Balisage: The Markup Conference 2009
Proceedings

Engineering Document Applications — From UML Models to XML Schemas

Dennis Pagano
Chair for Applied Software Engineering, Technische Universität München

Anne Brüggemann-Klein
Engineering Publishing Technology Group, Technische Universität München

Balisage: The Markup Conference 2009
August 11 - 14, 2009

How to cite this paper

Pagano, Dennis, and Anne Brüggemann-Klein. “Engineering Document Applications — From UML Models to XML Schemas.” Presented at Balisage: The Markup Conference 2009, Montréal, Canada, August 11 - 14, 2009. In *Proceedings of Balisage: The Markup Conference 2009*. Balisage Series on Markup Technologies, vol. 3 (2009). DOI: 10.4242/BalisageVol3.Bruggemann-Klein01.

Abstract

Modeling is a pivotal activity in the engineering of software systems; it is the key method to deal with complexity and change. Models support communication and drive software development processes. In the Engineering Publishing Technology Group, we aim to leverage modeling and other proven methods of software engineering for document engineering and electronic publishing. Particularly, in this paper, we align UML class diagrams, which model persistent data, with XML Schemas, which model the XML representation of persistent data as documents. Our method, which we call uml2xsd, transforms an XMI representation of a UML class diagram into an XML Schema that constrains XML instances of the UML model. It is patterned after transformations into XML Schema that XMI defines for models in its MOF language, which are closely related to UML class diagrams. With appropriate configuration, our method automates a transformation process that has been conceptually defined in our paper presented at Extreme Markup Languages 2007; it incorporates our handling of relations that has been presented at Balisage 2008.

Table of Contents

Organization of the paper

In the Engineering Publishing Technology Group, we aim to leverage modeling and other proven methods of software engineering for document engineering and electronic publishing. Particularly, in this paper, we align UML class diagrams, which model persistent data, with XML Schemas, their implementational counterparts that constrain the XML implementation of persistent data as documents. We have felt the need to provide, for the Balisage audience, in the first "real" section of this paper (Models in software engineering) some background information on the role of and technology for modeling in software engineering. We continue the paper with a section (Modeling persistent data) on the place that modeling of persistent data and schema engineering for XML implementations of persistent data should have in software engineering processes. In the next section (Focus of this paper), we present a management summary of our contributions in this paper.

The next four sections expand on a number of aspects: In the first of these four (Transforming UML models to XML Schema using XMI), we talk about the ways in which we can make use of the XML Metadata Interchange XMI for our purposes. The second (XMI schemas from UML class diagrams: The XMI mapping) reports on the way that XMI maps UML class diagrams into XML Schemas. The third (Adapting the XMI mapping to the document setting) talks about adaptations and modifications to the XMI mapping that are required in a document setting. The forth and final section (Realizing the layer shift using XSLT) introduces our implementation; particularly, it explains what we call the layer shift, our method that leverages the widely implemented XMI export of UML modeling tools.

We finish the paper with a section that gives our conclusions and discusses some related work.

Models in software engineering

In this section, we represent the well-established perspective of the software engineering discipline to modeling. The state of the art is discussed in many excellent textbooks. We have been working with the one by Brügge and Dutoit [BrüggeEtAlsOOSWE] .

Modeling is a pivotal activity in the engineering of software systems. Based on the dual principles of abstraction and implementation, and used in evolutionary phases that provide stepwise refinements, it is the key method to deal with complexity and change. Models support communication and drive software development processes. Modeling is common practice particularly in the earlier phases of software development. During requirement analysis, models support communication between software engineers and domain experts; during design, they document design intent to implementors; during implementation, they abstract from syntactic details of programming languages. The long-standing vision of software engineering, namely to drive the development process by models that are systematically transformed into lower-level models and software artifacts, has more recently found its expression in the approach of model-driven architecture. It is a widely acknowledged benefit of model-driven software development that it enforces uniform coding principles and strategies, leveraging best practices, and that it reduces error rates. The most striking advantages are:

1. Coherence between models and their implementations; this may go as far as to proven correctness.
2. Rational and documented transition between models on different levels of refinement, which enables accountability.

With the Unified Modeling Language UML, the Object Management Group OMG offers a graphical

notation for defining models. The UML provides several types of diagrams for the modeling of software systems. Software engineering methods such as the Rational Unified Process RUP are based on the UML. They are supported by so-called CASE tools; that is, WYSIWYG editors to create and edit UML diagrams. The more powerful CASE tools allow for code generation (forward engineering), in pursuit of the vision of model-driven software development. Over the last decade, the UML has replaced previous modeling languages in software engineering practice.

Modeling persistent data

Let us turn our attention now to a specific but important aspect of software systems, namely persistent data. Traditionally, if data volume and access requirements make custom formats unfeasible, data have been persisted in databases, with relational databases claiming the lion's share of the market. Increasingly, XML and XML database management systems come into play, the former as a format for persistent data and the latter for management and access of the XML data. In particular, XML establishes itself as the data format of choice if data are structured irregularly (semi-structured data) of if data are genuine textual documents [AbiteboulEtAlDataOnTheWeb] .

We pose three requirements for the modeling of persistent data:

1. Modeling of persistent data must be integrated into system modeling.
2. Modeling of persistent data must be independent of data implementation technology.
3. Modeling of persistent data should be amenable to forward engineering.

How do these requirements mesh with the reality of XML modeling?

With schema languages such as DTD, XML Schema, Relax NG and Schematron [W3CRecXMLSchemaStructures] [vdVlistSchemaTechnologies] [vdVlistXMLSchema] , the XML community has developed XML-specific modeling languages that are supported by XML-specific modeling methodologies [MalerEtAlDTDModeling] [EcksteinEtAlXMLDatenmodellierung] [ConradEtAlXMLConceptualModeling] [KhanEtAlSchemaDesignPatterns] [LainevoolXMLPatterns] [StephensonBestPractices] . Tool support is available in the form of graphical editors for schema languages, requiring XML domain knowledge from software engineers who develop models.

Fundamental weaknesses of this approach are that the modeling depends on the schema technology, that the modeling language is very close to the implementation language and that the modeling of documents is separated from and not integrated into the general system modeling. These deficits persist even if XML modeling tools are integrated as modules into general UML tools, as it is the case, for example, with Altova UModel or HyperModel.

What options do we have if we want to truly integrate schema engineering into software engineering? In this, we will be guided by proven software engineering procedure. In practice, in software projects persistent data are modeled as UML class diagrams. Historically, there are two main options to implement the data. The first option implements data as objects in a specific programming language and then “marshals” the objects into a custom data format for persistent storage. The second option stores data in a database. In both cases, not only the data but also their models have an implementation counterpart, either in the form of class declarations in a programming language or in the form of a database schema definition. If we persist data as XML documents, the obvious implementational counterparts to data models are XML Schemas. Hence, the sound software engineering position is to

regard XML as a target platform for persistent data and to transform data models into XML Schemas that constrain the XML-encoded data [PaganoDA] . This puts not only XML but also XML Schema and other XML schema languages firmly on the side of implementation technologies.

Focus of this paper

In this paper we present a method, which we call uml2xsd, to translate conceptual models that, following current software engineering practice, are written as UML class diagrams into XML Schemas, thus aligning UML class diagrams, which model persistent data, with XML Schemas, which constrain the XML representation of persistent data as documents. The paper is the quintessence of research that the first author did for his diploma thesis [PaganoDA] under the supervision of the second author.

We base this work on XMI, a method that the Object Management Group (OMG) has defined to turn models that are written in its modeling language MOF into equivalent XML Schemas. Due to the architectural alignment of MOF and UML, MOF models can be considered as specific UML models that represent concepts in the form of class diagrams. Hence, we were able to pattern uml2xsd after the OMG translation of MOF models in XML Schema, extending it to concepts that are present in UML class diagrams but not in MOF models.

The OMG method is defined on a conceptual level. We are not aware of any tools that actually implements this method. Hence, XMI provides a conceptual guideline for uml2xsd, but there is no precedence for an implementation.

There is, however, another aspect of XMI that we can build on: The goal of XMI is to facilitate exchange of models as XML documents. The MOF language is used as a language to define metamodels (models whose instances are models), such as the UML metamodel. The XMI method that we have mentioned above, is used to translate such metamodels into specific XML Schema documents that are called XMI schemas. Applied to the specific metamodel of UML, this method yields a single XMI schema that constrains XML representations of UML models, which are called XMI documents.

Our method uml2xsd leverages the common capability of modern UML tools to export UML models as XMI documents. It transforms an XMI representation of a UML class diagram into an XML Schema that constrains XML instances of the UML model. It is patterned after transformations into XML Schema that XMI defines for models in its MOF language, which, due to the architectural alignment, are closely related to UML class diagrams. Our method achieves a shift from the XML instance layer that XMI supports for model interchange to the XML Schema layer that supports modeling itself. Being based on XMI as a standardized interchange format for UML models, uml2xsd is independent of specific UML modeling tools or software development methods. Technically, uml2xsd is an XSLT program that operates on XML representations of UML class diagrams, the XMI documents, and turns them into XML Schemas.

The core idea behind uml2xsd is to generate from UML class diagrams not XML instances, as is the XMI way, but XML Schema documents, thus aligning conceptual instances of UML models with XML instances of XML Schema documents. We make use of the XMI support of UML modeling tools in that we first export UML class diagrams as XMI documents and then translate the results with XSLT into XML Schema documents. The translation rules adapt XMI translation rules from a higher level of the so-called MOF metamodel hierarchy, which is enabled by an alignment between the metamodel

hierarchy and UML class diagrams. This “layer shift”, which we are going to explain in more detail further on, validates the uml2xsd approach.

In principle, the uml2xsd method should work with XMI data that were exported from any UML tool. In practice, however, XMI allows for a wide spectrum of variants to represent any artifact in a UML diagram, so we hesitate to claim too much. We have, however, successfully tested the implementation with a number of UML tools, particularly with Enterprise Architect and Altova UModel.

At Extreme Markup Languages 2007, the Engineering Publishing Technology Group has presented principles, patterns and procedures of translating a conceptual document model in form of a simple UML class diagram into an XML Schema document [EPTatEML2007] . In a contribution [EPTatBalisage2008] at Balisage 2008, we have extended our investigation to more complex document models that exhibit relationships between classes, coding relationships on the XML side of documents with XLink and embedding processing support for XLink into XQuery. With appropriate configuration of uml2xsd, we are now able to automate these translations. We have applied uml2xsd to automatically generate XML Schema documents from conceptual models of two XML applications, XBlog and XTunes.

We have further evaluated uml2xsd by applying it to the UML *metamodel* (which is an instance of MOF and, hence, an UML class diagram); the result is an XML Schema for UML that is equivalent to the XMI encoding rules of UML class diagrams.

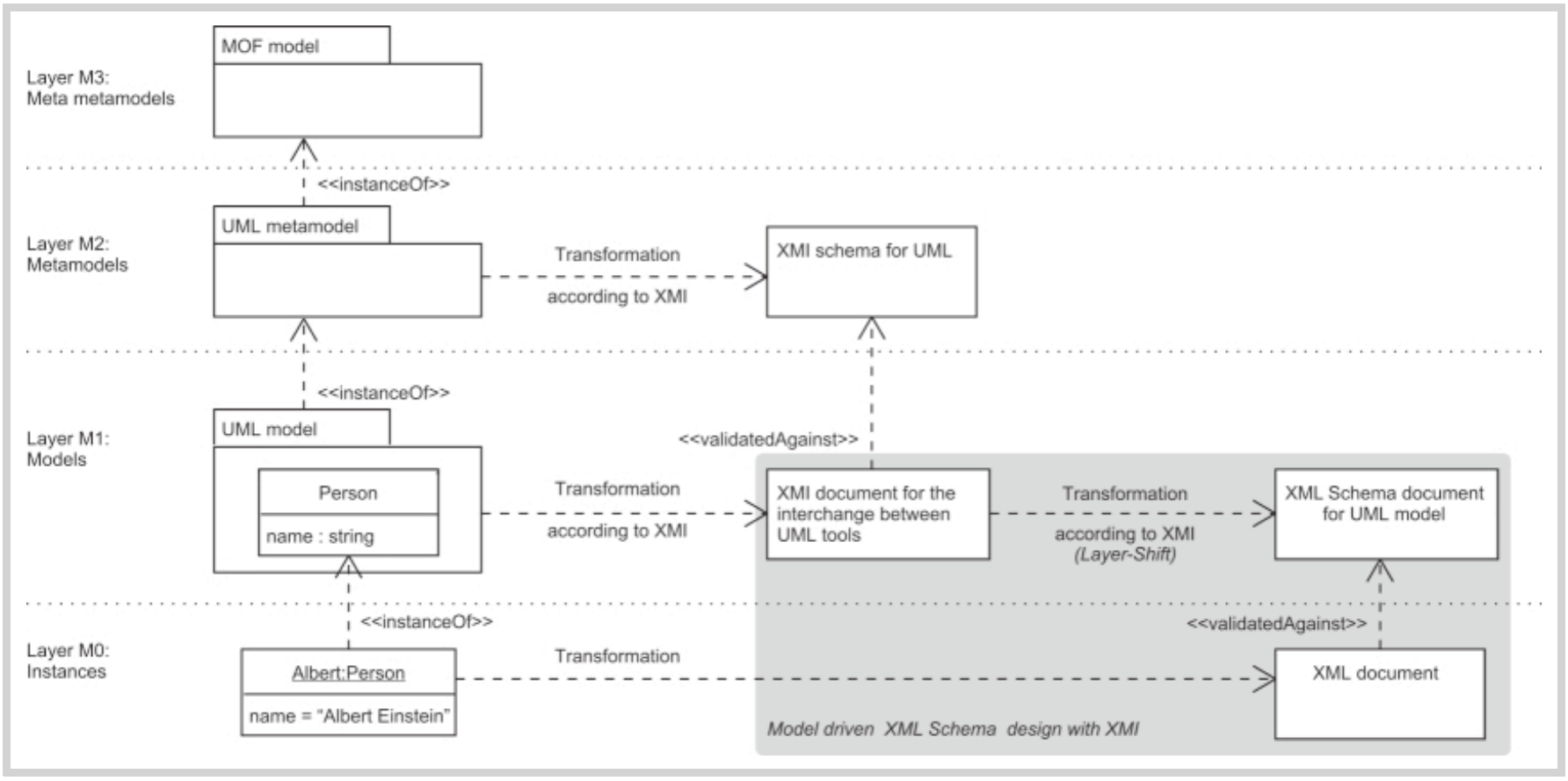
Transforming UML models to XML Schema using XMI

As we have briefly sketched out above, there already exists a standard for the generation of XML Schemas out of models namely XMI, the XML Metadata Interchange specification. With XMI, the OMG has standardized a data format that aims at the interchange of models. The goal is to support interoperability of UML modeling tools. There is urgent need for such support since, particularly in globally distributed software projects, a number of modeling tools are simultaneously in use. XMI not only supports UML models, but also MOF [OMGMOFCore] compliant models in general. It defines standards for the creation of so called XMI schemas, which represent MOF compliant metamodels such as the UML metamodel, and XMI documents, which represent the actual models such as UML models; the latter are instances of a metamodel's representation as an XMI schema. Thus XMI enables the validation of models in their XML representations against an arbitrary MOF compliant metamodel.

Since UML is designed using the MOF and because the UML metamodel itself is an instance of the MOF model [OMGMOF] , XMI can be (and is in most cases) used to exchange and validate UML models via XMI documents. As the names suggest, XMI documents are specific XML documents and XMI schemas are specific XML Schema documents. So when using XMI, UML models are actually transformed into XML documents, not into XML Schemas! Why that, didn't we state above that from the perspective of object persistence, we would expect conceptual UML models to be transformed into XML Schemas? That's still valid, but XMI was designed to validate and exchange models, and for that purpose it is best to transform models into XML documents; that is, document instances that can be validated against an XML Schema using common XML software.

Since XMI defines rules for generating XMI schemas (which are special XML Schema documents as we have seen) from MOF compliant metamodels, the question arises whether we can also use these rules to generate XMI schemas for conceptual UML models. The answer is yes. Because of the

architectural alignment of UML and MOF, conceptual UML models (models represented by UML class diagrams), are defined using the same concepts as MOF compliant meta models. [OMGMOFCore] Thus, we are able to interpret UML class diagrams as MOF compliant metamodels from an XMI point of view, which allows us to use the rules defined by the XMI specification to generate XMI schemas from conceptual models. We call this process, which effectively creates an XMI schema from a conceptual UML model encoded in an XMI document, *layer shift*. The attached figure illustrates the model-instance relationships, the mappings and the levels on which the models and their instances reside.



XMI schemas from UML class diagrams: The XMI mapping

In our research [PaganoDA] we have surveyed the XMI specification regarding its possibilities to map conceptual models to certain XML Schema documents, namely the XMI schemas. Our focus was to determine whether XMI allows a sound transformation of models for XML applications into XML Schema documents that represent their document classes. This question arises from the fact that XMI was designed with a completely different goal, namely the standardization of the interchange process of MOF compliant models.

First of all, although XMI targets MOF compliant models in general, the transformation of conceptual models in the form of UML class diagrams is possible due to architectural alignment in the form of the common infrastructure of UML and MOF [OMGUMLInfrastructure] . The OMG defines how to generate MOF-compliant models [OMGXMIMapping] as a set of predefined rules that can also be configured for special purposes. This configuration comprises the use of a specific set of so-called tagged values in the MOF models that are to be transformed. These tagged values are also defined in the XMI specification. In the following, we provide a short overview of the transformation rules specified in the XMI specification.

The general structure of an XMI schema is how we would expect an XML Schema document to look like, including namespace declarations and imports of XMI specific XML Schema types. By default, XMI software generates only one XMI schema for a complete UML model, without defining a target namespace therein. But the annotation of UML packages with an XML namespace results in the

generation of one XMI schema per UML package, each using the specified XML namespace as target namespace. This allows for the transformation of equally named UML elements within different packages, what would otherwise be impossible.

For UML packages XMI specifies the use of elements with locally defined complex types. Each complex type is then made up of a choice of the UML elements contained in the according package, which are referenced in the choice. Similarly, UML classes are transformed into element declarations and complex types, but in contrast to packages, the elements' types are declared globally. The elements and their types are put into the target namespace of the containing package. The content model of both packages and classes is very generous since it is realized as an XML Schema choice construct with open bounds, and thus hiding any multiplicity by default. But this representation can be changed using tagged values that enforce the use of the multiplicities as specified in the model.

UML attributes are transformed into element declarations, which are directly put into a choice in the definition of the complex type that represents the containing class. Additionally, an attribute declaration is appended for UML attributes with simple datatypes and a multiplicity of at most one. However, the type of declaration can be influenced using tagged values, for example to persist the order of the elements using sequences instead of choices. Every complex type by default references an additional attribute predefined in the XMI specific XML Schema types that is used as an identifier. However, by annotating a UML attribute with a certain tagged value, this attribute can be declared as the identifier attribute instead.

UML associations are defined using association ends that reside in the classes that participate in the association. XMI generates both an element declaration and an attribute declaration for each association end contained in a class. As the association end's type, XMI uses the predefined type “Any” that allows arbitrary content for the element. This allows the instantiation of classes in the content of other classes and therefore enables the construction of redundant but valid XML documents. We consider this redundancy a weak point of the XMI specification.

For inheritance, a specific type of association, XMI uses “copy-down-inheritance” [OMGXMIMapping] by default: The subclass is transformed into a complex type under the rules, but first the properties of all base classes are copied. Then these are extended or replaced by the properties defined in the subclass itself. The copy-down-inheritance is only a workaround for the realization of multiple inheritance that XML Schema does not support natively. In the sense of the term it is not really an inheritance mechanism, as the relation between base class and subclass gets lost during the transformation. If a UML model only uses simple inheritance, the use of the native XML Schema inheritance mechanism can be enforced by using the proper tagged values. In contrast to the way associations are represented, the use of copy-down-inheritance can't be seen as a drawback of XMI, because it is XML Schema that lacks the concept of multiple inheritance.

Adapting the XMI mapping to the document setting

Clearly the XMI specification was designed with the goal to facilitate the interchange and validation of models, not to represent conceptual models in the form of XML Schema documents. But most of the rules that transform UML model concepts into XML Schema concepts are sound from the perspective of a conceptual model by default or there are possibilities to influence the rules in a way that they become sound.

However, an important concept that is not considered by the XMI specification is abstraction, although abstract classes exist in the MOF specification. This may be a result of the aim of the XMI specification, since it should facilitate the interchange of models. This means XMI schemas shall facilitate the validation of models in the form of XMI documents, a process where abstraction plays an inferior role. Considering XML applications that need general schema concepts, abstraction would be desirable.

Another negative aspect in this context is the declaration of elements for classes, the use of *instance placeholders* as we call them. XMI creates an instance placeholder for each class without exception. This is always possible, but not always desirable in the context of XML applications, because it allows for a variety of valid XML documents implementing the schema.

Furthermore, the third drawback is the realization of associations in XMI, which allows for redundancy in instance documents. XMI however targets the validation of models regarding the definition of the according metamodel represented by an XMI schema. Hence, XMI does not emphasize well-structured and easy to use valid documents. Moreover, there are certain tagged values that make it possible to prevent element declarations for association ends and allow for the use of an URI-based linking construct that eliminates redundancy. Unfortunately the XMI specification doesn't make use of XLink [W3CRecXLink] at this point, which could be a helpful concept. This is because the XMI specification was created before XLink became a W3C standard.

Since the schema generation process can mainly be influenced using tagged values, we have to ask for a general set of tagged values to tailor the schema generation process for our needs. Having to annotate each and every model element with cryptic tags doesn't really simplify a software engineer's life. But leaving XMI aside, each transformation process has to be configurable on UML construct level to transfer design decisions into the schema.

Carlson [CarlsonXMLAppsUML] proposes the use of a special UML profile for XML Schema modeling purposes that defines according stereotypes and tagged values. For example he introduces the stereotype “XSDcomplexType” which applied to a class specifies that this class should be realized as a complex XML Schema type. But he also defines stereotypes and tagged values for other model elements like attributes or association ends (e.g. “XSDelement” or “XSDattribute”) and even stereotypes that target model groups whose function is to group other model elements (e.g. “XSDsequence”). This approach surely makes sense, but the terminology Carlson proposes for the profile belongs to the domain of XML Schema, not to the domain of modeling in general. Hence a software engineer has to have domain knowledge of XML Schema to ensure that the generated schema has the desired properties, and thus in fact has to model the schema, not the application. Here we consider it more useful and more general to hide such tags and preset their values sensibly from the perspective of XML applications or modeling in general. This way important patterns, principles and procedures could be ensured and the profile could be made more UML or model specific than XML Schema specific.

The major benefits of using XMI schemas are obviously the possibility of reusing an existing specification and the fact that most CASE tools currently support exporting their models as XMI documents. The latter fact is important when considering a way to make practical use of the XMI rules to create XML Schema documents from UML models in software engineering processes.

Realizing the layer shift using XSLT

XMI compatible UML CASE tools provide the possibility to export models in an XML based, standardized format, namely an XMI document. This is a welcome fact for the realization of the layer shift, which shall create an XMI schema from a conceptual UML model encoded in an XMI document. It allows for the insertion of an additional transform layer after the creation of the model during the development process. This means that we get models in a well defined format, which we can assume as input format for a transformation. It comes in handy that XMI documents are in fact XML documents, so we can use an XSLT stylesheet to transform a UML model encoded in an XMI document into XML Schema.

Naturally the question arises, whether we cannot simply influence the schema generation process of a UML tool to directly transform a UML model into an XMI schema. This question is inappropriate, though, because UML modeling tools with XMI functionality exclusively create XMI documents, not XMI schemas. The generation of XMI schemas is thus not part of the functionality of the mentioned tools, and hence has it to be done separately after the export of a model.

We realize the layer shift with an XSLT 2.0 stylesheet that targets XMI version 2.1 and above. Hereby we focus mainly on the XMI specification, which describes the transformation of MOF models into XMI schema documents, but we interpret UML models in particular. As we have found out, tagged values (amongst others) are realized in different ways by different tool manufacturers. But since they are most important for the tailoring of the schema generation process, the XSLT stylesheet has to be able to interpret them. As a matter of fact, a general interpretation is out of scope for our project and is basically difficult to achieve because of the high degree of flexibility within XMI. However, we support a number of some of the most popular tools and integrate an extension point in the stylesheet that makes it easy to provide additional interpretation modules for further tools. With our implementation we provide a conceptually sound open source realization of the layer shift that can be used in other processes.

To make it easy for reuse, we have designed our stylesheet to align with the translation rules defined by the OMG [OMGXMIMapping] by creating several templates that correspond to the several subrules and that have the same names as the according rules in the official mapping document. Besides interpreting some partially unclear points of the mapping specification like the handling of package level attributes, we focus on realizing the mapping as consistent with the specification as possible. Furthermore we implement features that allow for more flexibility than the specification or even provide additional functionality. Examples are the possibility to use substitution groups for derived model elements, the ability to generate model based name prefixes or the use of XLink for the realization of associations. To enable the use of certain configurations of tagged values, we also provided parameters that allow to set a default value for each tag.

Realizing the layer shift according to the XMI specification naturally leaves its mark in the resulting schemas: There are many additional elements and attributes that solely support the interchange process or provide extension points. These XMI-specific schema content can be removed for schemas that are generated in the case of document engineering.

Conclusions and related work

In this paper, we have presented a method uml2xsd that transforms conceptual data models from their

UML representations into XML Schema documents. From a software engineering point of view, our method provides an implementational counterpart to data models that constrains the XML implementations of model instances (data). Our method provides a missing piece for the puzzle of data modeling, extending good practice from object-oriented programming languages and databases to document engineering.

Clearly, our method provides a missing link in schema engineering: Data modeling can be integrated into system modeling, independently of data implementation technology.

The implementation of xml2xsd builds on XMI, a standard for model interchange that is widely supported by UML modeling tools. It follows mapping rules from data models to XMI schemas that the OMG has conceptually defined, providing the first publicly available implementation of these rules. Our implementation is open source, configurable and extensible. Hence, uml2xsd puts forward engineering into the toolbox of schema engineers.

Speaking of schema engineers, even if some project confines itself to pure schema development, without any ties to software development, our method provides two strong benefits: First, we enable modeling to be carried out independently of schema technologies. Second, in the vein of forward engineering, schemas can be generated automatically from models.

We consider the presentation [KimberEtAlUML2DTD] of Kimber and Heintz at Extreme Markup Languages 2000 as a direct predecessor of our work. We fully subscribe to their position on the relationships between software engineering, document engineering and XML technology. In our work, we have taken two further steps: First, with XMI we base our mapping from model to schema on an OMG standard, also facilitating configuration and extension of the XMI mapping. Second, we offer an implementation that works within any (reasonable) UML environment, requiring, with XSLT, only widely available XML technology.

The work that we have presented here is part of a larger endeavor in which we, the Engineering Publishing Technology Group, explore to what extent novel publishing applications can be composed from appropriately configured XML software with a minimum of programming. Our goal is to discover principles, patterns and procedures that reduce complexity and ensure sustainability when developing and maintaining Web applications. Modeling of persistent data and forward engineering of data models into XML schemas are one building block of a metamodel for publishing applications that Thomas Schöpf, another member of our group, develops in his PhD thesis “Eine Software-Architektur für webbasierte Publikationssysteme”.

References

[AbiteboulEtAlDataOnTheWeb] S. Abiteboul, D. Suciu: *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann 1999.

[BrüggeEtAlsOOSWE] B. Brügge and A.H. Dutoit. *Object-Oriented Software Engineering: Using UML, Patterns and Java*. Second Edition, Prentice Hall, 2004.

[EPTatEML2007] A. Brüggemann-Klein, Th. Schöpf, K. Toni: *Principles, Patterns and Procedures of XML Schema Design — Reporting from the XBlog Project*. Extreme Markup Languages 2007. Available from http://conferences.idealliance.org/extreme/.

[EPTatBalisage2008] A. Brüggemann-Klein, L. Singer: *Hypertext Links and Relationships in XML Databases*. Presented at Balisage: The Markup Conference 2008, Montréal, Canada, 2008, http://www.balisage.net/. Available from http://hyquery.ls-

softworks.de/HyperDataSystemsBalisage2008.pdf. doi:10.4242/BalisageVol1.Bruggemann-Klein01.

[CarlsonXMLAppsUML] D. Carlson: *Modeling XML Applications with UML: Practical E-Business Applications*. Redwood City, Addison Wesley Longman Publishing, 2001.

[ConradEtAlXMLConceptualModeling] R. Conrad, D. Scheffner, J.-C. Freytag. XML Conceptual Modeling Using UML. In A.H.F. Laender, S.W. Liddle, V.C. Storey (eds), *International Conference on Conceptual Modeling (ER 2000)*. LNCS 1920, pp. 558—571. Springer-Verlag 2000.

[EcksteinEtAlXMLDatenmodellierung] R. Eckstein, S. Eckstein. *XML und Datenmodellierung*. DPunkt-Verlag 2004.

[KhanEtAlSchemaDesignPatterns] A. Khan, M. Sum. *Introducing Design Patterns in XML Schemata*. Sun Developer Network 2006.

[KimberEtAlUML2DTD] W.E. Kimber, J.D. Heintz: *Using UML To Define XML Document Types*. Presentation at Extreme Markup Languages 2000. Available from http://www.gca.org/attend/2000_conferences/Extreme_2000/.

[LainevoolXMLPatterns] T. Lainevool. *Develop Effective XML Documents Using Structural Design Patterns*. http://www.LainevoolXMLPatterns.com/.

[MalerEtAlDTDModeling] E. Maler, J. El Andaloussi. *Developing SGML DTDs: From Text to Model to Markup*. Prentice Hall 1995.

[OMGMOF] Object Management Group: *Meta Object Facility (MOF) Specification*. Version 1.4.1, July 2005. http://www.omg.org/docs/formal/05-05-05.pdf

[OMGMOFCore] Object Management Group: *Meta Object Facility (MOF) Core Specification*. Version 2.0, January 2006. http://www.omg.org/docs/formal/06-01-01.pdf

[OMGXMIMapping] Object Management Group: *MOF 2.0/XMI Mapping*. Version 2.1.1. December 2007. http://www.omg.org/docs/formal/07-12-01.pdf

[OMGUMLInfrastructure] Object Management Group: *OMG Unified Modeling Language (OMG UML)*. Infrastructure, V2.1.2. November 2007. http://www.omg.org/docs/ formal/07-11-04.pdf

[PaganoDA] D. Pagano. *Modeling and Defining XML Applications with UML and XML Schema*. Diploma Thesis, Technische Universität München, 2008.

[StephensonBestPractices] D. Stephenson. *XML Schema Best Practices*. HP Dev Resource 2004. http://devresource.hp.com/drc/resources/vdVlistXMLSchemaBestPractices.jsp

[vdVlistSchemaTechnologies] E. van der Vlist. *Comparing XML Schema Languages*. XML.com 2001. http://www.xml.com/lpt/a/884.

[vdVlistXMLSchema] E. van der Vlist. *XML Schema*. O'Reilly 2002.

[W3CRecXLink] World Wide Web Consortium: *XML Linking Language (XLink)*. W3C Recommendation 27 June 2001. http://www.w3.org/TR/xlink/

[W3CRecXMLSchemaStructures] World Wide Web Consortium: *XML Schema Part 1: Structures Second Edition*. W3C Recommendation 28 October 2004. http://www.w3.org/TR/2004/REC-vdVlistXMLSchema-1-20041028/

Dennis Pagano
Chair for Applied Software Engineering, Technische Universität München

Dennis Pagano received his diploma in computer science in August 2008 from Technische Universität München. He currently works for the Chair for Applied Software Engineering at Technische Universität München as a research assistant and doctoral candidate. His research interests in the field of Software Engineering range from modeling, meta modeling and semantics to knowledge management and representation as well as intelligent learning. One of his special interests is the connection between

Software Engineering and XML based technologies, in particular MOF/UML and XML schema.

Anne Brüggemann-Klein
Engineering Publishing Technology Group, Technische Universität München

Anne Brüggemann-Klein is a professor of computer science at Technische Universität München. She received her PhD in Mathematics from Friedrich-Wilhelms-Universität Münster and her Habilitation in Computer Science from Albert-Ludwigs-Universität Freiburg. Her research interest is in document engineering. Earlier work, part of which is cited in the W3C XML Recommendation, focuses on the formal language theory foundation of document languages. Current research explores to what extent novel publishing applications can be composed from appropriately configured XML software with a minimum of programming. The goal is to discover principles, patterns and procedures that reduce complexity and ensure sustainability when developing and maintaining Web applications.