University of Twente
department of
computer science

Mapping UML Class Models Into XML Schema's Using Heuristics

M.Sc. Telematics Final Project

Anny Kartika Sari

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

July 2004

Abstract

XML (Extensible Markup Language) is becoming a pervasive choice for documentation, data interchange, application integration, business messages, and other areas. To ensure document conformity, XML Schema is recommended to be used. On the other hand, UML (Unified Modeling Language) has become established as the standard notation used to describe how systems are structured. UML Class Models is a subset of UML Models that is widely used. From the point of view of the system designer, UML Class Models in UML are closer to conceptual thinking than XML Schema. UML Class Models are easier to be created than directly creating XML Schema. Therefore, a mechanism to map UML Class Models into XML Schema's is needed.

An UML Class Model can be mapped into several valid XML schemas. Some alternatives could appear here. The goal of this project is to select the most suitable choice of those alternatives using heuristics. Heuristics describes certain situations in which given modelling constructs, in this case UML Class Models, such as classes, many-to-many relationships, one-to-many relationships, primitive data types, etc. must be mapped to XML Schema constructs in a way that guarantees certain properties of the result.The resulting XML schema may be used to validate XML documents.

Heuristic rules are constructed based on the properties of each construct of the ML Class Model. Each property is mapped into XML Schema construct, if it is available; otherwise the information brought by the property is lost. The mapping of constructs of the UML Class Model into constructs of the XML Schema is based on the similarity of the information or characteristics given by the constructs. In this project, the heuristic rules can be categorized into two types, i.e. rules that are based on quality properties of the source model and rules that are based on the fact that certain combinations of constructs in the target model are not allowed.

To identify the transformation from a given UML Class Model into valid XML schema, a technique called Design Algebra is used. It is used to model a set of alternative transformations for UML Model as the source model into an XML schema as the target model. Four steps in the process of constructing and utilizing a transformation space are performed, i.e. constructing transformation space, reducing transformation space, reducing transformation space on the basis of the quality properties of the target model, and refinement. As a case study, Telecommunication Service Access and Subscription (TSAS) UML Class Model is used.

This project concluded that this project has collected some available heuristics for selection of mapping rules from a UML Class Model into XML schema. It is shown in the case study that the rules can be used to select the alternatives from UML Class Model into XML schema using Design Algebra techniques.

Abbreviations

DTD	: Document Type Definition
ISO	: International Organization for Standardization
MOF	: Meta Object Facility
OMG	: Object Modeling Group
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
XSLT	: Extensible Stylesheet Language Transformation

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

This chapter contains the introduction to the project. The background of the project is described in section 1.1. Section 1.2 formulates the problem statement. The objectives are given in section 1.3. The solution approach is explained in section 1.4. The last section contains the outline of the report.

1.1 Background

XML (Extensible Markup Language) is a technology for creating markup languages to describe data of virtually any type in a structured manner [5]. XML is becoming a pervasive choice for documentation, data interchange, application integration, business messages, and other areas [3]. To ensure document conformity, especially in business-to-business (B2B) transactions where XML documents are exchanged, XML Schema is recommended to be used other than DTD (Document Type Definition). XML Schema defines an XML document’s structure, e.g. what elements, attributes, etc. are permitted in the document.

An XML Schema is a text-based schema, which can be created manually. However, designing a text-based XML Schema is error prone. The process of designing XML schemas may also be based on a model of the domain in which XML-based data exchange will be used. A conceptual model can be expressed as a UML (Unified Modeling Language) Model. UML is a standard graphical modeling language from OMG (Object Modeling Group) to specify, visualize, and document models of software systems, including their structure and design, in a way that meets all of the requirements for scalability, robustness, security, extendibility, and other characteristics [1]. This language is widely used. UML Class Models is a subset of UML Models. From the point of view of the system designer, UML Class Models in UML are closer to conceptual thinking than XML Schema. UML Class Models are easier to be created than directly creating XML Schema.

Generally, there are multiple alternative transformations of a UML Class Model into XML schemas. The solution space of alternative mappings is usually very large, and thus, it is difficult to evaluate every alternative. The designer may base his decisions on a certain heuristics. Heuristics describe certain situations in which given modelling constructs such as classes, many-to-many relationships, one-to-many relationships, primitive data types, etc. must be mapped to XML Schema constructs in a way that guarantees certain properties of the result.

1.2 Problem Statement

An UML Class Model may be transformed into several valid XML schemas. This project is aimed to select one or more suitable choices of those alternatives using heuristics. The resulting XML Schema may be used to validate XML documents. However, the production of these XML documents is out of the scope of this project.

The alternatives and the use of heuristics rules can be described by the figure below.

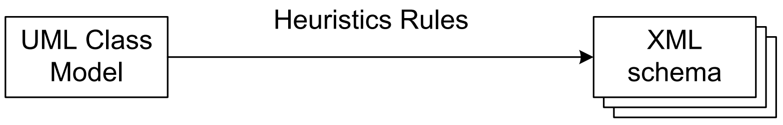


Figure 1-1: The alternatives raise in the mapping UML Class Model into XML Schema and the use of heuristics rules

Thus, heuristic rules are needed to select the alternatives of the mapping UML Class Model into XML schemas.

1.3 Objectives

Based on the above problem statement, the objective of this project is:

To study, collect and categorize available heuristics for selection of mapping rules from an UML domain model into an XML schema.

1.4 Solution Approach

To achieve the objectives above, the following solution approach is used:

1. Literature study

Articles, book, internet resources, etc. about XML, UML, Design Algebra, and methods for mapping UML models to XML schemas are studied in order to obtain the required theoretical background.

2. Analysis

   Investigation and categorization of the available heuristics for selection of mapping rules is conducted.

3. Case Study

   A case study is presented to show how the rules can be used to select the alternatives.

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 Generalization into XML schema. Two steps of alternatives generation are discussed. The second step of alternative generation for each possible mapping of UML Generalization is explained in detail. Also, the transformation process of mapping UML Generalization into XML schema is explained with an example.

Chapter 4 contains the description of the rules. The rules are described based on the properties of each construct of UML Class Model. Rules for each UML Class Diagram construct, i.e. class, attribute, association, and generalization are built.

Chapter 5 contains the transformation from UML Class Diagram into XML Schema using Design Space. Four steps of transformation are presented. Furthermore, the implementation of Design Space and reduction using the rules can reduce the alternatives.

Chapter 6 contains summary and suggestion of future work.

Chapter 2
Concepts

This chapter describes the basic concepts of this project. Section 2.1 introduces some basic concepts of UML. It is followed by the introduction of some basic concepts of XML in section 2.2. And the last section will explain basic concept of heuristics.

2.1 UML (Unified Modeling Language)

The Unified Modeling Language (UML) is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems [13]. The UML uses mostly graphical notations to express the design of software projects. In this project, only a subset of the language used for class diagrams will be discussed.

2.1.1 Class Diagram

Class diagram is one of the diagrams available in UML. The others are Use Case Diagram, Behaviour Diagrams (Statechart Diagram, Activity Diagram, and Interaction Diagram), and Implementation Diagrams (Component Diagram and Deployment Diagram) [22].

Class Diagram models class structure and contents using design elements such as classes and packages [1]. It also displays relationships such as containment, inheritance, associations and others. An example of class diagram is shown by Figure 2-1. The diagram is a modified version of *subscription information model of Telecommunication Service Access and Subscription (TSAS)* [16]. The model will be used in the case study (chapter 5).

A class diagram consists of classes and relationships between the classes. The next sub sections are the explanations of the components of class diagram that will be used in this project.

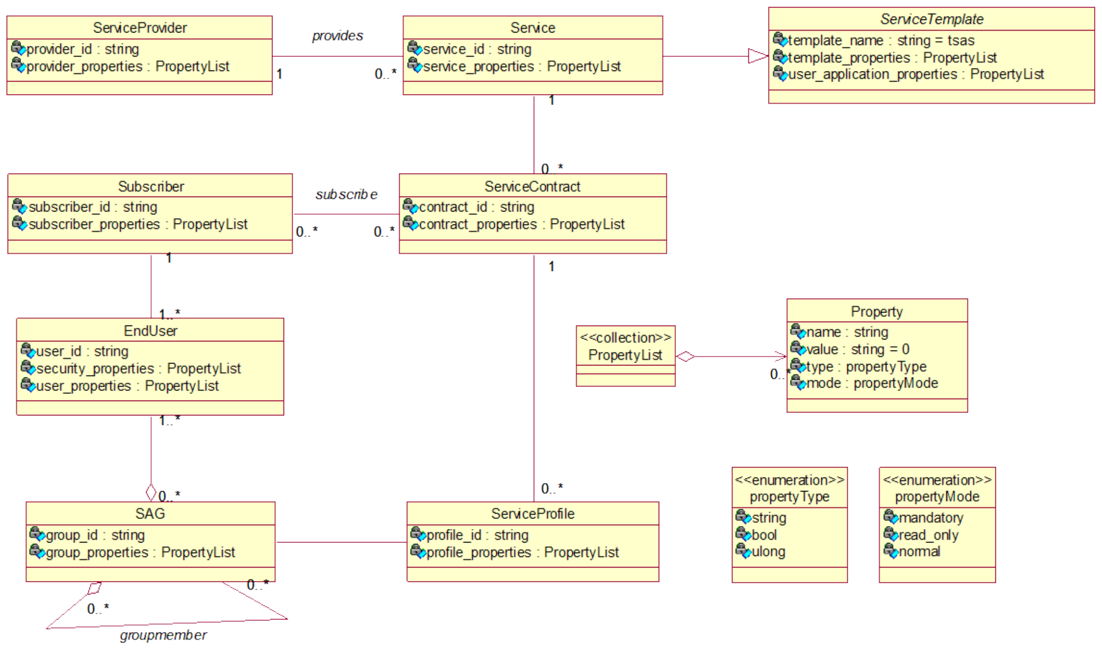


Figure 2-1: The Modified TSAS Class Diagram

2.1.1.1 Class

A class is a description of a set of objects that share the same attributes, operations, relationships, and semantics [13]. In UML, a class is represented by a rectangle. A class is composed of three things: a name, attributes, and operations. However, operations will not be used in this project, and thus, will not be discussed. Figure 2-2 is an example of a class.

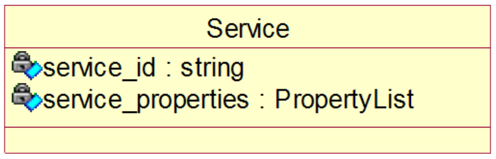


Figure 2-2: An example of a class

From the example, the following can be seen:

- The name of the class is *Service*.

  The name of a class must be unique and is textual string.

- The class has two attributes, i.e. *service_id* with the type of string, and *service_properties* with the type of PropertyList.

  Attribute describes the property of the class. A class can have any number of attributes. An attribute has name. It may have type, multiplicity, and initial value. Multiplicity may be omitted if it is exactly 1. Attribute's type can be a primitive data type (string, integer, etc.) or user-defined data type, for example the type of attribute service_properties in the example above.

2.1.1.2 Relationship

Relationship shows the collaboration between classes in UML class diagram. There are four types of relationships, i.e. association, generalization, dependency, and realization. Only association and generalization will be used in this project. Below are the explanations of them.

- Association

  Association specifies a bidirectional connection between classes. It is represented as a solid line that connects the classes involved in the association. One directional association uses an arrow head to represent the direction. Figure 2-3 shows an example of association.

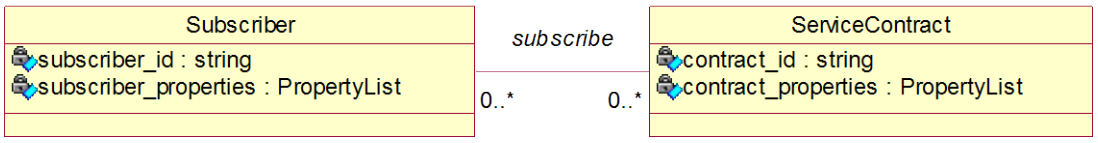


Figure 2-3: An example of association

Some important properties owned by association are:

- Name

  Association can have a name, but it is optional. In the above example, the name of the association is *subscribe*.
- 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. In the example, and 0 or more (0..*) instances of class *ServiceContract* may be connected across an instance of the association.
- Aggregation

When placed on one end of association, aggregation value specifies whether the class on the target end is an aggregation with respect to the class on the other end. There are three types of aggregations as follow:

♦ none, means that no aggregation is specified.
  This is the most common type of aggregation. The association that does not have aggregation value does not have special name.

♦ aggregate
  This type of aggregation indicates a 'has a' relationship. The association which has the aggregate type of aggregation is called *aggregation*. It is represented by a line with a diamond end. An example of aggregation is shown by Figure 2-4.

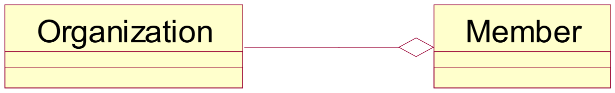


Figure 2-4: An example of aggregation

In the example, Organization has Member. However, the Member is not strongly owned by Organization, i.e. it can be a member of another organization.

♦ composite
  This type of aggregation indicates a 'is part of' relationship. The association which has the aggregate type of aggregation is called *composition*. It is represented by a line with a diamond end. An example of aggregation is shown by Figure 2-5.

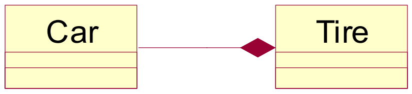
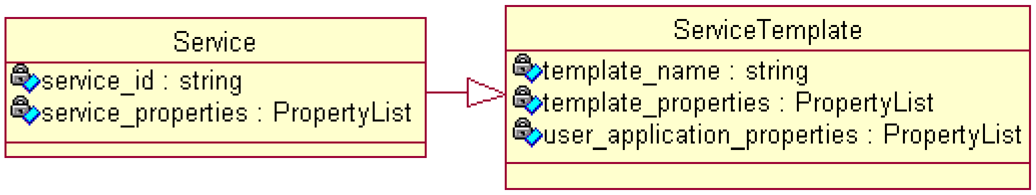


Figure 2-5: An example of composition

In the example, Tire is strongly owned by Car. It is part of Car and cannot be owned by another car.

▪ Generalization
  Generalization is a relationship between a general class (super or parent class) and a more specific class (sub or child class). The attributes and operations owned by the parent class are inherited by the child class. Generalization is represented as a solid directed line with a large

arrowhead. Figure 2-6 shows an example of a generalization. The specialized class or the child class is class *Service*, while the parent class is class *ServiceTemplate*.



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

Figure 2-6: An example of generalization

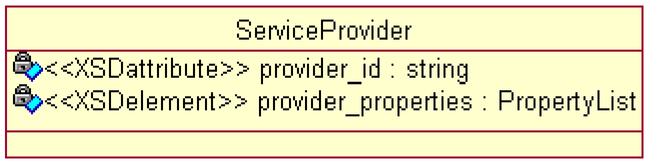
2.1.1.3 UML Profile

Generating code from UML for various languages such as Java, EJB, or XML schemas often requires additional information that guides the code-generation syntax or process [2]. These issues are solved through the use of UML extension profiles. UML profile will be used in chapter 3 to give the description how the mapping is done.

An UML profile has three key items: stereotypes, tagged values, and constraints [2]. Only stereotypes and tagged values will be used in this project, hence only those two will be explained.

- Stereotypes

  Stereotype is an extension to the UML itself [2]. A stereotype allows a modeler to attach a new meaning to one of the UML foundational elements such as class, attribute, and association. A stereotype is usually represented on a diagram as the name of surrounded by guillements << >>. Example of the use of stereotype is shown by Figure 2-7 as follows.



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

Figure 2-7: Example of the use of stereotype

- Tagged values

  Tagged value is an extension to the attributes of a UML model element [2]. Each model element has a standard set of attributes. For instance, each UML class has a name, visibility, etc. A tagged value defines a new attribute that is associated with a stereotyped model element. A tagged value is shown on the diagram as a name/value string: {tagName=value}.

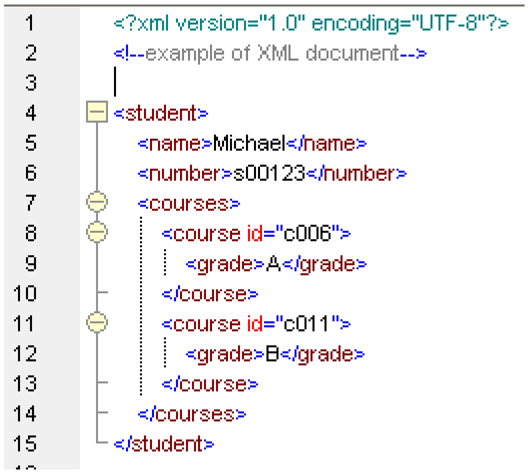
2.2 XML (Extensible Markup Language)

XML (Extensible Markup Language) is a technology for creating markup language to describe data of virtually any type in a structured manner [5]. XML is a restricted form of SGML, the Standard Generalized Markup Language (SGML). SGML is a standard from ISO (International Organization for Standardization). It is a meta language that creates markup language.

2.2.1 XML Document

XML describes a class of data objects called XML documents. W3C (World Wide Web Consortium) defines the specification of XML version 1.0 [20]. Each XML document has both a logical and a physical structure. Physically, the document is composed of units called entities. An entity may refer to other entities in the document. Logically, the document is composed of declarations, elements, comments, character references, and processing instructions.

An example of XML document about a student and his grades for some courses is as follows[1]:



```
<?xml version="1.0" encoding="UTF-8"?>
<!--example of XML document-->

<student>
    <name>Michael</name>
    <number>s00123</number>
    <courses>
        <course id="c006">
            <grade>A</grade>
        </course>
        <course id="c011">
            <grade>B</grade>
        </course>
    </courses>
</student>
```

Figure 2-8: An example of XML document

[1] Note: Document as it is displayed in the tool XML Spy 2004 Enterprise Edition

The first line is an XML declaration. It starts with <? and ends with ?>. It is not an obligation to write an XML declaration. The second line is comment. It is also optionally written. It starts with <!-- and ends with -->. There are some elements in the document. A document should contain at least one element. Element is everything between its start tag and its end tag. Tags are keywords contained in a pair of angle braces (<>). Tag <student> is a start tag and </student> is an end tag. Thus, student is an element. In the example, student is the root element. An XML document must contain exactly one root element. An element may contain other elements. For instance, student contains name, number, and courses elements. Element student is called parent, while name, number, and courses are each called child. An element may have one or more attribute. In the example, id is the attribute of element course. An attribute is placed in the start tag of an element. Its value is placed in quotes. All XML elements must be properly nested. For example, <student><name>Michael</student></name>is an error. Element name must end before element student.

2.2.2 XML Validation

An XML parser, which is a software program, is needed to process an XML document. It reads XML document, checks its syntax, and validates it. Concerning the parser, there are two important terms defined as follow:

- well-formed

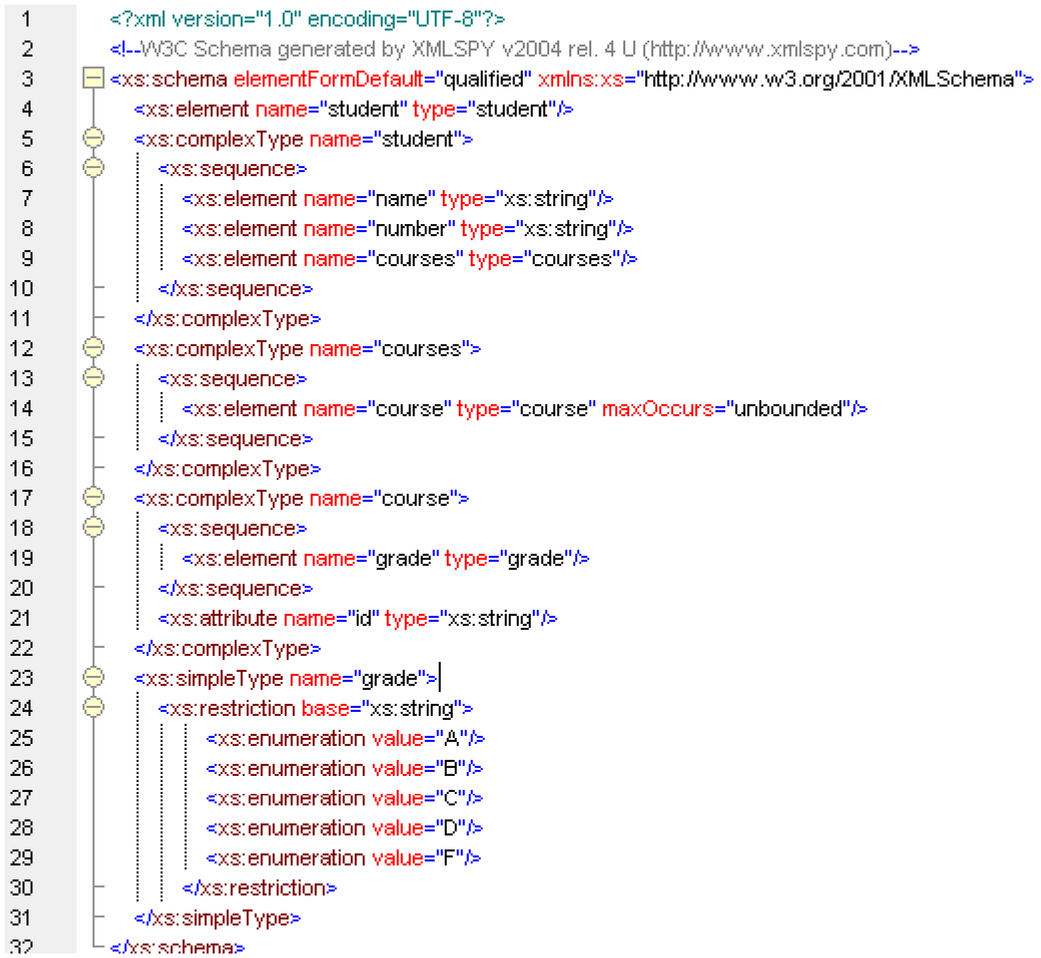
  An XML document is well-formed if its syntax is correct. It means that the document conforms to the XML grammar defined in the specification. It must have a single root element, a start tag and an end tag for each element, properly-nested tags, attribute values in quotes, and proper capitalization of its elements and attributes names.
- valid

  An XML document is valid if it conforms to the DTD (Document Type Declaration) or XML schema associated with the document. DTD and XML Schema define the structure of XML document. Since XML schema will be used in this project, it will be discussed in the next section.

2.2.3 XML Schema

As mentioned in the previous section, XML Schema as well as DTD defines the structure of XML document. Although schema is introduced later than DTD, it has some advantages that cannot be found in DTD. DTD uses non-XML syntax. On the other hand, schema uses XML syntax. It is an XML document and, thus, can be manipulated easily. Moreover, schema has richer of primitive types than DTD. Hence, XML Schema's are expected to replace DTDs as the primary means of describing document structure. The XML schema that will be used in this project is the one recommended by W3C [21] [22] [23].

An example of XML schema is described by Figure 2-9 below. It is the schema of the XML document in Figure 2-8[2].



```
<?xml version="1.0" encoding="UTF-8"?>
<!--W3C Schema generated by XMLSPY v2004 rel. 4 U (http://www.xmlspy.com)-->
<xs:schema elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
	<xs:element name="student" type="student"/>
	<xs:complexType name="student">
		<xs:sequence>
			<xs:element name="name" type="xs:string"/>
			<xs:element name="number" type="xs:string"/>
			<xs:element name="courses" type="courses"/>
		</xs:sequence>
	</xs:complexType>
	<xs:complexType name="courses">
		<xs:sequence>
			<xs:element name="course" type="course" maxOccurs="unbounded"/>
		</xs:sequence>
	</xs:complexType>
	<xs:complexType name="course">
		<xs:sequence>
			<xs:element name="grade" type="grade"/>
		</xs:sequence>
		<xs:attribute name="id" type="xs:string"/>
	</xs:complexType>
	<xs:simpleType name="grade">
		<xs:restriction base="xs:string">
			<xs:enumeration value="A"/>
			<xs:enumeration value="B"/>
			<xs:enumeration value="C"/>
			<xs:enumeration value="D"/>
			<xs:enumeration value="F"/>
		</xs:restriction>
	</xs:simpleType>
</xs:schema>
```

Figure 2-9: An XML schema of the XML document in Figure 2-8

[2] Note: schema as it is displayed in the tool XML Spy 2004 Enterprise Edition

In the above schema, there are some element declarations, for examples in line 4, 7, 8, and 9. Some complex types are defined, for example in line 5-11 and 12-16. One simple type is defined, i.e. in line 23-31 which uses derivation by restriction and `enumeration` facet. The complete description of XML schema is specified by W3C Recommendation in XML Schema Part 0: Primer [38], XML Schema Part 1: Structures [39], and XML Schema Part 2: Datatypes [40].

2.3 XMI (XML Metadata Interchange)

XMI is specified by OMG (Object Management Group). The main purpose of XMI is to enable easy interchange between modeling tools (based on the OMG-UML) and metadata repositories (OMG-MOF based) in distributed heterogeneous environments [14]. XMI integrates three key industry standards:

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.

XMI Version 1 [14] defines production rules for sharing objects with XML:

- Production of XML DTDs starting from an object model.
- Production of XML Schemas starting from an object model.
- Production of XML documents starting from objects.

XMI Version 2 [15], which is the successor of XMI Version 1, adds the production rules:

- Production of XML Schemas starting from an object model.
- Production of XML Documents compatible with XML Schemas.
- Reverse engineering from XML to an object model.

Some parts of the specification, especially the production of XML Schemas, will be used in this project as the reference in the development of rules.

2.4 Heuristics

A heuristic is a rule of thumb, simplification, or educated guess that reduces or limits the search for solutions in domains that are difficult and poorly understood [6]. Unlike algorithms, heuristics do not guarantee optimal, or even feasible, solutions and are often used with no theoretical guarantee. In *Computer Science*, heuristic is relating to or using problem-solving technique in which the most appropriate solution of several found by alternative methods is selected at successive stages of a program for use in the next step of the program [6].

In this project, heuristics will be used to select the alternatives in the mapping of UML class diagram into XML schema. In performing this, some rules must be developed first. The rules are built based on some particular conditions that will be explained in the next chapters.

2.5 Summary

This chapter has explained basic concepts that will be used in this project. In the first section, UML has been explained as well as some components of UML class diagrams. The components that are important for this project are class, attribute, association, composition, aggregation, and generalization. Following is the explanation about XML. It has included the discussion XML document, XML validation, and XML schema. Some examples have been given there. XMI has also been discussed. The last section briefly explained heuristics and how it will be used in this project.

Chapter 3
Mapping UML Generalization into XML Schema

This chapter contains the description of the mapping UML Generalization into XML Schema. This is a continuation of the previous work by Linda Gunawan [8]. Section 3.1 gives a description about the two steps of alternatives generation in mapping a UML Model into XML Schema. Section 3.2 discusses the schema skeleton refinement, i.e. the second step of alternatives generation, for UML Generalization. Section 3.3 addresses the representation of the alternatives found in the two steps of alternatives generation. Section 3.4 explains the steps of transformation of UML Generalization into XML Schema from the practical point of view. Summary of this chapter is given in the last section.

3.1 Two Steps of Alternatives Generation

An UML class model, including UML generalization, can be mapped into several XML Schemas. It will introduce alternatives. Gunawan [8] generates the alternatives in two steps, i.e. AG1 (Alternatives Generation Step 1) and AG2 (Alternatives Generation Step 2). The AG1 is also called schema skeleton generation, while AG2 is also called schema skeleton refinement. These two steps will be explained later. Figure 3-1 depicts the two steps of alternatives generation for mapping UML Model into XML Schema.

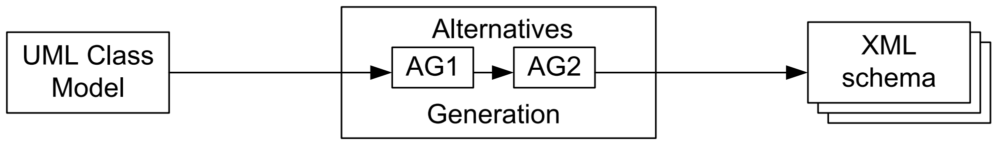


Figure 3-1: Two steps alternatives of generation

Another way to express the two steps of mapping of UML Class Model into XML Schema is by using set and operation as follows.

$$\mathbf{M_{UMLClass} \rightarrow XML\ Schema_{skeleton} \rightarrow XML\ Schema}$$

Where:

$M_{UMLClass}$ metamodel = {Class, Attribute, Generalization, Aggregation, Association}

XML $\text{Schema}_{skeleton}$ metamodel = (Component, Relation)

Component = {CT, ST, E, A, AG, MG}

Relation = {Der, Subst, Cont, Ref, InstanceOf, Copy}

$M_{UMLClass}$ is a UML class model. It can be identified several components based on UML metamodel, which defines UML class model. There are Class, Attribute, Generalization, Aggregation and Association. Similarly, components and relation can be identified from XML Schema metamodel, which defines XML Schema. 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), Reference (Ref), and Copy-down inheritance (Copy). Copy-down inheritance can be considered as special case of containment. It is used in XMI (XML Metadata Interchange) specification in mapping generalization into XML schema [14].

The expression above implies that XML Schema is expressed in XML Schema constructs. An XML Schema constructs has further alternatives which are based on its properties. For instance, an Element has a Name property, among all of its properties. The Name property has alternatives, i.e. Supplied and Derived, one of which must be chosen. This results in the more refined XML Schema which will be the required XML Schema. Thus, XML Schema is a refinement of XML Schema skeleton.

The two steps of mapping are explained below.

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

   This step corresponds to AG1 in Figure 3-1. Alternatives in this step are produced from the alternative mappings of a UML model component into several XML Schema components and relations. Table 3-1 shows the alternative generation step 1 of a UML Model component. This table is based on the thesis of Gunawan [8]. Three alternatives of UML Generalization mapping are newly introduced, i.e. into Element, Complex Type, and Copy-down Inheritance. The copy-down inheritance is a special case of mapping.

Table 3-1: Schema Skeleton Generation

UML Component	XML Schema Skeleton	Abbreviation
Class	Element	E
	Complex Type	CT

Attribute	Attribute	A
	Attribute Group	AG
	Element	E
	Model Group	MG
Association	Element	E
	Complex Type	CT
	Reference	Ref
	Containment	Cont
	Substitution	Subst
Aggregation	Treated just like ordinary association	
Generalization	Element	E
	Complex Type	CT
	Derivation	Der
	Containment	Cont
	Reference	Ref
	Copy-down Inheritance	Copy

2. Schema Skeleton Refinement (XML Schema$_{skeleton}$ $\rightarrow$ XML Schema)
   This step corresponds to AG2 in Figure 3-1. XML Schema construct will be refined further in this step. In this project, only the refinement of UML Generalization will be discussed. The alternatives are addressed in the next section.

3.2 Schema Skeleton Refinement for UML Generalization

Below is the figure of the UML Generalization metamodel as specified in the OMG UML Specification Version 1.5 [13].

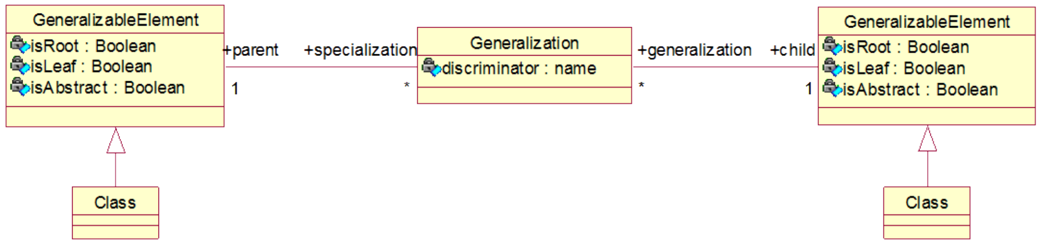


Figure 3-2: UML Generalization Metamodel

An UML Generalization can be mapped into several XML Schema constructs, i.e. Element (E), Complex Type (CT), Reference (Ref), Containment (Cont), Derivation (Der) and Substitution (Subst).

Mapping UML Association into Element, Complex Type, Reference and Containment will be discussed further. In fact, mapping into Substitution is possible. For instance, in mapping classes into element, the corresponding types must be related by derivation. However, the class that is substituted by other class will never be used. Thus, this mapping is excluded from the discussion.

A method called Design Algebra will be used to generate and evaluate alternatives for the mapping. The identified alternatives will be evaluated based on valid combination of XML Schema construct and constraint in XML Schema. Alternatives that are complicated will be eliminated for simplicity.

3.2.1 Mapping UML Generalization into XML Element

The GeneralizableElement in the UML Generalization metamodel does not need to be instantiated in the XML document, since it is an abstract class and it does not contain information that is important for generating XML document, nor does it have any essential information to the generalization itself. Thus, the GeneralizableElement will not be included in the mapping process.

A generalization from C2 as the child class to C1 as the parent class that will be mapped into XML element is depicted by the Figure 3-3 below. Note that <<XSDelement>>, defined by Gunawan, is the stereotype of the generalization to give the information that the generalization should be mapped into XML element. This will be explained further in this chapter.

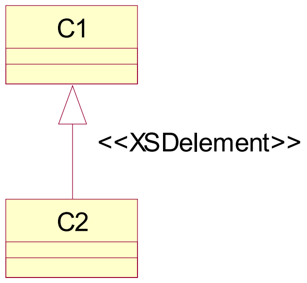


Figure 3-3: UML Generalization that will be mapped into XML element

According to UML metamodel of generalization shown by Figure 3-2, the above generalization can be represented by Figure 3-4. G is generalization. Note that the GeneralizableElement has been excluded from the mapping process. Moreover, it can be seen that the relationship between class generalization and the GeneralizableElement are associations. Thus, the classes (parent class and child class will inherit those association relationships. To simplify the model, only one direction of each association is considered, i.e. from the child class to generalization and from generalization to the parent class. Another reason is based on the assumption that the child should know the existence of the parent class since it will inherit the properties and behaviours of the parent class, while the parent class may not know the existence of the child class. In Figure 3-4, the association relationship is shown by arrow (→).

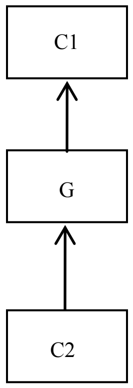


Figure 3-4: Mapping UML Generalization into XML element refined

Further, each component in the refinement other than G, i.e. class C1, class C2, association C2 → G, and association G → C1, can be mapped into several XML Schema constructs. The possibilities are shown by Figure 3-5. All of the mapping of classes, i.e. C1 and C2, and the mapping of associations, i.e. C2 → G, and G → C1, used in this project are based on the thesis of Gunawan [8].

The combinations of the mapping in Figure 3.5 will produce (2 x 6 x 1 x 6 x 2) possibilities or 144 possibilities of mappings. Two comes from the mapping of C1, 5 comes from the mapping of G → C1, 1 comes from the mapping of G, 5 comes from the mapping of C2 → G, and 2 comes from the mapping of C2.

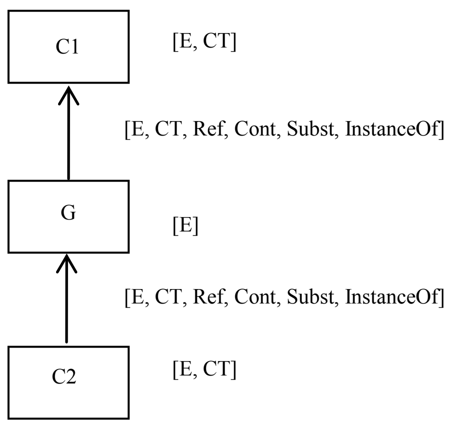

C1
[E, CT]
[E, CT, Ref, Cont, Subst, InstanceOf]
G
[E]
[E, CT, Ref, Cont, Subst, InstanceOf]
C2
[E, CT]

Figure 3-5: Mapping UML Generalization into XML element before elimination

Fortunately, there are some mappings that can be eliminated due to the constraints in XML Schema or simplicity reason. The eliminations are explained below:

- Mapping G → C1 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 3.1, and further, this will become an infinite refinement. This project will only consider the refinement until the first level of UML Metamodel.
- The same reason above applies to mapping C2 → G into XML element and complex type. Thus, mapping C2 → G into element and complex type are eliminated.
- As mentioned above, substitution is not addressed here. Thus, it is also eliminated in the mapping of G → C1 and C2 → G.
- Mapping C2 → G into reference is eliminated since it introduces indirect reference.
- Mapping C2 → G into instance of is eliminated for simplicity reason.
- Mapping G into element, C2 → G into Containment and C2 into complex type implies that an element contains a complex type. This is not a valid combination, and thus, is eliminated. This causes the elimination of mapping C2 into complex type.

The result of the mapping process after the eliminations is shown by Figure 3-6.

Furthermore, the mapping of G into element, G → C1 into instance of, and C1 into element is also eliminated. Class C1 is actually defined outside the generalization, i.e. as an UML class. If this class is mapped into element with named complex type, then the combination is valid. But it is not the case when the complex type is anonymous.

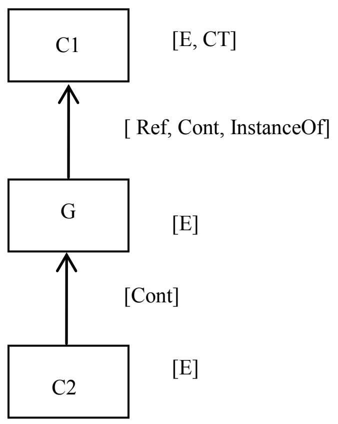


Figure 3-6: Mapping UML Generalization into XML element after elimination

The following table summarizes the possibilities of the mapping.

Table 3-2: The possible mappings of UML Generalization into XML Element

No	C1	G → C1	G	C2 → G	C2
1	E	Ref	E	Cont	E
2	E	Cont	E	Cont	E
3	CT	Ref	E	Cont	E
4	CT	Cont	E	Cont	E
5	CT	InstanceOf	E	Cont	E

The table shows that the variability of the mappings is in the mapping of the generalization to the parent class (C1). It can be mapped into reference, containment, or instance of.

3.2.2 Mapping UML Generalization into XML Complex Type

This mapping is similar to mapping UML generalization into XML Element in section 3.2.1 except that the generalization is mapped into a complex type instead of element. Hence, it will not be discussed further.

3.2.3 Mapping UML Generalization into Containment

Mapping UML Generalization into containment means that the components derived from the parent class is included as the children of the components derived from the child class. Figure 3-10 depicts a generalization that has to be mapped into containment.

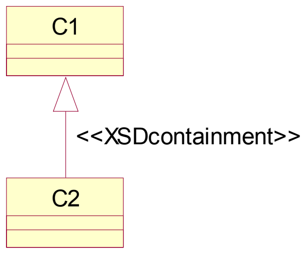


Figure 3-7: UML Generalization that will be mapped into XML Containment

Figure 3-11 shows the possible combinations of mapping generalization into containment.

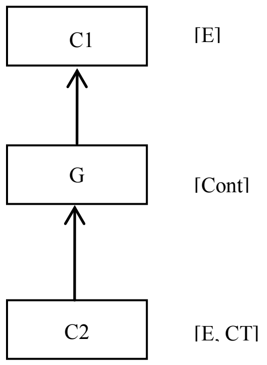


Figure 3-8: Mapping UML Generalization into Containment

Since the generalization is mapped into containment, C1 as the parent should be mapped into element because it will be contained in another element or complex type as the representation of the child class. Thus, the mapping class C1 (parent class) into complex type is eliminated. However, the type of the element representing C1 may be a complex type. The results of the mapping are summarized by Table 3-4.

Table 3-3: The possible mappings of UML Generalization into XML Containment

No	C1	G	C2
1	E	Cont	E
2	E	Cont	CT

3.2.4 Mapping UML Generalization into Derivation

Mapping UML Generalization into XML Derivation means that the XML representation of child class is a derivation of the XML representation of its parent class. Thus, the derivation is between the components to which the classes are mapped.

Since derivation is specified in the XML Schema specification as a property of complex type definition, alternatives can be identified based on the property. Feature diagram will be used to capture the points of variability from this property. In this section, feature diagram will be introduced briefly. Furthermore, feature diagram of derivation will be discussed to see the alternatives caused by the points of variability of the property.

A feature diagram is a visual representation of concepts, features, and their relationship [25]. It forms a tree diagram that consists of nodes, directed edges, and edge decorations. The root node of the diagram is a concept, while the remaining are features. An example of feature diagram is shown by Figure 3-12.

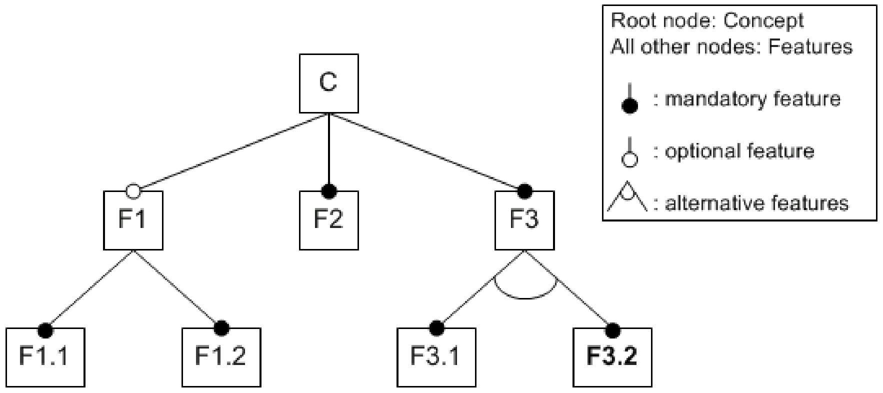
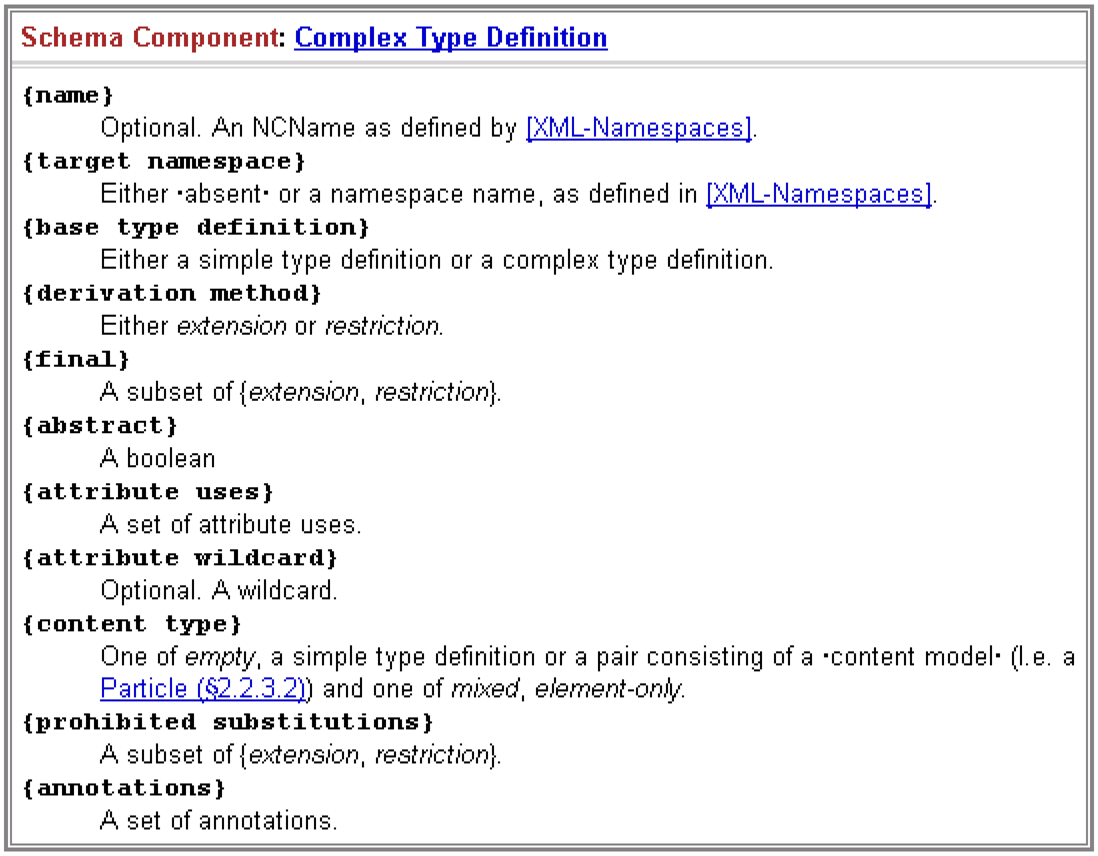


Figure 3-9: An example of feature diagram [8]

In the diagram above, C has three features, i.e. F1, F2, and F3. Feature F1 may be further refined into sub features F1.1 and F1.2, while feature F3 may be further refined into feature F3.1 and F3.2. Features F2, F3, F1.1, F1.2, and F3.1 are mandatory features, while F1 is an optional feature. F3.1 and F3.2 are in set a set of alternatives. F3.2 is bold typed, means that the feature is the default among the alternatives.

In her thesis, Gunawan [8] explains much about feature diagrams of element, complex type, attribute, attribute group, and model group. Here, as mentioned above, only the feature diagram of derivation, which is derived from the feature diagram of complex type, will be explained.

The properties of Complex Type definition schema components are shown by Figure 3-13.


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.

Figure 3-10: The properties of Complex Type schema definition [22]

The feature diagram of derivation is based on the definition above. However, not all of the properties are used. Only the properties related to derivation, i.e. base type definition and derivation method, appear in the feature diagram. The feature diagram of derivation is given by Figure 3-14.

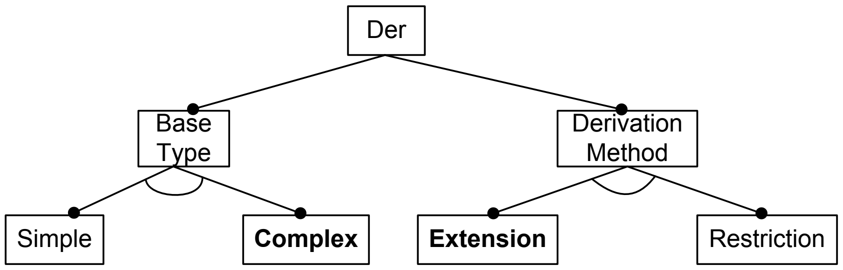


Figure 3-11: The feature diagram of Derivation

The edges ending with filled circles pointing to Base Type and Derivation Method features indicate that those two features are mandatory, which means that they must be included if and only if their parent, i.e. Derivation concept, is defined. The Simple and Complex features are pointed by edges connected by an arc, indicating that those features are alternative features. It means that if its parent, i.e. Base Type, is included, then exactly one of those two features is included. The Complex feature is bold typed which means that it is the default feature. It is chosen as default because simple type is not much discussed in this project due to its simplicity. Similarly, the Extension and Restriction features are alternative features, and Extension is the default value. The features Derivation Method and Base Type of Derivation must be included in the mapping because those features make the alternatives for the mapping. Those two features will be used in the definition of tagged values.

Figure 3-15 depicts UML Generalization that will be mapped into derivation, shown by the <<XSDderivation>> stereotype.

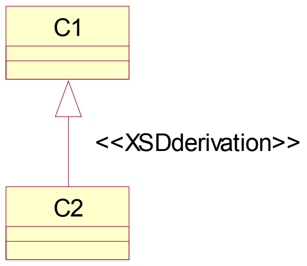


Figure 3-12: UML Generalization that will be mapped into derivation

Figure 3-16 shows the possible combinations of mapping generalization into derivation.

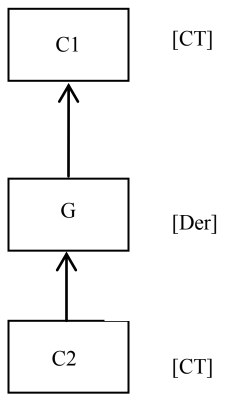


Figure 3-13: Mapping UML Generalization into XML Derivation

Since simple type is not used in this project, this mapping requires both C1 and C2 to be complex types. There is only one possible mapping as shown by Table 3-5, regardless the Base Type feature that is used. The feature will be used later for the tagged values.

Table 3-4: The possible mappings of UML Generalization into XML Derivation

No	C1	G	C2	Base Type
1	CT	Der	CT	Extension
2	CT	Der	CT	Restriction

3.2.5 Mapping UML Generalization into Reference

Mapping UML Generalization into Reference means that the child class refers to the parent class. Figure 3.17 depicts UML Generalization that will be mapped into reference, shown by the <<XSDreference>> stereotype.

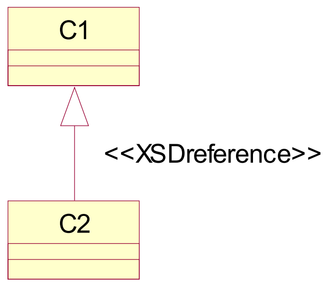


Figure 3-14: UML Generalization that will be mapped into XML Reference

The figure below depicts possible combinations of the mapping of UML Generalization into reference.

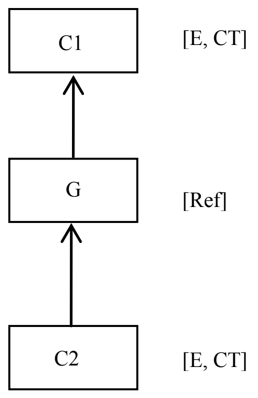


Figure 3-15: Mapping UML Generalization into XML Containment

All of the combinations are valid, and they are summarized by Table 3-6.

Table 3-5: The possible mappings of UML Generalization into XML Reference

C1	G	C2
E	Ref	E
E	Ref	CT
CT	Ref	E
CT	Ref	CT

3.2.6 Mapping UML Generalization Using Copy-down Inheritance

XMI version 2.0 [15] specifies that inheritance, which in this case is the UML generalization, will be copy-down inheritance in XML Schema. This is chosen because the extending type mechanism supported by XML Schema does not enable extending from more than one type. Moreover, XMI attempts to minimize order dependencies, which is imposed on the content models of the types that are derived from other types. Thus, in XMI schema extension is not used by default as the representation of inheritance.

The copy-down inheritance means that the definitions of attributes, association references, and associations from superclasses (and recursively to their superclasses) are simply copied down to the class being translated into XML. The copy-down inheritance applies to all of the attributes and compositions owned by the superclasses. For associations, i.e. AssociationEnds with references, the actual class references is used, and subclasses may be used on the other end of the reference.

Copy-down inheritance is not explicitly specified in the XML Schema Primer specification. Thus, this construct is not explicitly found in XML Schema metamodel. However, it can be considered that copy-down inheritance is a special case of containment. In this case, an element or complex type contains all properties of other element.

A simple example of copy-down inheritance is shown below.

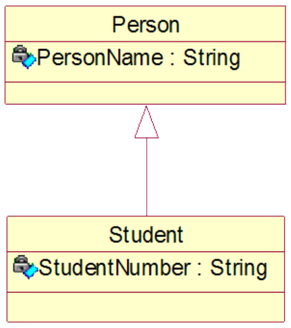


Figure 3-16: An example of UML Generalization

The above diagram suggests that the Student class is the specialization of the Person class, or in other words, the Person class is the generalization of the Student class. This means that the Student class inherits all the properties and relationship of the Person class, for instance the PersonName attribute. If this generalization relationship is mapped into XML Schema using copy-down inheritance, the PersonName attribute will also be owned by the class Student other than the StudentNumber attribute. If the Person class has an association relationship with other class, the Student class will also have the same association relationship.

The representation of the UML diagram in XML Schema will look like below. In this case, it is assumed that both the Person and Student classes are mapped into XML complex type, while the attributes are mapped into XML attributes.

```
<xs:complexType name="Person">
    <xs:attribute name="PersonName" type="xs:string"/>
 </xs:complexType>
<xs:complexType name="Student">
```

```
        <xs:attribute name="PersonName" type="xs:string"/>
        <xs:attribute name="StudentNumber" type="xs:string"/>
    </xs:complexType>
```

The Student complex type, as the representation of Student class, has two attributes, i.e. PersonName and StudentNumber. The PersonName attribute is inherited from the Person class, which in this schema is represented by Person complex type, while the StudentNumber attribute is the representation of StudentNumber attribute which is owned by the Student class itself.

Figure 3-20 shows a UML Generalization that will be copy-down inheritance in XML Schema, and this is indicated by the <<XSDcopyDown>> stereotype.

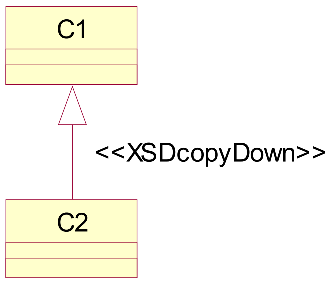


Figure 3-17: UML Generalization that will be mapped using copy-down inheritance rule

No alternatives can be derived for copy-down inheritance since the mapping only copies the components of the superclasses into the generalized class, other than the components owned locally.

3.3 Solution Approach

Gunawan [8] uses the diagram depicted by Figure 3-21 to show the approach that can be used in mapping an UML Class Model into XML Schema.

From the model, it is clear that an UML Class Model will be represented as an XML document as the result of the XMI serialization. The UML Profile will be used as the representation of alternative generations. It then becomes the guidance in doing the transformation from XML representation into XML Schema. In the figure, the strip and dotted arrow shows validation. Thus, from the diagram it can be seen that XML document is validated by XML Schema. Also, XML representation produced by XMI serialization is validated by XMI DTD.

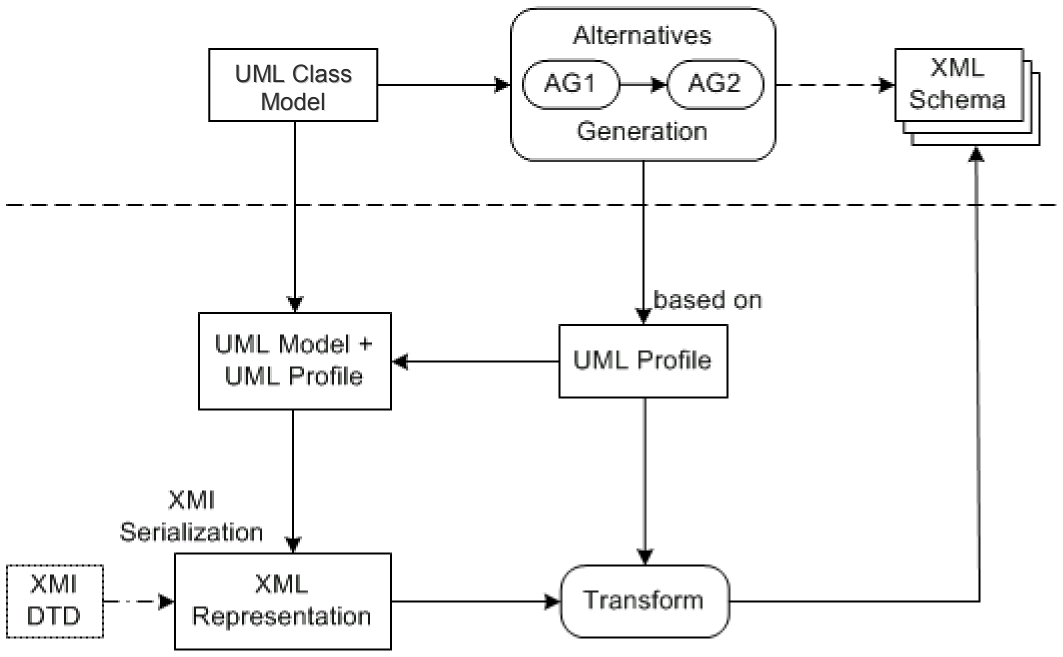


Figure 3-18: Diagram of the approach in mapping UML Model into XML Schema

3.4 Representation of Alternatives

The alternatives chosen in the two steps of Alternative Generation will be represented as UML Profile. UML Profile contains Stereotypes and Tagged Values. Those two will be addressed in the next two sections.

3.4.1 Stereotypes for UML Generalization

The result of Alternatives Generation step 1 is represented using stereotypes. Table 3-7 shows the stereotypes defined for mapping UML Generalization into XML Schema Skeleton.

Table 3-6: Stereotypes used for each possible schema skeleton

XML Schema Skeleton	Stereotype
Element	XSDelement
Complex Type	XSDcomplexType
Containment	XSDcontainment
Derivation	XSDderivation
Reference	XSDreference
Copy-down inheritance	XSDcopyDown

In the above table, the XSDelement and XSDcomplexType are defined by Carlson [2], while XSDcontainment, XSDderivation, and XSDreference are defined by Gunawan. However, both of them do not use those stereotypes for UML Generalization. There is one new definition of stereotype, i.e. XSDcopyDown. This stereotype is used in case a UML generalization is mapped into copy down inheritance in XML Schema. If a generalization in a UML model has a stereotype of XSDelement, then it will be mapped into an element. Thus, the stereotype indicates in what XML Schema skeleton a UML generalization will be mapped.

3.4.2 Tagged Values for UML Generalization

Tagged value can be used to represent the choice of alternatives from Alternatives Generation step 2. There is no tagged value needs to be defined for mapping UML Generalization into element, complex type, containment, and reference. Only mapping to derivation has tagged values. The tagged values defined for derivation are based on the feature diagram of Derivation in Figure 3-14. Those tagged values are:

- baseType (simple | *complex*)
  This tag value is used to represent the feature Base Type. The default tag value is complex.
- derivation (*extension* | restriction)
  This tag value represents the feature Derivation Method. The default is extension.

3.5 Transforming UML Generalization into XML Schema

After the schema skeleton for UML Generalization is generated, the transformation of UML Generalization into XML Schema can be performed using the UML Profile produced in the Alternative Generation step 2. There are some steps that must be taken to transform UML Generalization into XML Schema. Those steps, including the tools used, will be discussed in the next section. Furthermore, an example of transforming generalization into element will be shown.

3.5.1 The Steps for Transforming UML Generalization into XML Schema

Three steps can be identified to do the transformation, as explained below.

- Building UML Model

  Since this project only discusses the mapping of UML Generalization, the UML Model to be built is restricted to UML class diagram. There are several tools that can be used to build UML Model, such as Rational Rose, Rational XDE, Poseidon for UML, ArgoUML, Magicdraw, etc. In this project, Rational Rose Enterprise Edition 2003 is used to build the class diagram and add the UML Profile.

  The UML Profile, i.e. stereotype and tagged value, is added to the diagram in different way. The stereotype is added directly by putting it enclosed in guillemets in front of UML component's name (class, attribute, generalization) or putting it in stereotype field which located in the specification of the UML component. On the other hand, the tagged value cannot be added directly to the model. Rational Extensibility Mechanism can be used for this purpose. Tagged values are represented as user-defined properties. In order to add properties to UML model, Rose Scripting is used. Thus, a script that adds tagged values has to be created.

- XMI Serialization

  XMI Serialization produces an XML document from a UML Model. This mechanism is sometimes included in the tools of UML modeling. For instance, Rational XDE and Poseidon for UML can do the serialization without a requirement of an add-in. However, Rational Rose does not have such a mechanism. An add-in for it, i.e. Unisys Rose XMI Add-in 1.3.6, is used to do the XMI serialization. In this case, UML 1.3 specification and XMI 1.1 specification are used.

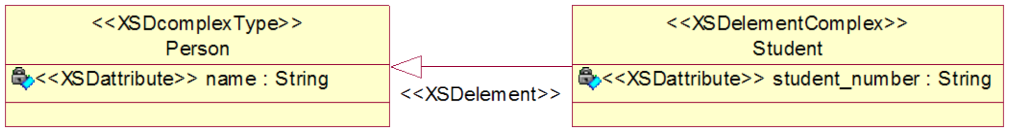
- Transformation using XSLT

XSLT is used to transform the XML document, which is produced by XMI serialization, into XML Schema. Some tools such as XML Spy and XSLT-process can be used for this purpose. In this project, the tool to create the XSLT and do the transformation is XML Spy Enterprise Edition Version 2004. The result of this transformation is the required XML Schema.

3.5.2 Example: Transforming UML Generalization into XML Element

An example of transformation process will be given. The example chosen is mapping of UML Generalization into XML Element.

A UML generalization will be mapped into element if it has XSDelement stereotype. Figure 3-22 shows the UML class diagram that contains generalization. The generalization has XSDelement stereotype, which means that it will be mapped into an element.



<<XSDcomplexType>>
Person
<<XSDattribute>> name : String
<<XSDelement>>
<<XSDelementComplex>>
Student
<<XSDattribute>> student_number : String

Figure 3-19: Example of UML Generalization with stereotype XSDelement

The mappings of class and attribute are based on the thesis of Gunawan [8]. In this case, the attributes are mapping into XML attributes, while the classes are mapped into a complex type and element with complex type. Since the UML generalization mapping to element does not use any tagged value, no tagged value is added to the model. Thus, there is no script crated for mapping UML Generalization into XML Element.

The UML class model in Figure 3-22 is then serialized into an XML document using the add-in of Rational Rose. The result of the XMI serialization an XML document as follows[3].

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- <!DOCTYPE XMI SYSTEM 'UML1311.dtd' > -->
<XMI xmi.version="1.1" xmlns:UML="href://org.omg/UML/1.3" timestamp="Mon Feb 09 02:39:03 2004">
  <XMI.header>
      <XMI.documentation>
          <XMI.exporter>Unisys.JCR.1</XMI.exporter>
```

[3] Note: XML document as the result of the serialization of UML Class Model in Figure 3-22, as it is displayed in the tool XML Spy 2004 Enterprise Edition.

```
            <XMI.exporterVersion>1.3.6</XMI.exporterVersion>
        </XMI.documentation>
        <XMI.metamodel xmi.name="UML" xmi.version="1.3"/>
    </XMI.header>
    <XMI.content>
        <!-- ==================== studentperson    [Model] ==================== -->
        <UML:Model xmi.id="G.0" name="studentperson" visibility="public" isSpecification="false" isRoot="false"
         isLeaf="false" isAbstract="false">
            <UML:Namespace.ownedElement>
                <!-- ==================== studentperson::Person    [Class] ==================== -->
                <UML:Class xmi.id="S.039.0238.48.1" name="Person" visibility="public" isSpecification="false"
                 isRoot="true" isLeaf="false" isAbstract="false" isActive="false" namespace="G.0"
                 specialization="G.2">
                    <UML:Classifier.feature>
                        <!-- ==================== studentperson::Person.name    [Attribute] ============== -->
                        <UML:Attribute xmi.id="S.039.0238.48.2" name="name" visibility="private"
                         isSpecification="false" ownerScope="instance" changeability="changeable"
                         targetScope="instance" type="G.1">
                            <UML:StructuralFeature.multiplicity>
                                <UML:Multiplicity>
                                    <UML:Multiplicity.range>
                                        <UML:MultiplicityRange xmi.id="id.0400138.1" lower="1" upper="1"/>
                                    </UML:Multiplicity.range>
                                </UML:Multiplicity>
                            </UML:StructuralFeature.multiplicity>
                            <UML:Attribute.initialValue>
                                <UML:Expression language="" body=""/>
                            </UML:Attribute.initialValue>
                        </UML:Attribute>
                    </UML:Classifier.feature>
                </UML:Class>
                <UML:Stereotype xmi.id="S.039.0238.49.0" name="XSDcomplexType" visibility="public"
                 isSpecification="false" isRoot="false" isLeaf="false" isAbstract="false" icon="" baseClass="Class"
                 extendedElement="S.039.0238.48.1"/>
                <UML:Stereotype xmi.id="S.039.0238.49.1" name="XSDattribute" visibility="public"
                 isSpecification="false" isRoot="false" isLeaf="false" isAbstract="false" icon="" baseClass="Attribute"
                 extendedElement="S.039.0238.48.2 S.039.0238.48.4"/>
                <!-- ==================== String    [DataType] ==================== -->
                <UML:DataType xmi.id="G.1" name="String" visibility="public" isSpecification="false" isRoot="false"
                 isLeaf="false" isAbstract="false"/>
                <!-- ==================== studentperson::Student    [Class] ==================== -->
                <UML:Class xmi.id="S.039.0238.48.3" name="Student" visibility="public" isSpecification="false"
                 isRoot="false" isLeaf="true" isAbstract="false" isActive="false" namespace="G.0"
                 generalization="G.2">
                    <UML:Namespace.ownedElement>
                        <UML:Generalization xmi.id="G.2" name="" visibility="public" isSpecification="false"
                         discriminator="" child="S.039.0238.48.3" parent="S.039.0238.48.1"/>
                    </UML:Namespace.ownedElement>
                    <UML:Classifier.feature>
                        <!-- ================ studentperson::Student.student_number    [Attribute] ========= -->
                        <UML:Attribute xmi.id="S.039.0238.48.4" name="student_number" visibility="private"
                         isSpecification="false" ownerScope="instance" changeability="changeable"
                         targetScope="instance" type="G.1">
                            <UML:StructuralFeature.multiplicity>
                                <UML:Multiplicity>
                                    <UML:Multiplicity.range>
```

```
                                        <UML:MultiplicityRange xmi.id="id.0400138.2" lower="1" upper="1"/>
                                    </UML:Multiplicity.range>
                                </UML:Multiplicity>
                            </UML:StructuralFeature.multiplicity>
                            <UML:Attribute.initialValue>
                                <UML:Expression language="" body=""/>
                            </UML:Attribute.initialValue>
                        </UML:Attribute>
                    </UML:Classifier.feature>
                </UML:Class>
                <UML:Stereotype xmi.id="S.039.0238.49.2" name="XSDelementComplex" visibility="public"
                 isSpecification="false" isRoot="false" isLeaf="false" isAbstract="false" icon="" baseClass="Class"
                 extendedElement="S.039.0238.48.3"/>
                <UML:Stereotype xmi.id="S.039.0238.49.3" name="XSDelement" visibility="public"
                 isSpecification="false" isRoot="false" isLeaf="false" isAbstract="false" icon=""
                 baseClass="Generalization" extendedElement="G.2"/>
            </UML:Namespace.ownedElement>
        </UML:Model>
        <UML:TaggedValue xmi.id="XX.9.238.49.2" tag="persistence" value="transient"
         modelElement="S.039.0238.48.1"/>
        <UML:TaggedValue xmi.id="XX.9.238.49.3" tag="persistence" value="transient"
         modelElement="S.039.0238.48.3"/>
    </XMI.content>
</XMI>
```

The XML document produced above should be transformed into XML Schema using XSLT. The XSLT files used are based on the thesis by Linda since a UML Generalization cannot stand alone separated from classes. The mapping of classes has been conducted by Gunawan [8]. Thus, some files created by Gunawan are reused, i.e. main.xslt, attribute.xslt, complex_type.xslt, element.xslt, and data_type_and_multiplicity.xslt. However, many new rules have to be defined since the mapping of generalization is not included in the files. To map UML generalization into an element, the element.xslt and complex_type.xslt files are modified and added by new rules.

The element.xslt file should contain UMLGeneralization2Element template, which is doing the mapping. Below is the UMLGeneralization2Element template[4].

```
  <xsl:template name="umlGeneralization2Element">
        <xsl:param name="child_id"/>
        <xsl:param name="parent_id"/>
        <xsl:param name="stereotype_generalization"/>
        <xsl:param name="generalization_name"/>
        <xsl:variable name="child_name">
            <xsl:value-of select="//UML:Class[@xmi.id=$child_id]/@name"/>
        </xsl:variable>
        <xsl:variable name="parent_name">
```

[4] Note: UMLGeneralization2Element template as it is displayed in the tool XML Spy 2004 Enterprise Edition.

```
            <xsl:value-of select="//UML:Class[@xmi.id=$parent_id]/@name"/>
        </xsl:variable>

        <xs:element>
            <xsl:attribute name="name">
                <xsl:choose>
                    <xsl:when test="not($generalization_name='')">
                        <xsl:value-of select="$generalization_name"/>
                    </xsl:when>
                    <xsl:otherwise>
                        <xsl:value-of select="$child_name"/>-<xsl:value-of select="$parent_name"/>
                    </xsl:otherwise>
                </xsl:choose>
            </xsl:attribute>
            <xs:complexType>
                <xs:sequence>
                    <xs:element>
                        <xsl:attribute name="name">child</xsl:attribute>
                            <xs:complexType>
                                    <xs:element>
                                        <xsl:attribute name="name">the<xsl:value-of select="$child_name"/>
                                        </xsl:attribute>
                                            <complexType>
                                                <xs:sequence/>
                                                    <xs:attribute>
                                                        <xsl:attribute name="name">idref</xsl:attribute>
                                                        <xsl:attribute name="type">xs:IDREF</xsl:attribute>
                                                    </xs:attribute>
                                            </complexType>
                                    </xs:element>
                            </xs:complexType>
                    </xs:element>
                    <xs:element>
                        <xsl:attribute name="name">parent</xsl:attribute>
                            <xs:complexType>
                                    <xs:element>
                                        <xsl:attribute name="name">the<xsl:value-of select="$parent_name"/>
                                        </xsl:attribute>
                                            <complexType>
                                                <xs:sequence/>
                                                    <xs:attribute>
                                                        <xsl:attribute name="name">idref</xsl:attribute>
                                                        <xsl:attribute name="type">xs:IDREF</xsl:attribute>
                                                    </xs:attribute>
                                            </complexType>
                                    </xs:element>
                            </xs:complexType>
                    </xs:element>
                </xs:sequence>
            </xs:complexType>
        </xs:element>

    </xsl:template>
```

The complex_type.xslt is changed a little. It now contains the conditional processing to test whether the stereotype of a generalization is XSDelement. Below is the conditional processing for the requirement.

```
<xsl:if test="not($generalization_id='') and ($stereotype_generalization='XSDelement')">
                <xsl:call-template name="umlGeneralization2Element">
                    <xsl:with-param name="child_id" select="$class_id"/>
                    <xsl:with-param name="parent_id" select="$parent_id"/>
                    <xsl:with-param name="stereotype_generalization" select="$stereotype_generalization"/>
                    <xsl:with-param name="generalization_name" select="$generalization_name"/>
                </xsl:call-template>
</xsl:if>
```

Also, there is a test to check whether id attribute of the child and parent class should be added due to the generalization mapping into an element. The conditional processing is as follow.

```
<!--check id attribute should be added due to the generalization-->
<xsl:if test="$stereotype_generalization='XSDelement' or $stereotype_generalization='XSDcomplexType'">
                <xs:attribute>
                    <xsl:attribute name="name">id</xsl:attribute>
                    <xsl:attribute name="type">xs:ID</xsl:attribute>
                </xs:attribute>
</xsl:if>
<xsl:if test="$stereotype_specialization='XSDelement' or $stereotype_specialization='XSDcomplexType'">
                <xs:attribute>
                    <xsl:attribute name="name">id</xsl:attribute>
                    <xsl:attribute name="type">xs:ID</xsl:attribute>
                </xs:attribute>
</xsl:if>
```

The XML schema as the result of the transformation is as follows. This schema is the mapping of the UML class diagram in Figure 3-22[5].

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:complexType name="Person">
        <xs:all/>
        <xs:attribute name="name" type="xs:string"/>
        <xs:attribute name="id" type="xs:ID"/>
    </xs:complexType>
    <xs:element name="Student" type="Student"/>
    <xs:complexType name="Student">
        <xs:all/>
        <xs:attribute name="student_number" type="xs:string"/>
        <xs:attribute name="id" type="xs:ID"/>
    </xs:complexType>
    <xs:element name="Student-Person">
```

[5] Note: XML schema produced from UML class diagram in Figure 3-22 as it is displayed in the tool XML Spy 2004 Enterprise Edition.

```
        <xs:complexType>
            <xs:sequence>
                <xs:element name="child">
                    <xs:complexType>
                        <xs:element name="theStudent">
                            <complexType>
                                <xs:sequence/>
                                <xs:attribute name="idref" type="xs:IDREF"/>
                            </complexType>
                        </xs:element>
                    </xs:complexType>
                </xs:element>
                <xs:element name="parent">
                    <xs:complexType>
                        <xs:element name="thePerson">
                            <complexType>
                                <xs:sequence/>
                                <xs:attribute name="idref" type="xs:IDREF"/>
                            </complexType>
                        </xs:element>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

The generalization between Student class and Person class is mapped into an element. This element has the type of complex type, which has two elements, i.e. child element and parent element. The child element has theStudent element which refers to the Student element defined before, while the parent element has thePerson element which refers to the Person complex type defined before.

3.6 Summary

This chapter explains the mapping of UML Generalization into XML schema. There are two steps of generation, i.e. schema skeleton generation and schema skeleton refinement. The refinement of UML Generalization has been discussed in detail for each possible mapping. The UML Profiles to represent the alternatives have been described. Furthermore, the transformation process of UML Generalization into XML Schema has been presented from the practical point of view, including an example to complete the explanation.

Chapter 4
Rules in Mapping UML Class Models into XML Schema's

This chapter describes the rules in mapping UML class models into XML schema's. The first section will discuss about the rules for class. The rules for attribute will be explained in the next section. Section 4.3 contains the rules for association, including aggregation and composition. Section 4.4 explains the rules for generalization. The last section is summary.

4.1 Rules for Class

In the UML Specification [13], especially in Figure 2-5 on page 2-13, it is defined that a Class is a specialization of a Classifier, a Classifier is a specialization of a GeneralizableElement, a GeneralizableElement is a specialization of a ModelElement, and a ModelElement is a specialization of an Element. This specification is stated in the Core Package – Backbone. This implies that some properties of class are owned by itself, some are inherited from Classifier, some are inherited by Classifier from GeneralizableElement, some are inherited by GeneralizableElement from ModelElement, and some are inherited by ModelElement from Element.

Table 4-1 contains the properties of UML Class and their mappings into XML Schema constructs, if available. Each property is mapped into XML Schema construct, if it is available; otherwise the information brought by the property is lost. The mapping of constructs of the UML Class Model into constructs of the XML Schema is based on the similarity of the information or characteristics given by the constructs. For instance, property *name* of a UML class is mapped into property *name* of element or complex type or simple type because both properties of each model bring the same information, i.e. name. Moreover, the last column is also provided to give comment whether information concerning a UML class implied by the property is lost, or information is added due to the mapping.

Table 4-1: Mapping UML Class Properties into XML Schema Constructs

Mapping Number	Property of UML Class	XML Schema Construct	Information Loss/Added
M.1.1	documentation (tagged value inherited from ModelElement)	annotation of complex type or simple type	-
M.1.2	name (attribute inherited from	name property of element or complex	-

	ModelElement)	type or simple type	
M.1.3	asArgument (association inherited from ModelElement)	-	information loss
M.1.4	clientDependency (association inherited from ModelElement)	-	information loss
M.1.5	constraint (association inherited from ModelElement)	-	information loss
M.1.6	implementationLocation (association inherited from ModelElement)	-	information loss
M.1.7	namespace (association inherited from ModelElement)	target namespace of the schema	-
M.1.8	presentation (association inherited from ModelElement)	-	information loss
M.1.9	supplierDependency (association inherited from ModelElement)	-	information loss
M.1.10	templateParameter (association inherited from ModelElement)	-	information loss
M.1.11	derived (tagged value inherited from ModelElement)	-	information loss
M.1.12	isAbstract (attribute inherited from GeneralizableElement)	- abstract property of element or complex type - choice between complex type and element	-
M.1.13	isLeaf (attribute inherited from GeneralizableElement)	block attribute to prevent extension	information loss
M.1.14	isRoot (attribute inherited from GeneralizableElement)	-	information loss
M.1.15	generalization (association inherited from GeneralizableElement)	used in the mapping of generalization	-
M.1.16	specialization (association inherited from GeneralizableElement)	used in the mapping of generalization	-
M.1.17	feature (association inherited from Classifier), consists of attributes and operations	Mapping attribute is discussed later	-
M.1.18	association (association inherited from Classifier)	used in the mapping of association	-
M.1.19	powertypeRange (association inherited from Classifier)	-	information loss
M.1.20	specifiedEnd (association inherited from Classifier)	-	information loss
M.1.21	<<metaclass>> (stereotype inherited from Classifier)	-	information loss
M.1.22	<<powertype>> (stereotype inherited from Classifier)	-	information loss
M.1.23	<<process>> (stereotype inherited from Classifier)	-	information loss
M.1.24	<<thread>> (stereotype inherited from Classifier)	-	information loss
M.1.25	<<utility>> (stereotype inherited from Classifier)	-	information loss
M.1.26	persistence (tagged value inherited from Classifier)	-	information loss

M.1.27	semantics (tagged value inherited from Classifier)	-	information loss
M.1.28	isActive (attribute owned by Class)	-	information loss
M.1.29	<<auxiliary>> (stereotype owned by Class)	-	information loss
M.1.30	<<focus>> (stereotype owned by Class)	-	information loss
M.1.31	<<implementation>> (stereotype owned by Class)	-	information loss
M.1.32	<<type>> (stereotype owned by Class), especially <<enumeration>>	- enumeration facet of simple type - choice ordering of complex type	-
M.1.33	un-ordering attributes in a class	group ordering of XML Schema, i.e. all, sequence, and choice	information added

Most of those constructs are ignored in the mapping discussed in this project. However, the mapping is possible in theory. At least, it is possible in XMI. In this project, only some constructs that are considered commonly used that will be used in the mapping.

From the mapping of UML Class properties into XML constructs, rule of mapping UML Class into XML Schema can be defined by combining two or more mapping of properties described above. Table 4-2 shows the mapping rule derived from the above mapping, and the explanation of the rule. The rules defined the most common cases found in class in the UML Class Diagram.

Table 4-2: Mapping Rules of UML Class into XML Schema

Rule Number	Derived from Mapping Number	Source Model (UML Class)	Target Model (XML Schema)	Rationale
R.1.1	M.1.2, M.1.12 and M.1.32	non-abstract class (class with isAbstract value 'false'), no <<enumeration>> stereotype	element declaration with complex type definition	UML class is a container of structural and behaviour features, such as attributes and associations. Its representation in XML should be a container of other XML Schema constructs, such as attributes or child elements. Thus, UML class is mapped into XML complex type so that it can contain child element or attributes as the representation of attributes and associations owned by the class. Furthermore, the 'false' value of isAbstract attribute means that the class needs to be instantiated; hence an element with the type of that complex type definition should also be declared.
R.1.2	M.1.12	abstract class (isAbstract value is 'true')	a. complex type	- The 'true' value of isAbstract attribute of UML class implies that the class is an abstract class and may not be directly instantiated [UML Specification], but it may still have some features such as attributes and associations which can be reused by

			b. element with complex type, their abstract attribute values are 'true' c. model group	other class. This characteristic is similar with complex type in XML Schema. Complex type cannot be instantiated without the declaration of element. - An element with abstract attribute value 'true' cannot be directly instantiated. This also corresponds to the meaning of isAbstract attribute of UML class. - Model group can also represent the abstract class since an abstract class does not need to be directly instantiated.
R.1.3	M.1.32 and M.2.19	<<enumeration>> stereotype, type of class attribute is primitive data type	simple type with enumeration facet	A UML class with a <<enumeration>> stereotype usually does not need to be directly instantiated, but is referred by other feature, such as attribute, which then will instantiate the element of the type. Thus the class does not need to be mapped into an element. When the data type defined in the enumeration as the attributes of the class is primitive data type, e.g. string, it can be mapped into the suitable built-in simple type in XML Schema. Thus, the restriction is based on the suitable built-in simple type of XML Schema. The attributes are each mapped into the enumeration value.
R.1.4	M.1.32 and M.2.19	<<enumeration>> stereotype, type of class attribute is user defined data type	complex type with choice ordering constraint as the container of the child elements	Since the type of attributes is not primitive data type, simple type cannot be applied, and thus complex type is chosen. Besides, class with a <<enumeration>> stereotype does not need to be directly instantiated. In enumeration, only one value can be used. Thus, the choice group ordering is the most suitable ordering constraint to be used. The value of attribute can be represented as the child elements.
R.1.5	M.1.32 and M.2.19	<<enumeration>> stereotype, type of attribute is not primitive data type	complex type with choice ordering constraint as the container of the child elements	Since the type of attributes is not primitive data type, simple type cannot be applied, and thus complex type is chosen. Besides, class with a <<enumeration>> stereotype does not need to be directly instantiated. In enumeration, only one value can be used. Thus, the choice group ordering is the most suitable ordering constraint to be used. The value of attribute can be represented as the child elements.
R.1.6	M.1.33	container of un-order attributes	a. all group ordering	- The UML attributes and associations contained in a UML class are not ordered. They can appear in any order. The only group ordering of XML Schema that allows un-order element is the all constraint order. The all is the wrapper of the child elements as the representation of UML attributes and

				relationship owned the class. However, some characteristics of the all group ordering, such as the allowed minoccurs and maxoccurs of element [21], have to be considered in the mapping of UML attributes and UML associations, as seen in the next sections.
			b. sequence group ordering	- It can be assumed that the order of attributes and relationships (associations and generalizations) are ordered. If this is the case, the sequence content model can be used. This will simplify the representation of attributes and relationships in the class since the sequence group ordering does not have constraint for the minoccurs and maxoccurs attributes of child elements.

One important rule mapping of UML Class that affects the definition of the rule mapping of UML Attribute, association, and generalization is rule R.1.5. This shows that there is dependency between the rules, i.e. the choice of one rule influences the choice of other rules. In this project, the first choice of the rule, i.e. use of all group ordering, is applied in order to keep the features of a class to be un-ordered.

4.2 Rules for Attribute

The UML Specification [13] for Core Package – Backbone described by Figure 2-5 on page 2-13 defines that an Attribute is a specialization of a StructuralFeature, a StructuralFeature is a specialization of a Feature, a Feature is a specialization of a ModelElement, and a ModelElement is a specialization of an Element. This means that some properties of Attribute are owned by itself, some are inherited from StructuralFeature, some are inherited by StructuralFeature from Feature, some are inherited by Feature from ModelElement, and some are inherited by ModelElement from Element.

Below is the table that contains the properties of UML Attribute and their mappings to XML Schema constructs, if available. Similar to the table for UML class, the last column is provided to give comment whether information concerning a UML Attribute implied by the property is lost, or information is added due to the mapping.

Table 4-3: Mapping UML Attribute Properties into XML Schema Constructs

Mapping Number	Property of UML Attribute	XML Schema Construct	Information Loss/Added
M.2.1	documentation (tagged value inherited from ModelElement)	annotation of element or attribute	-

M.2.2	name (attribute inherited from ModelElement)	name property of (child) element or attribute	-
M.2.3	asArgument (association inherited from ModelElement)	-	information loss
M.2.4	clientDependency (association inherited from ModelElement)	-	information loss
M.2.5	constraint (association inherited from ModelElement)	-	information loss
M.2.6	implementationLocation (association inherited from ModelElement)	-	information loss
M.2.7	namespace (association inherited from ModelElement)	target namespace of the schema	-
M.2.8	presentation (association inherited from ModelElement)	-	information loss
M.2.9	supplierDependency (association inherited from ModelElement)	-	information loss
M.2.10	templateParameter (association inherited from ModelElement)	-	information loss
M.2.11	derived (tagged value inherited from ModelElement)	-	information loss
M.2.12	ownerScope (attribute inherited from Feature)	-	information loss
M.2.13	visibility (attribute inherited from Feature)	the location of an element, directly as the child element of the root or not	-
M.2.14	owner (association inherited from Feature)	the parent of an element/attribute	-
M.2.15	changeability (attribute inherited from StructuralFeature)	-	information loss
M.2.16	multiplicity (attribute inherited from StructuralFeature)	minoccurs and maxoccurs properties of element	-
M.2.17	ordering (attribute inherited from StructuralFeature)	can be modeled only if a UML attribute is mapped into XML element	-
M.2.18	targetScope (attribute inherited from StructuralFeature)	-	information loss
M.2.19	type (association inherited from StructuralFeature)	type of element, either built-in simple type or user-defined complex type	-
M.2.18	persistence (tagged value inherited from StructuralFeature)	-	information loss
M.2.19	initialValue (attribute owned by Attribute)	default value constraint property of attribute or element	-
M.2.20	associationEnd (association owned by Attribute)	may be used in the mapping of association	-

From the mapping of UML Attribute properties into XML constructs, rule of mapping UML Attribute into XML Schema can be defined by combining two or more mapping of properties described above. Again, most properties are ignored. Table 4-4 shows the mapping rule derived from the above mapping, and the explanation of the rule. The rules defined the most common cases found in attributes of UML Class Diagram.

Table 4-4: Mapping Rules of UML Attribute into XML Schema

Rule Number	Derived from Mapping Number	Source Model (UML Attribute)	Target Model (XML Schema)	Rationale
R.2.1	M.2.19	type other than String	element with suitable type	An XML attribute is always of string type. Thus, UML attribute of type other than string must be mapped into XML element so that it can be represented by the suitable type of XML element. The type can be the built-in simple type of XML Schema or complex type defined as the representation of data type. This decision of type mapping must conform to the type of UML attribute.
R.2.2	M.2.19 and M.2.16	short string type and no multiplicity	attribute	A UML attribute of type String with simple value (no multiple spaces, linefeeds or large string values) can be mapped into XML attribute because XML attribute can only be string with simple value. Moreover, the UML attributes must not have multiplicity because XML attribute does not have a constraint for multiplicity.
R.2.3	M.2.19	expected to contain multiple spaces, linefeeds or large string values	element with suitable type (simple type if the data type is primitive data type, and complex type if data type is not primitive)	If a UML attribute is expected to contain multiple spaces, linefeeds or large string values, it cannot be mapped into XML attribute since XML attribute is white space normalized and cannot contains large string value [2]. Hence, the UML attribute with those characteristics should be mapped into XML element.
R.2.4	M.2.16 and M.1.33	with multiplicity (multi-valued attribute)	element with suitable type (simple type if the data type is primitive data type, and complex type if data type is not primitive)	An XML attribute does not have representation for multiplicity. Thus, a UML attribute that has multiplicity value must be mapped into XML element. The multiplicity is represented by the minoccurs and maxoccurs attributes. Furthermore, a class is decided to have the all constraint order. No element inside the all content model may appear more than once, which means that the permissible values of minOccurs and maxOccurs are 0 and 1. This problem can be solved by defining an anonymous complex type inside the

				element as the wrapper of a child element with minOccurs and maxOccurs more than 1. The name of the child element can be derived from the name of element, which is the same with the UML attribute name, by attaching some letters, such as 'the', in front of the element name.
R.2.5	M.2.13	visibility value is 'public' or 'package'	element as the direct child of the root element	A UML attribute has a 'public' or 'package' visibility value if it is visible outside the class (outside or inside the package). Thus, it can be accessed by other classes. The XML element that represents the UML attribute can only be referred by other element if it is a direct child of the root element.
R.2.6	M.2.19	initialValue	default of element or attribute	An initialValue of UML attribute specifies the value of the attribute upon initialization. This similar with the function of default value of XML element or attribute. Thus the value of UML attribute initialValue becomes the default value of XML element or attribute.
R.2.7	M.2.19, M.2.16 and M.2.13	group of UML attributes	attribute group	A group of attributes of a UML class can be mapped into attribute groups if: • Each of them can be mapped into XML attribute (M.2.19 and M.2.16). • There is expectation that it will be reused in several complex type definitions, for instance has the visibility value of 'public' (M.2.13).

4.3 Rules for Association

The UML Specification [13] for Core Package – Backbone (Figure 2-5 on page 2-13) defines that an Association is a specialization of a Relationship, while a Relationship is a specialization of a ModelElement. This means that some properties of Association are owned by itself, and some are inherited by Relationship from ModelElement (Relationship does not have any property). Furthermore, Association has two or more AssociationEnd (with a composition relationship). The AssociationEnd is a specialization of ModelElement, and thus inherits some of the properties of ModelElement.

Table 4-5 contains the properties of UML Association and their mappings to XML Schema constructs, if available. The last column is provided to give comment whether information concerning a UML Association implied by the property is lost, or information is added due to the mapping.

Table 4-5: Mapping UML Association Properties into XML Schema Constructs

Mapping Number	Property of UML Association	XML Schema Construct	Information Loss/Added
M.3.1	name (attribute inherited from ModelElement)	name property of (child) element or complex type	-
M.3.2	asArgument (association inherited from ModelElement)	-	information loss
M.3.3	clientDependency (association inherited from ModelElement)	-	information loss
M.3.4	constraint (association inherited from ModelElement)	-	information loss
M.3.5	implementationLocation (association inherited from ModelElement)	-	information loss
M.3.6	namespace (association inherited from ModelElement)	target namespace of the schema	-
M.3.7	presentation (association inherited from ModelElement)	-	information loss
M.3.8	supplierDependency (association inherited from ModelElement)	-	information loss
M.3.9	templateParameter (association inherited from ModelElement)	-	information loss
M.3.10	derived (tagged value inherited from ModelElement)	-	information loss
M.3.11	connection (association owned by Association)	-	information loss
M.3.12	<<implicit>> (stereotype owned by Association)	-	information loss
M.3.13	{xor} (constraint owned by Association)	choice content model	-
M.3.14	persistence (tagged value owned by Association)	-	-

The next table, i.e. Table 4-6, contains the properties of UML AssociationEnd and their mappings to XML Schema constructs, if available. The last column is provided to give comment whether information concerning a UML AssociationEnd implied by the property is lost, or information is added due to the mapping.

Table 4-6: Mapping UML AssociationEnd Properties into XML Schema Constructs

Mapping Number	Property of UML AssociationEnd	XML Schema Construct	Information Loss/Added
M.3.16	name (attribute inherited from ModelElement, becomes the role name of AssociationEnd)	name property of element or complex type	-

M.3.17	asArgument (association inherited from ModelElement)	-	information loss
M.3.18	clientDependency (association inherited from ModelElement)	-	information loss
M.3.19	constraint (association inherited from ModelElement)	-	information loss
M.3.20	implementationLocation (association inherited from ModelElement)	-	information loss
M.3.21	namespace (association inherited from ModelElement)	target namespace of the schema	-
M.3.22	presentation (association inherited from ModelElement)	-	information loss
M.3.23	supplierDependency (association inherited from ModelElement)	-	information loss
M.3.24	templateParameter (association inherited from ModelElement)	-	information loss
M.3.25	derived (tagged value inherited from ModelElement)	-	information loss
M.3.26	aggregation (attribute owned by AssociationEnd)	- (child) element with reference - (child) element with instance of	-
M.3.27	changeability (attribute owned by AssociationEnd)	-	information loss
M.3.28	ordering (attribute owned by AssociationEnd)	-	information loss
M.3.29	isNavigable (attribute owned by AssociationEnd)	- containment (element A contains element B) - reference (element A refers element B)	-
M.3.30	multiplicity (attribute owned by AssociationEnd)	minoccurs and maxoccurs properties of element	-
M.3.31	targetScope (attribute owned by AssociationEnd)	-	information loss
M.3.32	visibility (attribute owned by AssociationEnd)	the location of element declaration	-
M.3.33	qualifier (association owned by AssociationEnd)	-	information loss
M.3.34	specification (association owned by AssociationEnd)	-	information loss
M.3.35	participant (association owned by AssociationEnd)	-	information loss
M.3.36	(unnamed composite end) (association owned by AssociationEnd)	-	information loss
M.3.37	<<association>> (stereotype owned by AssociationEnd)	-	information loss
M.3.38	<<global>> (stereotype owned by AssociationEnd)	-	information loss
M.3.39	<<local>> (stereotype owned by AssociationEnd)	-	information loss
M.3.40	<<parameter>> (stereotype owned by AssociationEnd)	-	information loss
M.3.41	<<self>> (stereotype owned by AssociationEnd)	-	information loss

From the above mapping of properties of UML Association and UML AssociationEnd, some mapping rules can be derived. Table 4-7 shows the mapping rules produced. The rules defined the most common cases found in association in the UML Class Diagram. Although they are not mandatory mapping, the rules are recommended because they consider the important properties of associations, including the properties of association ends. If another type of mapping is chosen, i.e. the mapping that does not conform to the rule, it could be the case that one or more properties of association as well as association ends cannot be preserved.

Table 4-7: Mapping Rules of UML Association into XML Schema

Rule Number	Derived from Mapping Number	Source Model (UML Association/ AssociationEnd)	Target Model (XML Schema)	Rationale
R.3.1	M.3.26	aggregation value of one of the association ends is 'composite' (composition)	containment	An association end that has an aggregation attribute value of 'composite' indicates 'uses a' relationship between the involved class [UML Aggregation vs Composition]. It also suggests that that the related classes are owned by value, not by reference. The lifetime of the source class depends on the lifetime of composite class, which means that if the composite class is destroyed, the source class will also disappear. The XML containment is the only XML construct that can represent this characteristic, in which the element as the representation of the composite class contains the child element as the representation of the source class.
R.3.2	M.3.26	aggregation value of one of the association ends is 'aggregate' (aggregation)	containment	An association end that has an aggregation attribute value of 'aggregate' indicates 'has a' relationship between the involved class [UML Aggregation vs Composition]. Only containment of XML Schema can represent the 'has' relationship because containment implies that an element contains a child element, or in other words, an element has a child element.

R.3.3	M.3.26, M.3.1, and M.3.30	aggregation value of both association ends are 'none', name of association is "", and the multiplicities of both association ends are 1	reference	If the multiplicity values of both association ends are exactly 1, the aggregation values are "none", and no association name, the association is mapped into reference, since reference does not need any name. The navigability information is used to decide which element (as the representation of the involved class) refers to which element (as the representation of the other involved class).
R.3.4	M.3.26, M.3.29, and M.3.30	aggregation value of both association ends are 'none', bidirectional (both association ends are navigable), or both association ends are not navigable, one multiplicity of the association ends is 1	containment	This type of association can be mapped into containment, in which the element representing the class with multiplicity 1 contains the element representing the class with multiplicity other than 1. Reference is not chosen because if the association has name, the name cannot be preserved. Moreover, the multiplicity (other than 1) owned by one of the association ends is convenient to be mapped into a multiplicity of a child element.
R.3.5	M.3.26 and M.3.29	aggregation value of both association ends are 'none', bidirectional (both association ends are navigable), or both association ends are not navigable	element declaration with complex type definition	Since the values of the navigability of both association ends are the same, the representation in XML Schema does not need to consider the direction of the association. The complex type contains the child elements as the representations of the classes involved in the association. The definition of complex type must be accompanied with the declaration of element so that it can be instantiated (if the classes involved need to be instantiated). Reference is not chosen here due to the same reason as R.3.4 above.
R.3.6	M.3.26 and M.3.29	aggregation value of both association ends are 'none', unidirectional	containment	Since this type of association is unidirectional, the direction of the association must be considered in the mapping. The direction is shown by containment. The element that represents the

				class in the opposite direction of the navigation contains the child element that represents the class directed by the navigation. Reference is not chosen here due to the same reason as R.3.4 above.
R.3.7	M.3.13	two or more associations with {xor} constraint	model group with choice content model	Group of associations with {xor} constraint implies that exactly one association must be chosen among the association choices. This can be represented in XML Schema using the model group with choice content model. The class associated with all of the associations contains a child element in which an anonymous complex type is defined as the wrapper of the model group. Each element in the model group represents each association.

4.4 Rules for UML Generalization

The UML Specification [13] for Core Package – Backbone (Figure 2-5 on page 2-13) defines that a Generalization is a specialization of a Relationship, while a Relationship is a specialization of a ModelElement. This means that some properties of Generalization are owned by itself, and some are inherited by Relationship from ModelElement. Furthermore, Generalization has an association relationship with GeneralizableElement, which is the generalization of the classes (superclass and child class) involved in the generalization. Only two properties of GeneralizableElement, i.e. generalization and specialization, that is important for generalization. Generalization also has an association relationship with Classifier, which is the generalization of the class, in which the powertypeRange property is used in the association relationship. The mapping of GeneralizableElement and Classifier properties has been discussed in the mapping of UML Class properties, and thus, will not be re-discussed here.

Table 4-8 contains the properties of UML Generalization and their mappings to XML Schema constructs, if available. The last column is provided to give comment whether information concerning a UML Generalization implied by the property is lost, or information is added due to the mapping.

Table 4-8: Mapping UML Generalization Properties into XML Schema Constructs

Mapping Number	Property of UML Generalization	XML Schema Construct	Information Loss/Added
M.4.1	name (attribute inherited from ModelElement)	name property of (child) element or complex type	-
M.4.2	asArgument (association inherited from ModelElement)	-	information loss
M.4.3	clientDependency (association inherited from ModelElement)	-	information loss
M.4.4	constraint (association inherited from ModelElement)	-	information loss
M.4.5	implementationLocation (association inherited from ModelElement)	-	information loss
M.4.6	namespace (association inherited from ModelElement)	target namespace of the schema	-
M.4.7	presentation (association inherited from ModelElement)	-	information loss
M.4.8	supplierDependency (association inherited from ModelElement)	-	information loss
M.4.9	templateParameter (association inherited from ModelElement)	-	information loss
M.4.10	derived (tagged value inherited from ModelElement)	-	information loss
M.4.11	discriminator (attribute owned by Generalization)	-	information loss
M.4.12	child (association owned by Generalization)	- derived element - element that copies down all the content of element representing the superclass	-
M.4.13	parent (association owned by Generalization)	- the element whose type is the base of derivation - element whose contents are copied down by the element representing the child class	-
M.4.14	powertype (association owned by Generalization)	-	information loss
M.4.15	complete (constraint owned by Generalization)	-	information loss
M.4.16	disjoint (constraint owned by Generalization)	-	information loss
M.4.17	incomplete (constraint owned by Generalization)	-	information loss
M.4.18	overlapping (constraint owned by Generalization)	-	information loss

From the above mapping of properties of UML Generalization, some mapping rules can be derived. Table 4-9 shows the mapping rules produced. It also contains the explanation of the rule, and the possibility of subclassing. The rules defined the most common cases found in generalization in the UML Class Diagram.

Table 4-9: Mapping Rules of UML Generalization into XML Schema

Rule Number	Derived from Mapping Number	Source Model (UML Generalization)	Target Model (XML Schema)	Rationale
R.4.1	M.1.15 and M.4.12	single inheritance, i.e. child class inherits one superclass	a. derivation	- In inheritance, a child class should inherit the features of its superclass. In XML, this can be represented by derivation. The child class is represented as the derivation of the element representing the superclass. The derivation is the extension based on the type of the element representing the superclass.
			b. containment	- This type of generalization can also be mapped into containment, in which the element as the representation of the child class contains the element as the representation of the superclass. In this way, all the attributes and characteristics of the element representing the superclass are owned by the element representing the child class.
R.4.2	M.1.15 and M.4.12	multiple inheritance, i.e. child class inherits multiple superclasses	a. copy down inheritance	- A class that inherits multiple superclasses cannot be mapped into derivation since the derivation only allows one base type. However, the characteristic of inheritance in which the child class should inherit the features of the parent class should be maintained. This can be done by copying the attributes and child element of each element representing the superclass to the element representing the child class.
			b. containment	- Containment can also represent this type of generalization, in which the element representing the child class contains the all of the elements representing the superclasses.

4.5 Summary

This chapter contains the heuristics rules and how to obtain them. Heuristics rules are constructed based on the properties of each construct of UML Class Model. Each property is mapped into XML Schema construct, if it is available; otherwise the information implied by the property is lost. The mapping of the constructs of UML Class Model into the constructs of the XML Schema is based on the similarity of the information or characteristics given by the constructs. Furthermore, the rules for each UML construct, i.e. UML Class, UML Attribute, UML Association, and UML generalization have been developed. The rules will be useful for the next step of mapping UML Class Diagram into XML Schema.

Chapter 5
Transformation of UML Class Model into XML Schema Using Design Algebra

This chapter discusses the transformation of an UML class model into an XML schema using a technique called Design Algebra. In section 5.1, the four layer metamodeling architecture is explained. Following is the discussion about Design algebra. The next section contains the transformation of UML model into XML schema and the implementation to the case study. The last section is summary.

5.1 Four Layer Metamodeling Architecture

OMG Unified Modeling Language Specification specifies the four layer modeling. Table 5-1 that contains the four layer metamodeling architecture. It is a mixed of table of Four Layer Metamodeling Architecture specified in the UML Specification [13] with the table of OMG Metadata Architecture specified in XMI Specification [14].

Table 5-1: The four layer metamodeling architecture

Meta-level	MOF terms	Description	Examples
M3	meta-metamodel	The infrastructure for a modeling architecture. Defines the language for specifying metamodels.	The “MOF Model”
M2	meta-metadata metamodel	An instance of a meta-metamodel. Defines the language for specifying a model.	UML Metamodel, CWMI Metamodel(s), etc.
M1	metadata model	An instance of a metamodel. Defines a language to describe an information domain.	UML Models, Warehouse Schemas, etc.
M0	data	An instance of a model. Defines a specific information domain.	Modeled systems, Warehouse databases, etc.

Table 5-1 shows the correlation between the terms defined such as model, metamodel, and user data, with the terms of meta-level. From the table, it can be seen that UML Metamodel is defined in the second layer, i.e. metamodel, which is called M2 in the term of meta-level. Furthermore, it can also be understood that one layer defines one layer below. On the other hand, one layer is an instance of one layer above. For instance, a metamodel defines a model and is an instance of a meta-metamodel.

Figure 5-1 shows the generation of XML Schema from a UML Class Model related to four-layer modelling architecture [8].

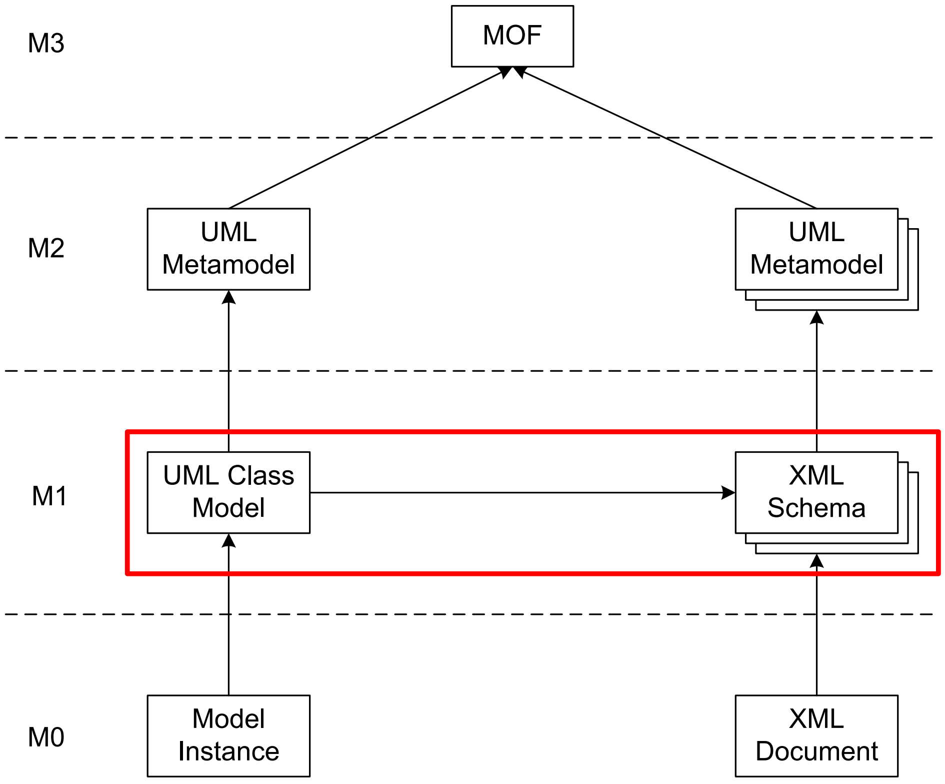


Figure 5-1: Mapping UML into XML Schema related with the four-layer metamodeling architecture

As depicted by the figure, UML Metamodel and XML Schema Metamodel are in layer M2. Those are the instance of MOF. UML Model, as the instance of UML Metamodel is in layer M1, while XML Schema as the instance of XML Schema Metamodel is also in layer M1.

The generation of XML Schema from UML Class Model is located in layer M2. In this case, the source model is UML Model, while the target model is XML Schema. This generation will be discussed in the next sections. A technique called Design Algebra is used to model a set of alternative transformations.

5.2 Design Algebra

This project uses the technique called Design Algebra to model a set of alternative transformations for UML Class Model as the source model into XML Schema as the target model. Design Algebra is a technique that can be used to do the transformation from a source model into the target model. The technique explicitly models a set of alternative transformation for the source model. This technique also allows the specification of quality properties of the target model which will be used as selection criteria among the alternatives.

A paper written by Kurtev, Berg, and Aksit [12] describes a process and technique for constructing and utilizing a set of alternative transformations for a given source model. The source model, its meta-model, and the meta-model of the target models are used as input to generate a *transformation space* for the source model. The transformation space contains a set of alternatives transformations for a given source model. It is a multidimensional space spanning a number of independent *dimensions*. The dimensions of a transformation space are determined from the constructs in the source model. A subset of the constructs in the source model is selected and one dimension is defined for each construct in the subset. However, some constructs may be skipped, and thus, do not introduce dimensions if the transformation alternatives for them are considered as unimportant for the target model.

Furthermore, each dimension relates with a number of coordinates. The coordinates form a *coordinate set*. The coordinate set for a particular dimension is determined on the basis of the construct from the source meta-model. The set of possible target constructs from the target meta-model is determined for the source meta-model construct. This set is used to create the coordinate set for the dimension.

According to that paper, there are 4 steps in the process of constructing and utilizing a transformation space. They are:

- Step 1: Constructing Transformation Space

A transformation space is constructed on a set of dimensions and a coordinate set for each dimension.

- Step 2: Reducing Transformation Space

  Operations for selection and exclusion are defined to reduce the space that most likely to be large.
- Step 3: Reducing Transformation Space on the basis of the quality properties of the target model

  The quality property of the result is expressed as a quality model. The quality property of the constructs from the source model is chosen, and the information is combined with the transformation space. Examples of quality properties are extensibility and sub-classing.
- Step 4: Refinement

  Once the space has sufficiently been reduced, the alternatives can be generated. However, the alternatives do not represent the complete transformation. Thus, some additional tuning is required. This step produces enough information for the specification of the transformation in a given language.

In the next section, the steps will be applied to a case study. However, step 4 introduces very large number of coordinate set because the coordinate set is also applied to attributes. Hence, this step will not be done in this project.

5.3 Transformation of UML Class Model into XML Schema: TSAS Case Study

As explained in the previous section, there are 4 steps to do to construct and utilize a transformation space in the mapping process. The implementation of those steps in doing the transformation from UML Model, especially UML Class Diagram, into XML Schema Model is explained in the next sections. To make the implementation clearer, a case study is used. The TSAS (Telecommunication Service Access and Subscription) class diagram [8] is chosen as the case study in this project. However, a little modification is made in order to show the different alternatives. The modification is on the type of class ServiceTemplate. In [8], it is not an abstract class, while in this project it is an abstract class. The TSAS class diagram is depicted by Figure 5-2 in the next page. Thus, TSAS Class Diagram is the source model, while the XML Schema is the target model. The implementation is mostly based on [12].

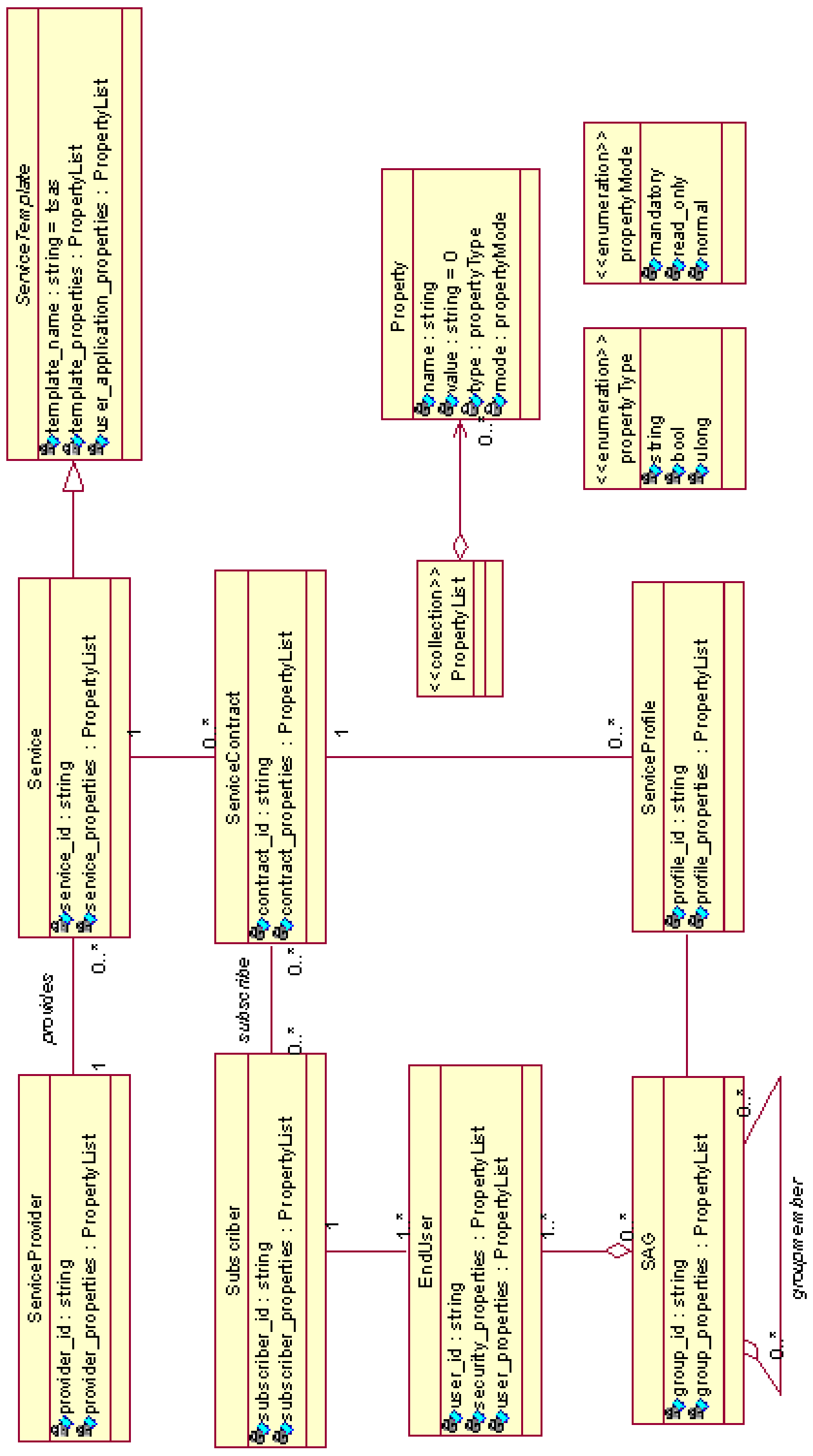


Figure 5-2: The modified TSAS Class Diagram

5.3.1 Constructing Transformation Space

First, the dimensions of a transformation space are determined from the constructs in the source model. The coordinate set for each dimension is determined on the basis of the construct from the source meta-model. The coordinate set for the dimension is created using the set of possible target constructs from the target meta-model which is determined for the source meta-model.

Two functions defined in [12] are used in this project. The first function is *dimensions(S)* which returns the set of dimensions for a given space S, while the second one is *coordinateSet(dimension, S)* which returns the coordinate set for a given dimension in a given space S.

Transformation space S_{TSAS} is constructed for TSAS Class Diagram, as the source model, depicted by Figure 2. For each class and relationship, a dimension is defined. Thus, the transformation space S_{TSAS} is as follows:

dimensions(S_{TSAS}) = (ServiceProvider, Service, ServiceTemplate, Subscriber, ServiceContract, EndUser, SAG, ServiceProfile, PropertyList, Property, propertyType, propertyMode, provides, Service-ServiceContract, subscribes, Subscriber-EndUser, ServiceContract-ServiceProfile, EndUser-SAG, SAG-ServiceProfile, groupmember, PropertyList-Property, Service-ServiceTemplate) (1)

ServiceProvider, Service, ServiceTemplate, Subscriber, ServiceContract, EndUser, SAG, ServiceProfile, PropertyList, Property, propertyType, propertyMode are dimensions defined for classes in TSAS class diagram. provides, Service-ServiceContract, subscribes, Subscriber-EndUser, ServiceContract-ServiceProfile, EndUser-SAG, SAG-ServiceProfile, groupmember, and PropertyList-Property are dimensions defined for associations. Dimension whose name contains '-' is the dimension defined for association that involves the classes whose names are separated with '-'. Service-ServiceTemplate is dimension defined for generalization in TSAS class diagram that connects class Service and class ServiceTemplate. The classes in the source model are instances of Class construct defined in the UML meta-model [13] and the associations and generalization are instances of Association and Generalization constructs.

Coordinate set for the dimensions are determined based on the target constructs defined in the target meta-model, i.e. XML Schema. As shown, by Table 1, XML Schema is the instance of XML Schema

Metamodel. Although there is no standard meta-model for XML Schema, a set of constructs can be identified based on the W3C Schema Specification. Another source to identify the constructs is OMG XMI (XML Metadata Interchange) Specification Version 2.0 which specifies the rules in describing objects, including UML Model, into XML. Using those two sources, the XML Schema model can be defined with a set of components, a set of relations, and a set of strategies among them as follows:

XMLSchemaMetaModel = (C, R) (2)

where C is a set of components, and R is a set of relations. C and R are mostly based on W3C Schema Specification, which also have been used in [12].

The set C corresponds to the XML Schema abstract data model components and has the following elements:

C = {CT, ST, E, ECT, EST, A, AG, MG} (3)

where :

CT	: Complex Type Definition
ST	: Simple Type Definition
E	: Element Declaration
ECT	: Element Declaration with Complex Type Definition
EST	: Element Declaration with Simple Type Definition
A	: Attribute Declaration
AG	: Attribute Group Definition
MG	: Model Group definition

The set R has the following elements:

R = {Der, Subst, Cont, Ref, Copy} (4)

where:

Der	: Derivation, corresponds to either extension or restriction mechanism
Subst	: Substitution, corresponds to the element substitution mechanism
Cont	: Containment, i.e. participation of a component in the content model of another
Ref	: Reference, i.e. the usage of elements or attributes of type ID and IDREF(S)
Copy	: Copy-down inheritance, the special case of containment

Although C and R are mostly based on W3C Schema Specification, one relation in R, i.e. copy-down inheritance, is defined based on the OMG XMI Specification Version 2.0 [15]. Copy corresponds to the copy-down inheritance strategy defined in XMI Specification Version 2. This strategy is defined especially to maintain the characteristic of inheritance. However, copy-down inheritance is actually a special case of containment, in which all of the features of the contained element are copied to the parent element, instead of just containing the element.

The coordinate set of each dimension in (1) should be identified. In this project, the class construct defined in the UML model is mapped to one of the components defined in (3). However, some possibilities are excluded as explained below.

- Class is not mapped into element (E) only, without the definition of type, because the declaration of element only will not have the capability to become a container of the representation of attributes and relationships of a UML class.
- Class is not mapped into attribute (A) because attribute only represents a simple value without the capability to contain other values, while class is the container of attributes and relationships which may become attributes, child elements or enumeration values.

Thus, element (E) and attribute (A) are excluded. The remaining components form a coordinate set that can be attached to the dimensions associated with the classes in the source model:

$$\text{coordinateSet (ServiceProvider, } S_{TSAS}) = \{ST, CT, ECT, EST, AG, MG\} \quad (5)$$
$$\text{coordinateSet (Service, } S_{TSAS}) = \{ST, CT, ECT, EST, AG, MG\} \quad (6)$$
$$\text{coordinateSet (ServiceTemplate, } S_{TSAS}) = \{ST, CT, ECT, EST, AG, MG\} \quad (7)$$
$$\text{coordinateSet (Subscriber, } S_{TSAS}) = \{ST, CT, ECT, EST, AG, MG\} \quad (8)$$
$$\text{coordinateSet (ServiceContract, } S_{TSAS}) = \{ST, CT, ECT, EST, AG, MG\} \quad (9)$$
$$\text{coordinateSet (EndUser, } S_{TSAS}) = \{ST, CT, ECT, EST, AG, MG\} \quad (10)$$
$$\text{coordinateSet (SAG, } S_{TSAS}) = \{ST, CT, ECT, EST, AG, MG\} \quad (11)$$
$$\text{coordinateSet (ServiceProfile, } S_{TSAS}) = \{ST, CT, ECT, EST, AG, MG\} \quad (12)$$
$$\text{coordinateSet (PropertyList, } S_{TSAS}) = \{ST, CT, ECT, EST, AG, MG\} \quad (13)$$
$$\text{coordinateSet (Property, } S_{TSAS}) = \{ST, CT, ECT, EST, AG, MG\} \quad (14)$$
$$\text{coordinateSet (propertyType, } S_{TSAS}) = \{ST, CT, ECT, EST, AG, MG\} \quad (15)$$
$$\text{coordinateSet (propertyMode, } S_{TSAS}) = \{ST, CT, ECT, EST, AG, MG\} \quad (16)$$

The association construct defined in the UML model can be mapped into the components defined in (3) or the relationship defined in (4). Some possibilities that are excluded are:

- Association is not mapped into simple type (ST) because simple type cannot have any attribute or child element, while UML Association has some properties that have to be represented, and this will not be sufficient by representing it as simple type.
- Association is not mapped into an element only (E) because the properties of element are not sufficient to represent the relationship between the involved classes.
- Association is not mapped into element with simple type definition (EST) with the same reason as of simple type.
- Association is not mapped into attribute (A) because UML Association has some properties that cannot be represented in a single attribute.
- Association is not mapped into substitution (Subst) because the class that is substituted by the other class will never be used.
- Association is not mapped into copy-down inheritance (Copy) because association does not have any inheritance characteristic, while copy-down inheritance are defined to maintain the inheritance characteristic.

Attribute (A), simple type (ST), element (E), element with simple type (EST)), and substitution (Subst) are excluded. Hence, the coordinate sets for dimensions of the associations in the TSAS class diagram are:

coordinateSet (provides, S_{TSAS}) = {CT, ECT, AG, MG, Der, Cont, Ref} (17)

coordinateSet (Service-ServiceContract, S_{TSAS}) = {CT, ECT, AG, MG, Der, Cont, Ref} (18)

coordinateSet (subscribes, S_{TSAS}) = {CT, ECT, AG, MG, Der, Cont, Ref} (19)

coordinateSet (Subscriber-EndUser, S_{TSAS}) = {CT, ECT, AG, MG, Der, Cont, Ref} (20)

coordinateSet (ServiceContract-ServiceProfile, S_{TSAS}) = {CT, ECT, AG, MG, Der, Cont} (21)

coordinateSet (EndUser-SAG, S_{TSAS}) = {CT, ECT, AG, MG, Der, Cont, Ref} (22)

coordinateSet (SAG-ServiceProfile, S_{TSAS}) = {CT, ECT, AG, MG, Der, Cont, Ref} (23)

coordinateSet (groupmember, S_{TSAS}) = {CT, ECT, AG, MG, Der, Cont, Ref} (24)

coordinateSet (PropertyList-Property, S_{TSAS}) = {CT, ECT, AG, MG, Der, Cont, Ref} (25)

The generalization construct defined in the UML model has a special characteristic, i.e. inheritance. Its mapping into XML Schema construct should maintain the inheritance characteristic. The generalization can be mapped into components defined in (3) or relationships defined in (4). However, still some possibilities are excluded as explained below.

- Generalization is not mapped into simple type (ST), element only (E), element with simple type (EST), or attribute (A) because a UML Generalization consists of child class and parent class. If it is represented as one of those XML constructs, the representation of child class and parent class will be a problem.
- Generalization is not mapped into substitution (Subst) because the class that is substituted by the inherited class will never be used.

Attribute, simple type, element, element with simple type, and substitution are excluded. Hence, the coordinate set for dimension of the generalization in the TSAS class diagram is:

coordinateSet (Service-ServiceTemplate, S_{TSAS}) = {CT, ECT, AG, MG, Der, Cont, Ref, Copy} (26)

5.3.2 Reducing Transformation Spaces using the Heuristics Rules

To calculate the number of alternatives for a space S with n dimensions, a formula is defined [12]:

numAlternatives (S) = numCoordinate(D_1) * numCoordinate(D_2) * … * numCoordinate(D_n) (27)

where:

numAlternatives (S) : function defined over a space S, returning the number of possible alternatives in the space

numCoordinate (D_i) : function defined over a dimension D_i, returning the number of elements in the coordinate set of dimension D_i

If the formula (28) is applied to the space S_{TSAS}, there will be 6 x 6 x 6 x 6 x 6 x 6 x 6 x 6 x 6 x 6 x 6 x 6 x 7 x 7 x 7 x 7 x 7 x 7 x 7 x 7 x 7 x 8 alternatives, or 70,272,815,129,1887,616 alternatives, that can be generated in the space. Certainly, it is unfeasible to generate the whole space of alternatives. The space can be reduced by selecting or excluding alternatives from the transformation space.

An operation for exclusion is defined in [12] and will be used in this project:

Exclude from S where <condition> (28)

Operation Exclude excludes from the space S the alternatives that satisfy the condition. This operation can be applied to the space S_{TSAS}.

Some alternatives can be excluded based on the requirements of the target model. The alternatives that are excluded are the mapping of class into attribute group (AG) or model group (MG), mapping association into derivation (Der), and mapping generalization into complex type (CT), element with complex type definition (ECT), attribute group (AG), model group (MG), and reference (Ref). Below is the implementation of the operation.

$S_{ReducedTSAS}$ = Exclude from S_{TSAS} where < (ServiceProvider.AG) and (Service.AG) and (ServiceTemplate.AG) and (Subscriber.AG) and (ServiceContract.AG) and (EndUser.AG) and (SAG.AG) and (ServiceProfile.AG) and (PropertyList.AG) and (Property.AG) and (propertyType.AG) and (propertyMode.AG) and (provides.AG or provides.Der) and (Service-ServiceContract.AG or Service-ServiceContract.Der) and (subscribes.AG or subscribes.Der) and (Subscriber-EndUser.AG or Subscriber-EndUser.Der) and (ServiceContract-ServiceProfile.AG or ServiceContract-ServiceProfile.Der) and (EndUser-SAG.AG or EndUser-SAG.Der) and (SAG-ServiceProfile.AG or SAG-ServiceProfile.Der) and (groupmember.AG or groupmember.Der) and (PropertyList-Property.AG or PropertyList-Property.Der) and (Service-ServiceTemplate.CT or Service-ServiceTemplate.ECT or Service-ServiceTemplate.AG or Service-ServiceTemplate.MG or Service-ServiceTemplate.Ref)> (29)

The reasons of the exclusion are as follow:

- Class is not mapped into AG (attribute group) because all of the attributes and especially the relationships in the class cannot simply be represented as attributes.
- Association is not mapped into derivation (Der) because derivation in XML Schema implicitly contains inheritance meaning, in which the derived type is inherited by other type. This can be seen from the reusability characteristic of derivation. On the other hand, association does not have inheritance characteristic.
- Association is not mapped into attribute group (AG) because attribute group only consists of attributes. The properties of associations, e.g. the classes involved, the multiplicity, the association end, cannot be represented by attributes only.
- Generalization is not mapped into complex type (CT), element with complex type (ECT), attribute group (AG), or model group (MG) because the representation of UML Generalization into one of those XML Schema constructs cannot preserve the inheritance characteristic of the

generalization. There is no inheritance characteristic in complex type, attribute group, or model group of XML Schema.

- Generalization is not mapped into reference (Ref) because in most cases, the inheritance characteristic cannot be represented clearly in XML Schema reference. Moreover, in most cases, reference is considered more suitable for association.

The coordinate sets are now becoming as follow:

coordinateSet (ServiceProvider, $S_{ReducedTSAS}$) = {ST, CT, ECT, EST, MG} (30)

coordinateSet (Service, $S_{ReducedTSAS}$) = {ST, CT, ECT, EST, MG} (31)

coordinateSet (ServiceTemplate, $S_{ReducedTSAS}$) = {ST, CT, ECT, EST, MG} (32)

coordinateSet (Subscriber, $S_{ReducedTSAS}$) = {ST, CT, ECT, EST, MG} (33)

coordinateSet (ServiceContract, $S_{ReducedTSAS}$) = {ST, CT, ECT, EST, MG} (34)

coordinateSet (EndUser, $S_{ReducedTSAS}$) = {ST, CT, ECT, EST, MG} (35)

coordinateSet (SAG, $S_{ReducedTSAS}$) = {ST, CT, ECT, EST, MG} (36)

coordinateSet (ServiceProfile, $S_{ReducedTSAS}$) = {ST, CT, ECT, EST, MG} (37)

coordinateSet (PropertyList, $S_{ReducedTSAS}$) = {ST, CT, ECT, EST, MG} (38)

coordinateSet (Property, $S_{ReducedTSAS}$) = {ST, CT, ECT, EST, MG} (39)

coordinateSet (propertyType, $S_{ReducedTSAS}$) = {ST, CT, ECT, EST, MG} (40)

coordinateSet (propertyMode, $S_{ReducedTSAS}$) = {ST, CT, ECT, EST, MG} (41)

coordinateSet (provides, $S_{ReducedTSAS}$) = {CT, ECT, MG, Cont, Ref} (42)

coordinateSet (Service-ServiceContract, $S_{ReducedTSAS}$) = {CT, ECT, MG, Cont, Ref} (43)

coordinateSet (subscribes, $S_{ReducedTSAS}$) = {CT, ECT, MG, Cont, Ref} (44)

coordinateSet (Subscriber-EndUser, $S_{ReducedTSAS}$) = {CT, ECT, MG, Cont, Ref} (45)

coordinateSet (ServiceContract-ServiceProfile, $S_{ReducedTSAS}$) = {CT, ECT, MG, Cont, Ref}
(46)

coordinateSet (EndUser-SAG, $S_{ReducedTSAS}$) = {CT, ECT, MG, Cont, Ref} (47)

coordinateSet (SAG-ServiceProfile, $S_{ReducedTSAS}$) = {CT, ECT, MG, Cont, Ref} (48)

coordinateSet (groupmember, $S_{ReducedTSAS}$) = {CT, ECT, MG, Cont, Ref} (49)

coordinateSet (PropertyList-Property, $S_{ReducedTSAS}$) = {CT, ECT, MG, Cont, Ref} (50)

coordinateSet (Service-ServiceTemplate, $S_{ReducedTSAS}$) = {Cont, Der, Copy} (51)

By the exclusion, the number of alternatives is now 5 x 5 x 5 x 5 x 5 x 5 x 5 x 5 x 5 x 5 x 5 x 5 x 5 x 5 x 5 x 5 x 5 x 5 x 5 x 5 x 5 x 3 alternatives, or 1,430,511,474,609,375 alternatives, or reduced by the factor 491,24 from the previous number of alternatives. This is still a big number, and thus must be reduced again. The reduction is now applied to each particular dimension or some dimensions which represent TSAS class diagram components that have the same characteristics.

The further reduction uses another operation, i.e. selection, which also has been defined in [12]. The operation is as follows:

```
    Select from S where <condition>                                    (52)
```

This operation selects from a given space S only the alternatives that satisfy the given condition.

For the selections applied to the dimensions defined for classes, the rules defined in Table 4.2 on page 4-4 are used. From the table, the decisions below are made.

Dimensions ServiceProvider, Service, Subscriber, ServiceContract, EndUser, SAG, ServiceProfile, PropertyList, and Property are dimensions defined for the classes in TSAS class diagram with the same names. Each of the classes has relationship(s) with other class(es). Furthermore, they are not abstract classes, and they do not have <<enumeration>> stereotype. According to rule R.1.1, each of them must be represented as element declaration with complex type definition (ECT). Thus, ECT is selected from the space $S_{ReducedTSAS}$ for the dimensions.

Class *ServiceTemplate* is an abstract class. According to rule R.1.2, the class can be represented as a complex type (CT), as an element with complex type definition (ECT) whose abstract attribute values are 'true', or as a model group (MG). Thus, CT, ECT, and MG are selected from the space $S_{ReducedTSAS}$ for the dimension ServiceTemplate.

Class *propertyType* and class *propertyMode* has <<enumeration>> stereotype. They do not have any relationship with other class. The attributes of the classes are primitive data type. Using rule R.1.3, simple type is chosen to represent the classes. ST is selected from the space $S_{ReducedTSAS}$ for the dimensions defined for the two classes.

The selections are performed by the operation below:

$S_{ReducedTSAS\text{-}I}$ = Select from $S_{ReducedTSAS}$ where < ServiceProvider.ECT and Service.ECT and Subscriber.ECT and ServiceContract.ECT and EndUser.ECT and SAG.ECT and ServiceProfile.ECT and PropertyList.ECT and Property.ECT and (ServiceTemplate.CT or ServiceTemplate.ECT or ServiceTemplate.MG) and propertyType.ST and propertyMode.ST > (53)

For the selections applied to the dimensions defined for associations, the rules defined in Table 4.7 on page 4-18 are used. From the rules, it can be seen that the mapping are determined by the aggregation value, the presence of the association name, the navigation, and the multiplicity. Table 5.2 on the next page lists the possible combinations of those factors and the possible alternative(s) for each combination.

In the table, the navigability is only divided into two types, i.e. unidirectional and bidirectional or no navigation. Bidirectional and no navigation are treated in the same way because there is an assumption that they are equivalent. Rumbaugh said that if navigability has not been decided, then it is bidirectional in the general case [17]. UML even represents both unstated and bidirectional navigability equivalently [13]. This is understandable since association is bidirectional by default.

Furthermore, for association with 'none' aggregation value, containment is not chosen for multiplicities of both association ends are 1, because this will lead to the repeating of containment, i.e. element A contains element B, and element B contains element A. Containment is also not chosen for many to many associations because it will lead to the repetition of the child element in many containments.

Table 5-2: Possible combinations of the factors that influence the choice of mapping association

			Navigation			
			Unidirectional		**Bidirectional or No Navigation**	
			Association Name Exists	**No Association Name**	**Association Name Exists**	**No Association Name**
Aggregation Value	**Composite**	**Multiplicities of both association ends are 1**	containment	containment	containment	containment
		One multiplicity of the association ends is 1	containment	containment	containment	containment
		Other kinds of multiplicity	containment	containment	containment	containment
	Aggregate	**Multiplicities of both association ends are 1**	containment	containment	containment	containment
		One multiplicity of the association ends is 1	containment	containment	containment	containment
		Other kinds of multiplicity	containment	containment	containment	containment
	None	**Multiplicities of both association ends are 1**	containment	- containment - reference	element declaration with complex type definition	- reference - element declaration with complex type definition
		One multiplicity of the association ends is 1	containment	containment	- element declaration with complex type definition - containment	- element declaration with complex type definition - containment
		Other kinds of multiplicity	containment	containment	element declaration with complex type definition	element declaration with complex type definition

Dimension EndUser-SAG, dimension groupmember, and dimension PropertyList-Property are dimensions that are defined for aggregations. This implies a 'has' relationship between the involved class. According to the above table, all types of aggregations are mapped into containment (Cont) of XML Schema. Thus, the Cont (containment) component for the representations of the three aggregations is selected from the space $S_{ReducedTSAS-I}$ (the space as the result of the first selections).

Dimension provides, dimension Service-ServiceContract, dimension Subscriber-EndUser, and dimension ServiceContract-ServiceProfile are dimensions defined for associations in TSAS class diagram. All of the associations are bidirectional and one multiplicity of their association ends is 1. Moreover, their aggregation values are 'none'. According to the above table, they can be mapped into either element declaration with complex type complex type definition or containment. Thus, the ECT and Cont components for the representations of those associations are selected from the space $S_{ReducedTSAS-I}$.

Dimension subscribe and dimension SAG-ServiceProfile are dimensions defined for associations in TSAS class diagram. The associations are bidirectional and their both multiplicities are not 1. Moreover, their aggregation values are 'none'. According to the above table, they are mapped into element declaration with complex type complex type definition. Thus, the ECT component for the representations of those associations is selected from the space $S_{ReducedTSAS-I}$.

Those selections are performed by the selection operation below:

$S_{ReducedTSAS-II}$ = Select from $S_{ReducedTSAS-I}$ where < EndUser-SAG.Cont and groupmember.Cont and PropertyList-Property.Cont and (provides.ECT or provides.Cont) and (Service-ServiceContract.ECT or Service-ServiceContract.Cont) and subscribe.ECT and (Subscriber-EndUser.ECT or Subscriber-EndUser.Cont) and (ServiceContract-ServiceProfile.ECT or ServiceContract-ServiceProfile.Cont) and SAG-ServiceProfile.ECT > (54)

For the selections applied to the dimensions defined for generalizations, the rules defined in Table 4.9 on page 4-24 are used. Using the rules, the generalization between the child class Service and the superclass ServiceTemplate can be represented as derivation or containment. Thus, Der and Cont are selected from the space $S_{ReducedTSAS-II}$ (the space as the result of the second selections) for the dimension defined for the generalization. The selection is performed using the selection operation below:

$S_{ReducedTSAS\text{-}III}$ = Select from $S_{ReducedTSAS\text{-}II}$ where < Service-ServiceTemplate.Der or Service-ServiceTemplate.Cont > (55)

As the consequences for the last selection, the superclass ServiceTemplate should be represented as complex type or element with complex type definition. Model group does not appropriately represent the class ServiceTemplate since for derivation representation of generalization the superclass must be represented as complex type. Thus, MG must be excluded from the space. The exclusion operation is performed below:

$S_{ReducedTSAS\text{-}IV}$ = Exclude from $S_{ReducedTSAS\text{-}IIi}$ where < ServiceTemplate.MG > (56)

The coordinate sets are now becoming as follow:

coordinateSet (ServiceProvider, $S_{ReducedTSAS\text{-}III}$) = {ECT} (57)

coordinateSet (Service, $S_{ReducedTSAS\text{-}III}$) = {ECT} (58)

coordinateSet (ServiceTemplate, $S_{ReducedTSAS\text{-}III}$) = {CT, ECT} (59)

coordinateSet (Subscriber, $S_{ReducedTSAS\text{-}III}$) = {ECT} (60)

coordinateSet (ServiceContract, $S_{ReducedTSAS\text{-}III}$) = {ECT} (61)

coordinateSet (EndUser, $S_{ReducedTSAS\text{-}III}$) = {ECT} (62)

coordinateSet (SAG, $S_{ReducedTSAS\text{-}III}$) = {ECT} (63)

coordinateSet (ServiceProfile, $S_{ReducedTSAS\text{-}III}$) = {ECT} (64)

coordinateSet (PropertyList, $S_{ReducedTSAS\text{-}III}$) = {ECT} (65)

coordinateSet (Property, $S_{ReducedTSAS\text{-}III}$) = {ECT} (66)

coordinateSet (propertyType, $S_{ReducedTSAS\text{-}III}$) = {ST} (67)

coordinateSet (propertyMode, $S_{ReducedTSAS\text{-}III}$) = {ST} (68)

coordinateSet (provides, $S_{ReducedTSAS\text{-}III}$) = {ECT, Cont} (69)

coordinateSet (Service-ServiceContract, $S_{ReducedTSAS\text{-}III}$) = {ECT, Cont} (70)

coordinateSet (subscribe, $S_{ReducedTSAS\text{-}III}$) = {ECT} (71)

coordinateSet (Subscriber-EndUser, $S_{ReducedTSAS\text{-}III}$) = {ECT, Cont} (72)

coordinateSet (ServiceContract-ServiceProfile, $S_{ReducedTSAS\text{-}III}$) = {ECT, Cont} (73)

coordinateSet (EndUser-SAG, $S_{ReducedTSAS\text{-}III}$) = {Cont} (74)

coordinateSet (SAG-ServiceProfile, $S_{ReducedTSAS\text{-}III}$) = {Cont} (75) coordinateSet (groupmember, $S_{ReducedTSAS-III}$) = {Cont} (76)

coordinateSet (PropertyList-Property, $S_{ReducedTSAS-III}$) = {Cont} (77)

coordinateSet (Service-ServiceTemplate, $S_{ReducedTSAS-III}$) = {Der, Cont} (78)

And now, the number of alternatives become 1 x 1 x 2 x 1 x 1 x 1 x 1 x 1 x 1 x 1 x 1 x 1 x 2 x 2 x 1 x 2 x 2 x 1 x 1 x 1 x 1 x 2 alternatives or 64 alternatives. This is the reduction by factor 22,351,741.790,771.48 from the previous number of alternatives. The alternatives are in the choice of the mapping of abstract class, associations, and generalization.

5.3.3 Reducing Transformation Space on the Basis of the Quality Properties of the Target Model

The third step in the process of constructing and utilizing a transformation space is reducing transformation space on the basis of the quality properties.

5.3.3.1 Scenario on the Requirement of the Quality Properties of the Target Model

One possible scenario that a quality property is needed by the target model is the requirement of the source model to be changed (see Figure 5-3). For instance, an attribute needs to be added to a class in the existing UML Class Diagram (as the source model). As the consequence, the existing XML Schema, which is the mapping of the UML Class Diagram, needs to be changed, i.e. the component as the representation of the class needs to be added with a new component as the representation of the new attribute. This little addition/modification must not much influence the existing Schema, i.e. only the component as the representation of the class is changed. In this case, extensibility is the quality property that is required by the target model. Furthermore, the modified XML Schema is the mapping of the modified UML Class Diagram. Another example, a class in a UML Class Diagram needs to be reused, for example in subclassing. The XML component as the representation of the class must be reusable as well. In this case, reusability is the quality property that is required by the XML Schema as the target model. Figure 5-3 shows how this scenario is applied.

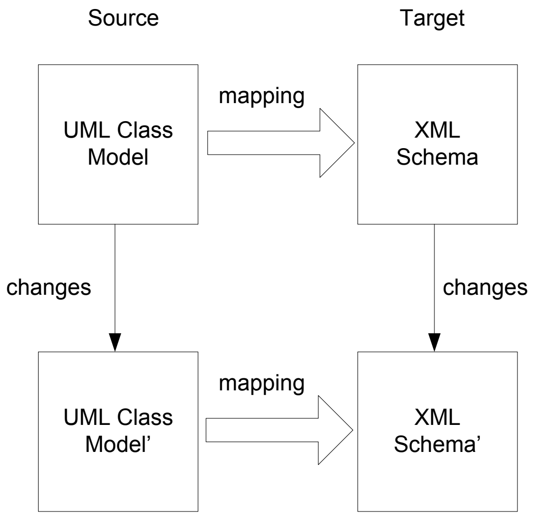


Figure 5-3: The changes on the source model causes the changes on the target model

5.3.3.2 Some Quality Properties

The ISO 9126 is concerned primarily with the definition of quality characteristics to be used in the evaluation of software products [19]. ISO 9126 sets out six quality characteristics, which are intended to be exhaustive.. The six quality characteristics are Functionality, Reliability, Usability, Efficiency, Maintainability, and Portability. Each of these characteristics is further refined into sub-characteristics. For instance, the quality characteristic 'maintainability' is divided into five quality characteristics: analysability, changeability, stability, testability, and compliance [18].

Other than the six quality characteristics mentioned in ISO 9126, there are many other quality attributes of software products. For example, in Software Engineering book, Huy mentions six quality attributes that are addressed by some standardization bodies, like ISO 9000, ANSI, IEEE, etc [10]. The six quality attributes are correctness, dependability, user friendliness, maintainability, efficiency, and portability. Similar with ISO 9126, some of the attributes can be divided into sub-attributes. For instance, maintainability is divided into three attributes: readability, extensibility, and testability.

In this project, only two quality properties will be explored for the target model, i.e. XML Schema. The first is *reusability* property. This is one of the sub-factors defined in the IEEE Standard 1061, which is standard for software quality metrics. According to Software Metrics Glossary [9], in term of software metrics, *reusability is the degree to which the application and the code in the application have been specifically designed, developed, and supported to be usable in other applications*. This implies that the application is used as it is, without any modification or addition. In the scenario described by Figure 5-3 above, UML Model' is the model that reuses the UML Model.

The second property is *extensibility*. In the IEEE Standard 1061, the term expandability is used instead of extensibility. In ISO 9126, the term changeability is used instead of extensibility, and it is one of the sub-characteristics of maintainability. In the term, changeability is defined *as the capability of the software product to enable a specified modification to be implemented* [18]. According to Huy, extensibility allows required modifications at the appropriate locations to be made without undesirable side effects [10]. Thus, the existing model will not be influenced by modifications at the appropriate location. The definition means that extensibility always implies reusability, because the modifications always need the reusing of the existing model.

UML recommends subclassing as the default mechanism for extensibility [7]. Thus, subclassing is an example of extensibility in UML. However, since extensibility always implies reusability, subclassing can be used also as an example of reusability. This is quite understandable since in subclassing all the attributes, operations, and characteristics of the superclass are reused (inherited) by the subclass. On the other hand, extensibility is shown by the fact that the additions of attributes or operations must be to the subclass, not to the superclass. Another example of extensibility is adding attribute to a class, as already mentioned above.

In XML Schema, an example of reusability mechanism is *inclusion of another schema*. In this case, the schema is reused as it is in the other schema. Another example is *redefinition* mechanism. Extensibility mechanism in XML Schema is shown by *derivation*. In this case, the derived type is reused by the new type, but there are some additions to the new type.

As the summaryTable 5-3 shows the examples of reusability and extensibility mechanisms in UML and XML Schema.

Table 5-3: Some quality properties of XML Schema

Quality Property	Example in UML	Example in XML Schema
Reusability	Subclassing	Inclusion of Schema
Extensibility	- Subclassing - Adding attribute	- Derivation - Redefinition

5.3.3.3 Applying the Requirement of the Quality Properties to the Case Study

An example of quality requirement that will be used is subclassing that applied to the given UML Class Diagram. This is an example of reusability requirement. In UML, subclassing can be done straight forward to the existing UML class. On the other hand, in XML Schema, this must be anticipated in the very beginning so that modifications to the existing XML Schema can be conducted without undesirable side effects and by using the allowed Schema operations. This is important to keep the extensibility of the schema.

Since quality properties will influence the selection of the alternatives of XML schema's, the description of how reusability and extensibility in each components of XML Schema will be discussed before applying the requirement of the quality properties to the case study. This will give an overview on how those two quality properties can be achieved in an XML Schema. Table 3.7 shows some quality properties applied to XML Schema components.

Table 5-4: Some quality properties of XML Schema

XML Schema Components	Quality Properties	
	Reusability	Extensibility
Element	The element can be reused, for example in containment or reference.	The element is of complex type so that new element or attribute can be added.
Complex Type	The type definition can be reused.	- New attributes/elements can be added. - It is possible to derive new type from it.

Simple Type	The type definition can be reused	- For simple type with enumeration facet, new enumeration value can be added. - It is possible to derive new type from it.
Attribute	The attribute can be reused, for example it can be referenced in another declaration. This implies that it must be declared globally.	Not relevant.
Attribute Group	The attribute group can be reused, for instance it can be referenced in another declaration.	New attributes can be added.
Model Group	The model group can be reused, for instance it can be referenced in another declaration.	New elements can be added.

Furthermore, XML Schema Best Practices [4] gives some rules in achieving different quality properties of XML Schema. However, the rules are originally designed for XML Schema, while quality properties in the scenario described in sub section 5.3.3.1 is needed for the source model (UML Class Model) and must be preserved in the target model (XML Schema). Thus, the rules from XML Schema Best practices are modified to fit the requirement. The rules produced are summarized by Table 5-5.

Table 5-5: Rules for some quality properties derived from XML Schema Best Practices

Rules Number	Source Model	Target Model	Rationale	Reference
RQ.1	Subclassing is expected for a class (this is an example of the requirement of the reusability	a. Create a (named) type definition and an element declaration for the mapping of the corresponding	- The properties and characteristics of the parent class are maintained in the type definition, while the instance of the parent class is represented by	- Article: Global versus Local [4]

	property)	class	the declaration of an element of that type. The subclass can be represented by creating a new element of the type or deriving a new type from the type represents the parent class. Reusability is preserved by the type definition of the parent class.	
		b. Define a type for the mapping of the class	- An element that represents the subclass can always be created from the type. The subclass can also be represented as the derivation of the type of the parent class.	- Article: Should it be an element or a type? [4]
RQ.2	An attribute is expected to be added to a class (this is an example of the requirement of the extensibility property)	a. Define a complex type for the mapping of the class, and insert <any> element into it.	- Inserting <any> element into a content model enables instance document to contain additional elements as the representation of the new attribute added to the class.	Article: Creating Extensible Content Models [4]
		b. Define a complex type for the mapping of the class, and <anyAttribute>	-Inserting <anyAttribute> into a content model enables instance document to contain additional attribute as the	- XML Schema Part 0: Primer [21]

		into it.	representation of the new attribute added to the class.	

Now, one example of reusability property, i.e. subclassing, is applied to the case study. This means that subclassing is expected to be possible for one or more classes in the TSAS class diagram. In this example, class ServiceProvider is expected to have subclass. Figure 5-4 shows the modified TSAS class diagram, in which subclassing for class ServiceProvider is applied. In this modification, a subclass of class ServiceProvider is added. Its name is SubServiceProvider.

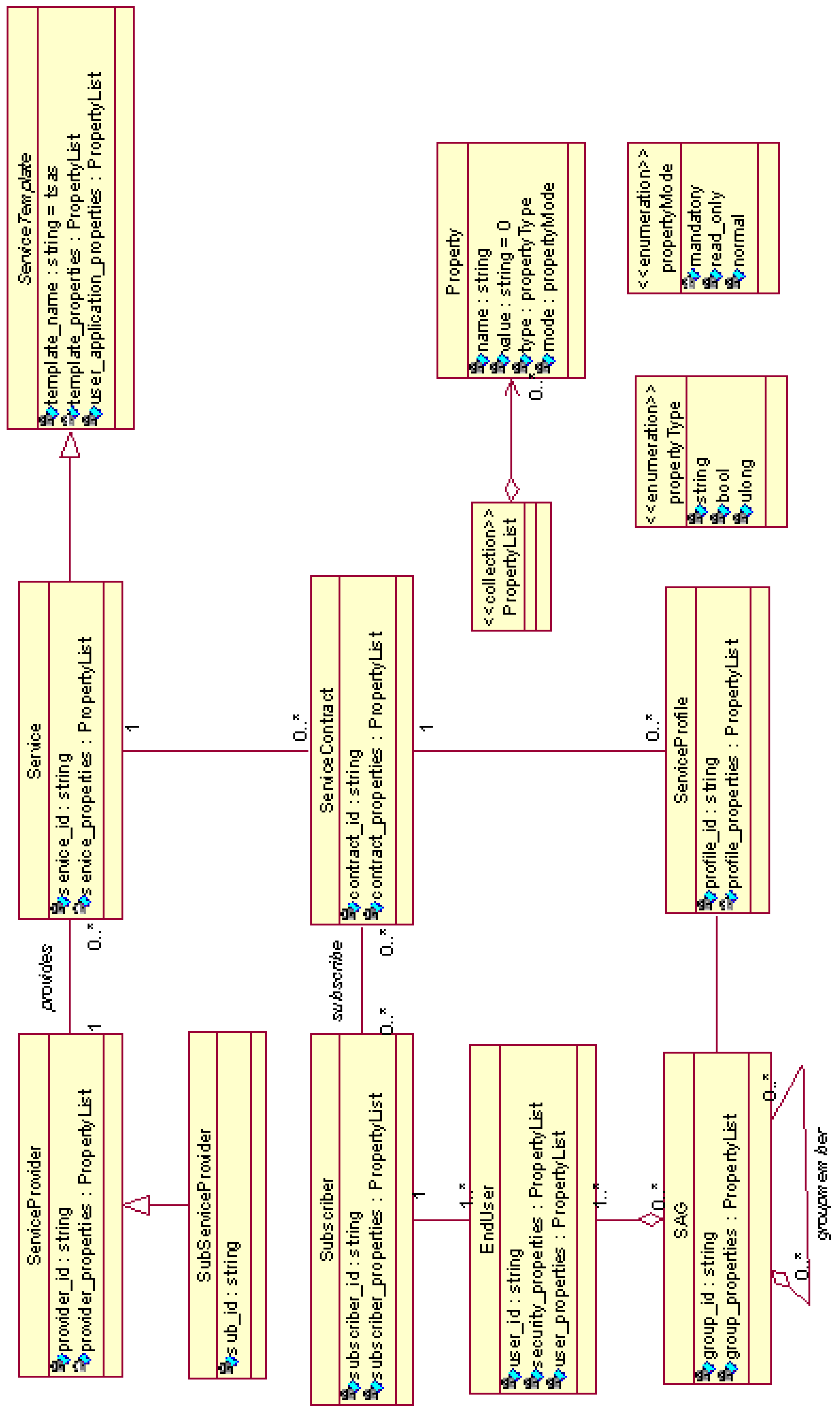


**Figure 5-4: The modified TSAS Class Diagram
(class SubServiceProvider is added)**

From rule RQ.1 in Table 5-5, if subclassing is expected for a class, then the class is mapped into a (named) type definition and an element declaration; or the class is mapped into a type definition. Those two ways of mapping enable the subclass to be mapped into an element which has type of the type of parent class. It also can be mapped into a type that is derived from the type of the parent class.

Apparently, this has been fulfilled by the coordinate set of class ServiceProvider. The previous coordinate set for class ServiceProvider is:

$$\text{coordinateSet (ServiceProvider, } S_{\text{ReducedTSAS-III}}) = \{\text{ECT}\} \tag{57}$$

Thus, the choice of mapping of class ServiceProvider does not need to be changed.

However, since class ServiceProvider has an association relationship with class Service, i.e. association *provides*, the subclassing will influence the choice mapping of the association. This happens because the subclass SubServiceProvider should also be possible to have the association relationship with class Service. From the previous space, the coordinate set for association *provides* is:

$$\text{coordinateSet (provides, } S_{\text{ReducedTSAS-III}}) = \{\text{ECT, Cont}\} \tag{69}$$

There are two alternatives of mapping, i.e. element with complex type (ECT) and containment (Cont). If ECT is chosen, which means that the association *provides* is mapped into element with complex type, the subclass SubServiceProvider will not be possible to have association relationship with class Service. On the other hand, if containment is chosen, the association between SubServiceProvider and Service is perfectly possible. Thus, if subclassing is required for class ServiceProvider, the association *provides* has to be mapped into containment.

One rule that can be derived from this is included in Table 5-6.

Table 5-6: Rule of quality properties concerning association

Rules Number	Source Model	Target Model	Rationale
RQ.3	Subclassing is expected for a class (class A) that has association relationship with another class (class B).	The association is mapped into containment (class A contains class B).	If the association is mapped into element, it needs another element declaration to represent the association between class B and the subclass.

In this third step (reducing transformation space on the basis of the quality properties), consider that the name of the transformation space is $S_{Reusable}$. The coordinate sets of transformation space $S_{Reusable}$ are becoming as follows:

coordinateSet (ServiceProvider, $S_{Reusable}$) = {ECT} (79)

coordinateSet (Service, $S_{Reusable}$) = {ECT} (80)

coordinateSet (ServiceTemplate, $S_{Reusable}$) = {CT, ECT} (81)

coordinateSet (Subscriber, $S_{Reusable}$) = {ECT} (82)

coordinateSet (ServiceContract, $S_{Reusable}$) = {ECT} (83)

coordinateSet (EndUser, $S_{Reusable}$) = {ECT} (84)

coordinateSet (SAG, $S_{Reusable}$) = {ECT} (85)

coordinateSet (ServiceProfile, $S_{Reusable}$) = {ECT} (86)

coordinateSet (PropertyList, $S_{Reusable}$) = {ECT} (87)

coordinateSet (Property, $S_{Reusable}$) = {ECT} (88)

coordinateSet (propertyType, $S_{Reusable}$) = {ST} (89)

coordinateSet (propertyMode, $S_{Reusable}$) = {ST} (90)

coordinateSet (provides, $S_{Reusable}$) = {Cont} (91)

coordinateSet (Service-ServiceContract, $S_{Reusable}$) = {ECT, Cont} (92)

coordinateSet (subscribe, $S_{Reusable}$) = {ECT} (93)

coordinateSet (Subscriber-EndUser, $S_{Reusable}$) = {ECT, Cont} (94)

coordinateSet (ServiceContract-ServiceProfile, $S_{Reusable}$) = {ECT, Cont} (95)

coordinateSet (EndUser-SAG, $S_{Reusable}$) = {Cont} (96)

coordinateSet (SAG-ServiceProfile, $S_{Reusable}$) = {Cont} (97)

coordinateSet (groupmember, $S_{Reusable}$) = {Cont} (98)

coordinateSet (PropertyList-Property, $S_{Reusable}$) = {Cont} (99)

coordinateSet (Service-ServiceTemplate, $S_{Reusable}$) = {Der, Cont} (100)

And now, the number of alternatives become 1 x 1 x 2 x 1 x 1 x 1 x 1 x 1 x 1 x 1 x 1 x 1 x 1 x 2 x 1 x 2 x 2 x 1 x 1 x 1 x 1 x 2 alternatives or 32 alternatives. This is the reduction by factor 2 from the previous number of alternatives. The reduction is the consequence of the implementation of the reusability property of class ServiceProvider.

5.4 Summary

This chapter contains the transformation of UML Class Model into XML Schema using Design Algebra. This transformation is applied to the case study, i.e. TSAS Class Diagram. The application of the rules defined in Chapter 4 is shown here. It is shown that using the design space and the rules, the alternatives can be reduced significantly.

Chapter 6
Conclusion

This chapter contains a summary of the project in section 7.1 and suggestion for future research in section 7.2.

6.1 Summary

This project shows a collection and categorization of the available heuristics for selection of mapping rules from an UML Class Models into XML Schema. The heuristics rules are then used to select the valid alternatives of the mapping.

To give the description of how to map UML Class Models into XML Schema's, mapping of UML Generalization into XML Schema is shown first. Two steps of alternatives generation are discussed. The second step of alternative generation for each possible mapping of UML Generalization is explained in detail. Also, the transformation process of mapping UML Generalization into XML Schema is explained with an example.

After that, the development of the rules is discussed. Heuristic rules are constructed based on the properties of each construct of UML Class Model. Each property is mapped into XML Schema construct, if it is available; otherwise the information implied by the property is loss. Rules for each UML Class Diagram construct are built. The heuristics rules can be categorized into two types, i.e. the rules that are based on the quality properties of the source model and the rules that are based on the fact that certain combinations of constructs in the target model are not allowed.

Suitable heuristic rules are used to select the valid alternatives among all the alternative mappings of a given UML Class Model into XML Schema. Design Algebra is used to model a set of alternative transformations from an UML Class Model as the source model into XML Schema as the target model. Four steps in the process of constructing and utilizing a transformation space are performed. The implementation of Design Space and reduction using the rules can reduce the alternatives.

This project has identified and collected some heuristic rules to select alternative mappings from UML Class Models into XML Schema as intended in the problem statement. It also shows the feasibility of the approach in a case study. However, the fourth step of transformation space, i.e. refinement, has not been conducted yet. This is due to the consideration that the number of alternatives will be much larger. Besides, the step is done by only repeating the first three steps. Thus, it is a kind of looping, and not important step.

The strong points of the approach in this project are:

- Using Design Algebra technique, the set of alternative transformations from UML Class Models as the source model into XML Schema as the target model can be derived systematically, step by step, so that all alternatives can be identified.
- Since heuristics rule are constructed based on the similarity characteristics of the properties of each construct of UML Class Model and the constructs of the XML Schema, the important characteristics of the UML Class Model, for instance the multiplicity of an association, can still be preserved in the XML schema produced.

However, there are also several drawbacks:

- Many properties of UML Class constructs, for example the isLeaf and isRoot properties of class, cannot be mapped into XML Schema constructs. This makes the information implied by the properties is lost (see chapter 4).
- On the other hand, there are some XML Schema constructs that are useless for the mapping because there are no UML Class constructs that bring the similar information. Example for this case is derivation by restriction.

- At first, the number of alternatives produced by Design Algebra is very large, since the technique considers all alternatives. The space is conceptual and the large number of alternatives is not explicitly generated.

6.2 Suggestion for Future Research

A tool can be built to map a UML Class Diagrams into XML Schema's using heuristic rules identified in this project. This will enable the system designers to map a UML Class Models into XML Schema's. The tool will give alternatives of the possible mapping to the user based on the rules. Thus, the user needs to give inputs to the tool. The output is the XML schema as the result of the mapping.

The reverse mapping, that is the mapping from XML Schema into UML Model, can be performed by considering the similarities of the information implied by the constructs of each domain model, as shown by this project. Therefore, this project can be useful as reference for such research.

References

[1] Braun, D., et al. *Unified Modeling Language (UML) Tutorial.* Spring 2001. Kennesaw State University.
http://pigseye.kennesaw.edu/~dbraun/csis4650/A&D/UML_tutorial/

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

[3] Champion, M. *The Future of XML*. 2004. XML Journal
http://www.sys-con.com/xml/article.cfm?id=754

[4] Costello, R.L., et al. *XML Schema: Best Practices*. February 12, 2003. xFront.
http://www.xfront.com/BestPracticesHomepage.html

[5] Deitel, H.M., et al. *XML How to Program*. 2001. Prentice-Hall, Inc.

[6] Dictionary.com
http://dictionary.reference.com/search?q=heuristic

[7] D'Souza, D., Sane, A., Birchenough, A. *First Class Extensibility for UML – Packaging for Profiles, Stereotypes, Patterns.* Second International Conference on the Unified Modeling Language: UML'99, 1999.
http://www.catalysis.org/publications/papers/1999-uml-99-submission-2.pdf

[8] Gunawan L.A., *Mapping UML Models onto XML Schemas using UML Profiles*. June 2003. MSc Telematics Thesis, University of Twente.

[9] Hierarchy Master. *Software Metrics Glossary-R.*
http://www.hmaster.com/Software-Project-Management/glossary/R.html

[10] Huy, N.Y. Software Engineering. Institute of Information Technology, Vietnam.
http://www.netnam.vn/unescocourse/se/software.htm

[11] IBM. *Rational Product Plug-ins and Add-ins. 19 July 2003.*
http://www-106.ibm.com/developerworks/rational/library/1376.html

[12] Kurtev, I., Berg, K., and Aksit, M. *UML to XML-Schema Transformation: a Case Study in Managing Alternative Model Transformations in MDA*. 2003. University of Twente.

[13] OMG. *OMG Unified Modeling Language Specification version 1.5* (OMG Document 03-03-01). March 2003.

http://www.omg.org/technology/documents/formal/uml.htm

[14] OMG. *OMG XML Metadata Interchange (XMI) Specification version 1.2* (OMG Document 02-01-01). January 2002.
http://www.omg.org/technology/documents/formal/xmi.htm

[15] OMG. *OMG XML Metadata Interchange (XMI) Specification version 2.0* (OMG Document 03-05-02). May 2003.
http://www.omg.org/technology/documents/formal/xmi.htm

[16] OMG. *Telecommunications Service Access and Subscription Specification, Version 1.0.* December 2002.
http://www.omg.org/technology/documents/formal/tsas.htm

[17] Rumbaugh, J., Jacobson, I., Booch, G. *The Unified Modeling Language Reference Manual*. 1999. Boston: Addison-Wesley.

[18] Solingen, R. *Product Focused Software Process Improvement SPI in the embedded software domain*. 2000. Eindhoven University Press.

[19] *The ISO 9126 Framework.*
http://mudhole.spodnet.uk.com/~kyrian/project/project.html

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

[21] W3C. *XML Schema Part 0: Primer*. W3C Recommendation. 2 May 2001.
http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/

[22] W3C. *XML Schema Part1: Structures*. 2 May 2001.
http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/

[23] W3C. *XML Schema Part 2: Datatypes*. W3C Recommendation. 2 May 2001.
http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/

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

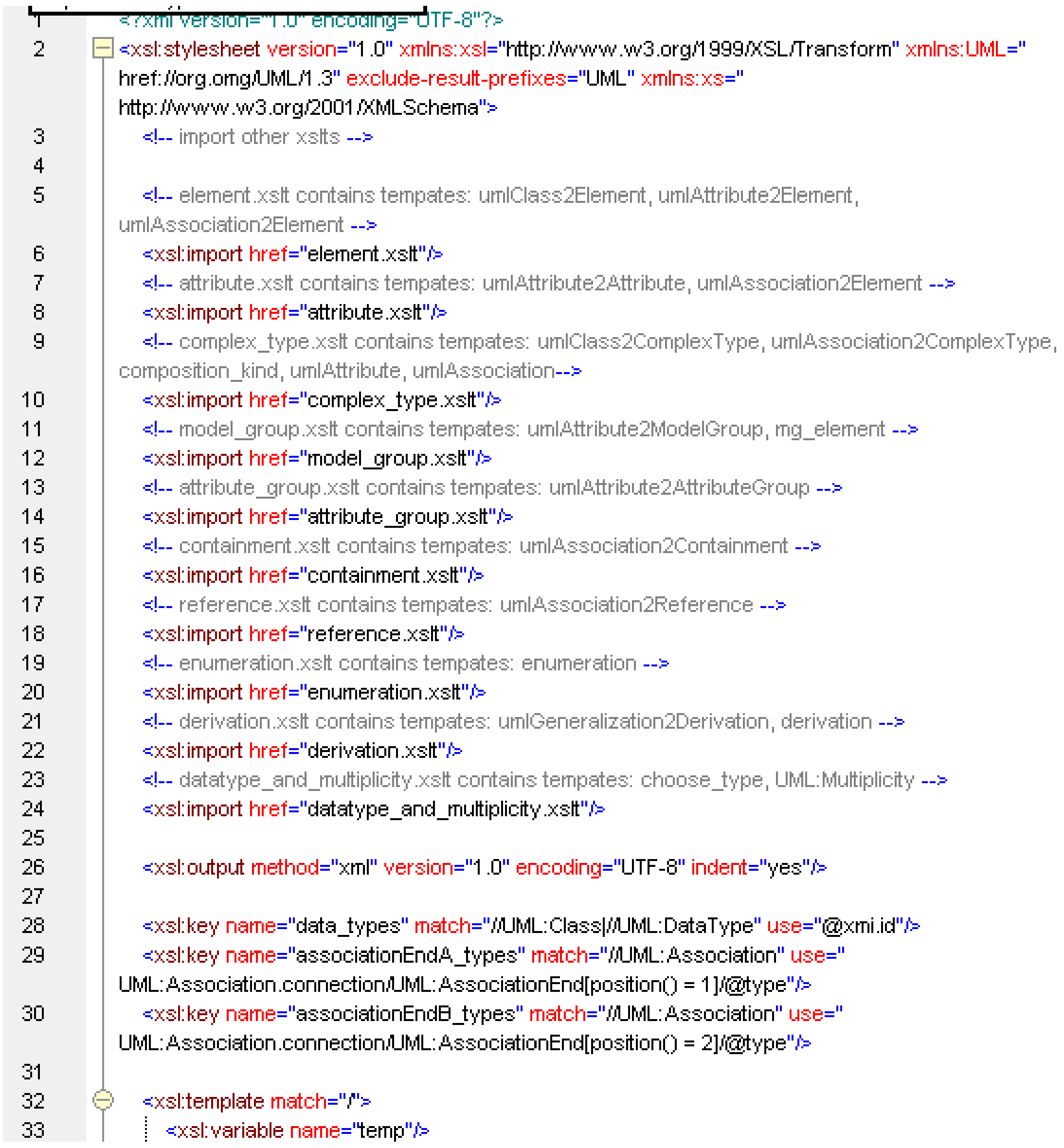
[25] ---, *Feature Diagram Overview*. Jarrett Interaction Design.
http://www.jarrettinteractiondesign.com/stories/storyReader$11

Appendix

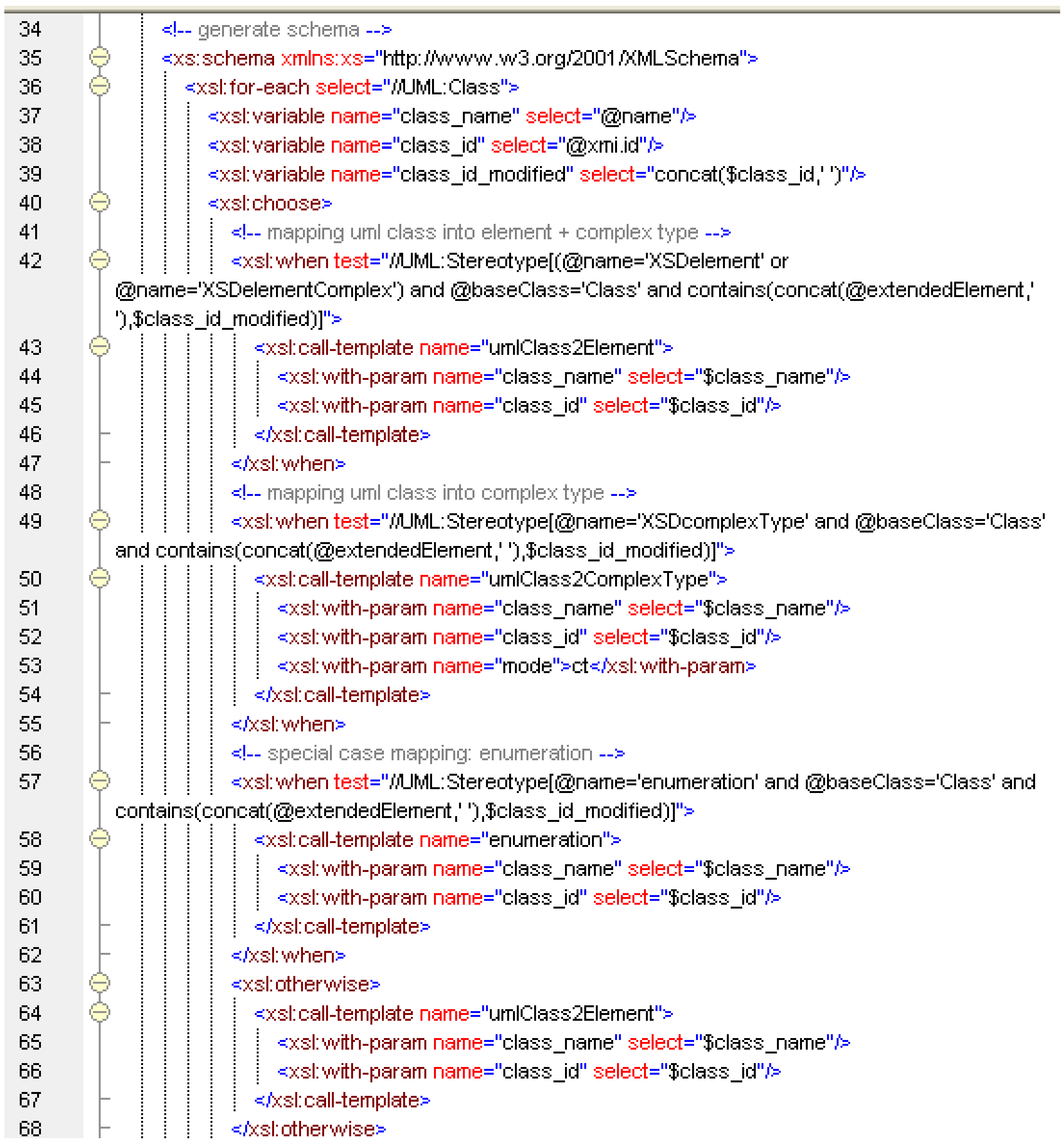
This appendix contains the xslt files used in chapter 3.

main.xslt

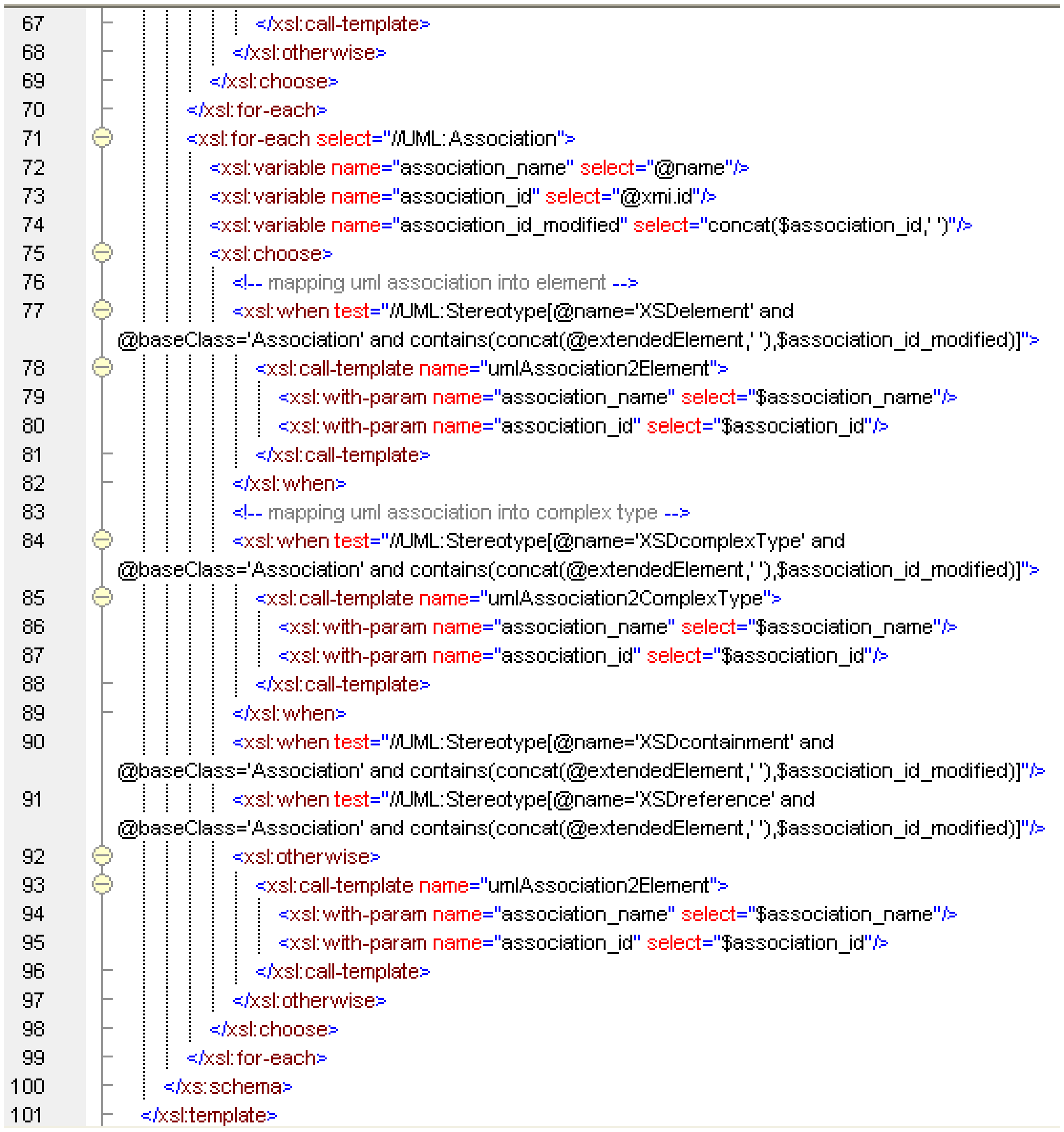
This file imports all other files and contains the main template.



```
<?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"/>
```



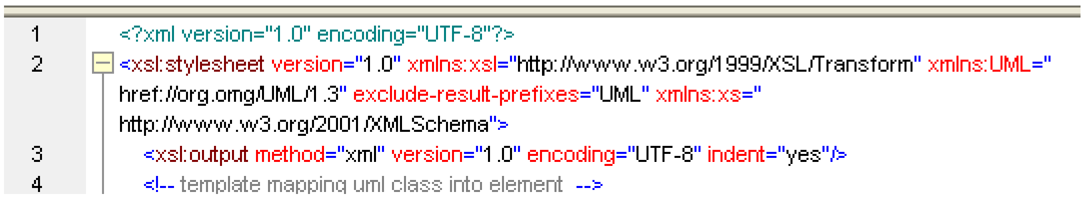
```
<!-- generate schema -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xsl:for-each select="//UML:Class">
    <xsl:variable name="class_name" select="@name"/>
    <xsl:variable name="class_id" select="@xmi.id"/>
    <xsl:variable name="class_id_modified" select="concat($class_id,' ')"/>
    <xsl:choose>
      <!-- mapping uml class into element + complex type -->
      <xsl:when test="//UML:Stereotype[(@name='XSDelement' or @name='XSDelementComplex') and @baseClass='Class' and contains(concat(@extendedElement,' '),$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>
```



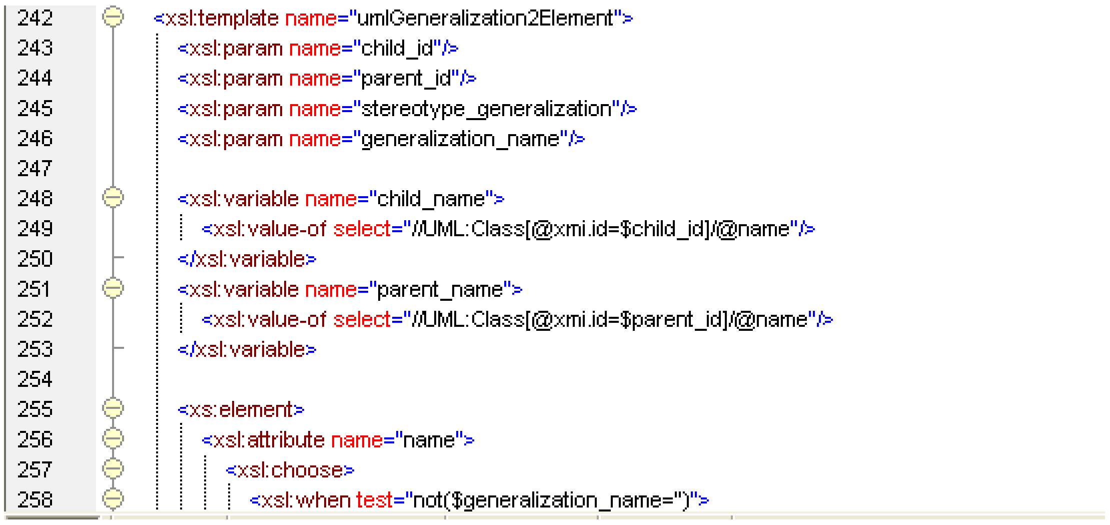
```
          </xsl:call-template>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:for-each>
    <xsl:for-each select="//UML:Association">
      <xsl:variable name="association_name" select="@name"/>
      <xsl:variable name="association_id" select="@xmi.id"/>
      <xsl:variable name="association_id_modified" 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>
```

element.xslt

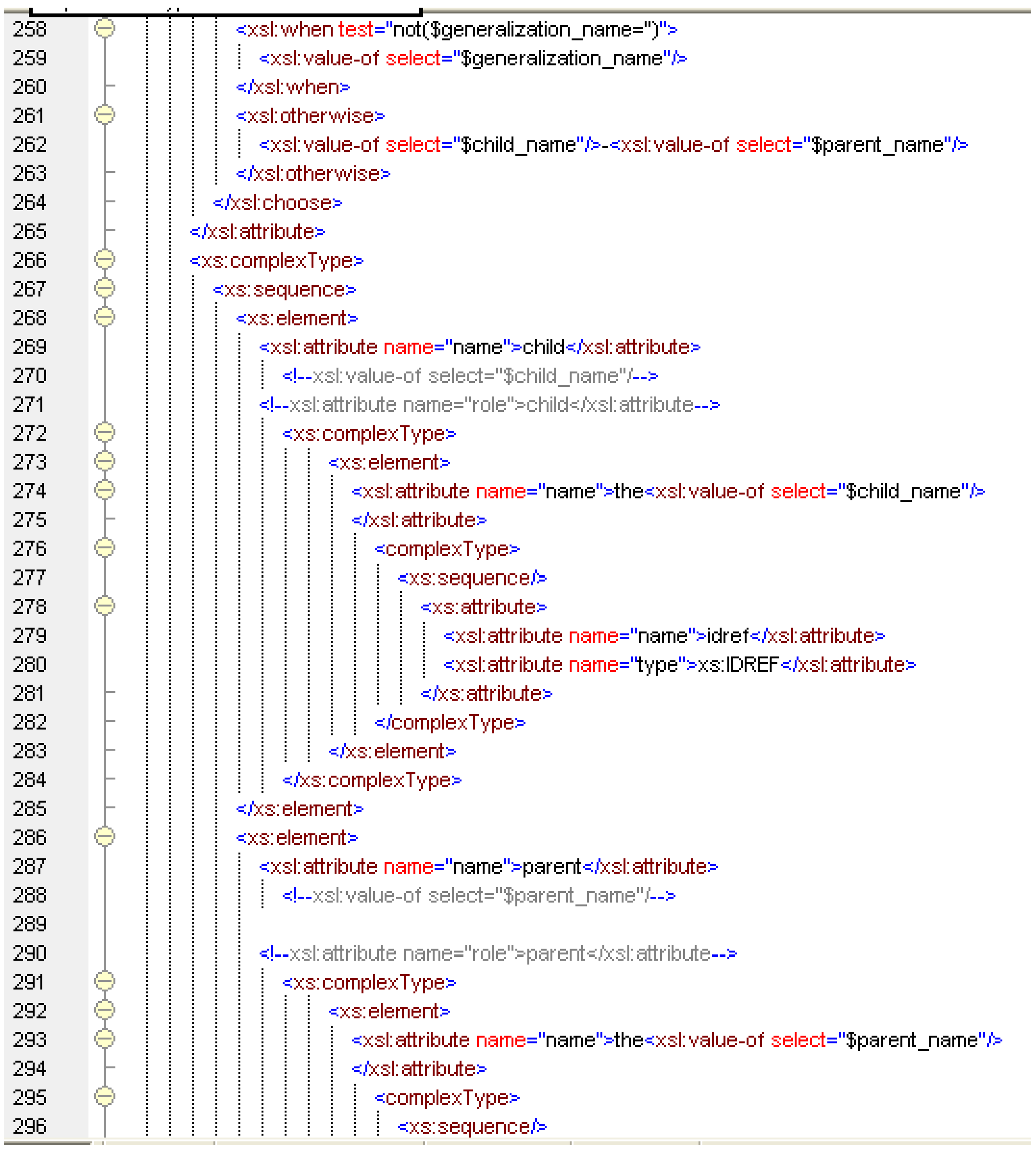
This file contains templates umlGeneralization2Element that maps UML generalization into element. Only template UMLGeneralization2Element is presented here.



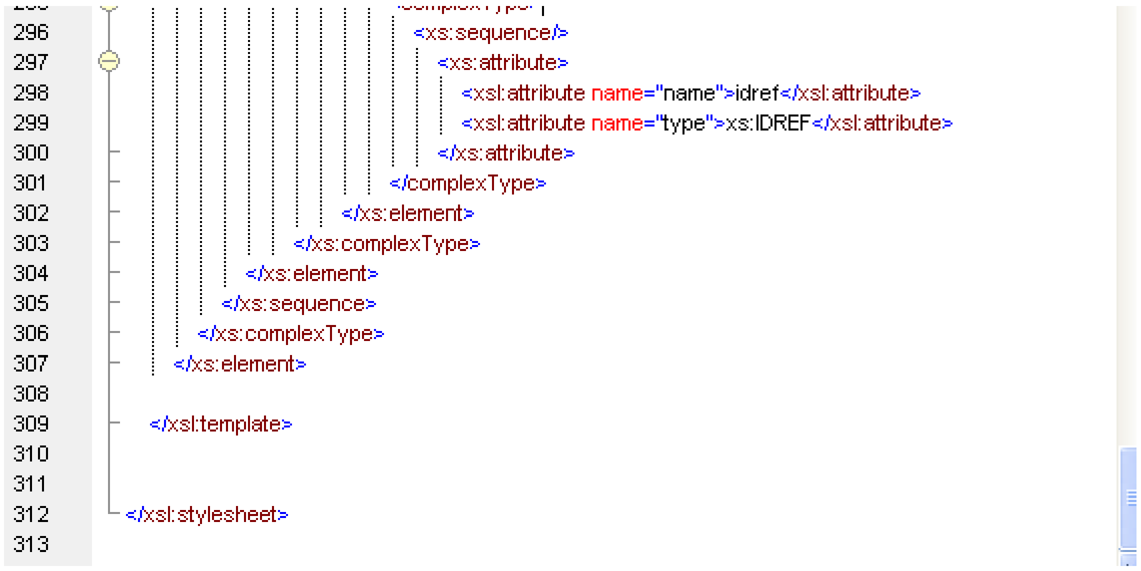
```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:UML="
href://org.omg/UML/1.3" exclude-result-prefixes="UML" xmlns:xs="
http://www.w3.org/2001/XMLSchema">
    <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>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



```
    <xsl:template name="umlGeneralization2Element">
        <xsl:param name="child_id"/>
        <xsl:param name="parent_id"/>
        <xsl:param name="stereotype_generalization"/>
        <xsl:param name="generalization_name"/>

        <xsl:variable name="child_name">
            <xsl:value-of select="//UML:Class[@xmi.id=$child_id]/@name"/>
        </xsl:variable>
        <xsl:variable name="parent_name">
            <xsl:value-of select="//UML:Class[@xmi.id=$parent_id]/@name"/>
        </xsl:variable>

        <xs:element>
            <xsl:attribute name="name">
                <xsl:choose>
                    <xsl:when test="not($generalization_name='')">
```



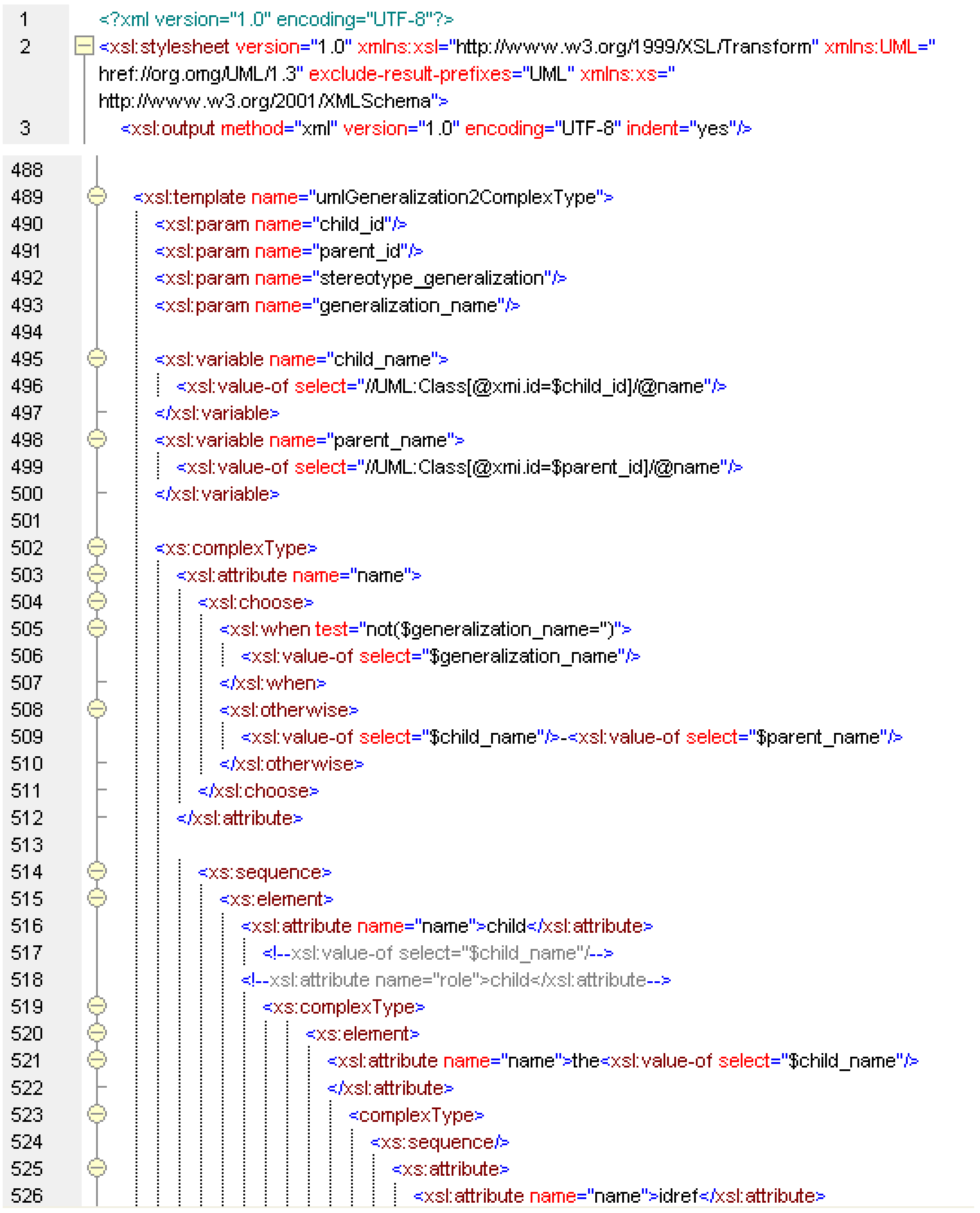
```
        <xsl:when test="not($generalization_name='')">
          <xsl:value-of select="$generalization_name"/>
        </xsl:when>
        <xsl:otherwise>
          <xsl:value-of select="$child_name"/>-<xsl:value-of select="$parent_name"/>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:attribute>
    <xs:complexType>
      <xs:sequence>
        <xs:element>
          <xsl:attribute name="name">child</xsl:attribute>
            <!--xsl:value-of select="$child_name"/-->
          <!--xsl:attribute name="role">child</xsl:attribute-->
            <xs:complexType>
                <xs:element>
                  <xsl:attribute name="name">the<xsl:value-of select="$child_name"/>
                  </xsl:attribute>
                    <complexType>
                      <xs:sequence/>
                        <xs:attribute>
                          <xsl:attribute name="name">idref</xsl:attribute>
                          <xsl:attribute name="type">xs:IDREF</xsl:attribute>
                        </xs:attribute>
                    </complexType>
                </xs:element>
            </xs:complexType>
        </xs:element>
        <xs:element>
          <xsl:attribute name="name">parent</xsl:attribute>
            <!--xsl:value-of select="$parent_name"/-->

          <!--xsl:attribute name="role">parent</xsl:attribute-->
            <xs:complexType>
                <xs:element>
                  <xsl:attribute name="name">the<xsl:value-of select="$parent_name"/>
                  </xsl:attribute>
                    <complexType>
                      <xs:sequence/>
```



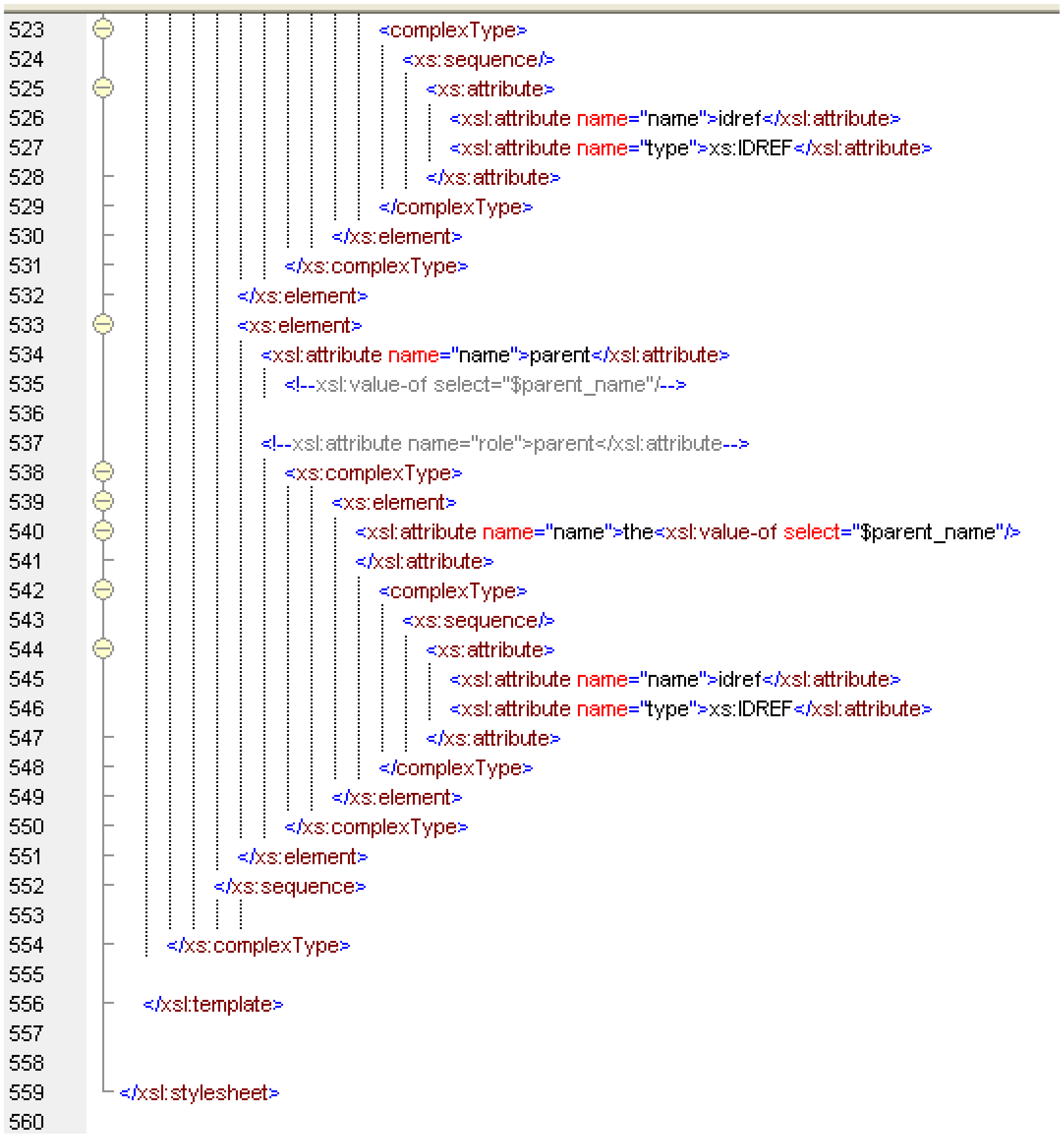
```
                        <xs:sequence/>
                          <xs:attribute>
                            <xsl:attribute name="name">idref</xsl:attribute>
                            <xsl:attribute name="type">xs:IDREF</xsl:attribute>
                          </xs:attribute>
                      </complexType>
                  </xs:element>
              </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>

  </xsl:template>


</xsl:stylesheet>
```

complex_type.xslt

This file contains templates umlGeneralization2ComplexType that maps UML generalization into complex type. Only template UMLGeneralization2ComplexType is presented here.



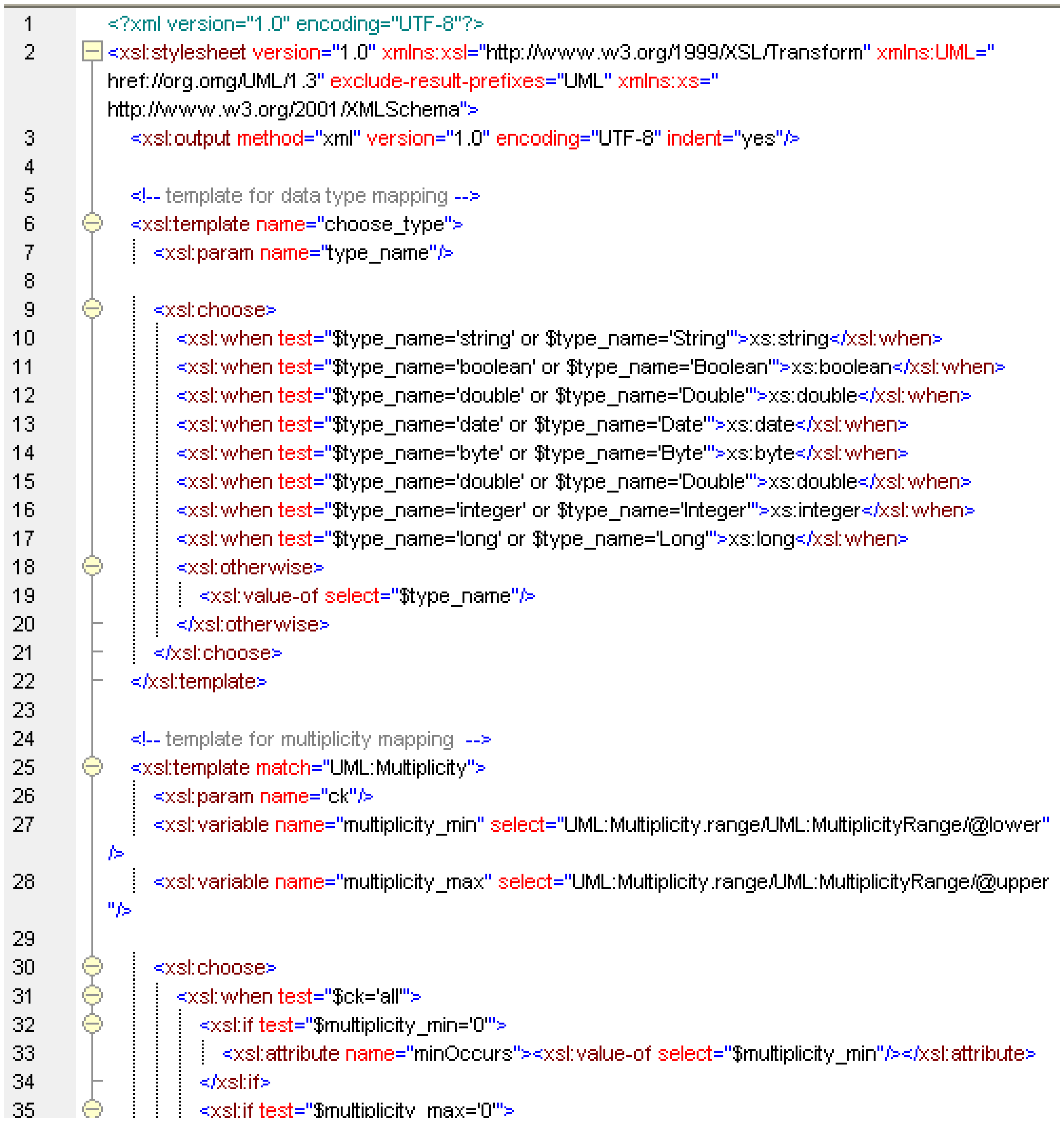
```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:UML="
href://org.omg/UML/1.3" exclude-result-prefixes="UML" xmlns:xs="
http://www.w3.org/2001/XMLSchema">
 <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

   <xsl:template name="umlGeneralization2ComplexType">
     <xsl:param name="child_id"/>
     <xsl:param name="parent_id"/>
     <xsl:param name="stereotype_generalization"/>
     <xsl:param name="generalization_name"/>

     <xsl:variable name="child_name">
       <xsl:value-of select="//UML:Class[@xmi.id=$child_id]/@name"/>
     </xsl:variable>
     <xsl:variable name="parent_name">
       <xsl:value-of select="//UML:Class[@xmi.id=$parent_id]/@name"/>
     </xsl:variable>

     <xs:complexType>
       <xsl:attribute name="name">
         <xsl:choose>
           <xsl:when test="not($generalization_name='')">
             <xsl:value-of select="$generalization_name"/>
           </xsl:when>
           <xsl:otherwise>
             <xsl:value-of select="$child_name"/>-<xsl:value-of select="$parent_name"/>
           </xsl:otherwise>
         </xsl:choose>
       </xsl:attribute>

         <xs:sequence>
           <xs:element>
             <xsl:attribute name="name">child</xsl:attribute>
               <!--xsl:value-of select="$child_name"/-->
             <!--xsl:attribute name="role">child</xsl:attribute-->
               <xs:complexType>
                   <xs:element>
                     <xsl:attribute name="name">the<xsl:value-of select="$child_name"/>
                     </xsl:attribute>
                       <complexType>
                         <xs:sequence/>
                           <xs:attribute>
                             <xsl:attribute name="name">idref</xsl:attribute>
```



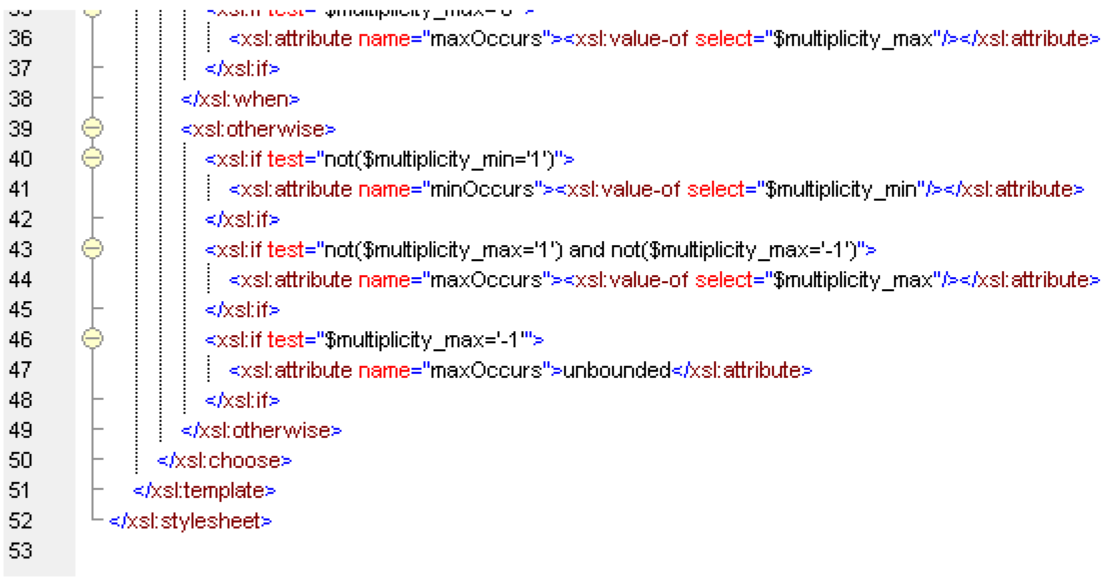
```
                      <complexType>
                        <xs:sequence/>
                          <xs:attribute>
                            <xsl:attribute name="name">idref</xsl:attribute>
                            <xsl:attribute name="type">xs:IDREF</xsl:attribute>
                          </xs:attribute>
                      </complexType>
                  </xs:element>
              </xs:complexType>
          </xs:element>
          <xs:element>
            <xsl:attribute name="name">parent</xsl:attribute>
              <!--xsl:value-of select="$parent_name"/-->

            <!--xsl:attribute name="role">parent</xsl:attribute-->
              <xs:complexType>
                  <xs:element>
                    <xsl:attribute name="name">the<xsl:value-of select="$parent_name"/>
                    </xsl:attribute>
                      <complexType>
                        <xs:sequence/>
                          <xs:attribute>
                            <xsl:attribute name="name">idref</xsl:attribute>
                            <xsl:attribute name="type">xs:IDREF</xsl:attribute>
                          </xs:attribute>
                      </complexType>
                  </xs:element>
              </xs:complexType>
          </xs:element>
        </xs:sequence>

    </xs:complexType>

  </xsl:template>


</xsl:stylesheet>

```

datatype_and_multiplicity.xslt

This file contains template to map a data type and template to map multiplicity of a UML attribute and UML association end.



```
<?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"/>

  <!-- template for data type mapping -->
  <xsl:template name="choose_type">
    <xsl:param name="type_name"/>

    <xsl:choose>
      <xsl:when test="$type_name='string' or $type_name='String'">xs:string</xsl:when>
      <xsl:when test="$type_name='boolean' or $type_name='Boolean'">xs:boolean</xsl:when>
      <xsl:when test="$type_name='double' or $type_name='Double'">xs:double</xsl:when>
      <xsl:when test="$type_name='date' or $type_name='Date'">xs:date</xsl:when>
      <xsl:when test="$type_name='byte' or $type_name='Byte'">xs:byte</xsl:when>
      <xsl:when test="$type_name='double' or $type_name='Double'">xs:double</xsl:when>
      <xsl:when test="$type_name='integer' or $type_name='Integer'">xs:integer</xsl:when>
      <xsl:when test="$type_name='long' or $type_name='Long'">xs:long</xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="$type_name"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

  <!-- template for multiplicity mapping  -->
  <xsl:template match="UML:Multiplicity">
    <xsl:param name="ck"/>
    <xsl:variable name="multiplicity_min" select="UML:Multiplicity.range/UML:MultiplicityRange/@lower"/>
    <xsl:variable name="multiplicity_max" select="UML:Multiplicity.range/UML:MultiplicityRange/@upper"/>

    <xsl:choose>
      <xsl:when test="$ck='all'">
        <xsl:if test="$multiplicity_min='0'">
          <xsl:attribute name="minOccurs"><xsl:value-of select="$multiplicity_min"/></xsl:attribute>
        </xsl:if>
        <xsl:if test="$multiplicity_max='0'">
```



```
        <xsl:attribute name="maxOccurs"><xsl:value-of select="$multiplicity_max"/></xsl:attribute>
      </xsl:if>
    </xsl:when>
    <xsl:otherwise>
      <xsl:if test="not($multiplicity_min='1')">
        <xsl:attribute name="minOccurs"><xsl:value-of select="$multiplicity_min"/></xsl:attribute>
      </xsl:if>
      <xsl:if test="not($multiplicity_max='1') and not($multiplicity_max='-1')">
        <xsl:attribute name="maxOccurs"><xsl:value-of select="$multiplicity_max"/></xsl:attribute>
      </xsl:if>
      <xsl:if test="$multiplicity_max='-1'">
        <xsl:attribute name="maxOccurs">unbounded</xsl:attribute>
      </xsl:if>
    </xsl:otherwise>
  </xsl:choose>
 </xsl:template>
</xsl:stylesheet>
```