Reference in which the child class refers to the parent class. Therefore, the alternatives for this mapping are the alternatives mapping of UML Attribute into Reference.

3.6 Summary

This chapter described the mapping from UML Model into XML Schema. These mapping is related to four-layer modeling architecture. There are two steps of alternatives generation, i.e. schema skeleton generation and schema skeleton refinement. In the first step, Design Algebra is used to generate and evaluate the alternatives. The refinement of schema skeleton is discussed in detail for each UML component. For UML class and attribute, which can be mapped into XML components, feature diagram is used to capture the variability of each XML component. For UML association and aggregation, the alternatives are generated based on model element of association and by using Design Algebra. Refinement for UML generalization is also discussed.

4 Transforming UML Model into XML Schema

This chapter discusses about the approach to solve the problem of this project. In section 4.1 the solution approach is presented. Section 4.2 contains about alternatives generation representation for the alternatives in chapter 3. In the next section, the detail of transformation is explained. Last section gives a summary of this chapter.

4.1 Solution Approach

The approach used in this project to solve the problem in figure 1.1 is depicted in the following figure. In the figure, the strip and dotted arrow shows validation. Thus, XML document is validated by XML Schema and XML Representation (result of XMI serialization) is validated by XMI DTD.

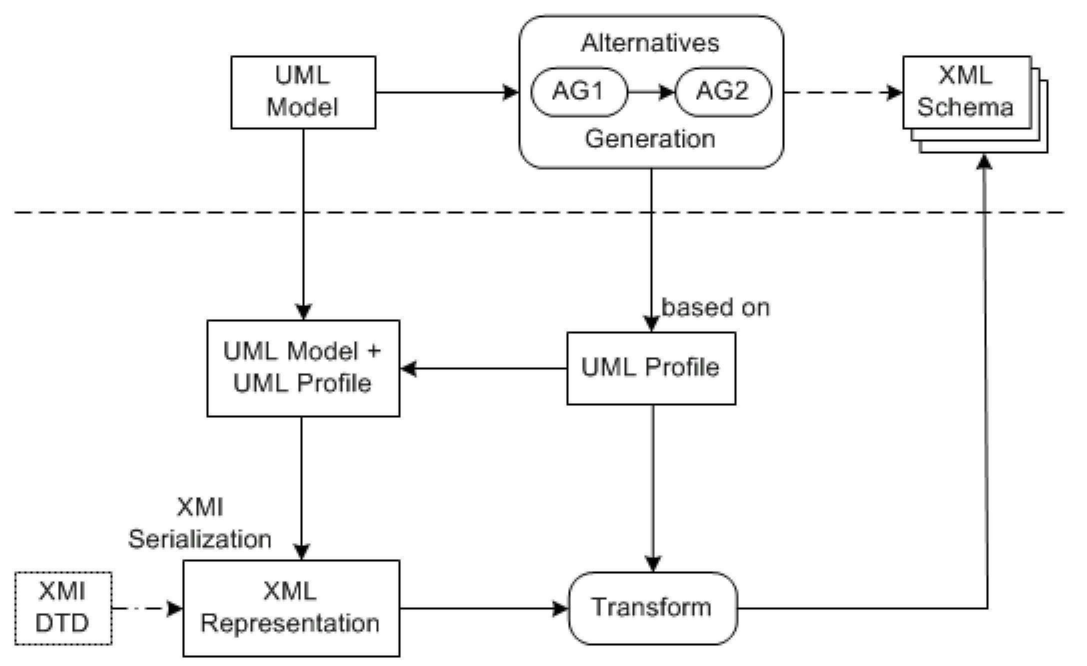


Figure 4.1 An approach to Solve the Problem

From the figure above, there are two things that will to be explained. They are the alternatives representation and the transformation.

4.2 Representation of Alternatives

In this project, alternatives are represented using UML Profile (Stereotypes and Tagged Values). Next two sections are the UML Profile defined for this project. The complete list of UML Profile for each UML construct is summarized in the appendix A. Carlson also defined UML Profile for XML Schema (see section 2.3.3). Part of his UML profile will be used in this project. When it is used, it will be stated in the text. The last part of this section will give an example how to use the stereotype and tagged values.

4.2.1 Schema Skeleton Generation

The result of Alternatives Generation step 1 is represented using stereotypes. The stereotypes defined for it is shown in table 4.1. Note that three of the stereotypes are taken from Carlson, i.e. `XSDcomplexType`, `XSDelement`, and `XSDattribute`.

Table 4.1 UML Profile for Alternatives Generation Step 1

UML	***XML Schema Skeleton***	***Stereotype***
Class	Element	XSDelementComplex
	Complex Type	XSDcomplexType
Attribute	Attribute	XSDattribute
	Attribute Group	XSDattributeGroup
	Element	XSDelement
	Model Group	XSDmodelGroup
Association	Element	XSDelement
	Complex Type	XSDcomplexType
	Reference	XSDreference
	Containment	XSDcontainment
	Substitution	XSDsubstitution
Aggregation	Treated just like association	
Generalization	Derivation	XSDderivation
	Containment	XSDcontainment
	Reference	XSDreference

The UML construct Class may be mapped into XML Element. This element may be refined further into element only, element with complex type, and element with simple type. Here only element with complex type is used, which has stereotype `XSDelementComplex`. The reason is because a UML Class usually has UML attributes and association. For that case, element only cannot be used. Element with simple type also cannot be used, since simple type cannot have any XML element or attribute as its child.

A UML construct that has stereotype `XSDelementComplex` may use the tagged values defined for XML Element and also XML Complex Type.

4.2.2 Schema Skeleton Refinement

Tagged value can be used to represent the choice of alternatives from Alternatives Generation step 2. The values of the tagged values may be an enumeration or a string that can be filled by the user.

4.2.2.1 UML Class and UML Attribute

Below are the tagged values for each XML Schema component. The italics font in the enumeration value of a tag means the default value for the tag.

Complex Type

Tagged values defined for this schema component are:

- `mixed(true|`*`false`*`)`

  This tagged value is used to show mixed content of a complex type, whether character data may appear alongside sub elements. It is corresponds to the variability point feature mixed (sub features: true or false). Value `false` is the default value of this tag.
- `complexTypecompositionKind(`*`all`*`|sequence|choice)`

  This tagged value is used to show how elements must appear in a content model. Value sequence means that elements in a content model must appear in order. On the other hand value all allows element appear in any order. Value choice allows one element from the content model to be appeared in instance document. This tagged value corresponds to variability point feature Composition Kind variability (sub features: all, sequence, choice) with `all` as the default one.

- `complexTypeName`

  This tagged value shows the name of complex type. It is used to represent optional feature Name (subfeatures: Supplied or Derived). If the value of this tagged is 'anonymous' then the complex type is anonymous. If it is filled with a string 'derived', then the name of the complex type is derived from the model. Other strings mean that is the name of the complex type (supplied by the user). By default, the value of this tagged is `derived`.
- `complexTypeAbstract(true|`*`false`*`)`

  If a complex type is declared as abstract, then it needs a type derived from it (the instance of an element that has this type must use xsi:type). This tagged used to represent feature Abstract (subfeatures: True or False). The default value of this tag is `false`.
- `modelGroupName`

  This tag shows the presence a named group of elements. If the tag is empty then there is no feature Group Reference.
- `attributeGroupName`

  This tagged value shows the presence of a group of attributes. It represents the optional feature Attribute Group in the definition of complex type. Empty tag means that there is no feature Attribute Group.

Tagged value `mixed(true|`*`false`*`)` is taken from Carlson. So is feature `complexTypeCompositionKind(`*`all`*`|sequence|choice)`, but in Carlson's it is named `modelGroup(`*`all`*`|sequence|choice)`.

Other features:

- Feature Namespace (subfeatures: Absent or Supplied)

  This feature is related to `targetNamespace` attribute of `<xs:schema>`, the root element of an XML Schema. To use that attribute, namespace prefix must also be defined. In this project, namespace prefix is not defined. Thus, this feature will not be used.
- Optional feature Element

  The presence of this feature is shown in the presence its sub features (Group and Group Reference).

- Optional feature Group

  The presence of this feature is determined by the presence its subfeature (Composition Kind). Thus, if tag `compositionKind(`***all***`|sequence|choice)` is used, then features Group and Element are included in the definition of complex type.
- Optional feature Attribute

  The presence of this feature in complex type is determined by the UML model, whether there is a UML component that will be mapped into XML attribute.
- Optional feature Base type and Derivation Method

  These two features are present in the complex type if the represented UML class is a child class of a UML generalization and the generalization is mapped into XML derivation.

Attribute

Tagged values defined for this schema component are:

- `attributeName`

  This tagged value shows the name of an attribute. It represents feature Name (sub features: derived or supplied). If this tagged has a value '`derived`', then the name of the attribute is derived from the model. If it is filled with a string, then the string is the name of the attribute. By default the name of an attribute is `derived`.
- `attributeConstraint(`***default***`|fixed)`

  This tag is related to the value of the attribute which specified in feature Value. It gives the constraint in the value, whether the value is a default value or a fixed one. This represents the feature Constraint (sub features: default or fixed). The presence of this feature is determined by the presence of feature Value. If feature value is present and the tag is not specified, then the value `default` is used.

Other features:

- Feature Namespace (subfeatures: Absent or Supplied)

  Similar to feature namespace in complex type.

- Optional feature Scope (sub features local: global or local)

  The presence and choices for this feature is based on other XML Schema components. If the attribute is under <complexType> element, then the scope is local. If it is under <xs:schema> directly, than its scope is global.
- Feature Type (sub features: built-in or user-defined)

  The choice is decided using information about attributes type in the model. If the type of the attribute is a primitive type (string, integer, Boolean, date, byte, double, long), then the type is built-in. Otherwise, the type is user-defined.
- Feature Value

  This feature is represented in the UML Attribute's initial value. If this feature is present, then feature Value Constraint is also present.

In the case Complex Type is the result of mapping a UML Class, the complex type must be a named one.

Element

Tagged values defined for this schema component are:

- elementName

  Similar to attribute's tagged value attributeName.
- elementConstraint(***default***|fixed)

  Similar to attribute's tagged value attributeConstraint.
- elementAbstract(true|*false*)

  Similar to complex type's tagged value complexTypeAbstract.
- substitutionGroup

  This tag shows that the mechanisms substitution groups, which allows element to be substitute by other element, is present. It represents optional feature Substitution Group. If this tagged value is empty, then the feature Substitution Group is not present. Otherwise, this tag is filled with a string which specifies an element name that it substitutes.

Other features:

- Feature Namespace (subfeatures: Absent or Supplied)

  Similar to feature namespace in complex type.

- Optional feature Scope (sub features local: global or local)
  Similar to attribute's feature Scope
- Feature Type (sub features: Simple Type or Complex Type) and Simple Type (sub features: built-in or user-defined)
  The choice for these features is determined by the mapping of other UML construct. Thus it is decided by the model. Here, an assumption is used, i.e. if the type of the element is a primitive data type (string, integer, Boolean, date, byte, double, long), then the element is mapped into a built-in simple type.
- Feature Value
  Similar to attribute's feature Value.

In the case element construct is the result of UML Attribute mapping, the multiplicity of UML Attribute is mapped into `minOccurs` and `maxOccurs` attribute of an XML element. There is a constraint to map the multiplicity, i.e. composition kind `all` only permits multiplicity `0` and `1`.

Attribute Group

No tagged value is defined for this schema component. Feature Namespace is not used in this project. Feature Name of this component is included in the complex type component.

Model Group

Tagged value defined for this schema component is:

- `modelGroupCompositionKind(`***`all`***`|sequence|choice)`
  Similar to complex type's tagged value `complexTypeCompositionKind`.

This tagged value is included in XML Complex Type construct. So is tagged for feature Name. As for other XML construct, feature namespace is not used.

The UML Profile defined in this project is summarized in the table below.

Table 4.2 UML Profile for Each XML Component

XML Construct	***Tagged Values***
Complex Type	mixed(true|*false*)
	complexTypeCompositionKind(*all*|sequence|choice)

	complexTypeName
	complexTypeAbstract(true|*false*)
	modelGroupName
	attributeGroupName
	modelGroupCompositionKind(*all*|sequence|choice)
Attribute	attributeName
	attibuteConstraint(*default*|fixed)
Element	elementName
	elementConstraint(*default*|fixed)
	elementAbstract(true|*false*)
	substitutionGroup

4.2.2.2 UML Association and UML Aggregation

Below are the tagged values for each possible mapping.

Element

One tagged value is defined for element, i.e. `associationEndToClass(containment|instanceOf|`*`reference`*`)`. This tagged value represents mapping an association end to a class. Reference is the default value for this tagged.

Notes:

- Association

  Association only considers the attribute name (association name). This is mapped into the name of element. Other features are assumed to be default. Name is a mandatory feature of an element. Thus, if the association name is empty and the association is connecting classes C1 and C2, then the association name is C1-C2.
- Association End

  Here, three things are considered (isNavigable, name, and multiplicity). If the association end is not navigable (isNaviagable is false) then the association end is not mapped. An element which represents Association End has an attribute `name` from association end name (role name) and attributes `minOccurs` and `maxOccurs` from the multiplicity. If the role name is empty, then a role name: The+`classname` is used.

Complex Type

Complex type has the same tagged value as Element above.

Containment

For this XML relation, two tagged values are used, i.e.

- `associationEndToClass(containment|instanceOf|`***`reference`***`)`

  Similar to element's tagged value `associationEndToClass`.

- `associationEnd(`***`element`***`|none)`

  This represents the mapping of an association end. The default value is element.

Reference

Reference has the same tagged value as Containment above.

4.2.2.3 UML Generalization

Derivation

Two tagged values are defined for this, i.e.:

- `baseType(simple|`***`complex`***`)`

  This is used to represent the feature Base type (subfeatures: Simple or Complex). The default value for this tag is complex.

- `derivation(`***`extension`***`|restriction)`

  This tagged value represents feature Derivation Method (sub features: Extension or Restriction). The default value for this tag is extension.

These tagged values are included in the child class of a generalization.

Containment

In this project, no tagged value is defined for Containment.

Reference

Reference has the same tagged value as Reference in UML Association above.

4.3 Example

An example of the usage of UML Profile in a UML Class and Attribute is depicted in the figure 4.2. Note that in the figure, the tagged values are represented in braces. This is the standard way to represent a tagged value. However, a UML tool may represent a tagged value in another way. For example Rational Rose represents tagged value as Model Properties. Rational Rose will be used in this project. How to add tagged values in Rational Rose will be explained in next chapter.

In that figure there are six items from the UML Profile.

- Stereotypes: `XSDcomplexType`, `XSDattribute`, and `XSDelement`
- Tagged values: `complexTypeCompositionKind=sequence`, `abstract=true`, and `elementName=property`

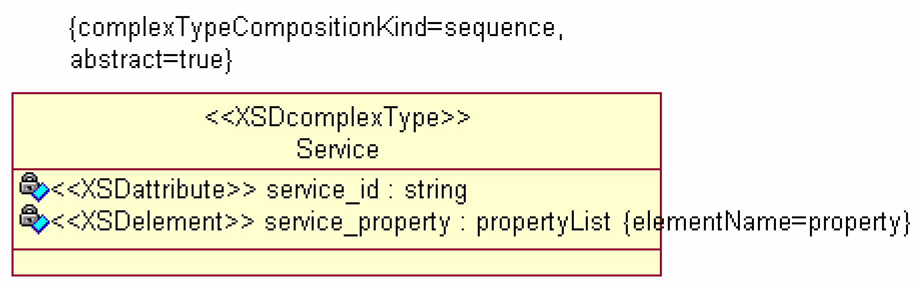

{complexTypeCompositionKind=sequence,
abstract=true}
<<XSDcomplexType>>
Service
<<XSDattribute>> service_id : string
<<XSDelement>> service_property : propertyList {elementName=property}

Figure 4.2 Example of UML Profile for a UML Class and UML Attributes

UML Class `Service` has stereotype `XSDcomplexType`. This means that the Class has to be mapped into an XML Complex Type. This class also has two tagged value, i.e. `complexTypeCompositionKind=sequence` and `abstract=true`. Those tagged values are defined for XML complex type construct. The second tagged value indicates that the complex type is abstract. The other tagged value means that the content model of the complex type is sequence. Thus, UML attributes contained in that class which are mapped into XML element must be ordered.

XML complex type has other tagged values, which are not displayed. This means that the default values of those tagged apply. For example, tagged value `complexTypeName` is not available, thus the default value for `complexTypeName` which is derived applies.

UML attribute `service_properties` has stereotype `XSDelement`. It means that this attribute is mapped into XML element. This attribute has a tagged value `elementName=property`. It means that the name of this attribute in the schema is property not `service_properties`. This is an example of user-defined name. The other tagged values are using the default values.

4.4 Transformation

One possible implementation of transformation from UML Class Diagram into XML Schema is by using XMI (XML Metadata Interchange) and XSLT (eXtensible Stylesheet Language Transformation). This implementation is shown in figure 4.3.

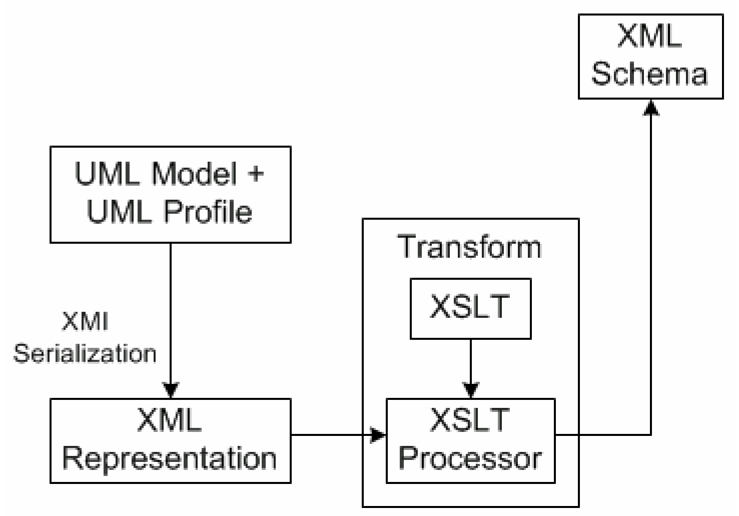


Figure 4.3 Transforming UML Model + UML Profile into XML Schema

The transformation is as follows.

1. UML Class Diagram + UML Profile → XML Representation (XMI Serialization)
   First, the UML class diagram is converted into an XML representation (XML document). In order to do this, XMI Serialization is used. XMI DTD is a Document Type Definition that validates the generated XML representation.
2. XML Representation (XMI) → XML Schema (XSLT and XSLT Processor)
   After that, the XML representation, generated using XMI, is then transformed into XML Schema. This transformation needs XSLT processor and XSLT files which contains the rules of transformation.

4.5 Summary

This chapter presents the approach used in the project to solve the problem of mapping UML model onto XML Schema. The approach is by using UML Profile for the alternatives and using XMI serialization and XSLT for the transformation. UML profile has been defined, part of them is from Carlson.

5 Implementation

This chapter describes the implementation of the approach presented in chapter 4. For this project, three tools are used, i.e. Rational Rose, Unisys Rose XML Tools 1.3.5 and XML Spy version 5.3. Refers to figure 4.1, Rational Rose is used for block UML Model + UML Profile and UML Profile. It is used to create UML class diagram and UML Profile. Unisys Rose XML Tools 1.3.5 which does the XMI serialization. In figure 4.1, it is the arrow from UML Model + UML Profile to XML Representation. XML Spy is used to do the transformation, i.e. transform block in the figure 4.1. These tools will be explained respectively in section 5.1, 5.2 and 5.3. Section 5.4 gives a summary of this chapter.

5.1 UML Model and UML Profile

Rational Rose is a modeling tools to create UML Model (in this project, it is restricted to UML Class Diagram) and UML Profile. The version of Rational Rose that is used in this project is Rose Enterprise Edition Release Version: 2002.05.20. The procedure to add UML Profile in class diagram in Rational Rose is as follows.

- Stereotype

  Rational Rose provides a standard method to add stereotypes. It is added by putting stereotype enclosed in guillemets in front of UML component's name or putting stereotype in stereotype field which located in the specification of the UML component. An example is depicted in figure 5.1. In the figure above, it is seen that UML Class named Service has stereotype `XSDelementComplex`, UML Attribute `service_id` has stereotype `XSDattributeGroup`, and `service_properties` has stereotype `XSDelement`.

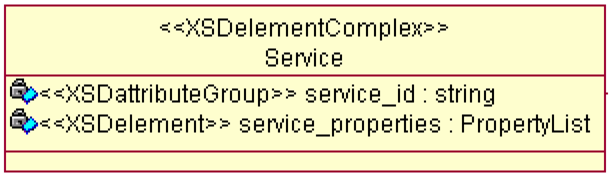
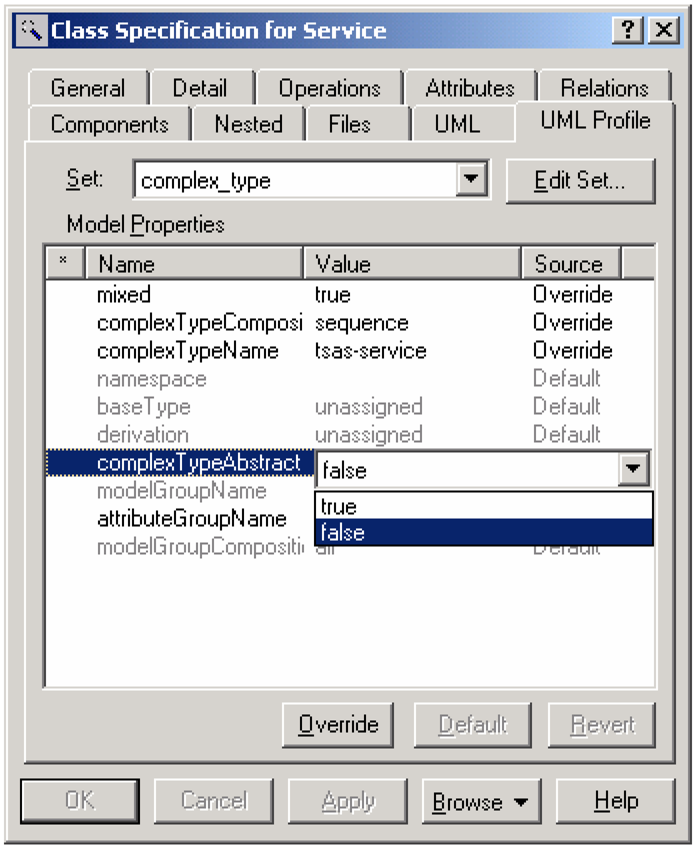

<<XSDelementComplex>>
Service
<<XSDattributeGroup>> service_id : string
<<XSDelement>> service_properties : PropertyList

Figure 5.1 Stereotype in Rational Rose

- Tagged Value

  In Rational Rose, there is no direct way (such as adding stereotype) to add tagged values to a model. Rational Extensibility Mechanism will be use for this.

  Tagged Values in Rational Rose are represented as user-defined properties. A property has a name and a value, corresponds to tag and value respectively. Tagged values are grouped in packages. To illustrate this consider an example in figure 5.2.



Set: complex_type

Model Properties

*	Name	Value	Source
	mixed	true	Override
	complexTypeComposi	sequence	Override
	complexTypeName	tsas-service	Override
	namespace		Default
	baseType	unassigned	Default
	derivation	unassigned	Default
	complexTypeAbstract	false	
	modelGroupName	true	
	attributeGroupName	false	
	modelGroupCompositi	all	Default

Figure 5.2 Tagged Values in Rational Rose

In the figure, there are ten properties shown, e.g. `complexTypeAbstract` and `complexTypeName`. Each property has a particular type. There are two types shown, i.e. string and user-defined enumeration. Property `complexTypeName` has string type, while `complexTypeAbstract` is a user-defined enumeration. The enumeration value is `true` or `false`.

Properties are grouped in Tools, which is represented as individual tabs. Here the tool name is `UML Profile`. In a tool, sets of default value of properties may be defined. The set is represented in the drop down list called `Set`. In the figure above, a set named `complex_type` is defined. Note that in the property, there is a column named `Source` which has value `Default` or `Override`. This means that a user may either leave the default property untouched or override it respectively.

In order to add properties in UML, Rose Scripting which is a component of Rational Extensibility Interface, is used. For this project, a script that adds tagged values defined in the previous chapter is created. This script must be run on a UML model. The script is provided in a separate file.

5.2 XMI Serialization

An add-in tool for Rational Rose called Unisys Rose XML Tools 1.3.5 is used to do the XMI serialization. For this project, UML 1.3 specification and XMI 1.1 specification are used.

The result of the serialization is an XML document which has `XMI` element as the root element. `XMI` element contains several child elements. One of them is `XMI.content`. Below this child element, the UML model is defined. Some UML components are explained below.

- UML Association

  An example of a UML association when serialized into XML is as follows.

```
<!-- ============= tsas-modified::provides{3E50F3250184}    [Association] ============= -->
  <UML:Association xmi.id="G.1" name="provides" visibility="public" isSpecification="false"
                   isRoot="false" isLeaf="false" isAbstract="false">
    <UML:Association.connection>
      <!-- ===== tsas-modified::provides{3E50F3250184}.(Role1)    [AssociationEnd] ===== -->
      <UML:AssociationEnd xmi.id="G.2" name="" visibility="public" isSpecification="false"
                          isNavigable="true" ordering="unordered" aggregation="none"
                          targetScope="instance" changeability="changeable"
                          type="S.061.1543.37.4">
        <UML:AssociationEnd.multiplicity>
          <UML:Multiplicity>
            <UML:Multiplicity.range>
              <UML:MultiplicityRange xmi.id="id.0621443.1" lower="0" upper="-1"/>
            </UML:Multiplicity.range>
          </UML:Multiplicity>
        </UML:AssociationEnd.multiplicity>
      </UML:AssociationEnd>
      <!-- ==== tsas-modified::provides{3E50F3250184}.(Role2)    [AssociationEnd] ====== ->
      <UML:AssociationEnd xmi.id="G.3" name="" visibility="public" isSpecification="false"
                          isNavigable="true" ordering="unordered" aggregation="none"
                          targetScope="instance" changeability="changeable"
                          type="S.061.1543.37.1">
        <UML:AssociationEnd.multiplicity>
          <UML:Multiplicity>
            <UML:Multiplicity.range>
              <UML:MultiplicityRange xmi.id="id.0621443.2" lower="1" upper="1"/>
            </UML:Multiplicity.range>
          </UML:Multiplicity>
        </UML:AssociationEnd.multiplicity>
      </UML:AssociationEnd>
    </UML:Association.connection>
  </UML:Association>
```

This example is a part of case study (see chapter 6), i.e. association which has name `provides` which connects class `ServiceProvider` and `Service`.

A `UML:Association` element represents an association. This element has an identification i.e. `xmi.id` as its attribute. The name of association is in the attribute `name`. A `UML:Association` element has two `UML:AssociationEnd` elements as its descendant. `UML:AssociationEnd` element represents the UML association role. This element also has an identification i.e. `xmi.id` and `name` as its attributes.

`UML:AssociationEnd` element also has some other attributes, e.g. `isNavigable`, `aggregation`, and `type`. The value of `type` attribute is an identification of a UML class which the shows the end of the association. If the association is not an aggregation, then the value of `aggregation` attribute is `none`. If it is, then the value is `aggregate`. If the UML association is a unidirectional association, then in one of the UML:AssociationEnd the value of isNavigable attribute is false.

UML:AssociationEnd element has UML:AssociationEnd.multiplicity as its child element. This shows the multiplicity of the association end. Attribute lower and upper of UML:MultiplictyRange, a descendant element of UML:AssociationEnd.multiplicity element, shows the value of multiplicity. If the value of the attribute is -1, then it means that the multiplicity is unbounded.

- UML Attribute

  An example of a UML attribute when serialized into XML is as follows.

```
<!-- =========== tsas-modified::ServiceProvider.provider_id    [Attribute] ============= -->
<UML:Attribute xmi.id="S.061.1543.37.2" name="provider_id" visibility="private"
                     isSpecification="false" ownerScope="instance" changeability="changeable"
                     targetScope="instance" type="G.28">
   <UML:StructuralFeature.multiplicity >
      <UML:Multiplicity>
         <UML:Multiplicity.range>
            <UML:MultiplicityRange xmi.id="id.0621443.19" lower="1" upper="1"/>
         </UML:Multiplicity.range>
      </UML:Multiplicity>
   </UML:StructuralFeature.multiplicity>
   <UML:Attribute.initialValue>
      <UML:Expression language="" body=""/>
   </UML:Attribute.initialValue>
</UML:Attribute>
```

  This example is a part of case study (see chapter 6), i.e. class ServiceProvider.

  A UML:Attribute element represents a UML attribute. This element has identification in xmi.id attribute, name in the name attribute, and type in the type attribute. The value of type attribute is the identification of a data type. The multiplicity of this attribute is represented in the same way as in the multiplicity of association end, only it is under UML:StructuralFeature.multiplicity instead of UML:AssociationEnd.multiplicity. The initial value of the UML attribute is shown in attribute body of element UML:Expression, a child element of UML:Attribute.intialValue.

The datatype of a UML attribute is declared in `UML:DataType` element. This element is in the same level as `UML:Class` element. An example of a representation of a datatype is as follows.

```
<!-- ==================== string    [DataType] ==================== -->
<UML:DataType xmi.id="G.28" name="string" visibility="public" isSpecification="false"
                        isRoot="false" isLeaf="false" isAbstract="false"/>
```

Just like `UML:Class` element, `UML:DataType` has an identification in `xmi.id` attribute and name in `name` attribute.

- UML Generalization

  The child class of a generalization when serialized into XML is as follows.

```
<!-- ==================== tsas-modified::Service    [Class] ==================== -->
<UML:Class xmi.id="S.061.1543.37.4" name="Service" visibility="public" isSpecification="false"
                isRoot="false" isLeaf="true" isAbstract="false" isActive="false" namespace="G.0"
                generalization="G.30">
   <UML:Namespace.ownedElement>
      <UML:Generalization xmi.id="G.30" name="" visibility="public" isSpecification="false"
                      discriminator="" child="S.061.1543.37.4" parent="S.061.1543.37.7"/>
   </UML:Namespace.ownedElement>
   <UML:Classifier.feature>
      <!-- ============= tsas-modified::Service.service_id    [Attribute] ============= -->
      <UML:Attribute … … …>

      … … …

      </UML:Attribute>
   </UML:Classifier.feature>
</UML:Class>
```

This example is a part of case study (see chapter 6), i.e. class `Service` as the child class and class `ServiceTemplate` (below) as parent class.

The difference of a `UML:Class` explained above (class that is not involved in a generalization) with the one that is a child class of a generalization is that the later type has `generalization` attribute. Moreover, it also has `UML:Namespace.ownedElement` as its child element, which has `UML:Generalization` as its child element. `UML:Generalization` element represents a UML generalization. This element has identifier in `xmi.id` attribute. Moreover, it shows the child entity identifier in `child` attribute and parent entity identifier in `parent` attribute.

The UML:Class element that represents the parent class of a generalization has specialization attribute which refers to the UML:Generalization identification. This is shown in the example below.

```
<!-- ================== tsas-modified::ServiceTemplate    [Class] ================== -->
<UML:Class xmi.id="S.061.1543.37.7" name="ServiceTemplate" visibility="public"
           isSpecification="false" isRoot="true" isLeaf="false" isAbstract="false"
           isActive="false" namespace="G.0" specialization="G.30">
… … …
</UML:Class>
```

- Stereotype

  A UML Stereotype when serialized into XML is as follows.

```
<UML:Stereotype xmi.id="S.126.1618.29.0" name="XSDelementComplex" visibility="public"
           isSpecification="false" isRoot="false" isLeaf="false" isAbstract="false" icon=""
           baseClass="Class" extendedElement="S.126.1618.28.1 S.126.1618.28.4
           S.126.1618.28.17 S.126.1618.28.24"/>
```

This example is a part of case study (see chapter 6), i.e. stereotype XSDelementComplex.

UML:Stereotype element represents a stereotype. It has xmi.id attribute as its identification. It also has name attribute that shows the stereotype name. Moreover there are two more important attributes, i.e. baseClass and extendedElement. The baseClass attribute shows which UML Component the stereotype has extends. In the example the stereotype is for UML Class. The extendedElement attribute contains the identifications of extended UML components.

- Tagged Values

  A UML Tagged Value when serialized into XML is as follows.

```
<UML:TaggedValue xmi.id="XX.7.1618.29.7" tag="RationalRose$UML Profile:elementName"
           value="properties" modelElement="S.126.1618.28.3"/>
```

This example is a part of case study (see chapter 6), i.e. tagged value elementName=properties.

UML:TaggedValue element represents a tagged value. This element is identified by xmi.id attribute. The tagged name of the tagged value is represented in the tag attribute. The value of the tagged is represented in the value attribute. Attribute modelElement indicates the identification of UML component to which the tagged value applies.

5.3 Transformation Using XSLT

XSLT file is created manually. An XML tool is used to create and edit XSLT files. The tool is XML Spy 5.3. It is also used to do the transformation. The XSLT processor used in the example is built-in in the XML Spy 5.3 tools.

There are 11 XSLT files.

- main.xslt
- attribute.xslt
- attribute_group.xslt
- complex_type.xslt
- containment.xslt
- datatype_and_multiplicity.xslt
- derivation.xslt
- element.xslt
- enumeration.xslt
- model_group.xslt
- reference.xslt.

File main.xslt and complex_type.xslt are given in appendix B. Others are provided in separate files. The file contains one or more templates. The file name shows the XML construct (the result of mapping a UML component). Thus, element.xslt contains templates umlClass2Element, umlAttribute2Element, and umlAssociation2Element. There are three exceptions as follows.

- File main.xslt imports all other files and contains the main template.
- File datatype_and_multiplicity.xslt contains template to map a data type and template to map multiplicity of a UML attribute and UML association end.
- File enumeration.xslt contain a template that maps class with stereotype enumeration.

The logic of the transformation is as follows. In the main template, each UML class and UML association is identified. The stereotype for those elements is checked. If a UML class has a stereotype `XSDcomplexType`, then template `umlClass2ComplexType` is called. If it has stereotype `XSDelementComplex`, then template `umlClass2Element` and `umlClass2ComplexType` are called. This is also the case if the class has no stereotype. For class with stereotype enumeration, the class is mapped into enumeration in XML (as a restriction of simple type).

If a UML association has a stereotype `XSDcomplexType`, then template `umlAssociation2ComplexType` is called. If it has a stereotype `XSDelement`, then template `umlAssociation2element` is called. This is also the case if the UML association has no stereotype. Note UML association and aggregation are treated the same.

In general, the templates examine the tagged values for element (class, attribute or association). Then, according to the tagged values, the XML Schema is generated. If the tagged values are not present then the default values apply.

In addition, inside the template `umlClass2ComplexType`, template `umlGeneralization2Derivation` is called if there is a generalization which is mapped into derivation. A template to examine UML attributes for the class (and for the parent class in case of generalization which mapped into containment) is called. If it has stereotype `XSDattribute`, then `umlAttribute2Attribute` is called. If it has stereotype `XSDelement` or no stereotype, then template `umlAttribute2Element` is called. Moreover, template to examine UML Association for the class is called. The stereotype that is evaluated is `XSDcontainment`. The template is `umlAssociation2Containment`.

Templates `umlAttribute2ModelGroup` is called if tagged value `modelGroupName` is present and not empty. For template `umlAttribute2AttributeGroup`, it is similar with tagged value is `attributeGroupName`. Moreover, Template `umlAssociation2Reference` is called if UML association is mapped into `XSDreference`.

5.4 Summary

This chapter explains about the implementation of the approach in chapter 4. Tools that are used are Rational Rose, Unisys Rational Tools, and XML Spy. The mechanism to add UML profile in Rational Rose is explained. So is the XML document as a result of XMI serialization. Moreover, the logic of XSLT file is also presented.

6 Case Study

This chapter contains a case study to show the feasibility of the approach described in chapter 4. Section 6.1 explains the case study, TSAS. Section 6.2 shows the UML Profile defined for the case study. The last sections show the transformation of TSAS to the required XML Schema and a summary of this chapter.

6.1 TSAS

Telecommunication Service Access and Subscription (TSAS) specification is an OMG document that provides a standardized framework for service provisioning [13].

TSAS is selected for case study in this project, because it is in the telecommunication application domain. The model contains the most important UML constructs. Moreover, this case is used in other research, such that different view points can be illustrated.

TSAS defines three domains, i.e. *Consumer Domain*, *Retailer Domain*, and *Service Provider Domain*. Each domain has one or more roles. Consumer domain has two roles, *end user* role and *subscriber* role. Retailer domain has one role, i.e. *retailer* role. Retailer connects service providers with consumer. Service provider domain also has one role, *service provider* role. Service provider is the one which offers services to end user or subscriber through a retailer.

In this project, only the subscription segment of TSAS will be used. That segment manages services and contract information between end user/subscriber with retailer and retailer with service provider. This project defines a modified subscription information model of TSAS. A modification is made because of the following reasons:

- The given specification is not complete and not consistent
- The specification should cover the subset of UML described in the UML section (see section 2.2)

The information model used in this project consists of the following entities:

1. ServiceProvider

   It represents service provider role. It provides many services for customers. The attributes of this entity are identifier of the service provider and some properties. The properties may contain the name, address, bank account and other details of service provider.
2. ServiceTemplate

   This is a generalization of service. Its attributes are template name and some properties.
3. Service

   This is a specialization of service. It defines the service in a more detailed level.
4. ServiceContract

   In order to subscribe to a service, a subscriber has to subscribe the service. This is done via a contract. It enables the end users to use the service. The attributes of this entity are identification of the contract and properties. The properties should be defined by retailer.
5. Subscriber

   As mention above, subscriber is the one who has a contract with retailer and subscribes the service. The attribute of subscriber is similar to service provider. A subscriber may have one or more end users.
6. EndUser

   End user is the actual user of the service. Besides user profile, it also has security profile as its attribute.
7. SAG

   Several end users, which have the same service characteristics and usage permission, can be grouped into Subscription Assignment Group (SAG). SAG can also contain one or more other SAGs.
8. ServiceProfile

   Service profile customizes the usage of service contract for a group of users.

9. PropertyList, Property, propertyType, and propertyMode

   PropertyList is a list of properties that an entity can have. It consists of several Property entities. A property has a name, value, type, and mode. The type of a property could be string, Boolean, unsigned long. The mode of a property is mandatory, read only or normal.

The information model described above can be depicted UML class diagram as in figure 6.1. The model below is produced using a UML tool, i.e. Rational Rose.

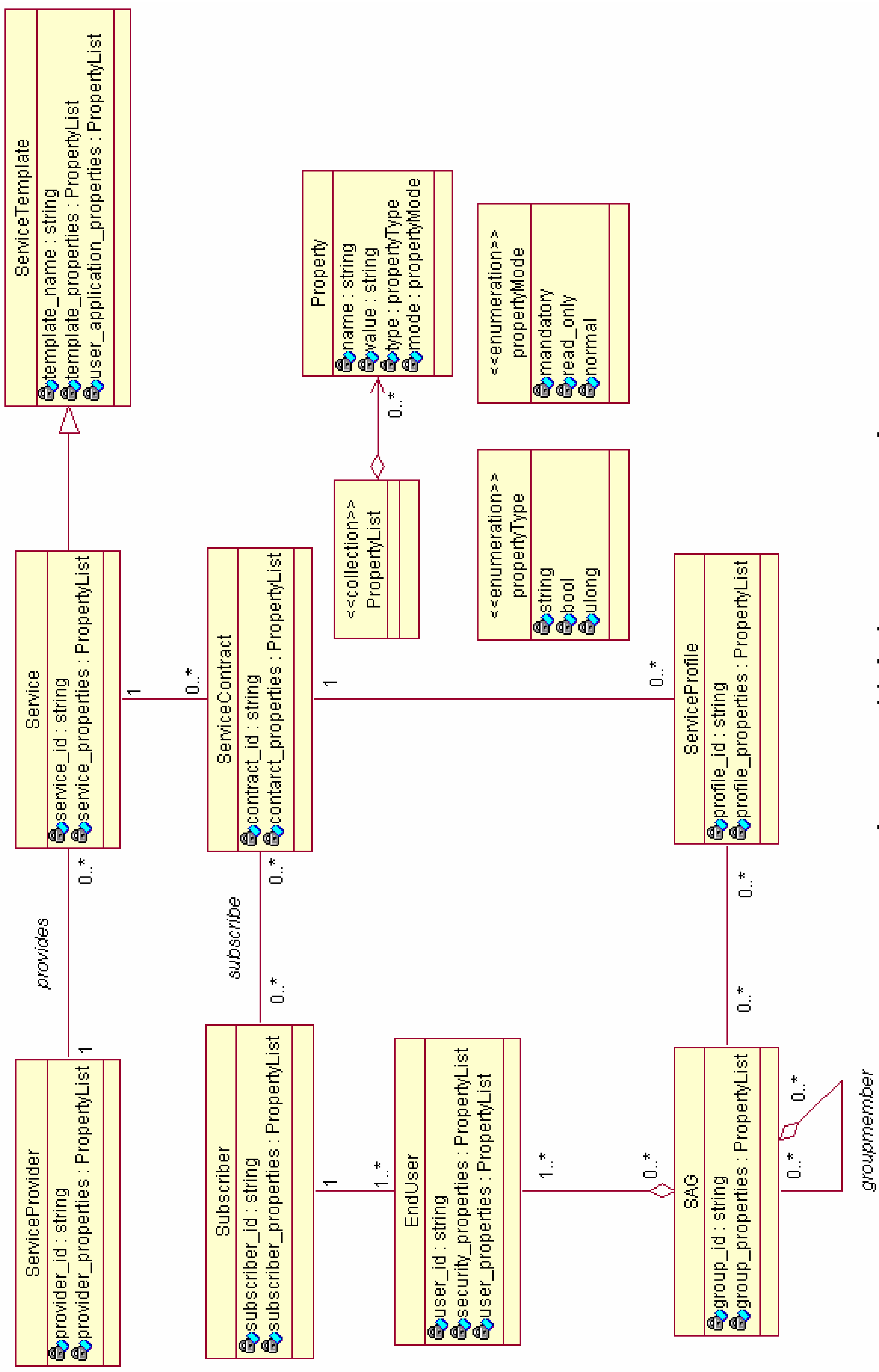


Figure 6.1 Information Model of Subscription Segment of TSAS

6.2 UML Profile Applied for The Case Study

UML Profile is used in the class diagram in figure 6.1. Those are listed in the tables below. Table 6.1 contains the stereotypes and tagged values for all UML classes of the TSAS model.

Table 6.1 UML Profile for TSAS: UML Class

UML Component	***Stereotype***	***Tagged Values***
ServiceProvider	XSDelementComplex	complexTypeCompositionKind=sequence
		complexTypeAbstract=true
Service	XSDelementComplex	elementName=Services
		mixed=true
		complexTypeCompositionKind=sequence
		complexTypeName=TSAS-service
		attributeGroupName=service-attr-group
ServiceTemplate	XSDcomplexType	mixed=true
		complexTypeName=TSAS-ServiceTemplate
		complexTypeAbstract=true
		modelGroupName=service-template-properties
		modelGroupCompositionKind=sequence
Subscriber	XSDcomplexType	mixed=true
		complexTypeCompositionKind=sequence
		modelGroupName= property
ServiceContract	XSDelementComplex	complexTypeCompositionKind=choice
		complexTypeName=TSAS-contract
		modelGroupName=service-contract-properties
		modelGroupCompositionKind=sequence
EndUser	XSDelementComplex	complexTypeCompositionKind=sequence
		complexTypeAbstract=true
SAG	XSDcomplexType	complexTypeAbstract=true
ServiceProfile		elementName=TSAS-profile
PropertyList	Collection	
Property		attributeGroupName=type-and-mode
propertyType	Enumeration	
propertyMode	Enumeration	

Table 6.2 below shows the stereotypes and attributes for all UML attributes of the TSAS model.

Table 6.2 UML Profile for TSAS: UML Attribute

UML Component	***Stereotype***	***Tagged Values***
provider_id	XSDattribute	attributeName=id-tsas-provider
provider_properties	XSDelement	elementName=properties
		elementAbstract=true
service_id	XSDattributeGroup	
service_properties	XSDelement	elementName=properties
template_name	XSDattribute	attributeName=name

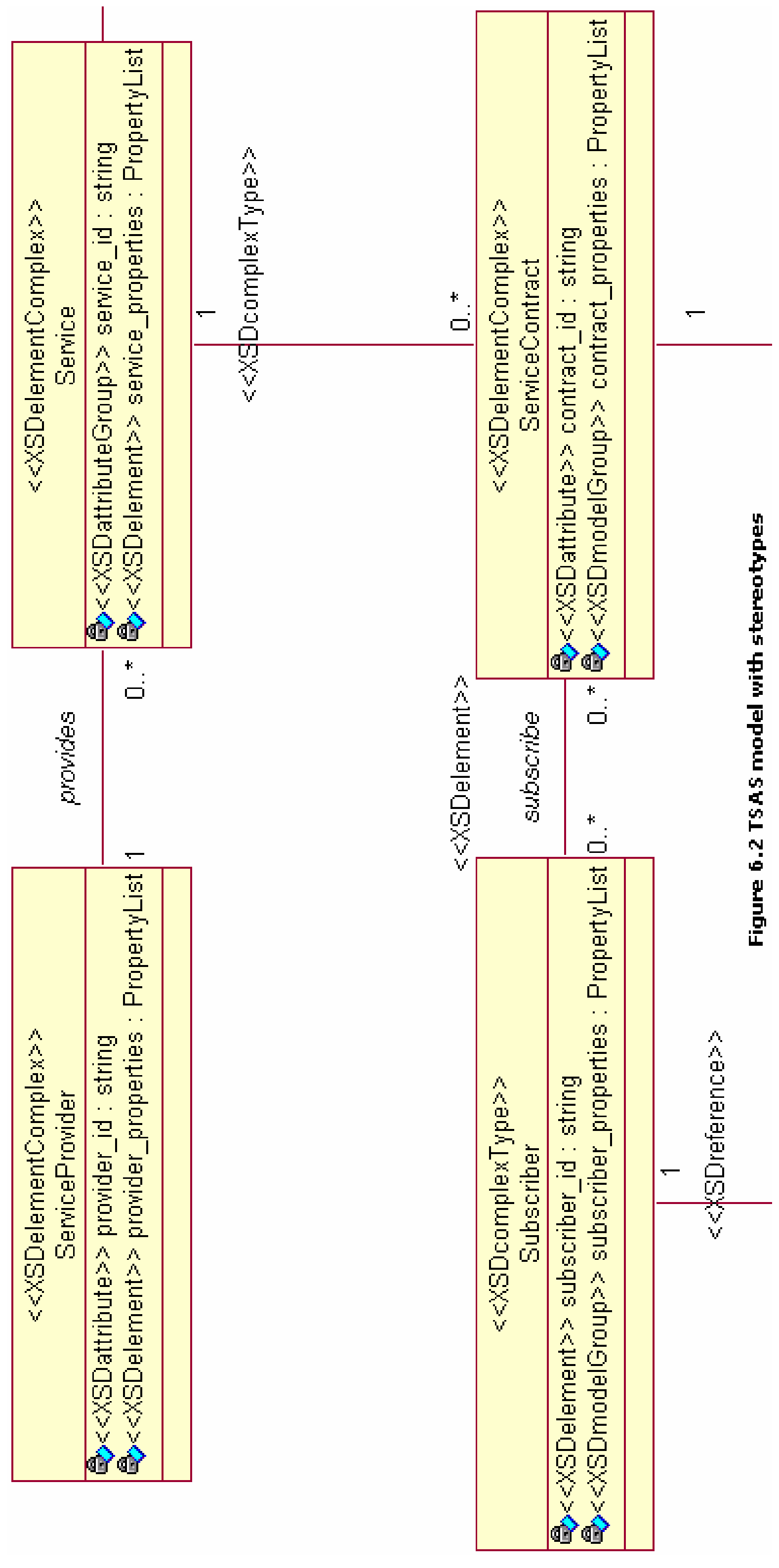
		(initial value=tsas)
template_properties	XSDmodelGroup	elementName=template
		elementAbstract=true
user_application_properties	XSDmodelGroup	elementName=user-application
subscriber_id	XSDelement	elementName=id-tsas-subscriber
subscriber_properties	XSDmodelGroup	
contract_id	XSDattribute	attributeName=no
contract_properties	XSDmodelGroup	
user_id	XSDelement	
security_properties	XSDelement	elementName=property-security
		elementAbstract=true
user_properties		elementName=properties-user
group_id	XSDattribute	
group_properties		elementName=property
profile_id		
profile_properties	XSDelement	
Name	XSDattribute	
Value	XSDattribute	(initial value=0)
Type	XSDattributeGroup	
Mode	XSDattributeGroup	
String		
Bool		
Ulong		
Mandatory		
read_only		
Normal		

Table 6.3 shows the stereotypes and tagged values for all UML associations and UML generalization of the TSAS model.

Table 6.3 UML Profile for TSAS: UML Relationship

UML Component	***Stereotype***	***Tagged Values***
UML Association		
Provides		(Profile B) associationEndToClass=containment
Service-ServiceContract	XSDcomplexType	(Profile A) associationEndToClass=instanceOf
Subscribe	XSDelement	
ServiceContract-ServiceProfile	XSDreference	(Profile B) associationEnd=none
Subscriber-EndUser	XSDreference	
SAG-ServiceProfile	XSDcomplexType	(Profile A) associationEndToClass=containment
UML Aggregation		
SAG-EndUser	XSDcontainment	(Profile A) associationEnd=none
		(Profile B) associationEndToClass=instanceOf
Groupmember	XSDcontainment	
PropertyList-Property	XSDcontainment	(Profile A) associationEnd=none
UML Generalization		
Service-ServiceTemplate	XSDderivation	

Figure 6.2 below shows part of TSAS model added with stereotypes. The tagged values defined for class Service is given in figure 5.2.



<<XSDelementComplex>>
Service
<<XSDattributeGroup>> service_id : string
<<XSDelement>> service_properties : PropertyList
<<XSDelementComplex>>
ServiceProvider
<<XSDattribute>> provider_id : string
<<XSDelement>> provider_properties : PropertyList
<<XSDelementComplex>>
ServiceContract
<<XSDattribute>> contract_id : string
<<XSDmodelGroup>> contract_properties : PropertyList
<<XSDcomplexType>>
Subscriber
<<XSDelement>> subscriber_id : string
<<XSDmodelGroup>> subscriber_properties : PropertyList
provides
subscribe
<<XSDcomplexType>>
<<XSDelement>>
<<XSDreference>>
1
0..*
1
0..*
0..*
0..*
1
1

Figure 6.2 TSAS model with stereotypes

6.3 Transformation of TSAS to XML Schema

The UML class diagram together with the profile is transformed into XML Schema using the XSLT stylesheet obtained from implementation. The result is given below.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="ServiceProvider" type="ServiceProvider"/>
    <xs:complexType name="ServiceProvider" abstract="true">
        <xs:sequence>
            <xs:element name="properties" type="PropertyList" abstract="true"/>
        </xs:sequence>
        <xs:attribute name="id-tsas-provider" type="xs:string"/>
    </xs:complexType>
    <xs:element name="Service" type="TSAS-service"/>
    <xs:complexType name="TSAS-service" mixed="true">
        <xs:complexContent mixed="true">
            <xs:extension base="TSAS-ServiceTemplate">
                <xs:sequence>
                    <xs:element name="properties" type="PropertyList"/>
                    <xs:group ref="service-template-properties"/>
                </xs:sequence>
                <xs:attributeGroup ref="service-attr-group"/>
                <xs:attribute name="id" type="xs:ID"/>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <xs:attributeGroup name="service-attr-group">
        <xs:attribute name="service_id" type="xs:string"/>
    </xs:attributeGroup>
    <xs:complexType name="TSAS-ServiceTemplate" abstract="true" mixed="true">
        <xs:sequence>
            <xs:group ref="service-template-properties"/>
        </xs:sequence>
        <xs:attribute name="name" type="xs:string" default="tsas"/>
    </xs:complexType>
    <xs:group name="service-template-properties">
        <xs:sequence>
            <xs:element name="template" type="PropertyList" abstract="true"/>
            <xs:element name="user-application" type="PropertyList"/>
        </xs:sequence>
    </xs:group>
    <xs:complexType name="Subscriber" mixed="true">
        <xs:sequence>
            <xs:group ref="property"/>
            <xs:element name="id-tsas-subscriber" type="xs:string"/>
        </xs:sequence>
        <xs:attribute name="id" type="xs:ID"/>
        <xs:attribute name="idrefs" type="xs:IDREFS"/>
    </xs:complexType>
    <xs:group name="property">
        <xs:all>
            <xs:element name="subscriber_properties" type="PropertyList"/>
        </xs:all>
    </xs:group>
    <xs:element name="theEndUser">
        <xs:complexType>
            <xs:sequence/>
            <xs:attribute name="idref" type="xs:IDREF"/>
            <xs:attribute name="id" type="xs:ID"/>
        </xs:complexType>
    </xs:element>
```

```
<xs:element name="ServiceContract" type="TSAS-contract"/>
<xs:complexType name="TSAS-contract">
    <xs:sequence>
        <xs:group ref="service-contract-properties"/>
    </xs:sequence>
    <xs:attribute name="no" type="xs:string"/>
    <xs:attribute name="id" type="xs:ID"/>
    <xs:attribute name="idrefs" type="xs:IDREFS"/>
</xs:complexType>
<xs:group name="service-contract-properties">
    <xs:sequence>
        <xs:element name="contract_properties" type="PropertyList"/>
    </xs:sequence>
</xs:group>
<xs:element name="theServiceProfile">
    <xs:complexType>
        <xs:sequence/>
        <xs:attribute name="idref" type="xs:IDREF"/>
        <xs:attribute name="id" type="xs:ID"/>
    </xs:complexType>
</xs:element>
<xs:element name="EndUser" type="EndUser"/>
<xs:complexType name="EndUser" abstract="true">
    <xs:sequence>
        <xs:element name="user_id" type="xs:string"/>
        <xs:element name="properties-security" type="PropertyList" abstract="true"/>
        <xs:element name="properties-user" type="PropertyList"/>
        <xs:element name="theSAG" type="SAG" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:ID"/>
    <xs:attribute name="idrefs" type="xs:IDREFS"/>
</xs:complexType>
<xs:element name="theSubscriber">
    <xs:complexType>
        <xs:sequence/>
        <xs:attribute name="idref" type="xs:IDREF"/>
        <xs:attribute name="id" type="xs:ID"/>
    </xs:complexType>
</xs:element>
<xs:complexType name="SAG" abstract="true">
    <xs:all>
        <xs:element name="property" type="PropertyList"/>
        <xs:element name="theSAG2" minOccurs="0">
            <xs:complexType>
                <xs:sequence/>
                <xs:attribute name="idref" type="xs:IDREF"/>
            </xs:complexType>
        </xs:element>
        <xs:element ref="EndUser"/>
        <xs:element name="theSAG1" minOccurs="0">
            <xs:complexType>
                <xs:sequence/>
                <xs:attribute name="idref" type="xs:IDREF"/>
            </xs:complexType>
        </xs:element>
    </xs:all>
    <xs:attribute name="group_id" type="xs:string"/>
    <xs:attribute name="id" type="xs:ID"/>
</xs:complexType>
<xs:element name="TSAS-profile" type="ServiceProfile"/>
<xs:complexType name="ServiceProfile">
    <xs:all>
        <xs:element name="profile_id" type="xs:string"/>
        <xs:element name="profile_properties" type="PropertyList"/>
    </xs:all>
    <xs:attribute name="id" type="xs:ID"/>
```

```
        <xs:attribute name="idrefs" type="xs:IDREFS"/>
    </xs:complexType>
    <xs:element name="PropertyList" type="PropertyList"/>
    <xs:complexType name="PropertyList">
        <xs:all>
            <xs:element ref="Property"/>
        </xs:all>
    </xs:complexType>
    <xs:element name="Property" type="Property"/>
    <xs:complexType name="Property">
        <xs:all/>
        <xs:attributeGroup ref="type-and-mode"/>
        <xs:attribute name="name" type="xs:string"/>
        <xs:attribute name="value" type="xs:string" default="0"/>
        <xs:attribute name="id" type="xs:ID"/>
    </xs:complexType>
    <xs:attributeGroup name="type-and-mode">
        <xs:attribute name="type" type="propertyType"/>
        <xs:attribute name="mode" type="propertyMode"/>
    </xs:attributeGroup>
    <xs:simpleType name="propertyType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="string"/>
            <xs:enumeration value="bool"/>
            <xs:enumeration value="ulong"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="propertyMode">
        <xs:restriction base="xs:string">
            <xs:enumeration value="mandatory"/>
            <xs:enumeration value="read_only"/>
            <xs:enumeration value="normal"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:element name="provides">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="theService" minOccurs="0" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence/>
                        <xs:attribute name="idref" type="xs:IDREF"/>
                    </xs:complexType>
                </xs:element>
                <xs:element ref="ServiceProvider"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:complexType name="ServiceContract-Service">
        <xs:sequence>
            <xs:element name="theServiceContract" type="TSAS-contract" minOccurs="0"
maxOccurs="unbounded"/>
            <xs:element name="theService">
                <xs:complexType>
                    <xs:sequence/>
                    <xs:attribute name="idref" type="xs:IDREF"/>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="subscribe">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="theServiceContract" minOccurs="0" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence/>
                        <xs:attribute name="idref" type="xs:IDREF"/>
                    </xs:complexType>
                </xs:element>
                <xs:element name="theSubscriber" minOccurs="0" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence/>
                        <xs:attribute name="idref" type="xs:IDREF"/>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:complexType name="ServiceProfile-SAG">
        <xs:sequence>
            <xs:element ref="TSAS-profile" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="theSAG" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:sequence/>
                    <xs:attribute name="idref" type="xs:IDREF"/>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

6.4 Summary

This chapter contains a case study to show that the implementation works. The case study is UML class diagram of a modification of subscription segment of TSAS. It is described in the first section. Then, UML profile is added to the class diagram. The profile is shown in the table 6.1. The result of transformation is shown in the last section.

7 Conclusion

This chapter contains a summary of the project in section 7.1, a comparison to other work in section 7.2, and suggestion for future research in section 7.3.

7.1 Summary

This problem of this project is how to map a UML Model onto XML Schema taking into account alternatives. In this project, the alternatives are organized in a two steps. The first step results in a schema skeleton. In the second step, the schema skeleton is further refined. This second step is studied in detail. The alternatives are represented using a UML Profile (as stereotypes and tagged values). This Profile together with a UML Model (in this project a subset of it, i.e. UML Class Diagram) is transformed into XML Schema.

The transformation is done by using XMI serialization and an XSLT sytlesheet. XMI serialization is used to generate an XML document from UML model augmented with UML profile. An XSLT stylesheet is created to transform the result from XML serialization into XML Schema.

This project has identified alternative mappings from UML model onto XML schema. It also provided a mechanism to map a UML model onto XML Schema as intended in the problem statement. An implementation also has been given to show the feasibility of the approach. Moreover, a case study has been taken to show the mapping mechanism using the implementation. However, in this project there are two XML schema metamodel construct that are not discussed, i.e. Simple Type and Substitution.

The strong points of the approach in this project are:

- With this approach, the generation of XML Schema can be guided in a detailed way.

- The XSLT stylesheet is extensible because it has a clear structure. If for example another UML component is added, then we can add a template on the stylesheet.

However, there are several drawbacks:

- Not all the construct of XML schema can be represented in UML, for example derivation by restriction, simple type, ordering, etc.
- Because the tagged value represent XML Schema in a very detailed way, then the user who creates UML model to model the schema must have knowledge about XML Schema. However, there are some default values that simplify this.
- The constraints from XML Schema cannot all be captured in UML Profile. In our case, XSLT which contains rules is used to impose the constraint.
- Processing of XSLT transformations is rather slow, especially for large UML models.

7.2 Comparison to Other Work

The approach in this project is compared to Carlson's [3] and Malik's [1].

Comparing this project to Carlson's Modeling XML Application with UML, there are similarities and differences.

- Similarities
  - use of UML Profile
  - use of XMI serialization and XSLT
- Differences
  - Generation of alternatives

    In this project, two steps of alternatives generation are used. Carlson uses a different method (he defined some criteria for strict and relaxed XML Schema). He also defined UML Profile to control the degree of strictness. There is an intersection between his profile and the profile in this project. Some stereotypes and tagged values are the same.

- mapping the UML association

  There is quite a big difference on mapping UML associations. Carlson only mapped association end (role) and multiplicity, but not the association (name). In this project, the approach is based on the UML metamodel.

The approach in this project is similar to Malik’s. The difference is the way of identifying the mappings. She maps the UML constructs into XML Schema constructs. But, not all the richness of XML Schemas can be captured by UML. To do this, she extends UML with stereotypes. She uses the stereotypes from Carlson. In her approach, XMI is also used for the transformation.

7.3 Suggestion for Future Research

As mentioned above, constraints from XML Schemas cannot be imposed solely by a UML Profile. A mechanism to impose this in UML is needed in future research. This will enable the use of technology other than XSLT and the generation of XML Schema could be automated (the generation could be a form of add-in in a UML tool).

The result of this project could be useful for other research, i.e. the reverse mapping (mapping from XML Schema into UML).

References

[1] Ayesha Malik. *Design XML Schema using UML*. 1 February 2003. http://www-106.ibm.com/developerworks/xml/library/x-umlschem/

[2] Dave Carlson. *Modeling XML Vocabularies with UML*. 2001. http://www.xml.com/pub/a/2001/08/22/uml.html

[3] David Carlson. *Modeling XML Applications with UML: Practical e-Business Application*. 2001. Addison-Wesley.

[4] David Skogan. *UML as a Schema Language for XML based Data Interchange*. 1999-05-14.

[5] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*. 1999. Addison-Wesley Longman, Inc.

[6] H. M. Deitel, P. J. Deitel, T. R. Nieto, T. M. Lin and P. Sadhu. *XML How To Program*. 2001. Prentice-Hall, Inc.

[7] Ivan Kurtev, Klaas van den Berg, and Mehmed Aksit. *UML to XML-Schema Transformation: a Case Study in Managing Alternative Model Transformations in MDA*. 2003.

[8] Jiri Jirat. *XMI Reference*. Zvon.org. 2001. http://www.zvon.org/xxl/XMI/Output/index.html

[9] John Hsia. *Creating Custom Model Properties*. 1999. http://www.therationaledge.com/rosearchitect/mag/archives/9901/extend.html

[10] Linda Bird, Andrew Goodchild, Terry Halpin. *Object Role Modeling and XML Schema*.

[11] Milosav Nic. *XSLT Reference*. Zvon.org. 2002. http://www.zvon.org/xxl/XSLTreference/Output/index.html

[12] OMG Analysis and Design Task Force. *White Paper on The Profile Mechanism version 1.0* (OMG Document ad/99-04-07). April 1999.

[13] OMG. *Telecommunication Service Access and Subscription* (OMG Document telecom/00-05-05). 2000.

[14] OMG. *OMG Unified Modeling Language Specification version 1.4* (OMG Document 01-09-67). September 2001.

[15] OMG. *OMG XML MetadataInterchange (XMI) Specification version 1.2* (OMG Document 02-01-01). January 2002.

[16] W3C. *Extensible Markup Language (XML) 1.0 (Second Edition)*. W3C Recomendation. 6 October. 2000.
http://www.w3.org/TR/REC-xml

[17] W3C. *XML Linking Language (Xlink) version 1.0*. W3C Recommendation. 27 Juni 2001.
http://www.w3.org/TR/xlink/

[18] W3C. *XML Path Language (Xpath) version 1.0*. W3C Recommendation. 16 November 1999.
http://www.w3.org/TR/xpath

[19] W3C. *XML Schema Part 0: Primer*. W3C Recommendation. 2 May 2001.
http://www.w3.org/TR/xmlschema-0/

[20] W3C. *XML Schema Part1: Structures*. 2 May 2001.
http://www.w3.org/TR/xmlschema-1/

[21] W3C. *XSL Transformation (XSLT) version 1.0*. W3C Recommendation. 16 November 1999.
http://www.w3.org/TR/xslt

Appendix A

UML Profile as outlined in this report.

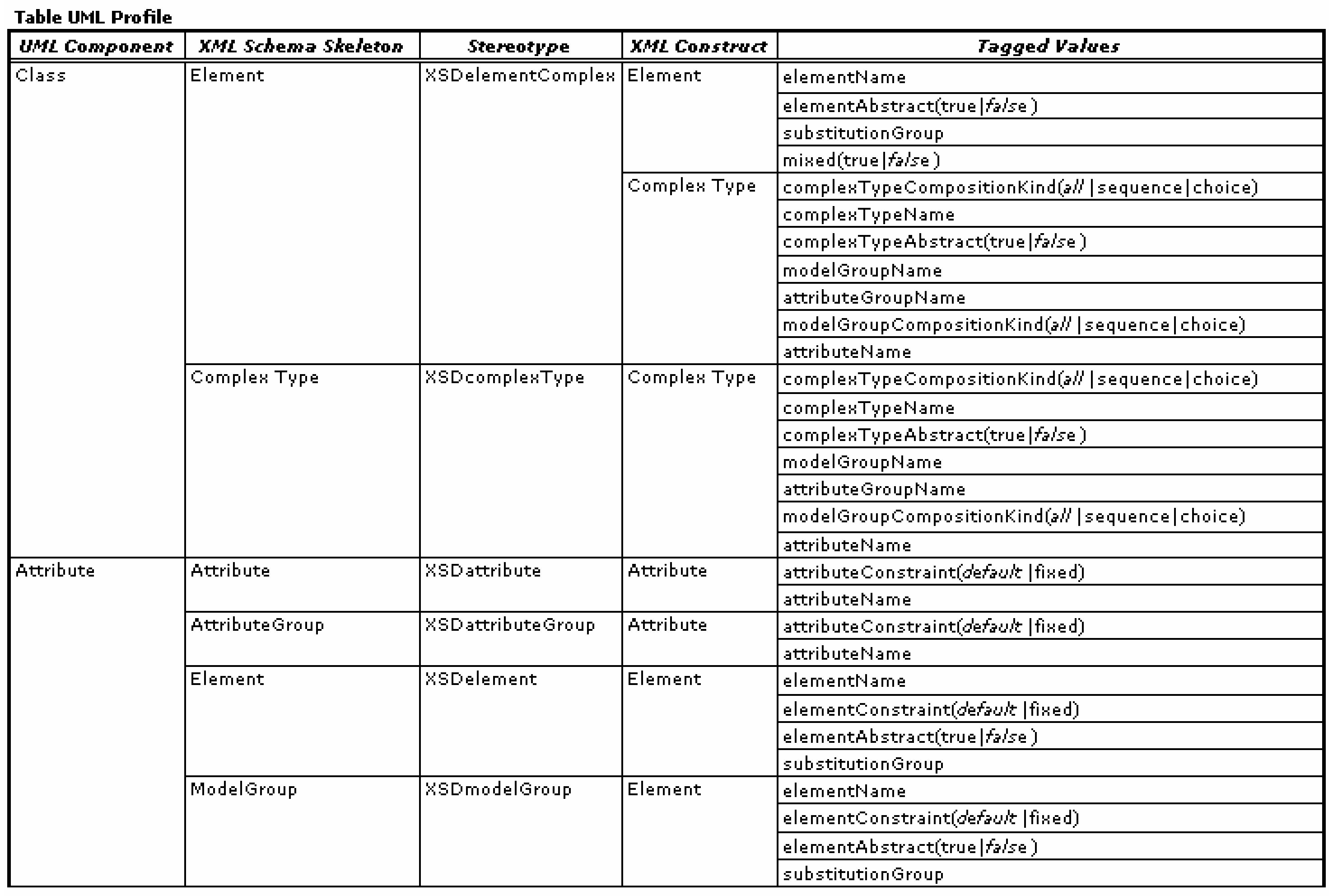
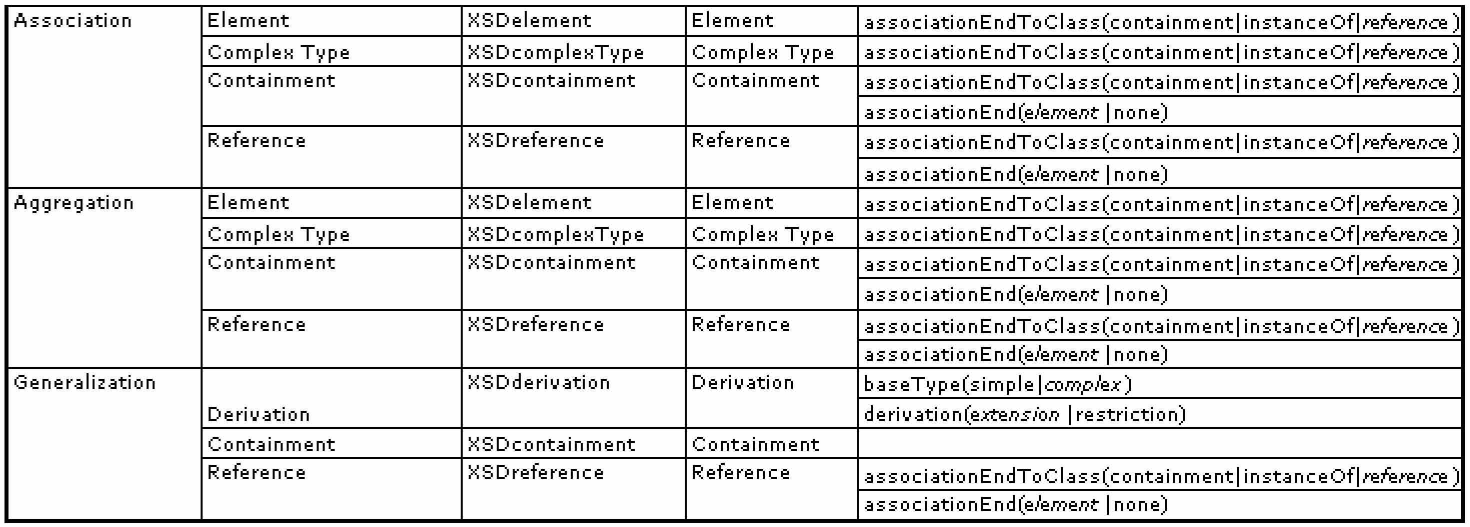


Table UML Profile

UML Component	*XML Schema Skeleton*	*Stereotype*	*XML Construct*	*Tagged Values*
Class	Element	XSDelementComplex	Element	elementName
				elementAbstract(true|*false*)
				substitutionGroup
				mixed(true|*false*)
			Complex Type	complexTypeCompositionKind(*all* |sequence|choice)
				complexTypeName
				complexTypeAbstract(true|*false*)
				modelGroupName
				attributeGroupName
				modelGroupCompositionKind(*all* |sequence|choice)
				attributeName
	Complex Type	XSDcomplexType	Complex Type	complexTypeCompositionKind(*all* |sequence|choice)
				complexTypeName
				complexTypeAbstract(true|*false*)
				modelGroupName
				attributeGroupName
				modelGroupCompositionKind(*all* |sequence|choice)
				attributeName
Attribute	Attribute	XSDattribute	Attribute	attributeConstraint(*default* |fixed)
				attributeName
	AttributeGroup	XSDattributeGroup	Attribute	attributeConstraint(*default* |fixed)
				attributeName
	Element	XSDelement	Element	elementName
				elementConstraint(*default* |fixed)
				elementAbstract(true|*false*)
				substitutionGroup
	ModelGroup	XSDmodelGroup	Element	elementName
				elementConstraint(*default* |fixed)
				elementAbstract(true|*false*)
				substitutionGroup



Association	Element	XSDelement	Element	associationEndToClass(containment|instanceOf|*reference*)
	Complex Type	XSDcomplexType	Complex Type	associationEndToClass(containment|instanceOf|*reference*)
	Containment	XSDcontainment	Containment	associationEndToClass(containment|instanceOf|*reference*)
				associationEnd(*element* |none)
	Reference	XSDreference	Reference	associationEndToClass(containment|instanceOf|*reference*)
				associationEnd(*element* |none)
Aggregation	Element	XSDelement	Element	associationEndToClass(containment|instanceOf|*reference*)
	Complex Type	XSDcomplexType	Complex Type	associationEndToClass(containment|instanceOf|*reference*)
	Containment	XSDcontainment	Containment	associationEndToClass(containment|instanceOf|*reference*)
				associationEnd(*element* |none)
	Reference	XSDreference	Reference	associationEndToClass(containment|instanceOf|*reference*)
				associationEnd(*element* |none)
Generalization		XSDderivation	Derivation	baseType(simple|*complex*)
	Derivation			derivation(*extension* |restriction)
	Containment	XSDcontainment	Containment	
	Reference	XSDreference	Reference	associationEndToClass(containment|instanceOf|*reference*)
				associationEnd(*element* |none)

Appendix B

main.xslt

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:UML="href://org.omg/UML/1.3" exclude-result-prefixes="UML"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <!-- import other xslts -->

    <!-- element.xslt contains tempates: umlClass2Element, umlAttribute2Element, umlAssociation2Element -->
    <xsl:import href="element.xslt"/>
    <!-- attribute.xslt contains tempates: umlAttribute2Attribute, umlAssociation2Element -->
    <xsl:import href="attribute.xslt"/>
    <!-- complex_type.xslt contains tempates: umlClass2ComplexType, umlAssociation2ComplexType,
        composition_kind, umlAttribute, umlAssociation-->
    <xsl:import href="complex_type.xslt"/>
    <!-- model_group.xslt contains tempates: umlAttribute2ModelGroup, mg_element -->
    <xsl:import href="model_group.xslt"/>
    <!-- attribute_group.xslt contains tempates: umlAttribute2AttributeGroup -->
    <xsl:import href="attribute_group.xslt"/>
    <!-- containment.xslt contains tempates: umlAssociation2Containment -->
    <xsl:import href="containment.xslt"/>
    <!-- reference.xslt contains tempates: umlAssociation2Reference -->
    <xsl:import href="reference.xslt"/>
    <!-- enumeration.xslt contains tempates: enumeration -->
    <xsl:import href="enumeration.xslt"/>
    <!-- derivation.xslt contains tempates: umlGeneralization2Derivation, derivation -->
    <xsl:import href="derivation.xslt"/>
    <!-- datatype_and_multiplicity.xslt contains tempates: choose_type, UML:Multiplicity -->
    <xsl:import href="datatype_and_multiplicity.xslt"/>

    <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

    <xsl:key name="data_types" match="//UML:Class|//UML:DataType" use="@xmi.id"/>
    <xsl:key name="associationEndA_types" match="//UML:Association"
            use="UML:Association.connection/UML:AssociationEnd[position() = 1]/@type"/>
    <xsl:key name="associationEndB_types" match="//UML:Association"
             use="UML:Association.connection/UML:AssociationEnd[position() = 2]/@type"/>

    <xsl:template match="/">
        <xsl:variable name="temp"/>
        <!-- generate schema -->
        <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
            <xsl:for-each select="//UML:Class">
                <xsl:variable name="class_name" select="@name"/>
                <xsl:variable name="class_id" select="@xmi.id"/>
                <xsl:variable name="class_id_modified" select="concat($class_id,' ')"/>
                <xsl:choose>
                    <!-- mapping uml class into element + complex type -->
                    <xsl:when test="//UML:Stereotype[(@name='XSDelement' or @name='XSDelementComplex') and
                                    @baseClass='Class' and contains(concat(@extendedElement,' '),
                                    $class_id_modified)]">
                        <xsl:call-template name="umlClass2Element">
                            <xsl:with-param name="class_name" select="$class_name"/>
                            <xsl:with-param name="class_id" select="$class_id"/>
                        </xsl:call-template>
                    </xsl:when>
                    <!-- mapping uml class into complex type -->
                    <xsl:when test="//UML:Stereotype[@name='XSDcomplexType' and @baseClass='Class' and
                                            contains(concat(@extendedElement,' '),$class_id_modified)]">
                        <xsl:call-template name="umlClass2ComplexType">
                            <xsl:with-param name="class_name" select="$class_name"/>
                            <xsl:with-param name="class_id" select="$class_id"/>
                            <xsl:with-param name="mode">ct</xsl:with-param>
                        </xsl:call-template>
                    </xsl:when>
                    <!-- special case mapping: enumeration -->
                    <xsl:when test="//UML:Stereotype[@name='enumeration' and @baseClass='Class' and
                                            contains(concat(@extendedElement,' '),$class_id_modified)]">
                        <xsl:call-template name="enumeration">
                            <xsl:with-param name="class_name" select="$class_name"/>
                            <xsl:with-param name="class_id" select="$class_id"/>
                        </xsl:call-template>
                    </xsl:when>
                    <xsl:otherwise>
                        <xsl:call-template name="umlClass2Element">
                            <xsl:with-param name="class_name" select="$class_name"/>
                            <xsl:with-param name="class_id" select="$class_id"/>
                        </xsl:call-template>
                    </xsl:otherwise>
                </xsl:choose>
            </xsl:for-each>
            <xsl:for-each select="//UML:Association">
                <xsl:variable name="association_name" select="@name"/>
                <xsl:variable name="association_id" select="@xmi.id"/>
                <xsl:variable name="association_id_modified" select="concat($association_id,' ')"/>
                <xsl:choose>
                    <!-- mapping uml association into element -->
                    <xsl:when test="//UML:Stereotype[@name='XSDelement' and @baseClass='Association' and
                                            contains(concat(@extendedElement,' '),$association_id_modified)]">
                        <xsl:call-template name="umlAssociation2Element">
                            <xsl:with-param name="association_name" select="$association_name"/>
                            <xsl:with-param name="association_id" select="$association_id"/>
                        </xsl:call-template>
                    </xsl:when>
                    <!-- mapping uml association into complex type -->
                    <xsl:when test="//UML:Stereotype[@name='XSDcomplexType' and @baseClass='Association'
                                            and contains(concat(@extendedElement,' '),$association_id_modified)]">
                        <xsl:call-template name="umlAssociation2ComplexType">
                            <xsl:with-param name="association_name" select="$association_name"/>
                            <xsl:with-param name="association_id" select="$association_id"/>
                        </xsl:call-template>
                    </xsl:when>
                    <xsl:when test="//UML:Stereotype[@name='XSDcontainment' and @baseClass='Association' and
                                            contains(concat(@extendedElement,' '),$association_id_modified)]"/>
                    <xsl:when test="//UML:Stereotype[@name='XSDreference' and @baseClass='Association' and
                                            contains(concat(@extendedElement,' '),$association_id_modified)]"/>
                    <xsl:otherwise>
                        <xsl:call-template name="umlAssociation2Element">
                            <xsl:with-param name="association_name" select="$association_name"/>
                            <xsl:with-param name="association_id" select="$association_id"/>
                        </xsl:call-template>
                    </xsl:otherwise>
                </xsl:choose>
            </xsl:for-each>
        </xs:schema>
    </xsl:template>
</xsl:stylesheet>
```

complex_type.xslt

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:UML="href://org.omg/UML/1.3" exclude-result-prefixes="UML"
                xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

    <!-- mapping uml class into complex type -->
    <xsl:template name="umlClass2ComplexType">
        <xsl:param name="class_name"/>
        <xsl:param name="class_id"/>
        <xsl:param name="mode"/>
        <!-- check tagged value: complexTypeName -->
        <xsl:variable name="tagged_complexTypeName">
            <xsl:value-of select="//UML:TaggedValue[(@modelElement=$class_id) and (@tag='RationalRose$UML
                                   Profile:complexTypeName')]/@value"/>
        </xsl:variable>
        <!-- check tagged value: mixed -->
        <xsl:variable name="tagged_mixed">
            <xsl:value-of select="//UML:TaggedValue[(@modelElement=$class_id) and (@tag='RationalRose$UML
                                   Profile:mixed')]/@value"/>
        </xsl:variable>
        <!-- check tagged value: complexTypeAbstract -->
        <xsl:variable name="tagged_complexTypeAbstract">
            <xsl:value-of select="//UML:TaggedValue[(@modelElement=$class_id) and (@tag='RationalRose$UML
                                   Profile:complexTypeAbstract')]/@value"/>
        </xsl:variable>
        <!-- check tagged value: complexTypeCompositionKind -->
        <xsl:variable name="tagged_complexTypeCompositionKind">
            <xsl:value-of select="//UML:TaggedValue[(@modelElement=$class_id) and (@tag='RationalRose$UML
                                   Profile:complexTypeCompositionKind')]/@value"/>
        </xsl:variable>
        <!-- check tagged value: attributeGroupName -->
        <xsl:variable name="tagged_attributeGroupName">
            <xsl:value-of select="//UML:TaggedValue[(@modelElement=$class_id) and (@tag='RationalRose$UML
                                   Profile:attributeGroupName')]/@value"/>
        </xsl:variable>
        <!-- check tagged value: modelGroupName -->
        <xsl:variable name="tagged_modelGroupName">
            <xsl:value-of select="//UML:TaggedValue[(@modelElement=$class_id) and (@tag='RationalRose$UML
                                   Profile:modelGroupName')]/@value"/>
        </xsl:variable>
        <!-- check tagged value: modelGroupCompositionKind -->
        <xsl:variable name="tagged_modelGroupCompositionKind">
            <xsl:value-of select="//UML:TaggedValue[(@modelElement=$class_id) and (@tag='RationalRose$UML
                                   Profile:modelGroupCompositionKind')]/@value"/>
        </xsl:variable>
        <xsl:variable name="stereotype_reference_id">
            <xsl:value-of select="//UML:Stereotype[@name='XSDreference' and
                                   @baseClass='Association']/@extendedElement"/>
        </xsl:variable>
        <xsl:variable name="stereotype_reference_id_modified" select="concat($stereotype_reference_id,' ')"/>
        <xsl:variable name="generalization_id">
            <xsl:value-of select="@generalization"/>
        </xsl:variable>
        <xsl:variable name="parent_id">
            <xsl:value-of select="UML:Namespace.ownedElement/UML:Generalization[@xmi.id=$generalization_id]
                                   /@parent"/>
        </xsl:variable>
        <xsl:variable name="stereotype_generalization">
            <xsl:value-of select="//UML:Stereotype[@extendedElement=$generalization_id and
                                   @baseClass='Generalization']/@name"/>
        </xsl:variable>
```

```
<!-- check tagged value: attributeGroupName of parent class -->
<xsl:variable name="tagged_parent_attributeGroupName">
     <xsl:value-of select="//UML:TaggedValue[(@modelElement=$parent_id) and (@tag='RationalRose$UML
                              Profile:attributeGroupName')]/@value"/>
</xsl:variable>
<!-- check tagged value: modelGroupName of parent class -->
<xsl:variable name="tagged_parent_modelGroupName">
     <xsl:value-of select="//UML:TaggedValue[(@modelElement=$parent_id) and (@tag='RationalRose$UML
                              Profile:modelGroupName')]/@value"/>
</xsl:variable>
<xs:complexType>
     <xsl:if test="$tagged_complexTypeName='derived' or $tagged_complexTypeName=''">
          <xsl:attribute name="name"><xsl:value-of select="$class_name"/></xsl:attribute>
     </xsl:if>
     <xsl:if test="not($tagged_complexTypeName='' or $tagged_complexTypeName='derived' or
                    $tagged_complexTypeName='anonymous')">
          <xsl:attribute name="name"><xsl:value-of select="$tagged_complexTypeName"/></xsl:attribute>
     </xsl:if>
     <xsl:if test="$tagged_mixed='true'">
          <xsl:attribute name="mixed"><xsl:value-of select="$tagged_mixed"/></xsl:attribute>
     </xsl:if>
     <xsl:if test="$tagged_complexTypeAbstract='true'">
          <xsl:attribute name="abstract"><xsl:value-of select="$tagged_complexTypeAbstract"/></xsl:attribute>
     </xsl:if>
     <xsl:choose>
          <xsl:when test="not($generalization_id='') and ($stereotype_generalization='XSDderivation' or
                         $stereotype_generalization='')">
               <xsl:call-template name="umlGeneralization2Derivation">
                    <xsl:with-param name="child_id" select="$class_id"/>
                    <xsl:with-param name="tagged_modelGroupName" select="$tagged_modelGroupName"/>
                    <xsl:with-param name="tagged_complexTypeCompositionKind"
                                   select="$tagged_complexTypeCompositionKind"/>
                    <xsl:with-param name="tagged_attributeGroupName"
                                   select="$tagged_attributeGroupName"/>
                    <xsl:with-param name="parent_id" select="$parent_id"/>
                    <xsl:with-param name="tagged_parent_modelGroupName"
                                   select="$tagged_parent_modelGroupName"/>
                    <xsl:with-param name="tagged_parent_attributeGroupName"
                                   select="$tagged_parent_attributeGroupName"/>
                    <xsl:with-param name="stereotype_generalization" select="$stereotype_generalization"/>
                    <xsl:with-param name="stereotype_reference_id_modified"
                                   select="$stereotype_reference_id_modified"/>
               </xsl:call-template>
          </xsl:when>
          <xsl:otherwise>
               <xsl:call-template name="composition_kind">
                    <xsl:with-param name="class_id" select="$class_id"/>
                    <xsl:with-param name="tagged_modelGroupName" select="$tagged_modelGroupName"/>
                    <xsl:with-param name="tagged_complexTypeCompositionKind"
                                   select="$tagged_complexTypeCompositionKind"/>
                    <xsl:with-param name="tagged_attributeGroupName"
                                   select="$tagged_attributeGroupName"/>
                    <xsl:with-param name="parent_id" select="$parent_id"/>
                    <xsl:with-param name="tagged_parent_modelGroupName"
                                   select="$tagged_parent_modelGroupName"/>
                    <xsl:with-param name="tagged_parent_attributeGroupName"
                                   select="$tagged_parent_attributeGroupName"/>
                    <xsl:with-param name="stereotype_generalization" select="$stereotype_generalization"/>
               </xsl:call-template>
          </xsl:otherwise>
     </xsl:choose>

     <xsl:if test="$generalization_id='' or not($stereotype_generalization='XSDderivation' or
                    $stereotype_generalization='')">
          <!-- check whether id attribute should be added -->
```

```
                <xsl:if test="(//UML:Association[./UML:Association.connection/UML:AssociationEnd[position()=1 and
                              @isNavigable='true' and @type=$class_id] and not(@xmi.id=//UML:TaggedValue[@tag=
                               'RationalRose$UML Profile A:associationEndToClass' and not(@value='reference')]
                              /@modelElement)]) or (//UML:Association[./UML:Association.connection
                              /UML:AssociationEnd[position()=2 and @isNavigable='true' and @type=$class_id] and
                              not(@xmi.id=//UML:TaggedValue[@tag='RationalRose$UML Profile
                              B:associationEndToClass' and not(@value='reference')]/@modelElement)])">
                    <xs:attribute>
                        <xsl:attribute name="name">id</xsl:attribute>
                        <xsl:attribute name="type">xs:ID</xsl:attribute>
                    </xs:attribute>
                </xsl:if>
                <!-- check wheter idref attribute should be added -->
                <xsl:if test="//UML:Association[contains($stereotype_reference_id_modified,concat(@xmi.id,'
                              '))]/UML:Association.connection/UML:AssociationEnd[@type=$class_id]">
                    <xs:attribute>
                        <xsl:attribute name="name">idrefs</xsl:attribute>
                        <xsl:attribute name="type">xs:IDREFS</xsl:attribute>
                    </xs:attribute>
                </xsl:if>
            </xsl:if>
        </xs:complexType>
        <xsl:if test="$mode='ct' and not($tagged_modelGroupName='')">
            <xsl:call-template name="umlAttribute2ModelGroup">
                <xsl:with-param name="class_id" select="$class_id"/>
                <xsl:w ith-param name="tagged_modelGroupName" select="$tagged_modelGroupName"/>
                <xsl:with-param name="tagged_modelGroupCompositionKind"
                                 select="$tagged_modelGroupCompositionKind"/>
            </xsl:call-template>
        </xsl:if>
        <xsl:if test="$mode='ct' and not($tagged_attributeGroupName='')">
            <xsl:call-template name="umlAttribute2AttributeGroup">
                <xsl:with-param name="class_id" select="$class_id"/>
                <xsl:with-param name="tagged_attributeGroupName" select="$tagged_attributeGroupName"/>
            </xsl:call-template>
        </xsl:if>
        <xsl:if test="$mode='ct'">
            <xsl:for-each select="//UML:Association[./UML:Association.connection/UML:AssociationEnd[position()=1
                                   and @type=$class_id]]">
                <xsl:variable name="association_id" select="@xmi.id"/>
                <xsl:variable name="association_id_modified" select="concat($association_id,' ')"/>
                <xsl:if test="UML:Association.connection/UML:AssociationEnd[position()=2]/@isNavigable='true' and
                              //UML:Stereotype[@name='XSDreference' and @baseClass='Association' and
                              contains(concat(@extendedElement,' '),$association_id_modified)]">
                    <xsl:call-template name="umlAssociation2Reference">
                        <xsl:with-param name="association_id" select="$association_id"/>
                        <xsl:with-param name="ae_a_or_b">A</xsl:with-param>
                    </xsl:call-template>
                </xsl:if>
            </xsl:for-each>
            <xsl:for-each select="//UML:Association[./UML:Association.connection/UML:AssociationEnd[position()=2
                                   and @type=$class_id]]">
                <xsl:variable name="association_id" select="@xmi.id"/>
                <xsl:variable name="association_id_modified" select="concat($association_id,' ')"/>
                <xsl:if test="UML:Association.connection/UML:AssociationEnd[position()=1]/@isNavigable='true' and
                              //UML:Stereotype[@name='XSDreference' and @baseClass='Association' and
                              contains(concat(@extendedElement,' '),$association_id_modified)]">
                    <xsl:call-template name="umlAssociation2Reference">
                        <xsl:with-param name="association_id" select="$association_id"/>
                        <xsl:with-param name="ae_a_or_b">B</xsl:with-param>
                    </xsl:call-template>
                </xsl:if>
            </xsl:for-each>
        </xsl:if>
    </xsl:template>
```

```
<xsl:template name="composition_kind">
    <xsl:param name="class_id"/>
    <xsl:param name="tagged_modelGroupName"/>
    <xsl:param name="tagged_complexTypeCompositionKind"/>
    <xsl:param name="tagged_attributeGroupName"/>
    <xsl:param name="parent_id"/>
    <xsl:param name="tagged_parent_modelGroupName"/>
    <xsl:param name="tagged_parent_attributeGroupName"/>
    <xsl:param name="stereotype_generalization"/>
    <xsl:choose>
        <xsl:when test="$tagged_modelGroupName='' and $tagged_parent_modelGroupName=''">
            <xsl:if test="$tagged_complexTypeCompositionKind='' or
                        $tagged_complexTypeCompositionKind='all'">
                <xs:all>
                    <xsl:call-template name="umlAttribute">
                        <xsl:with-param name="class_id" select="$class_id"/>
                        <xsl:with-param name="ck">all</xsl:with-param>
                        <xsl:with-param name="mapped">element</xsl:with-param>
                    </xsl:call-template>
                    <xsl:call-template name="umlAssociation">
                        <xsl:with-param name="class_id" select="$class_id"/>
                        <xsl:with-param name="ck">all</xsl:with-param>
                    </xsl:call-template>
                    <!-- containment/copy down inheritance-->
                    <xsl:if test="$stereotype_generalization='XSDcontainment'">
                        <xsl:call-template name="umlAttribute">
                            <xsl:with-param name="class_id" select="$parent_id"/>
                            <xsl:with-param name="ck">all</xsl:with-param>
                            <xsl:with-param name="mapped">element</xsl:with-param>
                        </xsl:call-template>
                    <xsl:call-template name="umlAssociation">
                        <xsl:with-param name="class_id" select="$parent_id"/>
                        <xsl:with-param name="ck">all</xsl:with-param>
                    </xsl:call-template>
                    </xsl:if>
                </xs:all>
            </xsl:if>
            <xsl:if test="$tagged_complexTypeCompositionKind='sequence'">
                <xs:sequence>
                    <xsl:call-template name="umlAttribute">
                        <xsl:with-param name="class_id" select="$class_id"/>
                        <xsl:with-param name="ck">seq</xsl:with-param>
                        <xsl:with-param name="mapped">element</xsl:with-param>
                    </xsl:call-template>
                    <xsl:call-template name="umlAssociation">
                        <xsl:with-param name="class_id" select="$class_id"/>
                        <xsl:with-param name="ck">all</xsl:with-param>
                    </xsl:call-template>
                    <!-- containment/copy down inheritance-->
                    <xsl:if test="$stereotype_generalization='XSDcontainment'">
                        <xsl:call-template name="umlAttribute">
                            <xsl:with-param name="class_id" select="$parent_id"/>
                            <xsl:with-param name="ck">all</xsl:with-param>
                            <xsl:with-param name="mapped">element</xsl:with-param>
                        </xsl:call-template>
                    <xsl:call-template name="umlAssociation">
                        <xsl:with-param name="class_id" select="$parent_id"/>
                        <xsl:with-param name="ck">all</xsl:with-param>
                    </xsl:call-template>
                    </xsl:if>
                </xs:sequence>
            </xsl:if>
            <xsl:if test="$tagged_complexTypeCompositionKind='choice'">
                <xs:choice>
                    <xsl:call-template name="umlAttribute">
                        <xsl:with-param name="class_id" select="$class_id"/>
                        <xsl:with-param name="ck">choice</xsl:with-param>
                        <xsl:with-param name="mapped">element</xsl:with-param>
                    </xsl:call-template>
                    <xsl:call-template name="umlAssociation">
                        <xsl:with-param name="class_id" select="$class_id"/>
                        <xsl:with-param name="ck">all</xsl:with-param>
                    </xsl:call-template>
                    <!-- containment/copy down inheritance-->
                    <xsl:if test="$stereotype_generalization='XSDcontainment'">
                        <xsl:call-template name="umlAttribute">
                            <xsl:with-param name="class_id" select="$parent_id"/>
                            <xsl:with-param name="ck">all</xsl:with-param>
                            <xsl:with-param name="mapped">element</xsl:with-param>
                        </xsl:call-template>
                    <xsl:call-template name="umlAssociation">
                        <xsl:with-param name="class_id" select="$parent_id"/>
                        <xsl:with-param name="ck">all</xsl:with-param>
                    </xsl:call-template>
                    </xsl:if>
                </xs:choice>
            </xsl:if>
        </xsl:when>
        <xsl:otherwise>
            <xs:sequence>
                <xsl:if test="not($tagged_modelGroupName='')">
                    <xs:group>
                        <xsl:attribute name="ref">
                            <xsl:value-of select="$tagged_modelGroupName"/>
                        </xsl:attribute>
                    </xs:group>
                </xsl:if>
                <xsl:call-template name="umlAttribute">
                    <xsl:with-param name="class_id" select="$class_id"/>
                    <xsl:with-param name="ck">seq</xsl:with-param>
                    <xsl:with-param name="mapped">element</xsl:with-param>
                </xsl:call-template>
                <xsl:if test="not($tagged_parent_modelGroupName='')">
                    <xs:group>
                        <xsl:attribute name="ref">
                            <xsl:value-of select="$tagged_parent_modelGroupName"/>
                        </xsl:attribute>
                    </xs:group>
                </xsl:if>
                <xsl:if test="$stereotype_generalization='XSDcontainment'">
                    <xsl:call-template name="umlAttribute">
                        <xsl:with-param name="class_id" select="$parent_id"/>
                        <xsl:with-param name="ck">seq</xsl:with-param>
                        <xsl:with-param name="mapped">element</xsl:with-param>
                    </xsl:call-template>
                </xsl:if>
            </xs:sequence>
        </xsl:otherwise>
    </xsl:choose>
    <xsl:if test="not($tagged_attributeGroupName='')">
        <xs:attributeGroup>
            <xsl:attribute name="ref"><xsl:value-of select="$tagged_attributeGroupName"/></xsl:attribute>
        </xs:attributeGroup>
    </xsl:if>
    <xsl:call-template name="umlAttribute">
        <xsl:with-param name="class_id" select="$class_id"/>
        <xsl:with-param name="ck"/>
        <xsl:with-param name="mapped">attribute</xsl:with-param>
    </xsl:call-template>
    <xsl:if test="not($tagged_parent_attributeGroupName='')">
        <xs:attributeGroup>
            <xsl:attribute name="ref"><xsl:value-of select="$tagged_parent_attributeGroupName"/></xsl:attribute>
        </xs:attributeGroup>
    <xsl:if test="$stereotype_generalization='XSDcontainment'">
        <xsl:call-template name="umlAttribute">
            <xsl:with-param name="class_id" select="$parent_id"/>
            <xsl:with-param name="ck"/>
            <xsl:with-param name="mapped">attribute</xsl:with-param>
        </xsl:call-template>
    </xsl:if>
</xsl:template>
<xsl:template name="umlAttribute">
    <xsl:param name="class_id"/>
    <xsl:param name="ck"/>
    <xsl:param name="mapped"/>
    <xsl:for-each select="//UML:Class[@xmi.id=$class_id]/UML:Classifier.feature/UML:Attribute">
        <xsl:variable name="attribute_id" select="@xmi.id"/>
        <xsl:variable name="attribute_name" select="@name"/>
        <xsl:variable name="attribute_type" select="@type"/>
        <xsl:variable name="attribute_id_modified" select="concat($attribute_id,' ')"/>
        <xsl:choose>
            <xsl:when test="$mapped='attribute' and //UML:Stereotype[(@name='XSDattribute' and
                            @baseClass='Attribute') and contains(concat(@extendedElement,'
                            '),$attribute_id_modified)]">
                <xsl:call-template name="umlAttribute2Attribute">
                    <xsl:with-param name="attribute_name" select="$attribute_name"/>
                    <xsl:with-param name="attribute_id" select="$attribute_id"/>
                    <xsl:with-param name="attribute_type" select="$attribute_type"/>
                </xsl:call-template>
            </xsl:when>
            <xsl:when test="//UML:Stereotype[(@name='XSDattributeGroup' and @baseClass='Attribute') and
                            contains(concat(@extendedElement,' '),$attribute_id_modified)]"/>
            <xsl:when test="//UML:Stereotype[(@name='XSDmodelGroup' and @baseClass='Attribute') and
                            contains(concat(@extendedElement,' '),$attribute_id_modified)]"/>
            <xsl:when test="$mapped='attribute' and not(//UML:Stereotype[(@name='XSDattribute' and
                            @baseClass='Attribute') and contains(concat(@extendedElement,'
                            '),$attribute_id_modified)])"/>
            <xsl:when test="not($mapped='attribute') and //UML:Stereotype[(@name='XSDattribute' and
                            @baseClass='Attribute') and contains(concat(@extendedElement,'
                            '),$attribute_id_modified)]"/>
            <xsl:otherwise>
                <xsl:call-template name="umlAttribute2Element">
                    <xsl:with-param name="attribute_name" select="$attribute_name"/>
                    <xsl:with-param name="attribute_id" select="$attribute_id"/>
                    <xsl:with-param name="attribute_type" select="$attribute_type"/>
                    <xsl:with-param name="ck" select="$ck"/>
                </xsl:call-template>
            </xsl:otherwise>
        </xsl:choose>
    </xsl:for-each>
</xsl:template>


<xsl:template name="umlAssociation">
    <xsl:param name="class_id"/>
    <xsl:param name="ck"/>

    <xsl:for-each select="//UML:Association[./UML:Association.connection/UML:AssociationEnd[position()=1 and
                         @type=$class_id]]">
        <xsl:variable name="association_id" select="@xmi.id"/>
        <xsl:variable name="association_id_modified" select="concat($association_id,' ')"/>
        <xsl:if test="UML:Association.connection/UML:AssociationEnd[position()=2]/@isNavigable='true' and
                    //UML:Stereotype[@name='XSDcontainment' and @baseClass='Association' and
                    contains(concat(@extendedElement,' '),$association_id_modified)]">
            <xsl:call-template name="umlAssociation2Containment">
                <xsl:with-param name="association_id" select="$association_id"/>
                <xsl:with-param name="ae_a_or_b">A</xsl:with-param>
                <xsl:with-param name="ck" select="$ck"/>
            </xsl:call-template>
        </xsl:if>
    </xsl:for-each>
    <xsl:for-each select="//UML:Association[./UML:Association.connection/UML:AssociationEnd[position()=2 and
                                   @type=$class_id]]">
        <xsl:variable name="association_id" select="@xmi.id"/>
        <xsl:variable name="association_id_modified" select="concat($association_id,' ')"/>
        <xsl:if test="UML:Association.connection/UML:AssociationEnd[position()=1]/@isNavigable='true' and
                    //UML:Stereotype[@name='XSDcontainment' and @baseClass='Association' and
                    contains(concat(@extendedElement,' '),$association_id_modified)]">
            <xsl:call-template name="umlAssociation2Containment">
                <xsl:with-param name="association_id" select="$association_id"/>
                <xsl:with-param name="ae_a_or_b">B</xsl:w ith-param>
                <xsl:with-param name="ck" select="$ck"/>
            </xsl:call-template>
        </xsl:if>
    </xsl:for-each>
</xsl:template>
<xsl:template name="umlAssociation2ComplexType">
    <xsl:param name="association_name"/>
    <xsl:param name="association_id"/>
    <xsl:variable name="classA_id"
                  select="UML:Association.connection/UML:AssociationEnd[position()=1]/@type"/>
    <xsl:variable name="classB_id"
                  select="UML:Association.connection/UML:AssociationEnd[position()=2]/@type"/>
    <xsl:variable name="classA_name" select="//UML:Class[@xmi.id=$classA_id]/@name"/>
    <xsl:variable name="classB_name" select="//UML:Class[@xmi.id=$classB_id]/@name"/>
    <xsl:variable name="roleA_name"
                  select="UML:Association.connection/UML:AssociationEnd[position()=1]/@name"/>
    <xsl:variable name="roleB_name"
                  select="UML:Association.connection/UML:AssociationEnd[position()=2]/@name"/>
    <xsl:variable name="navigableA"
                  select="UML:Association.connection/UML:AssociationEnd[position()=1]/@isNavigable"/>
    <xsl:variable name="navigableB"
                  select="UML:Association.connection/UML:AssociationEnd[position()=2]/@isNavigable"/>
    <xsl:variable name="tagged_associationEndAToClassA">
        <xsl:value-of select="//UML:TaggedValue[(@modelElement=$association_id) and
                                 (@tag='RationalRose$UML Profile A:associationEndToClass')]/@value"/>
    </xsl:variable>
    <xsl:variable name="tagged_associationEndBToClassB">
        <xsl:value-of select="//UML:TaggedValue[(@modelElement=$association_id) and
                                 (@tag='RationalRose$UML Profile B:associationEndToClass')]/@value"/>
    </xsl:variable>
    <xsl:variable name="tagged_classA_elementName">
        <xsl:value-of select="//UML:TaggedValue[(@modelElement=$classA_id) and (@tag='RationalRose$UML
                                 Profile:elementName')]/@value"/>
    </xsl:variable>
    <xsl:variable name="tagged_classA_complexTypeName">
        <xsl:value-of select="//UML:TaggedValue[(@modelElement=$classA_id) and (@tag='RationalRose$UML
                                 Profile:complexTypeName')]/@value"/>
    </xsl:variable>
    <xsl:variable name="tagged_classB_elementName">
        <xsl:value-of select="//UML:TaggedValue[(@modelElement=$classB_id) and (@tag='RationalRose$UML
                                 Profile:elementName')]/@value"/>
    </xsl:variable>
    <xsl:variable name="tagged_classB_complexTypeName">
        <xsl:value-of select="//UML:TaggedValue[(@modelElement=$classB_id) and (@tag='RationalRose$UML
                                 Profile:complexTypeName')]/@value"/>
    </xsl:variable>
    <xs:complexType>
        <xsl:attribute name="name">
            <xsl:choose>
                <xsl:when test="not($association_name='')">$association_name</xsl:when>
                <xsl:otherwise>
                    <xsl:value-of select="$classA_name"/>-<xsl:value-of select="$classB_name"/>
                </xsl:otherwise>
            </xsl:choose>
        </xsl:attribute>
```

```
<xs:sequence>
    <!-- association end A -->
    <xsl:if test="$navigableA='true'">
        <xs:element>
            <xsl:attribute name="name">
                <xsl:choose>
                    <xsl:when test="not($roleA_name='') and not($roleA_name=$roleB_name)">
                        <xsl:value-of select="$roleA_name"/>
                    </xsl:when>
                    <xsl:when test="not($roleA_name='') and $roleA_name=$roleB_name">
                        <xsl:value-of select="$roleA_name"/>1
                    </xsl:when>
                    <xsl:when test="$roleA_name='' and $classA_name = $classB_name">
                        the<xsl:value-of select="$classA_name"/>1
                    </xsl:when>
                    <xsl:otherwise>
                        the<xsl:value-of select="$classA_name"/>
                    </xsl:otherwise>
                </xsl:choose>
            </xsl:attribute>
            <xsl:apply-templates select="UML:Association.connection/UML:AssociationEnd
                                         [position()=1]/UML:AssociationEnd.multiplicity/UML:Multiplicity">
                <xsl:with-param name="ck">sequence</xsl:with-param>
            </xsl:apply-templates>
            <xsl:choose>
                <xsl:when test="$tagged_associationEndAToClassA=''">
                    <xs:complexType>
                        <xs:sequence/>
                        <xs:attribute>
                            <xsl:attribute name="name">idref</xsl:attribute>
                            <xsl:attribute name="type">xs:IDREF</xsl:attribute>
                        </xs:attribute>
                    </xs:complexType>
                </xsl:when>
                <xsl:when test="$tagged_associationEndAToClassA='containment'">
                    <xsl:attribute name="ref">
                        <xsl:choose>
                            <xsl:when test="$tagged_classA_elementName='' or
                                            $tagged_classA_elementName='derived'">
                                <xsl:value-of select="$classA_name"/>
                            </xsl:when>
                            <xsl:otherwise>
                                <xsl:value-of select="$tagged_classA_elementName"/>
                            </xsl:otherwise>
                        </xsl:choose>
                    </xsl:attribute>
                </xsl:when>
                <xsl:when test="$tagged_associationEndAToClassA='instanceOf'">
                    <xsl:attribute name="type">
                        <xsl:choose>
                            <xsl:when test="$tagged_classA_complexTypeName='' or
                                            $tagged_classA_complexTypeName='derived'">
                                <xsl:value-of select="$classA_name"/>
                            </xsl:when>
                            <xsl:otherwise>
                                <xsl:value-of select="$tagged_classA_complexTypeName"/>
                            </xsl:otherwise>
                        </xsl:choose>
                    </xsl:attribute>
                </xsl:when>
            </xsl:choose>
        </xs:element>
    </xsl:if>
    <!-- association end B -->
    <xsl:if test="$navigableB='true'">
        <xs:element>
```

```
            <xsl:attribute name="name">
              <xsl:choose>
                <xsl:when test="not($roleB_name='') and not($roleB_name=$roleA_name)">
                  <xsl:value-of select="$roleB_name"/>
                </xsl:when>
                <xsl:when test="not($roleB_name='') and $roleB_name=$roleA_name">
                  <xsl:value-of select="$roleB_name"/>2
                </xsl:when>
                <xsl:when test="$roleB_name='' and $classB_name = $classA_name">
                  the<xsl:value-of select="$classB_name"/>2
                </xsl:when>
                <xsl:otherwise>
                  the<xsl:value-of select="$classB_name"/>
                </xsl:otherwise>
              </xsl:choose>
            </xsl:attribute>
            <xsl:apply-templates select="UML:Association.connection/UML:AssociationEnd
                                [position()=2]/UML:AssociationEnd.multiplicity/UML:Multiplicity">
              <xsl:with-param name="ck">sequence</xsl:with-param>
            </xsl:apply-templates>
            <xsl:choose>
              <xsl:when test="$tagged_associationEndBToClassB=''">
                <xs:complexType>
                  <xs:sequence/>
                  <xs:attribute>
                    <xsl:attribute name="name">idref</xsl:attribute>
                    <xsl:attribute name="type">xs:IDREF</xsl:attribute>
                  </xs:attribute>
                </xs:complexType>
              </xsl:when>
              <xsl:when test="$tagged_associationEndBToClassB='containment'">
                <xsl:attribute name="ref">
                  <xsl:choose>
                    <xsl:when test="$tagged_classB_elementName='' or
                                    $tagged_classB_elementName='derived'">
                      <xsl:value-of select="$classB_name"/>
                    </xsl:when>
                    <xsl:otherwise>
                      <xsl:value-of select="$tagged_classB_elementName"/>
                    </xsl:otherwise>
                  </xsl:choose>
                </xsl:attribute>
              </xsl:when>
              <xsl:when test="$tagged_associationEndBToClassB='instanceOf'">
                <xsl:attribute name="type">
                  <xsl:choose>
                    <xsl:when test="$tagged_classB_complexTypeName='' or
                                    $tagged_classB_complexTypeName='derived'">
                      <xsl:value-of select="$classB_name"/>
                    </xsl:when>
                    <xsl:otherwise>
                      <xsl:value-of select="$tagged_classB_complexTypeName"/>
                    </xsl:otherwise>
                  </xsl:choose>
                </xsl:attribute>
              </xsl:when>
            </xsl:choose>
          </xs:element>
        </xsl:if>
      </xs:sequence>
    </xs:complexType>
  </xsl:template>
</xsl:stylesheet>
```