

Metaschema Layering for XML

Erik Wilde

Swiss Federal Institute of Technology Zürich (ETHZ)

<http://dret.net/netdret/>

Abstract:

The Extensible Markup Language (XML) is based on the concept of schema languages, which are used for validation of XML documents. In most cases, the meta-modeling view of XML-based application is rather simple, with XML documents being instances of some schema, which in turn is based on some schema language. In this paper, a metaschema layering approach for XML is presented, which is demonstrated in the context of various application scenarios. This approach is based on two generalizations of the standard XML schema language usage scenario: (1) it is assumed that one or more schema languages are acceptable as foundations for an XML scenario, but these schema languages should be customized by restricting, extending, or combining them; (2) for applications requiring application-specific schema languages, these schema languages can be implemented by reusing existing schema languages, thus introducing an additional metaschema layer. Metaschema layering can be used in a variety of application areas, and this paper shows some possible applications and mentions some more possibilities. XML is increasingly entering the modeling domain, since it is gradually moving from an exchange format for structured data into the applications as their inherent model. XML modeling still is in its infancy, and the metaschema layering approach presented in this paper is one contribution how to leverage the most important of XML feature's, which is the reuse of existing concepts and implementations.

1 Introduction

The *Extensible Markup Language (XML)* [Bra04] is the language of choice for many scenarios requiring data exchange in a heterogeneous environment. For many application areas, XML is nothing more than an exchange format, which needs to be used for data transmission. In this case, all that needs to be done is to find mappings between the local information model and the required XML exchange format (see RAHM and BERNSTEIN [RB01] for an extensive list of related work). However, XML is increasingly moving into the applications, because many technologies provide XML-based interfaces, so that XML is slowly changing its status from a pure exchange format to a format that is increasingly used as application information model. Testimonies to this observation are Web Services, XML support in database systems [ISO03], and Microsoft's *.NET* approach, which makes XML the key format in the whole application development environment.

Thus, XML is increasingly shifting its role from exchange format to application format. Taking this development and the recent focus on model-driven architectures, it becomes

clear that this new role of XML should be reflected in the way XML is modeled. In this paper, we present an approach to XML modeling which is not introducing a new modeling language, but instead presents a way how existing **schema languages for XML**, each being modeling methods themselves, can be combined to enable schema language reuse and rapid prototyping features through the reuse of existing implementation for these schema languages. The general approach of introducing different metamodel layers is well-known from the **OMG's Meta Object Facility (MOF)** [Obj02], but instead of using the M0-M3 layer names, we use layers named X0 and higher (and we do not limit our model to a fixed number of layers).

In Section 2, we take a close look at **XML Schema** from the point of view of metaschema layering, including some interesting observations about information vs. data models, and the implementation of many XML Schema tools. Since XML Schema is the most popular schema language and in many cases is used in slightly modified ways, we also discuss how XML Schema can be restricted and extended, and how these XML Schema modifications can be implemented from the metaschema layering point of view. However, many applications require dedicated schema languages, for example to enable users to define their own schemas, or simply to have a clean and clear way to manage application and schema evolution in parallel. These application-specific schema languages are described in Section 3, based on two real-life application scenarios.

The metaschema layering approach presented in this paper has been applied to some real-life examples, but it still is a novel method and would benefit from a systematic comparison with other approaches to XML modeling. Section 4 discusses these issues and describes some approaches for future work in this area, which is then described in Section 5.

2 XML Schema

XML Schema [BM04, Tho04] is the most important schema language for XML¹. It has been designed by the **World Wide Web Consortium (W3C)** and is increasingly used as a foundation for other XML-oriented standards, the most important being the **XML Query (XQuery)** language for querying XML databases, and the **SOAP/WSDL** standards for Web Services. Strictly speaking, XML Schema is not an XML Schema language, because it is defined on the **XML Information Set (XML Infoset)** rather than XML documents.

In fact, many of the upcoming XML technologies are not native XML technologies, but instead they are Infoset technologies. In order to understand the difference, an important distinction to make is the difference between *Information Models* and *Data Models* [Wil03c]. An information model is an abstract definition of concepts and their interrelations, while a data model is the implementation of an information model in a certain application context, such as a syntax for serialization, or an **API** for programmatic access. Using these concepts, it can be seen that while XML is a data model (a serialization of tree-structured

¹XML Schema has replaced XML's built-in **Document Type Definition (DTD)** schema language, which has very weak modeling features and, most importantly, lacks the datatype support that is essential for the majority of today's XML applications.

data), the Infoset is an information model for this data model, ignoring certain syntax details (for example, the order of attributes), and adding additional constraints (for example, the [namespace-well-formedness](#) of the XML document). In the context of XML Schema, the distinction between information and data models is shown in Figure 1, which shows the relevant parts of XML Schema from the perspectives of metaschema layers (vertical axis) and information and data models (horizontal axis).

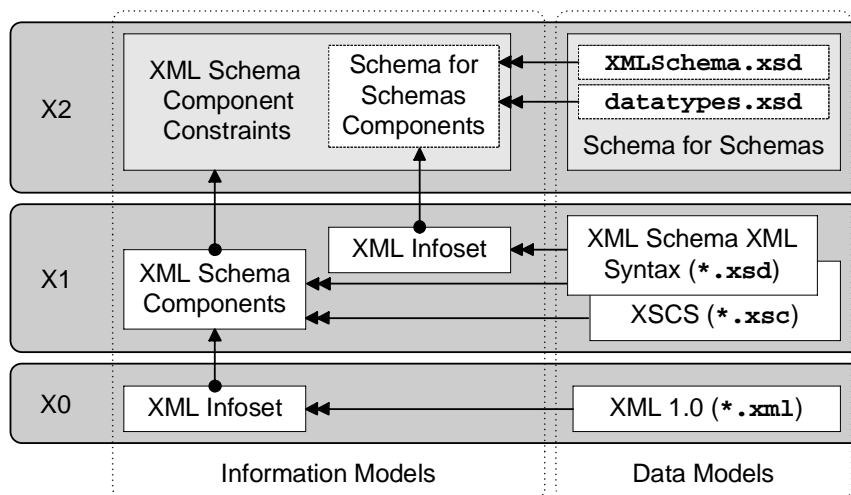


Figure 1: XML Schema Metaschema Layers and Model Views

The dot-started arrows in this diagram show “instance-of” relations, and the double-headed arrows show “represents” relations. The documents that usually are used in XML Schema scenarios (xsd files for schemas and xml files for instances) are serializations of data models, and for XML Schema validation, the documents are interpreted according to the information model of the respective layer (the Infoset for X0 and the XML Schema component model for X1), and the validation is performed on the information models.

From the metaschema layering point of view, it is interesting to note several things. As a very simple extension of the simplest view, it can be seen that there is an alternative syntax for XML Schema components, the *XML Schema Compact Syntax (XSCS)* [WS03]. This syntax simply is another way of serializing XML Schema components, it has the advantage of being better human-readable, at the price of being a non-XML syntax.

Another observation is that the information model of the X2 layer not only contains the components of the *Schema for Schemas* (implemented by the two normative schemas XMLSchema.xsd and datatypes.xsd), but also those aspects of the *XML Schema Component Constraints* which can not be expressed by XML Schema itself. Interestingly, there is no data model for these additional constraints, which are defined in the XML Schema specification using English prose. This is unfortunate, because the absence of a data model for these XML Schema Component Constraints means that it is impossible to work with them in advanced metaschema scenarios as described in Section 2.1.

As a side note, it should be mentioned that almost all software currently on the market dealing with XML Schema generation (most importantly, editors with XML Schema support) does not properly check all XML Schema Component Constraints when validating a schema (i.e., validating X1 instances against the X2 schema), in most cases effectively validating the Infoset of the XML Schema rather than the XML Schema components. This is a very serious error in the implementations, because it does not detect many subtle errors in the XML Schema, and consequently allows invalid XML Schemas to pass through the validation process, which in a properly implemented system should fully validate the X1 instance (the XML Schema components) against the X2 schema (the XML Schema Component Constraints).

While there are representation-oriented data models for XML Schema components (the XML Schema XML syntax and XSCS), so far there is no established API for XML Schema components. Popular approaches are the *XML Schema Object Model (XSOM)*, IBM's UML-based *Schema Infoset Model*, and the *XML Schema API* proposed in a W3C submission [Lit04], but so far no standardized XML Schema API exists, and working with XML Schemas from within applications consequently is necessarily bound to a specific product. To overcome this limitation in a portable way, it is possible to use an API for the data model rather than the information model, an approach which is described in the following section.

2.1 XML Schema Restrictions

In many scenarios including XML Schema as the schema language of choice, the complexity of XML Schema often is more than application designers want to deal with, so what they basically want is a subset of XML Schema, which supports all the modeling concepts that they want, but avoids some of XML Schema's complexity. For example, XML Schema's ability to have local and global declarations for elements, attributes, and simple and complex types makes for an amazing amount of possible permutations of these modeling alternatives. Many application developers want to avoid this complexity and thus reduce the modeling alternatives to a simpler model, for example forcing all these declarations to be global.

While it would be possible to define these application requirements as additional component constraints, there unfortunately is no data model for these constraints, so it is impossible to formalize and use the application requirements. It would be possible, however, to implement the application requirements as constraints that are programmatically checked through an XML Schema API, but this would have the drawback that (1) the application requirements would need to be written as procedural code rather than declarative constraints, and (2) this approach would bind the application to a programming language and a proprietary XML Schema API.

To overcome these limitations, PROVOST [Pro02] suggests to work with the Schema for Schemas, and to define the application requirements as constraints for the Schema for Schemas. One suggested way to implement XML Schema restrictions is by using XML

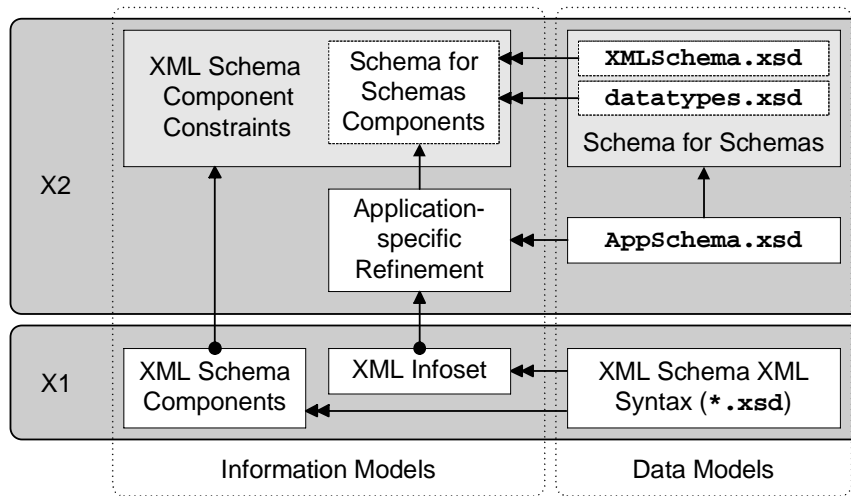


Figure 2: XML Schema Restrictions through Schema for Schemas Refinement

Schema's `redefine` mechanism to define the restricted variant of XML Schema as a refinement of the Schema for Schemas. While this solution, shown in Figure 2, is very elegant from the modeling point of view, it is not very powerful, because it is unable to access and thus reuse or redefine any of the information in the XML Schema Component Constraints. Thus, this solution looks elegant, but in practice often is not very useful because of its inherent limitations.

In order to overcome these limitations, it would be most useful to have a constraint language for XML Schema components. Unfortunately, such a language does not exist, but there is the *Schematron* [ISO04b] constraint language for XML documents (or, strictly speaking, for Infosets), which allows users to define **XPath**-based constraints (which are called patterns in Schematron). These constraints in essence are a set of rules against which the document is validated. Using this constraint language, it is thus possible to constrain the XML representation of XML Schema components. Since this variant works on the schema instance level (layer X1) rather than the metaschema level (layer X2), it is possible to define constraints that effectively serve as additional component constraints.

The general architecture of this approach is shown in Figure 3 (the Schematron constraints are represented in an XML document conforming to the Schematron metaschema, and these documents, representing Schematron schemas, conventionally use the `sch` file extension). It should be noted that the constraints are defined on the Infoset of the `xsd` file, rather than on the XML Schema components. For this reason, some constraints can be easily expressed (mostly those that are local to a component and can be defined on the syntax for this component), while others can be cumbersome or impossible to specify (constraints across multiple components, where the component relations are hard to express generically on the component syntax level).

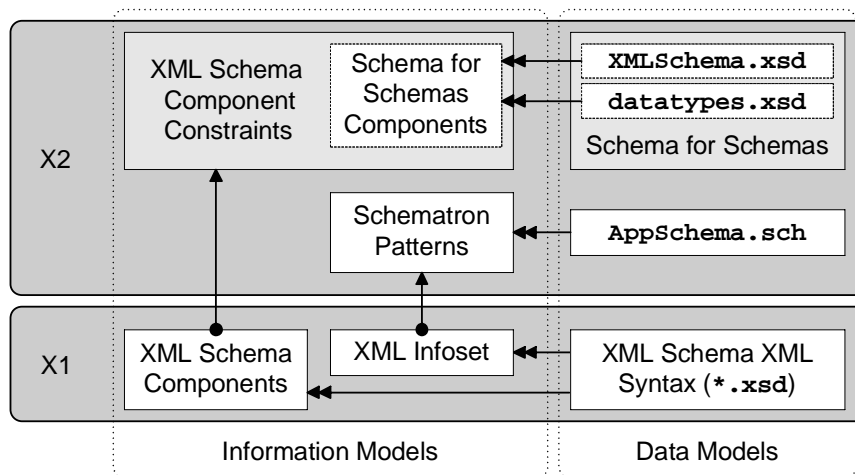


Figure 3: XML Schema Restrictions through Data Model Constraints

To summarize, there are several possible ways to implement XML Schema restrictions using metaschema layering, and the best way depends on the restrictions that should be implemented. While the schema refinement and schema data model constraint approaches have the advantages of being declarative, they both lack the generality of the programmatic approach, which implements restrictions by checking the components through an XML Schema API. The most important aspect of the declarative approaches is that they both work on the X1 Infoset of the XML Schema representation, rather than the actual XML Schema components.

2.2 XML Schema Extensions

In many scenarios, XML Schema is acceptable as the principal foundation for the schema of an application, but some additional constraints or information should also be captured in the schema. In this case, it is possible to use XML Schema's `appinfo` element, which enables users to embed application-specific information within an XML Schema. Annotations (the `annotation` element containing `appinfo` or `documentation` elements) may be associated with any XML Schema component, and in XML Schema's component model they are represented as *Annotation Schema Components* in the layer X1 XML Schema information model.

In Figure 4, the Annotation Schema Components are labeled as "Extension Components", because for our purpose they represent information extending XML Schema. They are part of the XML Schema Components, but they have a special status because they do not participate in the XML Schema validation process, and because they have semantics defined by some extension mechanism outside of XML Schema.

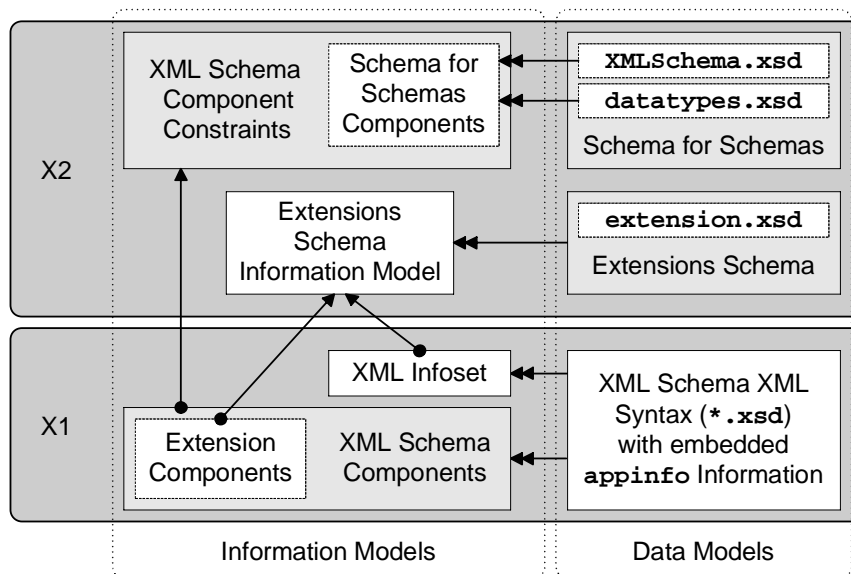


Figure 4: XML Schema Extensions

In the diagram it is shown that the Annotation Schema Components can be either regarded as XML Schema components, or from the perspective of their XML representation and thus through the corresponding Infoset. Since the components should be validated against a metaschema as well (XML Schema metaschema validation does not validate the semantics of annotation components), it is necessary to have a layer X2 schema for extensions. In our example, this schema is written in XML Schema (the `extension.xsd` document), thus minimizing the number of schema languages involved in the metaschema layering architecture, but in principle the schema for extensions could use any schema language (or none at all and implement extension validation through programmatic validation rather than a declarative schema language). In particular, it is not required that the schema for extensions validates extension instances (i.e., `appinfo` elements in the X1 schema) based on their component model, it could alternatively validate them based on their Infoset. Thus, the two “instance-of” arrows pointing to the “Extension Schema Information Model” are alternative implementations of XML Schema extensions. The validation of X1 “Extension Components”, however, would always have to be implemented using proprietary interfaces, because — as mentioned earlier — so far there is no standardized data model for XML Schema components.

As a special case for XML Schema extensions, it is interesting to look at scenarios where the schema should be described using XML Schema, but where the schema should be open to extensions throughout future versions of the application. ORCHARD and WALSH [OW03] discuss the basic idea of how XML vocabularies can be kept open and extensible for evolutionary changes. In the context of these changes, it may be required for forward

compatibility² reasons that additions to the application's information model must describe themselves through some predefined extension semantics, for example through information how to render new content. Thus, extensibility of a program, normally achieved through programmatic means such as APIs or plug-ins, could be increased to also encompass the information model, making extensions not only possible, but also semantically interpretable (limited to the initially foreseeable ways of how later revisions should be able to describe themselves) through an extension schema.

This schema could be named *extension schema* for two reasons, (1) because it extends the principal XML Schema and thus is a schema language *extending the principal schema language*, and (2) because it semantically describes the information model and thus enables the *semantic description of extensions to the application's information model*.

3 Application-specific Schema Languages

XML Schema has a special status in the area of **XML schema languages** because of its tight connection with other XML-related specifications. However, in many cases XML Schema is not the ideal schema language for application designers, and the same may be true for other existing XML schema languages. In this case, it may be appropriate to invent an application-specific schema language, an approach which is described by BERNAUER et al. [BKK03] (they discuss different possible strategies for designing application-specific schema languages). While in principle an application-specific schema language can be anything, an effective way to implement such a schema language is to base it on existing schema languages, thus reusing the functionality already implemented by these languages (the simplest approach, the restriction of an existing schema language, has already been discussed in Section 2.1).

Generally speaking, the *Document Schema Definition Languages (DSDL)* [ISO04a] initiative aims at replacing the single-schema-language approach exemplified by DTDs and XML Schema by a modular way of looking at XML validation (and thus schema languages). DSDL is still under development and so far did not gain a lot of support or momentum, but the general idea of treating validation as a multi-step process is useful and can be reused for the application-specific schema language approach discussed here.

As a first example of how schema languages can be modeled, Section 3.1 describes a generic schema language which is based (on the schema level) on existing schema languages, and is implemented (on the validation level) using existing XML technologies. As an alternative approach, Section 3.2 describes an application-specific schema language which on the schema as well as the validation level is based on the same existing XML schema languages, thus maximizing the reuse of existing technologies.

²*Forward compatibility* means that a program should be able to work with future revisions of the data format, handling previously unknown future features as gracefully as possible.

3.1 Schema Language Implementation

The *Character Repertoire Validation for XML Documents (CRVX)* [Wil03b] schema language is a specialized language for restricting the character repertoire used in XML documents (for example, it is possible to specify that all element and attribute names must be ASCII characters). It has been designed as one component of the DSDL framework mentioned in the previous section. CRVX allows the specification of constraints on used character sets, and it allows to scope these constraints on XML syntax structures (for example, comments, element names, attribute names, or character data), and on contexts identified by XPath patterns. The overall structure of an CRVX schema is pretty simple, and formally CRVX is defined by an XML Schema and a Schematron schema specifying additional patterns. This design is shown in layers X2 and X3 in the metaschema view of CRVX shown in Figure 5.

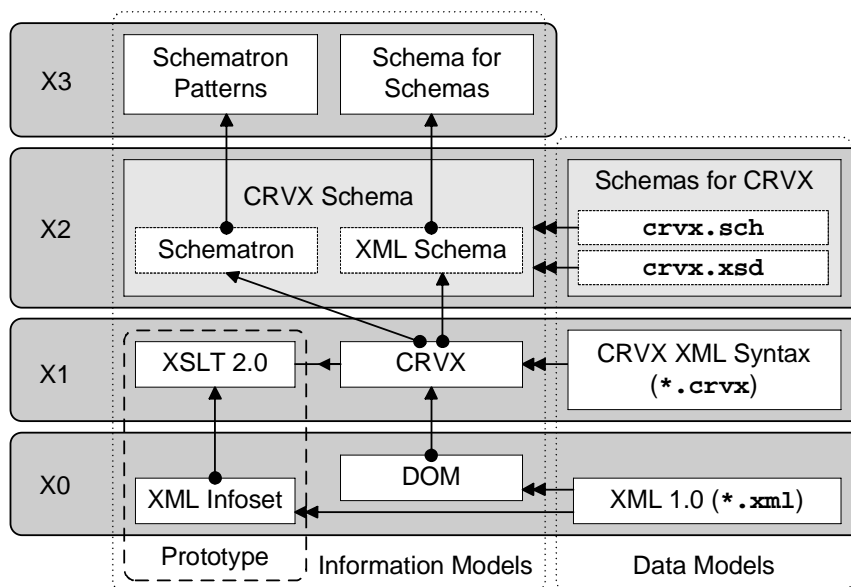


Figure 5: Definition and Implementation of CRVX

So far, this architecture is pretty easy to understand and is used in many similar scenarios. What makes it interesting and worth looking at in detail is the implementation of instance checking in layers X0 and X1. During the development of CRVX, it became clear that CRVX would need to be implemented incrementally, since only extensive testing could generate the feedback required to improve the language based on actual application requirements. Thus, to make the feedback cycle easier to handle and faster to develop, a prototype was created that allowed rapid testing of CRVX. This prototype is shown in the lower left corner of Figure 5. The prototype consists of an XSLT program that takes a CRVX schema as input and generates an XSLT program as output (represented by the

mid-arrow in layer X1). This generated XSLT program then can be used as validation code for XML documents. This approach reuses many of the useful features of XSLT, such as match patterns for applying certain constraints only to selected parts of XML documents. However, since only XPath 2.0 provides the advanced string handling functions [MMW03] required for character repertoire validation, the generated XSLT requires XSLT 2.0 (which is only available as prototype implementations).

When using XSLT, it is unavoidable that the code is operating on the Infoset of the input document, because XSLT's expression language XPath is based on the Infoset. Unfortunately, CRVX requires a more detailed information model than the Infoset, because it includes aspects such as **CDATA sections**, which are not represented in the Infoset. Thus, while CRVX is based on XML³ rather than the Infoset, the prototype implementation is based on the Infoset. This design was chosen because the benefit of a rapid development cycle outweighed the drawbacks of an incomplete implementation, and productive versions of CRVX are based on DOM information models (available, for example, through a **DOM** or **SAX API**) rather than the Infoset.

3.2 Metaschema Layering for Applications

For an application supporting the management of bibliographic information, the requirement was open and extensible schema handling, with schema extensions definable and installable by users [Wil03a], the requirement was to have a schema language that users could easily understand and use to extend the system. So instead of forcing users to define their schema extensions using a subset of XML Schema extended with Schematron pattern matching facilities, we designed our own schema language, which could be easily used and mapped to constructs of standard schema languages.

Figure 6 shows the design of the application-specific schema language *BibSchema*, which itself is based on a Schema (the *BibSchema Schema*). In order to avoid an overly complex picture, the figure does not differentiate between the information and data models, but it should be kept in mind that this differentiation is always present when working with XML.

The application architecture uses metaschema layer X3, which provides the schema language foundation for our application scenario. The *BibSchema Schema* is the application-specific schema language for user-defined *BibSchema* instances, and this schema language is defined through XML Schema and Schematron schemas. Since the *BibSchema Schema* remains stable, it is implemented using hard-coded XML Schema and Schematron schemas. On the other hand, *BibSchema* instances (layer X1) are user-definable and as such must be flexible and easy to generate and manage. Thus, users can define new *BibSchema* instances, which are then used for generating XML Schema and Schematron schemas (the mid-arrow lines in layer X1). These generated schemas are then used to validate X0 instances, the actual application data.

³In the figure, the information model is labeled “**DOM**”. This has been done to show that CRVX is based on a richer information model than the Infoset, and the DOM model (not the actual API definition implementing it) can be regarded as such a model.

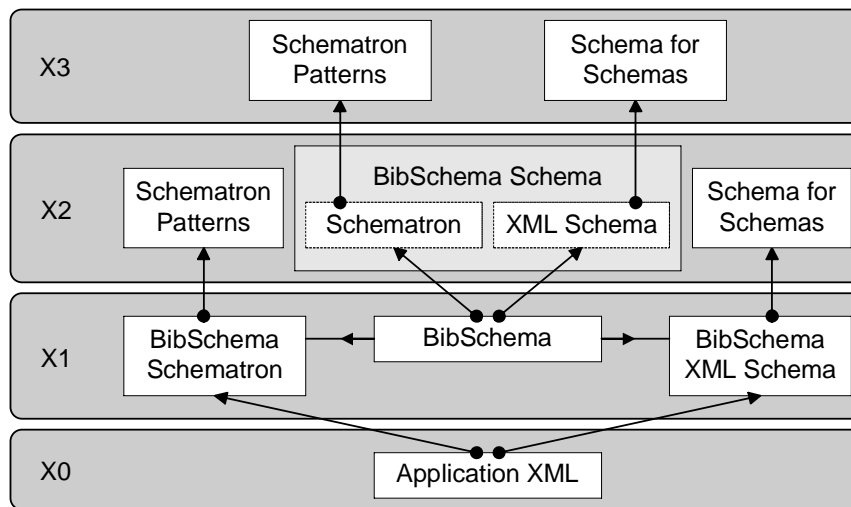


Figure 6: Application-specific Schema Language Design

From a software engineering point of view, it is interesting to note that the X3 schema languages are used for X2 schemas as well as X1 schemas (and hence for instance validation on layers X1 and X0). This has been done to minimize the number of schema languages used in the project. From a theoretical point of view, however, X2 schemas could have been based on an entirely different set of XML schema languages than X1 schemas. Some of the important issues of metaschema layering are to leverage the most efficient schema languages for the specific application scenario, to reuse existing implementations of these schema languages, and keep the number of required schema languages to a minimum.

3.3 Metaschema Layering and the Semantic Web

In the same way as we use schema layering for composing application-specific schema languages, additional schema layers can be used to add information relevant for the Semantic Web. For example, in an enhanced version for Semantic Web integration, the architecture of Figure 6 can be augmented with a layer X4, which then specifies semantic information for exposing BibSchema instances as Semantic Web information. This information is used to augment instances with semantic annotations. This way, Semantic Web applications can easily harvest BibSchema information by interpreting the semantic information associated with each instance.

The exact nature of the language being used for the semantic description is irrelevant (most likely RDF-based information based on OWL [MvH04] or some other RDF-based ontology language), as well as the question whether the ontology is application-specific or some existing ontology for bibliographic information. The important aspect is that this

additional Semantic Web layer would fit naturally into the overall model of metaschema layering, making the framework powerful enough to span the whole spectrum of XML schemas (describing *how* instances are being encoded) to XML semantics (describing *what* instances encode). Semantics in this case is just another layer of schema information, enabling users to describe their model in a more precise way.

4 Quality Assessment

According to SI-SAID CHERFI et al. [SAC02], the three dimensions for modeling quality are *usage*, *specification*, and *implementation*. While the metaschema layering framework presented in this paper is still missing some important pieces to become a full-fledged modeling framework, it already is used in practice in many places (not necessarily through applying it top-down, but because a bottom-up approach resulted in such an architecture). The innovation rate in the area of XML is very high and probably will continue to remain like this for some time to come, because there is an amazing amount of different uses and users employing XML technologies. The metaschema layering framework is an effective way to leverage existing technologies, and to reduce implementation costs and time by reusing modeling concepts and software components implementing them. Rating the metaschema layering approach against other methodologies would be very interesting, since so far no clear winner has emerged in the ongoing contest to become *the* XML modeling language rather than being *yet another* XML modeling language.

We believe that the modular approach pioneered by DSDL on the horizontal axis (by sequencing schema languages), and extended to include the vertical axis (the metaschema layering presented in this paper) has many advantages, in particular in the area of XML, which is being used for application areas as diverse as medieval text manuscripts being marked up as XML, and encoding DNA sequences in XML.

The metaschema layering approach is being used in some other of our projects, including projects where the modeling semantics go beyond validation and include application-specific processing semantics. Since these processing semantics often have very subtle side-effects and require very specific evaluation, it is difficult to capture the semantics in a purely declarative way. Thus, the metaschema layering framework needs to be extended to include external information, and to deal with application-specific validation that cannot be expressed declaratively. However, even though these extensions may compromise the purely declarative approach presented in this paper, we believe that it will improve the framework's quality by improving its expressiveness more than it reduces the understandability and the implementability.

5 Future Work

The metaschema layering approach presented in this paper shows an effective and flexible way to leverage the wealth of existing XML technologies, in this case XML schema

languages. While the XML Schema language is becoming the de-facto standard for XML schema modeling, other languages such as Schematron as a rather simple schema language for specifying constraints also should be taken seriously. Rather than having to choose exactly one schema language for an application scenario, through the flexible combination of schema languages and metaschema layers it is possible to reuse schema languages in novel and interesting ways.

When using metaschema layering for validation purposes only, it is interesting to look and some other approaches to XML validation beyond the most popular schema languages, such as *XML Constraint Specification Language (XCSL)* [Jac02] (a constraint language comparable to Schematron) and *xlinkit* [Nen03], which is a much more powerful approach than a single schema language, containing a complete framework for consistency checking for XML documents.

However, broadening the perspective beyond validation, it becomes clear that metaschema layering could be useful in a variety of ways, not only for validation, but also for processing of XML, where different processing steps could be controlled and described by different schema. While this is an intriguing idea, work on an XML processing model [LW04] is in its early stages, and the integration of a horizontal dimension (pipelining XML processing steps) with a vertical model (such as the metaschema layering approach) requires a lot of research efforts. Some of the most interesting questions are the following:

- *Application Semantics Beyond Validation:* Using XML Schema extensions for forward compatible information models has been discussed at the end of Section 2.2, but so far this area is mostly unexplored. Rather primitive extension semantics (such as HTML's rule to ignore unknown elements and attributes) could be refined to include more elegant and more powerful ways of extending an information model, including ways to validate an extended information model against an extension schema.
- *Information Model Issues:* Most figures in this paper show the information as well as the data model of the metaschema layers. In some cases, it can be seen that one information model may be represented by several data models (XML Schema components can be represented as XML or as XSCS, shown in Figure 1), or that one data model can be interpreted according to different information models (an XML Schema XML document can be interpreted as XML Schema components or as Infoset, as shown in Figures 2–4). Many publications discussing XML issues do not make this distinction in a sufficiently clear way, and in some cases this can lead to confusion or even conceptual errors.
- *Processing Pipelines:* The idea of processing pipelines appears in many different contexts, and even though there have been some approaches to defining generic XML processing pipeline facilities, none has succeeded so far. Defining a generic, flexible, but still easily applicable XML processing pipeline model is a challenge, and it still remains to be seen whether one language will be able to gain some popularity. Integrating the pipeline model with metaschema layering concepts could be an interesting way to go, but there may be other dimensions which should be taken into account, too.

Some of these issues have been discussed in application-specific ways already, but it still remains to be seen whether generic solutions are possible, and if so, whether they will succeed. The innovation rate in the XML area is very high, and thus there seems the tendency to simply invent *yet another markup language* (or sometimes *yet another modeling language* or *yet another schema language*) in many cases, rather than reusing existing ones. One reason for this inflation of XML dialects is the shortage of powerful and flexible ways to reuse XML technologies on a higher level than just reusing XML parsers. Future work in the area of XML modeling and XML processing should focus on reusing existing technologies in the most flexible way, trying to cover as many potential XML applications as possible, rather than targeting on a narrow and specialized application domain.

6 Conclusions

The metaschema layering framework presented in this paper provides an interesting perspective on XML modeling, in particular with the focus on reusing existing modeling mechanisms for XML. We have used this approach in several projects, and it has provided a very useful way to structure, compare, and rate various XML modeling alternatives. Our next goals are to further investigate this approach in the area of schema evolution, in particular in the context of Web Services. XML openness and extensibility are among the most important arguments for using XML within applications, but appropriate modeling approaches are required to really use XML in an open and extensible way. The most promising path is to reuse existing methods rather than to reinvent the wheel by introducing yet another modeling language. This paper is a contribution to encouraging reuse and combination of existing and proven XML technologies.

References

- [BKK03] Martin Bernauer, Gerti Kappel, and Gerhard Kramler. Approaches to Implementing a Tailored Metaschema in XML. In Johann Eder and Tatjana Welzer, editors, *Short Paper Proceedings of the 15th Conference on Advanced Information Systems Engineering*, volume 74 of *CEUR Workshop Proceedings*, pages 133–140, Klagenfurt, Austria, June 2003. Technical University of Aachen (RWTH).
- [BM04] Paul V. Biron and Ashok Malhotra. XML Schema Part 2: Datatypes Second Edition. World Wide Web Consortium, Proposed Edited Recommendation PER-xmlschema-2-20040318, March 2004.
- [Bra04] Tim Bray, Jean Paoli, C. Michael Sperberg-McQueen, Eve Maler, and François Yergeau. Extensible Markup Language (XML) 1.0 (Third Edition). World Wide Web Consortium, Recommendation REC-xml-20040204, February 2004.
- [ISO03] International Organization for Standardization. Information Technology — Database languages — SQL — Part 14: XML-Related Specifications (SQL/XML). ISO/IEC 9075-14, December 2003.
- [ISO04a] International Organization for Standardization. Information Technology — Document Schema Definition Languages (DSDL). to be published as ISO/IEC 19757.

- [ISO04b] International Organization for Standardization. Information Technology — Document Schema Definition Languages (DSDL) — Part 3: Rule-based validation — Schematron. to be published as ISO/IEC 19757-3.
- [Jac02] Marta Henriques Jacinto, Giovanni Rubert Librelotto, José Carlos Leite Ramalho, and Pedro Rangel Henriques. Constraint Specification Languages: Comparing XCSL, Schematron and XML-Schemas. In *Proceedings of XML Europe 2002*, Barcelona, Spain, May 2002.
- [Lit04] Elena Litani. XML Schema API. World Wide Web Consortium, Member Submission SUBM-xmlschema-api-20040309, March 2004.
- [LW04] Dmitry Lenkov and Norman Walsh. XML Processing Model Requirements. World Wide Web Consortium, Note NOTE-proc-model-req-20040405, April 2004.
- [MMW03] Ashok Malhotra, Jim Melton, and Norman Walsh. XQuery 1.0 and XPath 2.0 Functions and Operators. World Wide Web Consortium, Working Draft WD-xpath-functions-20031112, November 2003.
- [MvH04] Deborah L. McGuinness and Frank van Harmelen. OWL Web Ontology Language Overview. World Wide Web Consortium, Recommendation REC-owl-features-20040210, February 2004.
- [Nen03] Christian Nentwich, Wolfgang Emmerich, Anthony Finkelstein, and Ernst Ellmer. Flexible Consistency Checking. *ACM Transactions on Software Engineering and Methodology*, 12(1):28–63, January 2003.
- [Obj02] Object Management Group. *Meta Object Facility (MOF) Specification — Version 1.4*, April 2002.
- [OW03] David Orchard and Norman Walsh. Versioning XML Languages. World Wide Web Consortium, Proposed TAG Finding, September 2003.
- [Pro02] Will Provost. Working with a Metaschema. *xml.com*, October 2002.
- [RB01] Erhard Rahm and Philip A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *The International Journal on Very Large Data Bases*, 10(4):334–350, December 2001.
- [SAC02] Samira Si-Said Cherfi, Jacky Akoka, and Isabelle Comyn-Wattiau. Conceptual Modeling Quality — From EER to UML Schemas Evaluation. In Stefano Spaccapietra, Salvatore T. March, and Yahiko Kambayashi, editors, *Proceedings of the 21st International Conference on Conceptual Modeling*, volume 2503 of *Lecture Notes in Computer Science*, pages 414–428, Tampere, Finland, October 2002. Springer-Verlag.
- [Tho04] Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn. XML Schema Part 1: Structures Second Edition. World Wide Web Consortium, Proposed Edited Recommendation PER-xmlschema-1-20040318, March 2004.
- [Wil03a] Erik Wilde. Towards Federated Referatories. In *Proceedings of the SINN03 Conference on Worldwide Coherent Workforce and Satisfied Users*, Oldenburg, Germany, September 2003.
- [Wil03b] Erik Wilde. Validation of Character Repertoires for XML Documents. In *Proceedings of the Twenty-fourth Internationalization and Unicode Conference*, Atlanta, Georgia, September 2003.
- [Wil03c] Erik Wilde. XML Technologies Dissected. *IEEE Internet Computing*, 7(5):74–78, September 2003.
- [WS03] Erik Wilde and Kilian Stillhard. A Compact XML Schema Syntax. In *Proceedings of XML Europe 2003*, London, UK, May 2003.