International Journal of Software Engineering and Its Applications
Vol. 8, No. 12 (2014), pp. 75-90
http://dx.doi.org/10.14257/ijseia.2014.8.12.07

Formalization of Transformation Rules from XML Schema to UML Class Diagram

Hannani Aman and Rosziati Ibrahim

Faculty of Computer System and Information Technology, University Tun Hussein Onn Malaysia, Johor, Malaysia
hanani@uthm.edu.my, rosziati@uthm.edu.my

Abstract

XML Reverse Engineering is a research that focuses on getting a conceptual model using an XML schema. In integration issue, previous XML reverse engineering researchers apply the reverse method of XML schema or document in order to generate a class diagram. However, to generate a complete class diagram, XML constructs are not used entirely. Therefore, this paper describes XML transformation that focuses on each XML construct transforming to a class diagram. In order to generate a complete class diagram, formal method is used. There are several steps involved in constructing and transforming each XML into a class diagram. In order to ensure the formalization is complete, the ebXml case study is used and from the result obtained, this method can be used as an alternative solution to show a complete reverse XML schema.

Keywords: *XML Reverse Engineering, Class Diagram, Formalization*

1. Introduction

XML Reverse Engineering is a potential mechanism used to represent XML in conceptual level that accept not as much of interest for the researchers. In conceptual level, conceptual model need to be generated to illustrate data structure and relationships in traditional database [1–4]. It is acknowledged that conceptual models are helpful in designing a system or a database. It is also as an alternative medium for communication between end user and developer. However, there is a conflict between XML conceptual model and data organization. In conceptual model, it needs high abstraction and graphical notation in making the conceptual model independent of XML schema languages, whereas in data organization, the detail should not be exposed. Meanwhile, if we want to capture XML-specific in the conceptual model, we have to reveal some features in its data organization.

The existing approaches have significant contribution of forward engineering of XML which to transform the conceptual model to an XML document or schema for integration issue [5–7]. But less work is conducted on generating a conceptual model back from an XML document or schema which is important as an alternative way for developers whom adapting Agile methodology. Some researchers [3, 8–10] focus on the Data Type Definition (DTD), a type of XML schema, but some other researchers use XML Schema as a source [2, 3, 11]. XML Schema has been used because of its popularity nowadays. Reverse engineering process results a few kinds of conceptual model which are Class Diagram[2, 3, 8, 11], Conceptual XML(C-XML) [12], Entity Relationship Diagram (ERD) [3] and an XML Tree Model(XTM) [10]. Various kinds of transformation rules have been done by these researchers such as rule based conversion, transformation rules and semi automatic formal method that maps a few primary components of XML Schema.

ISSN: 1738-9984 IJSEIA

This research is motivated in generating UML diagrams from XML Schema because the previous research did not formalize the transformation rules using all primary components of XML Schema. This formal model can be used to ensure that the transformation rules generated are consistent in the class diagram.

The rest of the paper is organized as follows: the review of the XML document and XML Schema is discussed in Section 2. An overview on UML Class Diagram is given in Section 3. Section 4 discusses about syntax and semantics transformation rules. Section 5 formalizes the XML reverse transformation rules. Section 6 provides a discussion about the case study of XML Reverse Transformation. Finally, Section 7 concludes the paper.

2. XML and UML Class Diagram Overview

2.1. XML Document

XML documents recommend languages that use tags to mark up the text. The tags describe their content as XML is called as 'self describing' language. XML is flexible enough to allow for the creation of many different types of languages to describe data. The only constraint on XML vocabularies is that they are well-formed. Well formed XML is based on a few criteria. The first criterion is that the document contains one or more elements. The second criterion is that its document contains a single document element, which may contain other elements. The third criterion is each element closes correctly, case-sensitive and its attribute values are enclosed in quotation marks and cannot be empty. The fourth criterion is the order of the tags, tag that open first, must close last. In XML documents, those pieces contained in between open and close tags is called an element. An XML document that has fulfilled XML construction rules are as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<book isbn="0836217462">
      <title>Being a Dog Is a Full-Time Job</title>
      <author>Charles M. Schulz</author>
      <character>
            <name>Snoopy</name>
            <friend-of>Peppermint Patty</friend-of>
            <since>1950-10-04</since>
            <qualification>      extroverted beagle    </qualification>
      </character>
      <character>
            <name>Peppermint Patty</name>
            <since>1966-08-22</since>
            <qualification>bold, brash and tomboyish</qualification>
      </character>
</book>
```

From the example above, the self describes tags such as <book>, <title>, <author>, <character>, <name>, <friend-of>, <since> and <qualification> are organized in pairs with </book>, </title>, </author>, </character>,</name>,</friend-of>, </since> and </qualification> tags. The pairing organization is called well formed. The correctness of an XML document are based on its well formed in XML schema syntax. The order of the tag is nested pairing as it shows the importance of element ordering.

2.2. XML Schema

XML schema definition language (XSD) has been a mainstream for describing XML documents. XML schema primary component contains simple type, complex type, element,

attribute, attribute group, identity-constraints, model group and notation. According to Booch and friends[13], XML schema is a language for defining the structure of XML document instances that belong to a specific document type. XML is replacing its previous XML syntax, XML DTD. It provides strong data typing, modularization and reuse mechanism which is unavailable in XML DTDs. Al Kamha and friends[14] claimed that XML schema has a hierarchical structure to describe the document. In addition, Mani and friends[15] explained that with the hierarchical structure, XML schema emphasizes the ordered relationship between the elements.

XML schema also supports constraints such as key and foreign key constraints. In the work of Belen and Esperanza[16], XML Schema is the definition of a specific XML structure. An XML Schema itself is a special kind of XML document that defines the content and structure of an XML document, which describes the components that may be contained in an XML document and the ways the components may be arranged within the document structure. Moreover, Salim[11] added that Schema languages are used to provide a means for describing the structure, content and semantics of XML instance documents. XML instance documents are XML documents that adhere to the structure and the rules specified in the space schemas of XML. Therefore, XML Schema is the most recent emerging standard in describing XML instance documents which also serve as validation support and design documentation for a set of XML instance documents.[4].

XML Schema is a document that describes what and how XML document tags are used. It is a meta-data that describe the element in the XML document. XML Schema contains descriptions of element and relationship. Basically, XML schema consists of 5 different components, which are: type of definition components (simple and complex types), declaration components (elements and attributes), model group components, constraint components, group definition components and annotations components.

XML schema components that fulfill previous XML document requirement are as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schemaxmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="book">
<xs:complexType>
<xs:sequence>
<xs:element name="title" type="xs:string"/>
<xs:element name="author" type="xs:string"/>
<xs:element name="character" minOccurs="0" maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:element name="name" type="xs:string"/>
<xs:element name="friend-of" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="since" type="xs:date"/>
<xs:element name="qualification" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
<xs:attribute name="isbn" type="xs:string"/>
</xs:complexType>
</xs:element>
</xs:schema>
```

The previous XML Schema shows that one element named <book> contains attribute named <isbn> with string type. Another element named as <title>, <author>, and <character>

are nested in the <book> element. Each <title> and <author> elements has string type. While <character> element has a set of element named <name>, <friend-of>, <since> and <qualification> nested with it. <name>, <friend-of> and <qualification> elements are also used string type, while <since> element use date type. <character> element may occur within <book> element either with empty element or maximum unbounded element. Other elements that are nested to <book> element, which are <title> and <author> occur only once. <friend-of> element may occurs with either empty element or maximum unbounded element related to <character> element.

XML Schema presentation is easily understood by the developer himself, but not to the user who wants to review the application content. Therefore, there is need to change back (or reverse) the XML Schema to a conceptual view that is more easily recognized and understand by the end user. The conceptual diagram selected is class diagram which will be explained in the Section 3.

2.3. Class Diagram: An Overview

A class diagram is one of Unified Modeling Language diagrams that are used to view a static structure of a system. Each class diagram contains attributes and method, described as its most relevant features/properties e.g. visibility and type. It also shows the relationships between other classes in the diagram, thus to allow for communication and interaction among them. In short, this provides an informative summary of design decisions about a system organization. A class in class diagram is an element in the system. The element's feature is described within the class recognized as attributes and properties. Each attribute may also have its own descriptions which are named as type, visibility and scope. Relationships between classes are used to show how a class is able to access features of other class. Generalization, aggregation, composition and association relationships are used to indicate that a class has access to other classes' features.

A generalization relationship connects two classes when a class (super class) inherits features to other class (subclass). A subclass that inherits features from other class may add its own features to complete its class. Aggregation is a form of strong relationship. A class is having an aggregation relationship to another class if the end (second) class of connection is a part of prior (first) class. The end class exists independently in cooperation with the prior class. In the other words, object of prior class has an object of the end class. The stronger relationship is composition, where the end (second) class is part of the prior class. The prior class needs the end class as a whole class.

An association is a weaker relationship among classes. There are 2 types of association: bidirectional and unidirectional. Bidirectional association occurs when there is a possibility to navigate from an object instantiating the first class to an object of second class and vice versa. A unidirectional association exists when there is only one direction that is possible. In this association, the prior class may access the end class at any time. Previous XML Schema example, generated from a class diagram as in Figure 1. Discussion about transformation process is provided in Section 3.

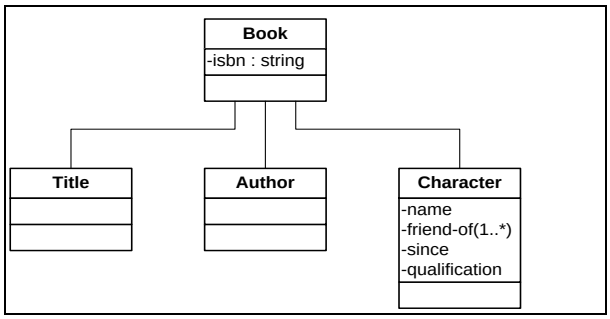

Figure 1. Class Diagram of Book schema

Every element is a class and the association is based on the minimum or maximum occurrences between the nested elements. Details complex transformation will be explained in Section 4.

3. Transformation Rules

XML is known as predefined tags that can be created by any developer whereas XML schema is a metadata describing how the XML document is organized. Class Diagram is a diagram that represents XML schema in a logical view of data used in XML document. XML Schema need to be presented in a class diagram as its will shorten the communication time between developer and end user.

XML schema consists of type definitions, element declarations, attribute declarations, attribute group definition and model group definition. Type definition is used to constrain the structure of element and attribute. It is divided into 2 types of type definition which are simple and complex type. Simple type is a set of constraint on strings and it save information about value attributes or text valued elements. Complex type is another kind of type definition that uses a structure of a set of attribute declarations or a set of element. A set of element in complex type may have a set of child element. When a set of elements occurs more than once in a schema, the set of elements will be a group that has a name to refer to called model group definition. When a set of attributes occurs more than once in an element, the set of attributes will be a group that has a name to refer to called attribute group definition. This explanation of schema definition is in Rule 1.

Rule 1: XML schemas consist of:

- Type definitions
- Element declarations
- Attribute declarations
- Attribute group definitions
- Model group definition

where

- Type definition is used to constrain the structure of element and attribute.
- Element declaration is an association of a name with a type definition, either simple or complex. It also has an attribute within. Each element may occur more than once in the schema.
- Attribute declaration is an association between a name and a simple type definition, together with occurrence information.

- Attribute group definitions is a group of attribute declaration with a name of the group. This group is embedded in element together in complex type definition or element or with attribute in an element.
- A model group definition is an association between a name and a model group, enabling re-use of the same set in several complex type definitions.

To represent the XML schema in a logical view, class diagram of UML is described. Class diagram consist of two main components, which is class and association. The definition of class diagram is represented in Rule 2.

Rule 2: Class diagram consist of:
- Class
- Association

where
- A class consists of a set of attributes and each attribute may have multiplicity.
- Association is a connection between classes. Each association may have multiplicity.

In XML schema, element is the main component to be referred to and has its own content. This is similar with a class diagram where a class has attributes as its content. This similarity definition enable for the element to be transformed to a class. In the meantime, an attribute of an element will be transformed into an attribute of a class. For each global simple type definition, it will be transformed into a stereotype class of <<type>>. Rule 3 explains the transformation rules for XML Schema construct.

Rule 3: Transformation rules for XML Schema construct:
- A global element declaration is transformed to a UML class.
- A global simple type definition is transformed to a UML class.
- A local attribute declaration is transformed to a UML attribute
- A local element declaration is transformed to a UML attribute.
- An attribute group definition is transformed to a UML class.
- A model group definition is transformed to a UML class.
- A local complex type definition located within element declaration is transformed to a UML class.
- Each element that only has type definition is transformed into attribute and its type definition will be attribute type.

Sometimes, the global or local element did not have data type declared directly. It only shows the component is nested to other component. These make the representation of classes empty and need an artificial name before a class is created. Rule 4 explains the rules for generating artificial names for element which uses predefined declaration. Predefined type of element is where an element is declared with a type definition only. There is no nested element within its element.

Rule 4: Rules for generating artificial names for element.
- An artificial name for a UML attribute is composed of the UML class name followed by capital letter of first attribute text and “attribute” text.
- An artificial name for a UML class is composed of the name of the transformed XML Schema construct and “element” text.

- An artificial name for each element that has complex type with sequence ordering is transformed into a UML class that has an attribute "{ordered}" composed of string and a number.
- An artificial name for each element that has complex type with choice ordering is transformed into a UML class that has attributes "{or}" composed of string and a number.

Rule 5 is a relationship transformation among element declaration. It is used to relate all the elements used in XML Schema to create a class diagram. Rule 6 show that all the compositor of complex type is transformed into a specific component in a class diagram. Finally, when the complex type is nested within another XML Schema component, Rule 7 is used.

Rule 5: Relationship transformation:
- A global element declared as complex type with other nested global element, is transformed to association relationship.
- Each element or attribute that have occurrences with maxOccur or minOccurs construct is transformed to UML multiplicity with value of zero-to-one, one-to-many, zero-to-many or by default one-to-one.

Rule 6: Complex Type Transformation Rules
- Each element that has complex type with sequence ordering is transformed into a UML class that has an attribute "{ordered}", followed by the attributes in the sequence. This applies to the child element that uses predefined types without occurrences and no nested element.
- Each element that has complex type with choice ordering is transformed into a UML class that has attribute "{or}". This applies to the child element that uses predefined types without occurrences and no nested element.
- "all" is a compositor that allows the immediate child elements to appear in any order with multiplicity either 0 or 1. An all can only contain element declarations; thus it could not hold a choice or a sequence compositor.

Rule 7: When there is compositor nested within one another, the combination of each compositor rule is applied.

Finally, these are seven rules defined to explain the definition and transformation of an XML construct to a Class Diagram. In the next section, the use of formalization language to ensure the correctness and completeness of the reverse transformation is presented.

4. Formalization of Transformation Rules

In an XML schema, a collection of elements, type definitions, attributes, attributes group and model group component are drawn. The element is adhered to Element component. The type definition is adhered to Type Definition component. The attribute is adhered to attribute component. The attributes group is adhered to Attribute Group component. Meanwhile, model group is adhered to the Model Group component.

In Definition 1, Element component is represented as a set named E, Type Definition component is represented as a set named T, Attribute component is represented as a set named AT, Attributes Group component is represented as a set named ATG and Model Group component is represented as a set named EG. We assume T, E, ATG, EG constructs are global definition and they are a direct child of S.

Definition 1. *Let "S" be an XML schema construct. The "S" consists of a finite set of type definitions, a finite set of elements, a finite set of attributes, a finite set of attribute groups and a finite set of model group component. Thus, it can be defined as*

$S = \{T, E, AT, ATG, EG\}$
where
$T = ST \vee CT$
$E = \{e_i \mid 1 \le i \le n\}$ is a finite set of elements;
$AT = \{at_i \mid 1 \le i \le n\}$ is a finite set of attributes;
$ATG = \{atg_i \mid 1 \le i \le n\}$ is a finite set of attribute groups;
$EG = \{eg_i \mid 1 \le i \le n\}$ is a finite set of model groups

Definition 2. *Let "T" be a set of type definition of schema, "S". A type definition may be a complex type definition, "CT" or a simple type definition, "ST". A simple type definition may be an enumerated of string, while complex type definition may consist of an ordering, a finite set of elements, a finite set of attributes, a finite set of attribute groups and a finite set of model group. Thus, it can be defined as*

$st \in ST; ct \in CT;$
where
$ct_i = \{odr(ct_i), e(ct_i), at(ct_i), atg(ct_i), eg(ct_i)\};$

In definition 3, an enumerated string is defines as "A". A simple type may use an enumerated of string.

Definition 3. *Let "A" be an enumerated of string. A simple type definition, "st" is an enumerated of string. Thus, it can be defined as*

$A := \{a..z|A..Z|.\infty\};$
$st \subseteq A;$

In definition 4, an ordering component in complex type construct has 3 values which are sequence, choice and all.

Definition 4. *Let "odr" be an ordering of complex type that consist value of sequence, choice and all. Thus, it can be defined as*

$odr = \{sequence, choice, all\}$

In Definition 5, an attribute of a schema or a complex type is defined. An attribute must have a name and a type definition. A set of attribute that can be grouped together and may be used in other complex type is defined in Definition 6.

Definition 5. *Let "$atg \in ATG$" be a set of attributes of a category in schema, S with a name. Let "$at \in AT$" that consist of a name and simple type definition. Thus, it is defined as*

$$at(atg_i) = \{< at_1 : st_1 >, < at_2 : st_2 >, .. < at_n : st_n >$$

Definition 6. *Let "$eg \in EG$" be a set of element of a model category in schema,"S" with a name. Let "$e \in E$". It can either be from complex type, "ct" or a set of attribute, "at". Thus, it can be defined as*

$$e(eg) = \{e_1, e_2, ...e_n\}$$
where
$$e_i = \{ct_i(e_i) \vee at_i(e_i)\}$$
where
$ct(e_i)$ *is a complex type*
$ct(e_i) = \{ct(e_i), ct(e_i), ...ct(e_n)$ *;*
$at(e_i)$ *if a finite set of attribute of* e.

Each element and attribute may have occurrences of maximum or minimum occurences. maxOccurs is defined as the value of maximum occurrences of element or attribute in that class while minOccurs is defined as the value of minimum occurrences of element or attribute. This explanation is defined in Definition 7.

Definition 7. *Let "$occurences$" be an element or attribute occurrences of maxOccurs or minOccurs. maxOccurs is defined as the number of maximum occurrences of an element while MinOccurs shows the number of minimum occurrences of an element. Thus, it can be defined as*

$$occurences = \{\max Occurs \vee \min Occurs\}$$
where
$$\max Occurs = \min Occurs = \{0 \mid 1 \mid \infty\}$$

Definition 8 defines a class diagram consist of a finite set of classes and a finite set of associations. Definition 9 defines a class that consists of a set of attributes of class with their type definition. Each attribute may have a multiplicity value within. The association is defined between classes which are denoted as the n-ary relationship of class in Definition 10.

Definition 8. *Let "cd" be a class diagram.* cd *consist of a finite set of classes and a finite set of associations. Thus, it can be denoted as*
$cd = \langle CLS, ASSOC \rangle$ *;*
where
$CLS = \{cls_i \mid 1 \le i\}$ *is a finite set of class*
$ASSOC = \{assoc_i \mid 1 \le i \le n\}$ *is a finite set of association*
Definition 9. *Let* $cls \in CLS$. *The class consists of a finite set of attribute. Thus, it can be denoted as*

$$cls_i = att(cls_.)$$

where
$att(cls_i)$ *is a set of attribute of class* $\{\langle att_1 : T_1\rangle,....\langle att_n : T_n\rangle\}$
and T_i *denoted to type of attribute,* att_i *of a class,* cls_i *with a multiplicity value.*

Definition 10. *Let "association" be an association between classes. Thus, it can be denoted as n-ary relationship of class.*

$$\underbrace{CLS \times ... \times CLS}_{n}$$

where
$association = (cls_1,....cls_n)$
where
$association \in ASSOCIATION$ *and* $cls \in CLS$

Definition 11. *Let "* $mutiplicity$ *" be a multiplicity of classes or attribute. Multiplicity is the number of minimum and maximum occurrences of association of class or attribute in each class. Thus, it can be denoted as*

$$multiplicity_i \in \{Z^+, \infty\}$$

XML constructs is transformed into UML class diagram using Definition 1 to Definition 10. The transformation rules are explained in the following Transformation Rule (TR).

TR 1. *Let* $Trans(e_i)$ *be a transformation rule for the element. "e" transforms to a class if the class is a subset to a Schema, S, while element is transformed into attribute if element is pre-defined. Thus, it can be denoted as*

$$Trans(e_i) = \begin{matrix} cls_i \\ att_i \end{matrix} \; if \; \begin{matrix} e_i = \{ct_i(e_i) \vee at_i(e_i)\} \\ \{\langle e_i : T\rangle\} \end{matrix}$$

TR 2. *Let* $Trans(atg_i)$ *be a transformation rule for attribute groups. "* atg *" is transformed to a UML class if the attributeGroup is a subset to a Schema,* S *. All attributes in the group will be UML class's attributes. Thus, it can be denoted as*

$$Trans(atg_i) = cls_i \;\; if \;\; atg_i \subseteq S$$

TR 3. *Let* $Trans(eg_i)$ *be a transformation rule for model groups. "* eg *" is transformed to a UML class if the modelGroup is a subset to a Schema,* S *. All elements in the group will be UML class's attributes. Thus, it can be denoted as*

$$Trans(eg_i) = cls_i \;\; if \;\; eg_i \subseteq S$$

TR 4. *Let* $Trans(at_i)$ *be a transformation rule for attributes. "* at *" is transformed to a UML class diagram attribute of referring class. Thus, it can be denoted as*

$Trans(at_i) = att_i(cls_i)$

TR 5. *Let* $Trans(T_i)$ *be a transformation rule for type definition. "T " is transformed to an UML stereotype class <<type>> if the type definition is a subset to* S *. Thus, it can be denoted as*

$Trans(T_i) = cls_{type}$ if $st_i \vee ct_i \subseteq S$

TR 6. *Let* $Trans(odr_i)$ *be a transformation rules for ordering of complex type. "*odr*"is transformed to a UML attribute "{ordered}" if* odr *is a sequence but if* odr *is a choice, it is transformed to a UML attribute "{or}". When the* odr *is "all", by default, no attribute is generated. Thus, it can be denoted as*

$$Trans(odr_i) = \begin{matrix} att_{\{ordered\}} \\ att_{\{or\}} \end{matrix} \; if \; \begin{matrix} odr_i = \{sequence\} \\ odr_i = \{choice\} \end{matrix}$$

TR 7. *Let* $Trans(e_a, e_b)$ *be a transformation rule for association in UML class diagram if* e_a is a subset element of e_b.

$Trans(e_a, e_b) = assoc$ if $e_a \subseteq e_b$

TR 8. *Let* $Trans(occurences, e_i)$ *be a transformation rules for multiplicity of association between classes. If minOccurs is 0, then multiplicity of* e_i *is 0 to 1. If maxOccurs = unbounded, then multiplicity of* e_i *is 1 to infinity. If maxOccurs or minOccurs of* e_i *is not stated, then multiplicity of* e_i *is 1 to 1. Else the number of multiplicity is based on number of maxOccurs or minOccurs stated. Thus it can be denoted as*

$Trans(occurences, e_i) =$

$$\begin{cases} multiplicity_{cls_{e_i}} = \{0..1\} & if\ minOccurs(e_i) = \{0\} \\ multiplicity_{cls_{e_i}} = \{1..*\} & if\ maxOccurs(e_i) = \{unbounded\} \\ multiplicity_{cls_{e_i}} = \{1..1\} & if\ minOccurs(e_i) \vee maxOccurs(e_i) = null \end{cases}$$

TR 9. *Let* $Trans(occurences, att_i)$ *be a transformation rule for multiplicity of attributes. If minOccurs is 0, then multiplicity of* att_i *is 0 to 1. If maxOccurs = unbounded, then the multiplicity of* att_i *is 1 to infinity. If maxOccurs or minOccurs of* att_i *is not stated, then multiplicity of* att_i *is 1 to 1. Else the number of multiplicity is based on the stated number of maxOccurs or minOccurs. Thus, it can be denoted as*

$Trans(occurences, att_i) =$

$$\begin{cases} multiplicity_{cls_{att_i}} = \{0..1\} & if\ minOccurs(att_i) = \{0\} \\ multiplicity_{cls_{att_i}} = \{1..*\} & if\ maxOccurs(att_i) = \{unbounded\} \\ multiplicity_{cls_{att_i}} = \{1..1\} & if\ minOccurs(att_i)\ \vee maxOccurs(att_i) = null \end{cases}$$

These transformation rules need to be proven. In this research a case study is used to prove that these transformations rules are complete and reliable.

5. Case Study using Transformation Rules

In this section, a case study on transformation rules defined in Section 4 is presented. The case study is taken from Liquid Technology web entitled Electronic Business using eXtensible Markup Language (ebXml,http://schemas.liquid-technologies.com/ EBXML/2.0/). This case study is based on 1 of 6 schema ebXml that involve all XML construct definitions (element, attribute, attribute group, complex type, model group and simple type) as in Section 4. ebXml is an open specification of XML based infrastructure which provide global use of electronic business information for an interoperable, secure and consistent manner sponsored by Organization for the Advancement of Structured Information Standards (OASIS) and UN/CEFACT.

This case study consists of 60 global elements, 9 global complex types, 9 global simple types and 2 attribute groups. The ebXml schema design is based on the Garden of Eden which consists of global element and type. This design allows the reuse of both elements and types. The design benefits the high reuse of class. However, this kind of schema is difficult to read and understand. The difficulty makes this case study suitable to complete reverse transformation successfully.

XML constructs transformation uses only element, complex type, simple type, attributes and group attribute. Other XML constructs that are not listed, will not be used. Table 1 shows the transformation steps applied in this case study. The first step is to let all global elements, attribute groups and model groups to be transformed into a class.

It is based on Transformation Rule (TR) 1 – 3. During the transformation in TR1, let XML attribute is the attribute of UML Class Diagram as in TR4. The UML Class Diagram attribute multiplicity also needs to be transformed from occurrences of XML attribute using TR9. Next step is to generate a stereotype <<type>> class from the construct type definition based on TR5. In the complex type transformation step, ordering construct is transformed into new attribute "{ordered}" or "{or}". Thirdly, the association of classes is generated using TR7 and finally, multiplicity of the association is generated based on TR8.

Table 1. Transformations Steps

1. Every global construct is transformed into a class. a. XML attributes for each element is transformed to UML attribute. b. Every XML attributes occurrences are transformed to UML attribute multiplicity.
2. Every global complex type and simply type is transformed into a stereotype class <<type>>. 2. 1 Every ordering in complex type is transformed into new attribute({ordered}/{or})
3. For each class generated, to ensure the association, TR7 is applied. 3.1 If the element is a predefined class, then generate it to attribute. The classes generated are associated using pre order deep first search traversal. 3.2 For every class association created, generate the multiplicity of a class using TR8.

Table 2 shows some of the generated transformation based on Table 1. As an example, PartyInfo element is used to illustrate the transformation. As PartyInfo is an element, it is

transformed to Class using TR1. The attribute of PartyInfo is PartyName, is transformed to partyNameAttr using TR4. The association generated between PartyInfo associate to element inside of it, named PartyId. Attribute of XML is transformed to PartyNameAttr using TR4. When the element inside the PartyId is a predefined element, then PartyId transform into PartyIdAttr using TR1. Finally the occurrences transformation of elements, which is PartyInfo to PartyId is associated with multiplicity of one-to-many using TR8. An attribute will also have multiplicity in the class which is zero-to-infinity using TR9.

Table 2. Some Generated Transformation

#	Transformation	ElementXML	UMLElement	#TR
1	ElementToClass	PartyInfo	PartyInfo	TR1
2	AttributeGroup ToClass	grp.pkg	grouppkg	TR2
3	SimpleTypeToClass	Service	type>> Service>>	TR5
4	ElementToAttr	PartyId	PartyIdAttr	TR1
5	AtToAttr	partyName	partyNameAttr	TR4
6	Association	PartyInfo, PartyId	PartyInfo associate to PartyId	TR7
7	ElementsOfOccurences	PartyInfo, PartyId	Multiplicity {1.. * }	TR8
8	trOccurencesOfAt	Application CertificateRef	Multiplicity {0.. * }	TR9

Transformation of Table 2 is generated in Figure 2. Figures 2 and 3 shows applied transformation rules in generating a class diagram from the case study. Based on these figures, it clearly indicates that the formalization of transformation rules helps in generating the class diagram based on XML schema.

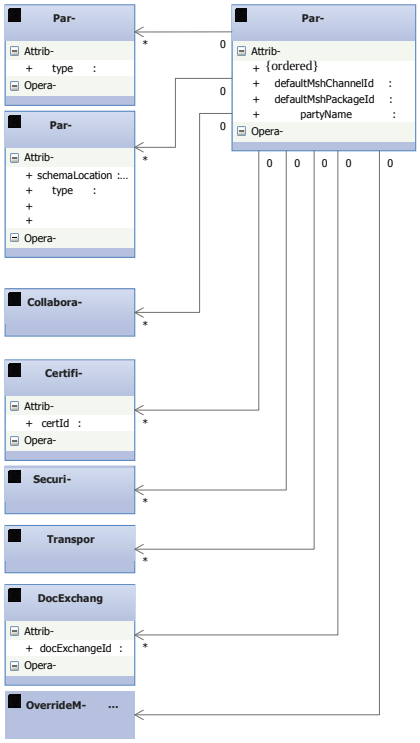

Figure 2. Part of Class Diagram generated as fromTable 2

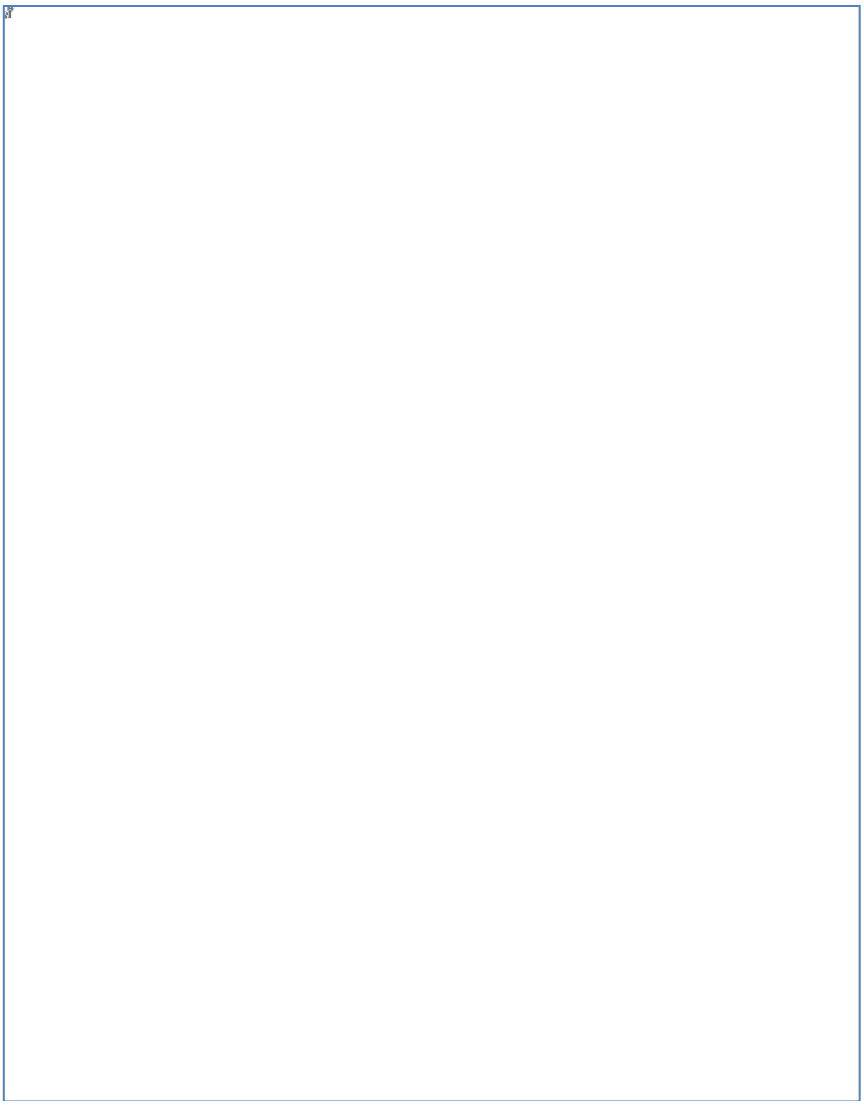

Figure 3. A part of ebXml Class Diagram (CollaborationProtocolAggrement Class)

6. Conclusion

Reversing XML Schema to UML Class Diagram is a tedious and time consuming task. The conceptual diagrams that are used as a base document for communication between user and developer can be generated easily by using this formalization transformation method. The complete class diagram generated using more XML schema constructs is better developed. The formal transformation method is complete as ebXml case study has been tested.

References

[1] C. Haitao, "A Survey to Conceptual Modeling for XML", Proc. 2010 3rd Int. Conf. Sci. Inf. Technol., vol. 8, **(2010)**, pp. 473–477.
[2] M. Necasky, "Reverse Engineering of XML Schemas to Conceptual Diagrams", Proceedins 6th Asia PAcific Conference on Conceptual Modelling, vol. 96, **(2009)** January, pp. 117–128.

[3] Y. Weidong, G. Ning and S. Baile, "Reverse Engineering XML", Proc. First Int. Multi-Symposiums Comput. Comput. Sci., **(2006)**.
[4] F. D. Salim, R. Price, M. Indrawan and S. Krishnaswamy, "Graphical Representation of XML Schema," in APWeb 2004, Springer Verlag Heidelberg, **(2004)**, pp. 234–245.
[5] T. Kudrass and T. Krumbein, "Rule-Based Generation of XML DTDs from UML Class Diagrams", Advances in Databases and Information Systems(ADBIS) 2003, Springer Verlag Berlin Heidelberg, **(2003)**, pp. 339–354.
[6] E. Dominquez, J. Lloret, B. Pérez, A. Rodriquez, A. L.Rubio and M. A.Zapata, "A survey of UML models to XML schemas transformations", WISE 2007, Springer-Verlag Berlin Heidelberg, **(2007)**, pp. 184–195.
[7] M. Necasky, "XSEM - A Conceptual Model for XML", Proceedings of the 4th Asia-Pacific conference on Conceptual modelling, vol. 67, **(2007)**.
[8] M. R. Jensen, T. H. Møller and T. B. Pedersen, "Converting XML Data To UML Diagrams For Conceptual Data Integration", Data Knowl. Eng., vol. 44, no. 3, **(2003)**, pp. 323–346.
[9] S. Mello and C. A. Heuser, "A Rule-Based Conversion of a DTD to a Conceptual Schema", ER 2001, Springer Verlag Berlin Heidelb., **(2001)**, pp. 133–148.
[10] J. Fong, S. K. Cheung and H. Shiu, "The XML Tree Model – toward an XML conceptual schema reversed from XML Schema Definition", Data Knowl. Eng., vol. 64, no. 3, **(2008)** March, pp. 624–661.
[11] F. D. Salim, "A UML Representation of XML Schemas", Monash University, **(2003)**.
[12] R. Al-Kamha, "Conceptual Xml For Systems Analysis," Brigham Young University, **(2007)**.
[13] G. Booch, M. Christerson, M. Fuchs and J. Koinstinen, "UML for XML schema Mapping Specification", **(1999)**.
[14] R. Al-Kamha, D. W. Embley and S. W. Liddle, "Foundational Data Modeling and Schema Transformations for XML Data Engineering", Springer-Verlag Berlin Heidelb., **(2008)**, pp. 25–36.
[15] M. Mani, D. Lee and R. R. Muntz, "Semantic Data Modeling Using XML Schemas", Springer-Verlag Berlin Heidelb. 2001, **(2001)**, pp. 149–163.
[16] V. Belén and M. Esperanza, "Extending UML to represent XML Schemas", CAiSE Short Paper Proceedings, vol. 74, **(2003)**.

Authors

Hannani bt Aman received her master's degree in Computer Science in 2005 from University of Putra Malaysia. Currently she is a Ph.D student at Department of Software Engineering, University Tun Hussein Onn Malaysia(UTHM). Her areas involve Reverse Engineering in Software Engineering.

Rosziati bt Ibrahim received her Ph.D degree in Information Technology (Software Specification) in March 2000 from Queensland University of Technology (QUT), Brisbane, Australia. Currently, she works at Department of Software Engineering, University Tun Hussein Onn Malaysia(UTHM) as a Professor. She also is a Dean of Research and Development Center in Office for Research, Innovation, Commercialization and Consultancy Management (ORICC), UTHM. Her areas involve software specification and image processing.