UML Documentation Support for XML Schema

Flora Dilys Salim¹, Rosanne Price², Shonali Krishnaswamy¹, Maria Indrawan¹

¹*School of Computer Science and Software Engineering* ²*School of Business Systems*
Monash University, Victoria, Australia
floradilys@yahoo.com.au,
[Maria.Indrawan|Shonali.Krishnaswamy|Rosanne.Price]@infotech.monash.edu.au

Abstract

With the proliferation of XML as the lingua franca of internet information exchange, engineering XML documents and maintaining their databases becomes a major challenge. In this context, the use of XML Schema to constrain and validate XML document content is becoming a standard part of web information production. However, adoption of XML Schema poses new problems; specifically, the difficulty in understanding even a relatively small schema due to its textual nature and XML syntax. This problem can be addressed by use of a graphical modelling language such as UML to document XML Schema and conforming XML document structure. This paper discusses the issues involved in representing XML Schema using UML and presents a transformation approach which is both comprehensive and thorough.

Keywords: XML Schema, UML, transformation, graphical modelling, conceptual, reverse engineering.

1. Introduction

XML Schema is increasingly replacing DTD as the standard specification of XML document structure [Duck01]. XML Schema, compared to DTD, provides a more powerful mechanism for describing and constraining the content of the XML instance documents [Bird00]. XML Schema serves as validation support and design documentation for a set of XML instance documents [List02]. In particular, XML Schema supplies the following additional features: XML syntax, rich datatyping, support for namespaces, constraints, type derivations, and groupings [Bird00]. However, although XML Schema is powerful and flexible, it has a longer learning curve than DTD [Duck01, Lee00].

As the number and size of XML documents developed based on XML Schema grow, maintenance of these XML Schema documents is necessary. Maintenance requires understanding and, given that XML Schema is textual and is described using XML syntax, this becomes tedious. In general, the larger the XML Schema document, the more difficult it is to understand and maintain [Bird00]. Graphical and higher level conceptual languages can be used to clarify XML Schema structure. In particular, graphical modelling languages are a more effective means of detailing and communicating data requirements [Bird00]. As a result, research in the area of graphical and higher-level modelling of XML Schema is becoming increasingly relevant.

Significant contributions in the area of forward engineering from existing graphical and higher-level models to various XML schemas include those in [Bird00, Carls01a, Carls01b, Conrad00, Hucka99, Rout02a, Skogan99]. The design methodologies for developing XML Schema, as presented in [Carls01a, Carls01b, Conrad00, Hucka99, Rout02a, Skogan99], have been developed to solve the issue of XML Schema maintenance. However, forward engineering's design methods are only applicable for new XML Schema that will be developed. They do not facilitate understanding and maintaining schema that have already been developed without a formal design process. Therefore, there is a need for graphically visualising existing XML Schema documents.

In the area of reverse engineering XML schemas (such as DTD, XML Schema) to graphical models, comparatively less work has been done [Booch99b, Mani01, Mello01, Rout02b, Widj02]. In addition to existing research, there are also commercial tools available for both forward and reverse engineering schemas for XML [Ratio00, Sun].

The limitations of existing work in the area of reverse engineering XML schemas are that:

- they deal with the transformation of XML Schema predecessors (such as SOX, DTD), as in [Booch99b, Mello01],
- they do not use UML (the de-facto standard in Object-Oriented conceptual modelling) as the higher-level model, as in [Mani01, Mello01, Widj02],
- the transformation rules are not comprehensive or are missing completely, as in [Booch99b, Mani01, Rout02b, Widj02],
- the resulting diagrams are not simple and require special training as new UML constructs are created to represent XML constructs, as in [Booch99b, Rout02b],
- they depend on user defined rules and so are not automatic (i.e. existing automation tools), as in [Ratio00, Sun].

The most closely related work, involving XML Schema as a source of transformation and UML as a result of transformation is [Rout02b]. The transformation work is oriented towards reverse engineering; therefore, a new UML construct is created to represent each XML Schema construct. The method used relies on a one-to-one transformation of XML Schema to UML constructs, which has the advantage that most of the information in the original XML Schema document is preserved. However, the disadvantage of this method is that it employs an extensive collection of new constructs that must be learned and complicate the resulting UML diagram. In addition, this research does not cover transformation of all thirteen XML Schema building blocks.

The research presented in this paper aims to: (1) assist understanding and documentation of XML Schema by converting them to graphical form, i.e. UML, and (2) to adopt a transformation approach that does not require additional training beyond standard UML.

In terms of the research goals of the transformation work described here, having clarity in the resulting UML diagram takes precedence over complete reverse engineering (allowing generation of the original XML Schema document from the UML diagram). Reverse engineering an XML Schema requires that everything in the schema, including less important details and processing instructions, be transformed to UML constructs and captured within the resulting diagram. Including less important syntactic details of XML Schema complicates the resulting UML diagrams. On the other hand, total reverse engineering is only needed when the XML Schema is not preserved, which is not the case considered here. The XML Schema is still maintained because XML instance documents need to refer to an XML Schema document to be validated.

Therefore, in this paper, maintaining clarity is our focus, rather than preserving completeness in the UML diagram resulting from the transformation process.

Using existing constructs instead of creating new constructs or datatypes have several advantages. Firstly, it simplifies the resultant UML diagram. Secondly, the transformation result can be understood by anyone who knows UML. There is no need to learn special symbols or additional stereotypes. This contrasts with the transformation approach of Routledge et al. [Rout02b]. As existing UML constructs are used to represent the semantics of each given XML Schema construct in our transformation approach, this implies the use of a pattern-based approach as described in [Price00]. In resolving which UML modeling constructs should be used to represent a set of given XML Schema constructs, several alternative UML patterns (UML diagram fragments) are provided with guidelines as to the context in which each alternative should be used. Existing UML constructs are used to specify a UML pattern [Price00], as illustrated in the transformation examples in Section 4. These examples illustrate representative constructs of XML Schema that are commonly found in a XML Schema document. In this paper, the use of C++ to describe individual constructs is for illustrative purposes. In fact, any commonly understood language, whether programming, natural, or logic-based could be used instead, as specified by [Rumb99]. As stated previously, this approach has the advantage over stereotypes of being immediately understandable and thus not requiring training. Furthermore, the readers of UML are not distracted by an array of special symbols.

This paper is structured as follows. Section 2 discusses transformation issues that occur because of the differences between XML Schema and UML. Section 3 presents the overview of transformation rules. Section 4 demonstrates the application of the transformation rules by example. Section 5 concludes the paper and discusses future work.

2. XML Schema vs. UML

There are several issues that arise in the process of defining the transformation rules from XML Schema to UML. One issue to be addressed is the difference in the conceptual level between XML Schema and UML. UML is primarily architectural and conceptual. In particular, the class diagram (the most relevant diagram to modelling XML Schema) is intended for conceptual modelling at the design development stage [Booch99a]. XML Schema, on the other hand, is more programmatic and implementation oriented. XML

Schema includes programmatic and syntactic details that are not generally captured in UML. For example, the scope of each XML Schema component in a given schema is limited structurally, i.e. statically scoped. This information about scope in XML Schema is a programmatic detail that is generally not part of conceptual models. XML Schema contains details on syntax, editing, and processing instructions for interpreters, parsers, and validation tools of XML and XML Schema. Examples of such details are: syntax in data types (e.g. length, pattern / regular expression) and means of text substitution in XML Schema.

The next issue is that of ordering. In UML, attributes of a class are unordered. On the contrary, in XML Schema, ordering information is very important. In general, XML Schema can specify whether a set of elements is ordered or unordered, i.e. specific XML Schema constructs are used to indicate whether a set of elements is ordered or not. To preserve XML Schema ordering semantics, resultant UML diagrams should be able to display ordering and non-ordering characteristics for a set of XML elements in the XML Schema.

Another issue is that XML Schema is textual, whereas UML is graphical. Incorporating excessive textual information in a graphical model causes an overloaded and complex diagram. Therefore, the importance of a text in XML Schema from a semantic point of view should be considered before incorporation in the resulting UML diagram.

Finally, the different treatment of derivation in XML Schema and UML must be considered. Class derivation in UML only allows attributes to be added, not removed, from an existing class definition. This is modelled using generalization in UML. In contrast, derivation in XML Schema allows attribute addition (by extension) or elimination (by restriction). Therefore, although generalization in UML can be

used to represent XML extension, another method is required to represent XML restriction in UML.

3. Overview of the transformation rules

The XML Schema constructs transformed comprise the thirteen XML building blocks defined in the XML Schema Recommendation [W3C], which are categorized into three types of components:

Primary components that consist of *element declarations*, *attribute declarations*, *simple type definitions*, and *complex type definitions*. Every schema has at least one of these components.

Secondary components that consist of *model group definitions*, *attribute group definitions*, *identity constraint definitions*, and *notation declarations*. These components are optional in XML Schema.

Helper components that consist of *annotations*, *model groups*, *particles*, *wildcards*, and *attribute uses*. These components are used in primary and secondary components.

In UML, a construct should not be anonymous; however, in XML Schema, it is possible to find anonymous constructs. In such cases, artificial names need to be used for naming UML constructs that are transformed from anonymous XML Schema constructs. For consistency, in generating artificial names, a set of rules should be followed. The naming rules employed in this paper follow the rules described in [Sal03].

Table 1 summarizes the rules of transforming XML Schema constructs to UML. Refer to [Sal03] for full details and examples of transformation of each XML Schema construct. The working definition of each XML Schema construct in this research is based on [Duck01, W3C], whereas the definition of each UML construct is based on [Booch99a, Rumb99b].

Table 1. Summary of Transformation Rules of XML Schema Constructs

XML Schema Constructs		UML Constructs
Namespace		Package
Element declaration	Global element	Class and {root} property (if necessary)
	Local element	Attribute
	Occurrence constraints (minOccurs, maxOccurs)	Multiplicity of attribute
	fixed constraint	{frozen} constraint
	default constraint	{changeable} constraint
	Nilable	Individual constraint {can be NULL}
Attribute declaration	Global attribute	Class
	Local attribute	Attribute

	fixed constraint		{frozen} constraint
	default constraint		{changeable} constraint
	optional constraint		Multiplicity [0..1]
	required constraint		Multiplicity [1..1]
Simple type definition	Global simple type		Class and <<type>> stereotype (if necessary)
	Local simple type		Attribute
	restriction		Attribute
		Child constructs	Individual constraints of the attribute
		Base type	Type of the attribute
		enumeration	C++ enum datatype
	list datatype		C++ list datatype
	union datatype		C++ union datatype
Complex type definition	sequence compositor		C++ struct datatype, or {ordered} running constraint in attribute compartment
	choice compositor		C++ union datatype
	all compositor		Transform each child element using the rules for element transformation.
	Simple content		Follows simple type transformation rules. Additional XML Schema attributes will be transformed to UML attributes.
	Empty content		Follows the rules for transforming elements of complex type.
	Mixed content		A class with optional artificial string attributes and either a running constraint or a UML note specifying the UML class must contain text or string
	Derived by extension		Subclass with generalization relationship to the class that represents the base type of the XML Schema derived type
	Derived by restriction		A new class
Substitution group			C++ union, or classes associated with {xor} constraint
Model group			Not transformed. Text substitution rule is used if necessary.
Attribute group			Not transformed. Text substitution rule is used if necessary.
Identity constraint definition			Qualifier on an association between the qualified class and the target class
	key element		Qualified class
	keyref element		Target class
Notation declaration			Note
Ancillary XML Schema construct			Note

To maintain conceptual clarity in the UML transformation, programmatic and textual details in XML schema such as those described in Section 2 are not routinely transformed. However, when necessary, such details can be included in UML notes. Now, we apply the rules in Table 1 to specific application examples to illustrate representative constructs of XML Schema.

4. Transformation Examples

This section displays the application of transformation rules in converting elements and complex types, the primary components of XML Schema. The child constructs of complex types that are discussed in this section are *sequence*, *choice*, and *extension*. These constructs are used for illustrative purposes since they are widely used in XML Schema.

In the following example (Code Listing 1), *Publication* is a global complex type. The *Publication* global complex type contains a sequence compositor. A sequence is a compositor that constrains the elements in the instance documents to appear in the order in which they are declared in the schema. There are several alternative rules concerning the transformation of sequence compositors. The application of each rule depends on the complexity of the type definition, as illustrated below. In addition, these alternative rules are used to illustrate how the issue of preserving XML Schema ordering (discussed in Section 2) is addressed in our transformation approach.

As described in [Rumb99], any programming language can be used to describe attribute datatypes in UML. Therefore, the first alternative for transforming sequence compositor is that a sequence compositor will become a structure (i.e. a C++ struct datatype) with an artificial name and all the direct child elements of sequence become the member variables of the structure. This alternative is applicable only to those sequence compositors whose child elements use predefined types without occurrence constraints and without nested compositors (because each of the C++ struct member types is required, occurs only once, and cannot be nested). C++ struct is compliant with the XML Schema sequence construct since all the member variables of the C++ struct are ordered. C++ is used throughout to describe UML attribute datatypes because it is one of the most widely recognised and established programming languages. Using C++ is different than using stereotypes because there is no need to learn constructs specific to the transformation approach. Any language can be used to

specify datatypes in UML, as specified in [Rumb99]. We choose C++ as an example.

The second transformation alternative for transforming sequence compositors is to apply a UML running constraint. A running constraint specified as the text string {ordered} can be used to represent each sequence compositor in the UML. UML attributes are not ordered. Consequently, the running constraint has to be applied to enforce ordering of attributes. The {ordered} constraint is a predefined UML constraint that is only used as a constraint in an association. The application of the {ordered} constraint to an association end means that all of the instances of the class on the {ordered} side will be an ordered set of objects. However, an {ordered} constraint in the current context, which is used in this section and is a construct transformed from a sequence compositor, means that all the subsequent attributes will be ordered in the class instances. Therefore, the application of {ordered} constraint in this section represents a change in its semantics, as the {ordered} constraint is applied on a group of attributes in each class instance, not on the class instances as a whole. However, as a running constraint can have any arbitrary textual string, the usage of {ordered} is valid in this context and represents a standard UML running constraint.

An XML Schema attribute declaration cannot be a part of a compositor, as attributes cannot be structured. Therefore, a UML attribute transformed from an XML attribute will always be put before the {ordered} running constraint in the attribute compartment, as a running constraint applies the rules to the subsequent attributes of the UML class.

If the type is more complex, for example if the occurrence of the sequence compositor is specified, or if the sequence compositor is nested within another compositor; the third alternative of transforming sequence must be employed. A new UML class is used to represent the XML sequence compositor with XML child elements represented as attributes of the new UML class and a UML composition relationship to the owner class. The {ordered} UML running constraint described in the second alternative is used to preserve the order of XML child elements represented as UML attributes.

```
<xs:complexType name = "Publication">
  <xs:sequence>
    <xs:element name = "Title"
      type = "xs:string" />
    <xs:element name = "Author"
      maxOccurs = "unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Surname"
```

```

        type = "xs:string" />
        <xs:element name="FirstName"
            type = "xs:string"/>
    </xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name = "YearPublished"
    type = "xs:gYear" />
</xs:sequence>
</xs:complexType>

```

Figure 1: Publication Complex Type

Based on the above rules for transforming complex types with sequence compositors, the `Publication` global complex type is transformed to a class named `Publication` with an `{ordered}` constraint. The transformation of the sequence compositor in the `Publication` element follows the second alternative rule, since `Publication` element contains nested compositors.

The `Title` local element is transformed to an attribute named `Title` that belongs to the `Publication` class.

The `Author` local element is transformed to an attribute named `Author` with a `struct` datatype. That is because the `Surname` element and the `Firstname` element within `Author` element have no occurrence constraints and are using predefined types. Since the `Author` XML local element has an occurrence constraint `maxOccurs = "unbounded"`, the `Author` UML attribute has multiplicity `[1..*]`.

The `YearPublished` local element is transformed to an attribute named `YearPublished` that belongs to the `Publication` class. As an existing XML datatype is preserved in our transformation approach, the `YearPublished` attribute in the `Publication` class is of `gYear` XML Schema datatype as defined in the Code Listing 1.

The resulting UML diagram transformed from Figure 1 is displayed in Figure 2.

Figure 2: Publication Class

A complex type can be derived by applying extension or restriction on an existing type. Extension of an existing complex type produces a complex type with *complex content*. Complex type derivation by extension allows groups of elements or attributes to be added to an existing complex type. On the other hand,

placing restrictions on a complex type allows child elements or attributes to be removed and allows the application of additional constraints to existing child elements or attributes. This is where the issue of the different treatment of derivation between XML Schema and UML discussed in Section 2 arises in practice. To address this issue, the UML inheritance concept is used to transform XML Schema complex types derived by extension. On the other hand, XML Schema complex types that are derived by restriction will be transformed to new UML classes not directly related to the base complex type since UML does not have the concept of class derivation through restriction. Continuing with the publication example used earlier, the transformation of an XML complex type derived by extension is illustrated as follows.

For instance, books and conference papers will have a title, author, and year of publication; however, books will have publisher information in addition to the basic information, whereas conference papers will have conference title and conference dates. The following Figure 3 is the XML Schema code fragment containing complex types that are derived by extension.

Inheritance is the fundamental UML concept used to transform an XML Schema complex type that is derived by extension. The derived complex type by extension will be transformed to a child class of the parent class, which is transformed from the base type. The child class will have a generalization relationship to the parent class.

In the resulting UML diagram (Figure 4), `Book` and `ConferencePaper` are the child classes of `Publication` class, since the base type of the local complex type within the `Book` element and the `ConferencePaper` element is the `Publication` complex type. The new elements added specifically to the `Book` complex type and the `ConferencePaper` complex type are converted to attributes of the `Book` class and the `ConferencePaper` class respectively. In both classes, sequence compositors are transformed to `{ordered}` running constraint.

```

<xs:element name = "ConferencePaper">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension
        base="Publication">
        <xs:sequence>
          <xs:element name =
            "ConferenceTitle" type =
            "xs:string"/>
          <xs:element name =
            "ConferencePlace" type =
            "xs:string"/>
          <xs:element name =

```

```

        "StartDate" type =
        "xs:date" />
        <xs:element name
        ="EndDate"
        type = "xs:date" />
        </xs:sequence>
        </xs:extension>
        </xs:complexType>
    </xs:element>
    <xs:element name = "Book">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base= "Publication">
                    <xs:sequence>
                        <xs:element name =
                        "PublisherName" type =
                        "xs:string" />
                        <xs:element name =
                        "PlacePublished" type =
                        "xs:string" />
                    </xs:sequence>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>

```

Figure 3: Book Element and ConferencePaper Element

Figure 4: Book Class and ConferencePaper Class

A complex type can also be derived by adding additional attributes to an existing XML Schema complex type. In Figure 5, OnlinePublication is an extension of Publication with an additional attribute named URL.

```

<xs:element name = "OnlinePublication">
    <xs:complexType>
        <xs:complexContent>

```

```

        <xs:extension base="Publication">
            <xs:attribute name = "URL"
            type = "xs:anyURI" />
        </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:element>

```

Figure 5: OnlinePublication Element

The OnlinePublication complex type is transformed to OnlinePublication class that is an extension of Publication class. The additional XML attribute is converted to a UML attribute of the OnlinePublication class (Figure 6).

Figure 6: OnlinePublication Class

In Figure 7, the XML Schema code fragment shows that Literature Review is a collection of books, conference papers, and online publications. The LiteratureReview element has a choice compositor as the child construct of the local complex type. A choice is a compositor that enforces the constraint that only one immediate child element can appear in the instance document, unless the maximum occurrence is specified to be more than one [Duck01].

```

<xs:element name = "Bibliography">
    <xs:complexType>
        <xs:choice maxOccurs = "unbounded">
            <xs:element ref = "Book" />
            <xs:element ref =
            "ConferencePaper" />
            <xs:element ref =
            "OnlinePublication" />
        </xs:choice>
    </xs:complexType>
</xs:element>

```

Figure 7: LiteratureReview Element

The choice compositor is transformed to union datatype (as in C++); likewise, every choice

compositor is transformed to an artificial UML attribute of union datatype (as in C++). Each child element of the choice compositor is transformed to a C++ variable that is initialized with the child element's type as its type. All these variables are the members of the C++ union.

The resulting UML diagram (Figure 8) displays the LiteratureReview class (transformed from the LiteratureReview element) that has a union that represents the choice compositor and is comprised of class-valued attributes of the Book class, ConferencePaper class, and OnlinePublication class.

Bibliography
choice1[1..*]: union{Book_attr: Book; ConferencePaper_attr: ConferencePaper; OnlinePublication_attr: OnlinePublication}

Figure 8: Bibliography Class

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name = "Publication">
    <xs:sequence>
      <xs:element name = "Title" type = "xs:string" />
      <xs:element name = "Author" maxOccurs = "unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name = "Surname" type = "xs:string" />
            <xs:element name = "FirstName" type = "xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name = "YearPublished" type = "xs:gYear" />
    </xs:sequence>
  </xs:complexType>
  <xs:element name = "ConferencePaper">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base= "Publication">
          <xs:sequence>
            <xs:element name = "ConferenceTitle" type = "xs:string"/>
            <xs:element name = "ConferencePlace" type = "xs:string"/>
            <xs:element name = "StartDate" type = "xs:date" />
            <xs:element name = "EndDate" type = "xs:date" />
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name = "Book">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base= "Publication">
          <xs:sequence>
            <xs:element name = "PublisherName" type = "xs:string" />
            <xs:element name = "PlacePublished" type = "xs:string" />
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name = "OnlinePublication">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base = Publication">
          <xs:attribute name = "URL" type = "xs:anyURI" />
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
</xs:schema>
```



```

</xs:complexType>
</xs:element>
<xs:element name = "Dependent">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name = "age" type = "xs:integer" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
<xs:element name = "Bibliography">
  <xs:complexType>
    <xs:choice maxOccurs = "unbounded">
      <xs:element ref = "Book" />
      <xs:element ref = "ConferencePaper" />
      <xs:element ref = "OnlinePublication" />
    </xs:choice>
  </xs:complexType>
</xs:element>
</xs:schema>

```

Figure 9: The Whole XML Schema Document

Figure 10: The Whole UML Diagram

The entire XML Schema document that includes the previous code listings is presented in Figure 9. The resulting UML diagram of the transformation of the whole document is displayed in Figure 10. The association between instances of Bibliography and Publication can be represented in the diagram either by an association line between the two classes concerned or by the use of class-valued attributes (i.e. an attribute

Therefore, to address this issue, we have aimed to, firstly, assist understanding and documentation of

whose domain is a class) [Rumb99]. We chose to use class-valued attributes for illustrative purposes.

5. Conclusion and Future Work

A major challenge in using XML Schema is the difficulty in reading and understanding even a relatively small XML Schema because of its textual nature and its XML syntax.

XML Schema by converting them to graphical form, i.e. UML, and, secondly, adopt a transformation

approach that does not require additional training beyond standard UML.

The primary contributions of this research are as follows:

The transformation work includes all thirteen building blocks of the XML Schema.

The UML constructs used to form a UML pattern are predefined in UML. No new UML constructs are created. This makes the resulting UML diagram simpler and generally understandable (i.e. not requiring special training beyond that required for understanding standard UML).

There are several future directions proposed following from the work in this paper. One could be case studies or user testing to evaluate the degree of improved understanding using the graphical UML transformation versus the original textual XML schema. Other future work includes implementation of an automated transformation tool based on the transformation rules proposed in this research. Another direction would be to develop rules for mapping XML Schema datatypes to other datatypes in one or more programming languages. For example, one could map XML Schema datatypes to equivalent Java or C++ datatypes. We also intend to investigate the feasibility of extending this approach to support reverse engineering.

6. References

- [Bird00] L. Bird, A. Goodchild, and T. Halpin. "Object Role Modeling and XML-Schema". In *19th International Conference on Conceptual Modeling (ER'2000)*, Salt Lake City, USA, October 2000, Springer, pp. 309-322.
- [Booch99a] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*, Addison-Wesley, Reading, M.A., 1999.
- [Booch99b] G. Booch, M. Christerson, M. Fuchs, and J. Koistinen. *UML for XML Schema Mapping Specification*. Rational Software and CommerceOne, 1999. Available at: http://www.rational.com/media/uml/resources/media/uml_xmlschema33.pdf
- [Carls01a] D. Carlson. *Modeling XML Applications with UML: Practical e-Business Applications*, Addison-Wesley, Boston, 2001.
- [Carls01b] D. Carlson. *Integrating XML and non-XML data via UML*, 2001. Available at: <http://xmlmodeling.com/articles/IntegratingXMLData.pdf>
- [Conrad00] R. Conrad, D. Scheffner, and J.C. Freytag. "XML Conceptual Modeling using UML". In *19th International Conference on Conceptual Modeling (ER'2000)*, Salt Lake City, USA, October 2000, Springer, pp. 558-571.
- [Duck01] J. Duckett, O. Griffin, S. Mohr, F. Norton, I. Stokes-Rees, K. Williams, K. Cagle, N. Ozu, J. Tennison. *Professional XML Schemas*, Wrox Press Ltd, Birmingham, U.K., 2001.
- [Hucka99] M. Hucka. "SCHUCS: An UML-Based Approach for Describing Data Representations Intended for XML Encoding." In *Proceedings of the 2nd International Conference on The Unified Modeling Language (UML'99)*, October 1999.
- [Lee00] D. Lee and W. Chu. "Comparative Analysis of Six XML Schema Languages". In *SIGMOD Record (ACM Special Interest Group on Management of Data)*, vol. 29, no. 3, 2000, ACM, pp. 76-87.
- [List02] E. v. d. List. *XML Schema*. O'Reilly, CA, 2002.
- [Mani01] M. Mani, D. Lee, and R.R. Muntz. "Semantic Data Modeling using XML Schemas". In *Proceedings. 20th International Conference on Conceptual Modeling (ER 2001)*, Yokohama, Japan, November 2001, Springer, pp. 149-163.
- [Mello01] R.d.S. Mello and C.A. Heuser. "A Rule-Based Conversion of a DTD to a Conceptual Schema". In *Proceedings. 20th International Conference on Conceptual Modeling (ER 2001)*, Yokohama, Japan, November 2001, Springer, pp. 133-148.
- [Price00] Price, R., N. Tryfona, and C.S. Jensen. *Extended Spatiotemporal UML: Motivations, Requirements, and Constructs*, Journal of Database Management, Vol. 11, Issue 4, Oct-Dec 2000, pp. 14-27.
- [Ratio00] Rational Software Corporation. "Migrating from XML DTD to XML Schema using UML". *Rational Software White Paper*, 2000. Available at: <http://www.rational.com/media/whitepapers/TP189draft.pdf>
- [Rout02a] N. Routledge, L. Bird, and A. Goodchild. "UML and XML Schema". In *Proc. Thirteenth Australasian Database Conference (ADC 2002)*, Melbourne, Australia, ACS.
- [Rout02b] N. Routledge, A. Goodchild, and L. Bird. *XML Schema Profile Definition*. Honours thesis extract, DSTC, Queensland, Australia, 2002. Available at: <http://titanium.dstc.edu.au/papers/xml-schema-profile.pdf>
- [Rumb99] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*, Addison Wesley, 1999.
- [Sal03] F.D. Salim. *UML Representation of XML Schemas*. Honours thesis, Monash University, Australia, 2003.
- [Skogan99] D. Skogan. "UML as a Schema Language for XML based Data Interchange". In *Proceedings of the 2nd International Conference on The Unified Modeling Language (UML'99)*, October 1999.
- [Sun] Sun Corporation. *Java™ Architecture for XML Binding (JAXB)*. Available at: <http://java.sun.com/xml/jaxb/index.html>
- [W3C] W3C. *XML Schema*. Available at: <http://www.w3.org/XML/Schema>.
- [Widj02] N. D. Widjaya, D. Taniar, J.W. Rahayu, and E. Pardede. "Association Relationship Transformation of XML Schemas to Object-Relational Database", In *Proceedings of IIWAS2002*, Bandung, Indonesia, 2002.