1999-05-14: Submitted to UML'99. http://www.cs.colostate.edu/UML99/

UML as a Schema Language for XML based Data Interchange

David Skogan [1,2]

[1] Department of Informatics, University of Oslo, P.O. Box 1080 Blindern, N-0316 OSLO, NORWAY
davids@ifi.uio.no
http://www.ifi.uio/~davids
[2] SINTEF Telecom and Informatics, P.O. Box 124 Blindern, N-0314 OSLO, NORWAY
david.skogan@informatics.sintef.no

Abstract. *The Unified Modeling Language (UML) is here used as a schema language to define data interchange formats based on the Extensible Markup Language (XML). UML is a powerful and flexible modeling language and XML is expected to be the next generation data interchange format for the Web. UML's declarative expressiveness and intuitive visual form overcome XML's current declarative powers. The use of UML as a schema language combined with XML as a data representation language addresses both semantic and syntactic interoperability. A mapping from UML to XML is defined and two prototype implementations are presented. The mapping is inspired by Object Management Group's (OMG) XML Metadata Interchange specification (XMI). It is developed as a part of the ongoing standardization work creating an international standard for geographic information (ISO 15046). It is generic and may easily be adapted to other application domains.*

Keywords: UML, XML, data interchange, schema language, interoperability

1 Introduction

The Unified Modeling Language (UML) [1] is a powerful and flexible modeling language, with capabilities of describing static, dynamic and environmental aspects of software systems. This paper describes how to use UML as a schema language to define system independent data interchange formats based on the Extensible Markup Language (XML). A mapping from UML to XML is defined and two different prototype implementations are presented. The mapping and prototypes are discussed and the implications on the use of UML and XML are outlined. The conclusion is that UML with certain restrictions is a suitable schema language for defining the structure and semantics of data, and that the resulting XML datasets are adequate for data interchange and long term storage when the primary objective is to achieve interoperability.

The mapping and its implementations are created based on the experiences gained from using UML and XML in the DISGIS European ESPRIT project no. 22.084 [2] and in ISO Technical Committee no. 211 Geographic Information (ISO/TC 211) [3]. The mapping is defined in detail in part 18 of the ISO 15046 family of standards, ISO CD 15046-18 Encoding [4].

The main idea is to use UML class and package diagrams and map the various concepts to XML elements. The mapping described is inspired by the Object Management Group's (OMG) XML Metadata Interchange (XMI) [5], but is simplified in the sense that only the static and organizational parts of UML's metamodel are mapped (package, class, attribute, and association). The CORBA data types have been replaced by a smaller set of more conceptual primitive data types, and an optional tagging mechanism has been defined that may reduce the number of characters in the element names.

The paper is organized as follows: Section 2 gives an overview of important interoperability aspects and relevant concepts of UML and XML. Section 3 defines the mapping. Section 4 presents ways of implementing the mapping, section 5 discusses the approach and section 6 concludes.

2 Background

This section gives an overview of the data interchange and interoperability issues, important UML concepts, and XML concepts and limitations.

2.1 Data Interchange and Interoperability

Data interchange is an essential element in achieving interoperability between heterogeneous systems. A lot of effort is being put into the field of interoperability research [6-8]. Two fundamental interoperability issues are addressed here. The first issue, semantic interoperability, is to define and agree on the semantics of the content and logical structures of data. UML with additional documentation is an intuitive and powerful choice for describing the semantics and logical structures of data as well as behavioral aspects. The term application schema is used to denote such an UML model. The second issue, syntactic interoperability, is to define a system and platform independent data structure that can represent data corresponding to the application schema. XML has been chosen as a data interchange format because of its simplicity, Web conformance, and extensively tool support.

The objective of ISO/TC 211 is to enable interoperability between heterogeneous geographic information systems. The focus is not to create a complete system, but to create a framework of guidelines, rules and model libraries that allow application domains to create models (application schemas) that formally describe the structure and semantics of their data. The final international standard (ISO 15046) will consist of many parts. Part 18 Encoding defines the mapping from an application schema written in UML to a data interchange format specified in XML. The application schemas developed based on this family of standards will be implemented in many ways using different programming languages, databases and software development tools. To ensure interoperability between different implementations it is of vital importance to define and agree on how to use of UML and how to interpret application schemas.

2.2 UML

UML is a general-purpose visual modeling language that can be used to specify, visualize, construct and document a software system. It has been standardized by OMG [1] and contains static, dynamic, environmental, and organizational parts. UML models can be interchanged using OMG's XMI specification.

The UML notation is intuitively appealing and can be used on many levels of abstractions, but it suffers from lack of precise semantics [9]. This is somewhat intended and the UML Reference Manual [10] cautions developers to be aware of UML's semantic variation points and to define appropriate profiles if needed. Within a data interchange and standardization project it is essential that users interpret a model in the same way to achieve semantic interoperability. This becomes apparent when an UML model are mapped to different implementation technologies, which may influence or restrict the model in different ways.

From a data interchange purpose it is natural to focus on the static parts of UML. The static structure defines the kind of objects important to a system and to its implementation, as well as the relationships among the objects. Important concepts for capturing the static aspects of a system are package, class, attribute and association. A step towards achieving semantic interoperability is to specify exactly what are meant by these concepts. One way to do this is to define and agree on an abstract model for representation of data. The focus is to use UML as an object definition language, and model persistent objects and their structure. This is also called modeling a logical database schema in the UML User Guide [11]. To achieve syntactic interoperability there should be a canonical mapping from the abstract data representation model to the data interchange format.

2.3 Extensible Markup Language (XML)

The Extensible Markup Language (XML) [12] is a subset of SGML ISO 8879:1996. XML defines a class of data objects called XML documents. XML's hierarchical structure combined with its linking capabilities [13, 14] can encode a wide variety of information structures. The logical structure of an XML document comprises of properly nested XML elements. An XML element has a name, may have attributes and a content model. This corresponds to UML's class name, attributes and composition associations. An element is encoded syntactically with a start tag, which includes the element's attributes, the content and the end tag. The tag is made up of the element's name. Only three data types are supported for attributes, i.e. character data, special purpose XML tokenized types and enumerated types. XML has a global namespace and does not support inheritance. Recently a new namespace mechanism that allows several namespaces within an XML document has been published [15].

The declarative part of XML, which defines the XML elements, their attributes and how they are structured, is called a Document Type Declaration or DTD. The expressiveness of the DTD has been criticized and W3C's XML Schema Working Group is currently developing an XML schema language that will replace the current DTD syntax. The working group has recently published a note describing the requirements for an XML schema

language [16]. This is a response on different activities of enhancing the expressive powers of XML DTD [17, 18]. The requirements are divided into three groups: structural, data type and XML conformance requirements. The structural requirements include support for inheritance and constraints on structural constructs. The data type requirement calls for primitive data types, well-defined lexical representation, and support for user-defined data types.

Declaring XML elements with pre-defined XML attributes indicate their behavior for the XML processor. An XML element that can refer to another element within the XML document or to external resources using XML's linking capabilities are called a linking element. Linking elements point to a target resource through a Universal Resource Identificator (URI) reference. The special purpose attribute **xml:link** indicates that this is a linking element and the value of the attribute **href** is a URI pointing to another XML document. Here is an example of a XML element, which also is a linking element.

```
<Link xml:link="simple" href="http://ex.org/d.xml|i2">
        Click here for more information!
</Link>
```

3 The UML to XML Mapping

This section defines the mapping from UML to XML. The mapping is defined based on an abstract representation model that comprises of two models, the schema model and the instance model. The schema model corresponds to the subset of the UML metamodel that is essential for data representation, and the instance model contains a minimal model for structuring and representing data that is close to the representation form of XML. The two models (metamodels) are abstract in the sense that they are not dependent of any programming language, database or application schema and that they do not have to be implemented in order to realize encoding services. They are tools for reasoning about application schemas and data, for expressing examples and for defining the mapping.

Fig. 1 gives an overview of the mapping. It comprises of two categories of conversion rules. A conversion rule specifies how to convert instances of the input data structure into instances in the output data structure. The first category is the schema conversion rules, which define rules for how to convert instances of the schema model to XML element declarations in a DTD file. The second category is the instance conversion rules, which define rules for how to convert instances of the instance model into instances of the resulting data structure, the XML document. Notice that the schema conversion rules are one way only, because of the limitations of XML's DTD syntax. The instance conversion rules are of course two ways.

Before data can be interchanged the application schema should be transformed into the schema model. The schema conversion rules should be applied to generate a DTD file, and the DTD file together with the application schema encoded according to the XMI standard should be made available to the public. This enables the recipient to access the application schema to prepare his system and the XML processor to access the DTD to validate the XML document. The data interchange process involves transformation of appropriate application data to the instance model, application of the instance conversion rules to produce an XML dataset and to send or publicize the dataset. The recipient has to get the dataset, apply the inverse instance conversion rules, and map the data of the instance model into his/hers private database.

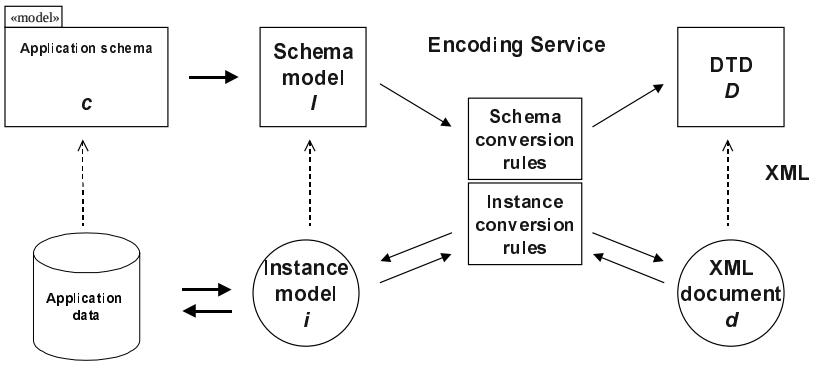

Fig. 1. An overview of the UML to XML mapping.

3.1 Schema Model

This section defines the schema model, and how to transform an application schema to the schema model. Fig. 2 shows the schema model. The main UML concepts that define the data structure are package, class and association. Packages group classes together and classes define their objects' possible state through attributes and associations. Classes stereotyped as interfaces do not contain any state and can therefore not be mapped. All attributes and associations are mapped to ensure that state is not lost. Since operations cannot be mapped all attributes and associations necessary to recreate the state of the dataset must be modeled in the application schema.

From an encoding point of view there is a need to differentiate between attributes of primitive types and attributes of model types because they require different representation. Attributes of primitive data types are represented as property objects (cProperty), and attributes of model types are represented as composition objects (cComposition). The primitive data types allowed in the application schemas are:

- Fundamental types for representing values: String, Integer, Real, Binary, Boolean, Date, Time, DateTime and DirectPosition[1].
- Collection data types, which are template types for representing multiple occurrences of other types and give semantics to attributes with explicit multiplicity. Three of the types are based on the collection types of OCL and are Set, Bag, and Sequence. The last is the Dictionary type, which is a special type of collection with set semantics where the elements of the collection consist of value pairs instead of a single value. Thus the Dictionary template type takes two arguments. The first argument is the type of the key and the second is the type of the object.
- Enumerated data types, which provides a mechanism for defined a list of legal values, where each value is a mnemonic word with associated semantics.

[1] A DirectPosition is a primitive data type that holds a position in a coordinate system. It may store any number of coordinates.

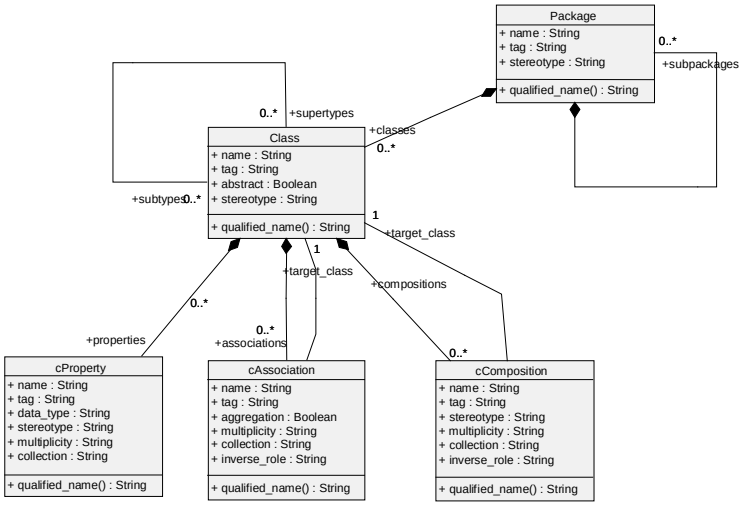

Fig. 2. The Schema Model is a subset of the UML's Metamodel and it may be used to represent application schemas.

Associations that are not compositions are represented as association objects (cAssociation) encapsulated in its respective target classes. Only associations with explicit role names are represented and the role name becomes the name of the association object. The inverse role refers to the name of the association object of the corresponding target class. Composite associations shall be represented as composition objects. The composite objects are likely to be nested as a part of its parent object in a hierarchical manner. Aggregation associations are treated the same way as associations because an object participating in one aggregation may participate in other aggregations. From an encoding point of view it will be unfavorable to duplicate objects in a hierarchical nesting, that in fact should be shared.

Packages contain their classes and are organized in a hierarchical manner. The topmost package corresponds to the model package.

Notice the occurrence of the name and tag attributes of the classes in the schema model. The tag attribute may be used as an alternative name for identifying the object and must be unique within the dataset. The name attribute must be unique within its namespace, which in this case are package and class. The schema model handles all classes the same way, still the stereotype and collection semantics are represented in case of future extensions. The instances can be identified by a qualified name. The form of the qualified names is *name1.name2.name3*, where *name1* is the name of the outermost package, *name2* is a name that appears within the namespace of *name1* and *name3* is a name that appears within the namespace of *name2*. The full-stop character "." is used as the name separator.

3.2 Instance Model

The instance model may be used to represent application data, see Fig. 3. This model depends on an instantiated schema model and is close to the representation form of XML. The objects refer to their class by a class name and are organized according to the package structure (iPackage) defined in the application schema.

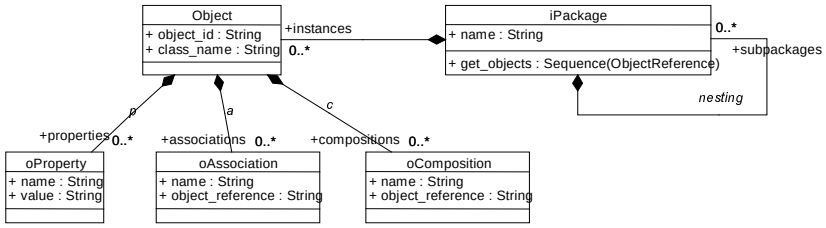

Fig. 3. The Instance Model may be used to represent data corresponding to an application schema.

An object has a unique object-identifier and its state is represented as a list of name-value pairs that corresponds to the property, association and composition objects of its class and its supertypes as represented in the schema model. The values stored in the property objects (oProperty) are encoded as text strings. The values shall be according to the value domain of their data type and may further be restricted to the value domains stated in the constraint statements of the class. The object references stored in the association (oAssociation) and composition (oComposition) objects are valid object-identifiers.

3.3 Output Data Structure

The output data structure is depicted in Fig. 4. It is a hierarchical structure composed of XML elements. The root element (GI) contains a header (GI.header) and a content (GI.content) element. The header consists of an optional documentation element, an element that refers to the application schema, and an optional lookup table. The content element contains the application specific XML elements, organized hierarchical according to the package structure of the application schema. The attributes of the root element indicates the naming policy, the containment model of the dataset, and whether the data is verified against the constraints specified in the application schema.

The naming policy indicates how the application schema based XML elements are named. Because of XML's global namespace all element names must be unique. Two alternative ways of naming the XML elements are defined: qualified or tag based. A qualified-name shall be a valid XML name that corresponds to a qualified name in the schema model. A tag-name shall be a valid XML name that corresponds to the tag attributes in the schema model. The rationale for using tag-names is to operate with language independent XML tags and also to reduce the amount of markup in the XML document, because the qualified names tend to get very long. Since UML does not support tag-names a lookup table must be included in the header element whenever tag-names are used. The lookup table shall consist of a list of tag-name qualified-name pairs separated by white-space. Another alternative, that has not been considered so far, is to use the XML namespace mechanism. Then a namespace for each package and for each object element must be defined.

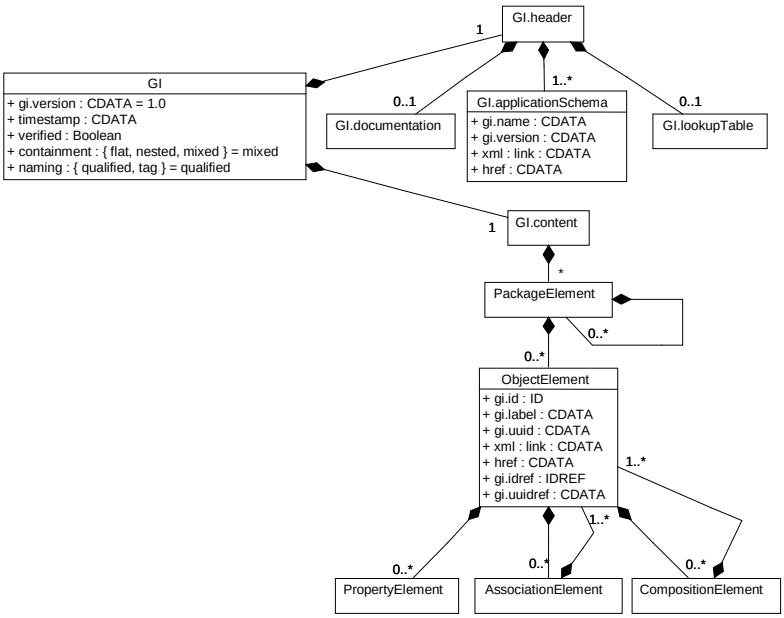

Fig. 4. An overview of XML document structure.

The containment model of the dataset indicates whether the composition elements contain their object elements or refer to their object elements. If the containment model is flat all composite objects are stored separately under their package extent. If the model is nested all composite objects are given directly. If the containment model is mixed a combination of a flat and mixed structure can occur.

Notice the attributes of the object elements. The first three attributes are used for identification. The purpose of these attributes is to allow other XML elements to refer to an object element utilizing XML's linking capabilities. The **gi.id** attribute corresponds to the unique object-id and must be unique within an XML document, but does not have to be globally unique. The **gi.label** can be used to identify a particular XML element by a user-defined string. The **gi.uuid** can hold a universal unique identifier for an object. Application communities are free to define the structure of this identifier [19, 20]. The next four attributes enable object elements to refer to other objects and thus act as proxy objects, that instead of giving the state of the object refers to its corresponding object. The first two attributes **xml:link** and **href** enable an XML element to act as a linking element and hence refer to object outside the dataset. The **gi.idref** attribute allows an XML element to refer to another XML element within the same document, and the **gi.uuidref** attribute allows the proxy object to refer to an object element that has a corresponding **gi.uuid** attribute.

3.4 Conversion rules

As mentioned earlier the conversion rules are divided into two categories. The first category is the schema conversion rules and the second category is the instance conversion rules. The result of a schema conversion is an XML DTD file. The purpose of this file is not to exchange schemas, but to ensure valid XML documents. The result of an instance conversion is an XML document that conforms to the DTD. In the following an overview of the schema and instance conversion rules are described.

A *package* is mapped to a package element declaration. The content model of a package element is the list of subpackages and choice of object elements corresponding to the classes declared within the package. An example of a package declaration is:

```
<!ELEMENT TopPackage ( (SubPackage1, ..., SubPackageN), ( Class1 | Class2
... | ClassN )*)>
```

A *class* is mapped to an object element declaration with attributes as described in Sect. 3.3. The content model of an object element is the list of property, association and composition elements.

A *property* is mapped to a property element declaration. If the property is of enumerated or Boolean data type then the value is mapped as an attribute of enumerated type. If not, the value is mapped to the element's content

model as parsed character data (#PCDATA). The multiplicity of the property is mapped to the corresponding multiplicity of XML, which are one-or-more (+), zero-or-more (*) or zero-or-one times (?). The multiplicity of the property is given in the class' content model. An example of a property element declaration is:

```
<!ELEMENT RM.Road.road_no (#PCDATA)>
```

An *association* is mapped to an association element declaration with a content model that is a choice of the XML elements that corresponds to the target class and any of the target class' concrete subclasses. The multiplicity of the association is stated at the end of the content model of the association element. An example of an association element declaration with multiplicity "1..*" is:

```
<!ELEMENT RM.Road.intersection (RM.Intersection)+>
```

A *composition* is mapped to a composition element declaration, with the same syntax as the association element. The difference between association and composition elements depends on the containment model. The object's composition elements shall either contain its target object element directly or its target proxy object element. The containment model of the dataset decides which of the two alternative conversion rules that shall be applied. The association elements shall always contain a proxy object element.

The content model of the object element declaration contains the list of its properties, association and composition elements, e.g.:

```
<!ELEMENT RM.Road
          (RM.Road.road_no, RM.Road.intersection)?>
```

Notice the optional content model (?) of the object element declaration. This is necessary to allow an empty object element to act as a proxy element and refer to the actual object.

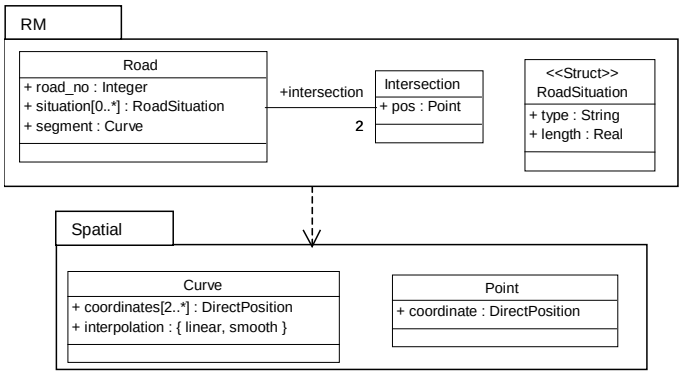

Fig. 5. An example of an application schema. The schema describes the class of objects in a road map data set.

3.5 Example

Fig. 5 shows a small example application schema that is subdivided into two packages, RM and Spatial. The important classes are Road and Intersection, which models a simple road map. A Road corresponds to a road segment that is associated to two Intersections. An example of the XML encoding of a Road object is given below. The Road class defines three attributes, **road_no** is of primitive type and is represented as a property element, whereas **situation** and **segment** are of model types and are therefore represented as composition elements. The association with role name **intersection** is mapped to an association element.

Here is an example of the XML encoding of a Road object using qualified names and a nested containment model:

```
<RM.Road gi.id="i01"> <RM.Road.road_no>66</RM.Road.road_no>
   <RM.Road.intersection>
      <RM.Intersection gi.idref="i05"></RM.Intersection>
      <RM.Intersection gi.idref="i06"></RM.Intersection>
   </RM.Road.intersection>
   <RM.Road.situation>
      <RM.RoadSituation gi.id="i02">
         <RM.RoadSituation.type>gravel
         </RM.RoadSituation.type>
         <RM.RoadSituation.length>159.5
```

```
        </RM.RoadSituation.length>
      </RM.RoadSituation>
    </RM.Road.situation>
    <RM.Road.segment>
      <Spatial.Curve gi.id="i09">
        <Spatial.Curve.coordinates>23 44
        </Spatial.Curve.coordinates>
        <Spatial.Curve.coordinates>59 33
        </Spatial.Curve.coordinates>
        <Spatial.Curve.interpolation value=”linear”/>
      </Spatial.Curve>
    </RM.Road.segment>
  </RM.Road>
```

4 Implementation

There are many ways to implement this mapping. Within the DISGIS project two different implementations have been made. The first is a part of the DISGIS distributed communication framework and the second is a prototype of a generic export/import facility.

The main DISGIS result is a distributed communication framework that comprises of an object-oriented data model and a server interface, specified in UML and implemented in C++. The framework supports both XML and binary encoding on top of CORBA and Socket communication. The code that handles the interchange format is generated from the UML models using Rational Rose [21] and its scripting language. The script generates XML and binary read/write operations according to the visitor pattern [22]. The distributed communication framework is currently being optimized and tested with respect to performance. The framework is portable, but the binary encoding and the data model are dependent on C++. The XML encoding, however, allows portable data as well.

The prototype generic export/import facility, called the GI Browser, is a Java application that has implemented the mapping described in ISO CD 15046-18. It is a tool that allows users to browse, edit, encode and decode datasets. It uses the Document Object Model [23] interface from SunSoft [24] to read and write XML. The prototype can be downloaded from [25]. The tool implements the schema model and the instance model, and allows users to browse data according to the models. A Rational Rose script has been developed that generates XML based schema model documents, which enables the tool to be configured according to the application schema.

5 Discussion

UML is a general-purpose modeling language for developing software systems. It is very popular and has become the language of choice for various standardization activities (ISO/TC 211, SQL/MM, CEN/TC 251). A natural question to ask is whether UML is powerful enough to facilitate semantic and syntactic interoperability?

The mapping should be sufficient to ensure syntactic interoperability. The abstract representation model, i.e. the instance model and the schema model, is the basis for defining the exact interpretation of the UML concepts used (package, class and association), and for creating a canonical mapping to XML. The package and class structure is well suited for organizing both schemas and datasets into manageable parts and the association for defining relationships between objects. The state of an object is stored as name-value pairs in three different groups, i.e. properties for the primitive data types, associations for relationships and compositions for composite objects.

It is essential that both the schema developers and the schema implementators are aware of the abstract representation model in order to ensure the same interpretation and understanding of an application schema and application data. Extensive documentation, use cases and appropriate example data are also needed. The models should be as formal as possible and could be further restricted using the Object Constraint Language (OCL) [26]. Even though semantic interoperability cannot be guaranteed.

XML is expected to be the successor of HTML on the Web. It is a generic, character and tag-based format that can encode a wide variety of data structures. XML is based on SGML and is therefore suitable for encoding documents, but is it also suitable for encoding large complex data structures, such as geographic information? XML’s extensive markup information, lack of primitive data types, global namespace and character encoding

imply large and ineffective files. On the other hand character encoding is well suited for compression and experience has shown that the file size can be reduced to 80-90% using standard compression software (WinZip). The tag-based naming policy has promises to reduce the amount of markup with 80%. This is based on an estimate that qualified names have an average length of 30 characters and that they are replaced with tag-names of no more than 6 characters. XML is a simple format and is easy to read because of extensive tool support. XML is neither dependent on any implementation nor programming language. Even if more efficient formats exist both in performance and size, the simplicity and broad support of XML and the possibility to use efficient de facto compression techniques weights more when interoperability is the primary goal. Applications with high-speed requirements however, should use a more effective format.

The mapping is inspired by XMI in that properties, associations and compositions are mapped to XML elements. Alternatively one could use XML attributes. The properties could instead have been mapped to XML attributes of type character data. But XML does not support repeating attributes, so encoding mechanisms to handle multiple occurrences of a property value must be created. Associations could be mapped to XML attributes of type IDREF, but this only allows an association to refer to objects within the same dataset and the links becomes untyped. Using XML attributes more extensively will reduce the amount of markup, because an XML attribute does not have an end-tag. But this will make the mapping more complex and restrict the possibility to refer to objects outside the dataset.

6 Conclusion and Further Work

The mapping presented and its implementations show that by restricting the semantic variation point of UML with regards to primitive data types and data structure, UML can be used as a schema language for defining XML based interchange formats and that this facilitates syntactic interoperability. The abstract representation model is the key to ensure a common interpretation of an application schema, and to ensure interoperability between different implementations of the same application schema. The mapping can easily be adapted to other application areas by defining a suitable set of primitive data types. XML is not the most efficient data interchange format, but it gets the job done. Future XML developments may lead to improvements, especially on schema declaration and handling of primitive data types. UML's intuitive and visual form extends XML's somewhat limited declarative powers and the combination of UML and XML may simplify data interchange between applications that are developed based on UML models.

The implementations show that the mapping is feasible, but large scale testing has not yet been done. Neither has the tag based naming policy been tested. It should be considered along with XML's new namespace mechanism. Other future work is to improving the efficiency of the export and import libraries with regards to large-scale data sets.

References

1. Soley, R., et al., eds. *Unified Modeling Language version 1.1*. 1997, Object Management Group. 97-08-[02..10]. http://www.omg.org/.
2. *Distributed Geographical Information Systems - Models, Methods, Tools and Frameworks*. 1999, ESPRIT project no. 22.084. http://www.disgis.com/.
3. *ISO/TC 211 Geographic Information/Geomatics*. 1999, http://www.it.statkart.no/isotc211/.
4. Skogan, D., ed. *ISO CD 15046-18 Geographic Information - Encoding.* ISO/TC 211 N-709 1999-03-16. 1999.
5. Iyengar, S. and S.A. Brodsky, eds. *XML Metadata Interchange (XMI).* Proposal to the OMG Object Analysis & Design Task Force RFP 3: Stream-based Model Interchange Format (SMIF) 1998, Object Management Group. http://www.omg.org.
6. Sheth, A.P., *Changing focus on interoperability in information systems: From system, syntax, structure to semantics*, in *Interoperating Geographic Information Systems*, M.F. Goodchild, *et al.*, Editors. 1999, Kluwer Academic Pub. ISBN 0792384369.
7. Kim, W., ed. *Modern Database Systems: The Object Model, Interoperability, and Beyond.* 1995, Addison-Wesley. ISBN 0-201-59098-0.
8. Bishr, Y., *Overcoming the semantic and other barriers to GIS interoperability.* International Journal of Geographic Information Science, 1998. **12**(4): p. 299-314.
9. France, R., et al., *The UML as a formal modeling notation.* Computer Standards & Interfaces, 1998. **19**(7): p. 325-334.
10. Rumbaugh, J., I. Jacobson, and G. Booch, *The Unified Modeling Language Reference Manual.* Object Technology Series. 1999, Addison-Wesley. ISBN 0-201-30998-X.

11. Booch, G., J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide.* Object Technology series. 1999, Addison-Wesley. ISBN 0-201-57168-4.
12. Bray, T., J. Paoli, and C.M. Sperberg-McQueen, eds. *Extensible Markup Language (XML) 1.0.* REC-xml19980210. W3C Recommendation 1998. http://www.w3.org/TR/REC-xml.
13. Maler, E. and S. DeRose, eds. *XML Linking Language (XLink).* WD-xlink-19980303. W3C Working Draft 1998. http://www.w3.org/TR/WD-xml-link.
14. Maler, E. and S. DeRose, eds. *XML Pointer Language (XPointer).* WD-xptr-19980303. W3C Working Draft 1998. http://www.w3.org/TR/WD-xptr.
15. Bray, T., D. Hollander, and A. Layman, eds. *Namespace in XML.* REC-xml-names-19990114. W3C Recommendations 1999. http://www.w3.org/TR/REC-xml-names.
16. Malhotra, A. and M. Maloney, eds. *XML Schema Requirements.* NOTE-xml-schema-req-19990215. W3C note 1999. http://www.w3.org/TR/NOTE-xml-schema-req.
17. Bray, T., C. Frankston, and A. Malhorta, eds. *Document Content Description for XML.* NOTE-dcd-19980731. W3C Note 1998. http://www.w3.org/TR/NOTE-dcd.
18. Fuchs, M., M. Maloney, and A. Milowski, eds. *Schema for Object-oriented XML.* NOTE-sox-19980930. W3C Note 1998. http://www.w3.org/TR/NOTE-SOX.
19. Sargent, P. *Feature Identities, Descriptors and Handles* in *Interoperating Geographic Information Systems: Second International Conference*. 1999. Zurich, Switzerland. Springer Verlag. ISSN 0302-9743.
20. Bishr, Y.A. *A Global Unique Persistent Object ID for Geospatial Information Sharing* in *Interoperating Geographic Information Systems: Second International Conference*. 1999. Zurich, Switzerland. Springer-Verlag. ISSN 0302-9743.
21. Rational, *Rational Rose 98,* http://www.rational.com/.
22. Gamma, E., et al., *Design Patterns: elements of reusable object-oriented software.* 1994, Addison-Wesley. ISBN 0-201-63361-2.
23. Apparao, V., et al., eds. *Document Object Model (DOM) Level 1 Specification.* REC-DOM-Level-1-19981001. W3C Recommendation 1998. http://www.w3.org/TR/REC-DOM-Level-1/.
24. JavaSoft, *Java Project X*, Early Access. DOM library. http://www.javasoft.com/xml/.
25. Rolfsen, R.K. and D. Skogan, *GI Browser*, 1.0. Java Application. http://www.informatics.sintef.no/UML2XML/.
26. Warmer, J. and A. Kleppe, *The Object Constraint Language - Precise Modeling with UML*. 1999, Addison-Wesley. ISBN 0-201-37940-6.