Balisage: The Markup Conference 2010

Proceedings

Reverse Modeling for Domain-Driven Engineering of Publishing Technology

Anne Brüggemann-Klein
Fakultät für Informatik, Technische Universität München

Tamer Demirel
Fakultät für Informatik, Technische Universität Münchenn

Dennis Pagano
Fakultät für Informatik, Technische Universität München

Andreas Tai
Fakultät für Informatik, Technische Universität München

Balisage: The Markup Conference 2010
August 3 - 6, 2010

How to cite this paper

Brüggemann-Klein, Anne, Tamer Demirel, Dennis Pagano and Andreas Tai. “Reverse Modeling for Domain-Driven Engineering of Publishing Technology.” Presented at Balisage: The Markup Conference 2010, Montréal, Canada, August 3 - 6, 2010. In *Proceedings of Balisage: The Markup Conference 2010*. Balisage Series on Markup Technologies, vol. 5 (2010). DOI: 10.4242/BalisageVol5.Bruggemann-Klein01.

Abstract

We report in this paper on a technique that we call reverse modeling. Reverse modeling starts with a conceptual model that is formulated in one or more generic modeling technologies such as UML or XML Schema. It abstracts from that model a custom, domain-specific meta-model and re-formulates the original model as an instance of the new meta-model. We demonstrate the value of reverse modeling with two case studies: One domain-specific meta-model facilitates design and user interface of a so-called instance generator for broadcasting productions metadata. Another one structures the translation of XML-encoded printer data for invoices into semantic XML. In a further section of this paper, we take a more general view and survey patterns that have evolved in the conceptual modeling of documents and data and that implicitly suggest sound meta-modeling constructs.

Taken together, the two case studies and the survey of patterns in conceptual models bring us one step closer to our superior goal of developing a meta-meta-modeling facility whose instances are custom meta-models for conceptual document and data models. The research that is presented in this paper brings forward a core set of elementary constructors that a meta-meta-modeling facility should provide.

Table of Contents

Introduction

Modeling is a pivotal activity in the engineering of software systems; it is the key method to deal with complexity, adaptivity requirements 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. One focus has been on the modeling of persistent data [BST07,ES08,PB09]. In a 2009 Balisage paper [PB09], we have discussed the relationships between conceptual data models and their implementations in an XML schema language; furthermore, we have, following current thinking in the field, posed three requirements for the modeling of persistent data:

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

Our previous work and the three requirements cited above have been inspired by a 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, that has found its expression in the approach of model-driven architecture [MSUW04]. Recently, the role of modeling in software engineering has been even further emphasized: The emerging software engineering discipline of domain-driven design [E04] puts domain models at the heart of software development, letting them delineate so-called ubiquiteous languages that are used by domain experts and software engineers alike and that are embodied both in the code and in the user interface of systems. Hence, we should like to replace the last requirement in the list above by the following item:

- Models of persistent data should be amenable to being embodied in every aspect of the system.

Over the last two years, in two projects we have deviated from the much-travelled road of modeling to support forward engineering. In fact, in both projects, we apply a technique that we call reverse modeling. Starting with a pre-existing conceptual model that is formulated in one or more generic modeling technologies such as UML or XML Schema, we abstract from that model a custom, domain-specific meta-model and re-formulate the original model as an instance of the new meta-model. In both cases, the new domain-specific meta-model explicitly captures salient points of the model that come to bear on system and interface design. We are convinced that in both cases reverse modeling leads to a better understanding of the domain and to a more flexible system that is resilient against future changes in the model.

This paper is organized into four further main sections. In the next section, we present our main case

study. More precisely, we apply reverse modeling to broadcasting productions meta data and demonstrate, how the domain-specific meta-model that we develop reduces complexity when building a so-called instance generator for broadcasting productions meta data. In the section after that, we reflect on reverse modeling, arguing for domain-specific meta-models. The next section applies reverse modeling in the context of translating XML-encoded printer data of invoices into semantically meaningful XML. In this case study, the meta-model that we develop structures the procedural method of the translator. In a further section of this paper, we take a more general view and survey patterns that have evolved in the conceptual modeling of documents and data and that implicitly suggest sound meta-modeling constructs. Taken together, the two case studies and the survey of patterns in conceptual models bring us one step closer to our superior goal to develop a meta-meta-modeling facility whose instances are custom meta-models for domain document and data models. They bring forward a core set of elementary constructors that a meta-meta-modeling facility should provide.

An instance generator for broadcasting productions metadata

In his Master's Thesis [T09] project, Andreas Tai deals with several incarnations of a huge data model for broadcasting productions metadata, the Broadcast Metadata Exchange Format (BMF). The model incarnations are formulated in a number of meta-model technologies, among others UML, XML Schema and a domain-specific registry format; these incarnations are aligned with each other but are not equivalent, and the relationship between them is not documented in detail. Project goal is a so-called BMF instance generator; that is, editing support for XML instances of metadata that illustrate sections of the model by example. Intended users are domain experts who explore sections of the model by generating XML instances when devising mappings from other metadata models into BMF. The initial users are XML experts; yet it is considered a valuable asset if later users need to be conversant neither in XML nor in XML Schema. Use of typical schema-driven XML editors has been tried out, but has not been found particularly helpful; hence, it is ruled out. The BMF instance generator must be resilient against changes in the model. A domain-specific meta-model that is constructed by what we call reverse modeling is pivotal to the solution, as explained below.

Broadcasting productions metadata and their use cases

The Broadcast Metadata Exchange Format (BMF) is a data model that has been developed over nearly a decade by the Institut für Rundfunktechnik (IRT, Institute for Broadcasting Technology). A universal format, BMF enables the platform independent exchange of metadata in IT-based TV production, serving as a lingua franca that intermediates between different metadata formats in the domain of TV production.

Briefly, BMF consists of types of different metadata sets such as *Programme* that group metadata elements such as *MainTitle* or *Award*. Metadata elements associate with simple data types or with other metadata sets. Figure 1 demonstrates that metadata element *MainTitle* is associated with simple type *String* and that metadata element *Award* is associated with a metadata set that is also called *Award*.

Figure 1: A segment of the BMF model in UML class-diagram notation

BMF is formulated with different modeling technologies. It was first developed as an entity-relationship diagram which later has been replaced with a UML class diagram. To leverage data exchange by means of XML documents an XML Schema was dynamically generated from the BMF UML class diagram. Furthermore the data model is currently being registered at the Society of Motion Picture and Television Engineers (SMPTE), which requires its own format for data models [SMPTE03,SMPTE01,SMPTE08].

BMF meets its proclaimed goal of covering every use case of metadata exchange in the domain of TV production at the price of being large and complex. Yet there are related use cases that BMF on its own does not support well, due to its complexity. In this paper, we are concerned with the specific such use case of generating sample XML-encoded metadata that conform to BMF, e.g. for the purposes of documentation or of designing mappings between BMF and other metadata formats. Each of the sample documents provides a keyhole view into a small part of BMF. What is needed tool-wise for generating such sample documents is a facility to deeply zoom into parts of BMF, to select a few illustrative building blocks and to instantiate them, the focus being on structures that may be filled with ad-hoc or dummy content.

In the experience of the BMF modeling team at IRT, generic schema-aware XML editors are inadequate tools for the specific task of generating sample metadata. This seems to be due to the discrepancy in the sizes of the large BMF model and the relatively tiny focus that is relevant for the sample, making it quite time-consuming to generate sample documents even for XML experts who otherwise routinely use schema-aware editors.

Hence, the group decided to look into specific editing support for BMF sample instances, initiating the project “BMF instance generator”, which became the topic of Andreas Tai's Master's Thesis T09 at TU München.

There are two main requirements for the instance generator: First, it must be resilient against changes in the BMF model. More precisely, it must be able to adapt itself to any new version of the BMF XML Schema. Second, it must cut down time and effort to generate sample metadata that conform to the BMF model.

In the remainder of this section, we report on design and implementation of the BMF instance generator before drawing conclusions.

The quest for the domain model

The leitmotiv question that needs to be addressed in the project is: What makes the generation of BMF instances so difficult? Apparently, problems that arise from the inherent complexity and sheer size of the BMF model are aggravated by the choice of technologies that is used to formulate the model, namely UML and XML Schema. These technologies bring a complexity to the model that is more due to the intrinsic logic of these technologies than to the underlying domain model that they express. They obscure the domain model of BMF. Furthermore, the different technologies that are in use to represent model information lead to inconsistent conclusions about the BMF domain model itself. One point of inconsistency is the order of metadata elements in metadata sets: In the UML class diagram, metadata elements are designed as class attributes which in UML are unordered; in the XML Schema metadata elements are attached to metadata sets within sequence content models that impose order on them.

From this analysis, there arises naturally the question what the real domain model of BMF is. In order

to get the domain aspect of the model back into focus, we need to strip BMF from the specifics of the modeling technologies that it is formulated in and to arrive at a more direct formulation of the model than the BMF UML class diagram and BMF XML Schema provide. In a process that we call reverse modeling, we first identify the building principles that constitute BMF. These building principles must meet the following requirements:

- They apply to every incarnation of BMF.
- They represent the concept that the modeling team of BMF has of the model.
- They are expressed as directly as possible, and certainly independently of implementation technology.

Naturally, there is no magic wand for reverse modeling. We just have to apply sound software engineering practices. First, we carefully analyze the existing incarnations of BMF, namely the BMF UML class diagram and the BMF XML Schema as well as the registration data for BMF at SMPTE. Second, we conduct extensive interviews with the modeling team of BMF on the conceptual ideas behind the model. The result is a small set of rules the most important of which we describe in natural language:

- BMF consists of types of metadata sets that group metadata elements.
- The types of metadata sets are partitioned into packages.
- Metadata elements are unordered
- Metadata elements reference either simple data types or metadata sets.
- The existance of a metadata set that is referenced from a metadata element can be of one of two types: dependent on or independent of the existence of the referer.

We view the building principles as a model for the domain model and hence as a domain-specific meta-model. With such a domain-specific meta-model, we restrict the terminology of the model to the domain. Instead of generic terms such as classes, attributes or complex type definitions we only use terms of the meta-model such as metadata set and metadata element.

The meta-model reduces complexity in two ways: First, it facilitates a direct expression of the model that is independent of implementation technology. Second, it explicitly defines and limits the building principles that are used in BMF, thus imposing structure on the domain model and, hence, raising understanding to an accidental to a more principled level.

Leveraging the domain-specific meta-model for the instance generator

Our quest for the BMF domain model has led us to discover a domain-specific meta-model of which the BMF domain model is an instance. We demonstrate in this section how we can leverage the meta-model for the instance generator, reducing complexity and making the instance generator resilient against changes in the BMF model.

We structure the instance generator into the following three components:

- **Model advisor**: a component that provides all model-related information.
- **Shopping mall**: a component that lets users select model elements for later use.
- **Building site**: a component that enables users to build instances of the BMF model.

To illustrate the role of the domain-specific meta-model we look a bit closer at the implementation details of the three components.

The model advisor is the key component of the instance generator. The component provides an interface to answer all questions that might be directed at the BMF domain model, for example:

- Which metadata sets are available?
- Which metadata elements are required for a metadata sets?
- Does a metadata element reference a simple type value or another metadata set?

By the project requirements, the instance generator and, hence, the model advisor must refer to the BMF XML Schema for model information. Hence, two questions arise: First, the access question, how do we query XML Schema? Second, the interface question, how do we relate the BMF domain model to the BMF XML Schema?

As to the access question, one would expect to be able to use XML technology for that. Indeed, the work of Wilde and Michel on SCX, an alternative XML representation of XML Schema [WM07c], and SPATH, an XSLT library to query XML schema components [WM07a,WM07b], looks promising. Unfortunately, there is no full implementation yet, and development seems to have been discontinued. Therefore, we resort to a programming-language interface to XML Schema, namely the Java framework Eclipse XSD. Eclipse XSD is part of the Eclipse Modeling Framework (EMF) and closely mirrors the abstract data model of XML Schema.

As to the interface question, technically, the model advisor exposes a query interface of Java methods for dynamic access of the BMF XML Schema. The model advisor's class model defines the interface in terms of the domain meta-model, with query methods refering to domain concepts such as metadata sets and metadata elements, not to XML Schema concepts such as type definitions and content models; see Figure 2. The implementation is based on Eclipse XSD.

Figure 2: Signature of a Java method that queries the BMF domain model

```
List<String> getMetadataSetNames(String packageName)
```

Hence, the query interface encapsulates the mapping between the BMF domain model and its XML Schema incarnation, and the model advisor's system of classes acts as an adaption layer that takes care of changes in the BMF XML Schema. Thus, the model advisor is resilient against changes in the BMF XML Schema as long as new versions of the schema still conform to the domain meta-model.

To illustrate this with examples, the system will adapt if new types of metadata sets are defined or the selection of metadata elements within a type of metadata set is modified, because such a change in the domain model only requires Eclipse XSD to refer to a different schema at startup time or even at run time. The system will, however, not be able to adapt if a new kind of metadata element container is introduced that requires metadata elements to be ordered, because such a change in the domain meta-model demands an extension of the query interface.

With its stacked architecture of access layer based on Eclipse XSD and adaption layer implementing a domain-specific query interface, the model advisor dynamically exposes the BMF domain model in its XML Schema incarnation — robust in the face of changes in the domain model that respect the boundaries of the domain meta-model.

The other two components of the instance generator, namely the shopping mall and the building site, obtain the model information they need through the model advisor.

The shopping mall component of the instance generator offers a graphical user interface to explore the BMF domain model and to select the metadata sets and metadata elements of interest; see Figure 3 and Figure 4. The selected model elements are put in a shopping cart for later use in the instance generation process. The shopping-mall interface is dynamically created by querying the model advisor. Whereas the model advisor gives the programmer access to the domain model, the shopping mall visualises the domain model for the end user.

Please note how the shopping mall interface integrates domain concepts into the shopping-mall metaphor.

Figure 3: The shopping mall user interface Shopping for metadata sets

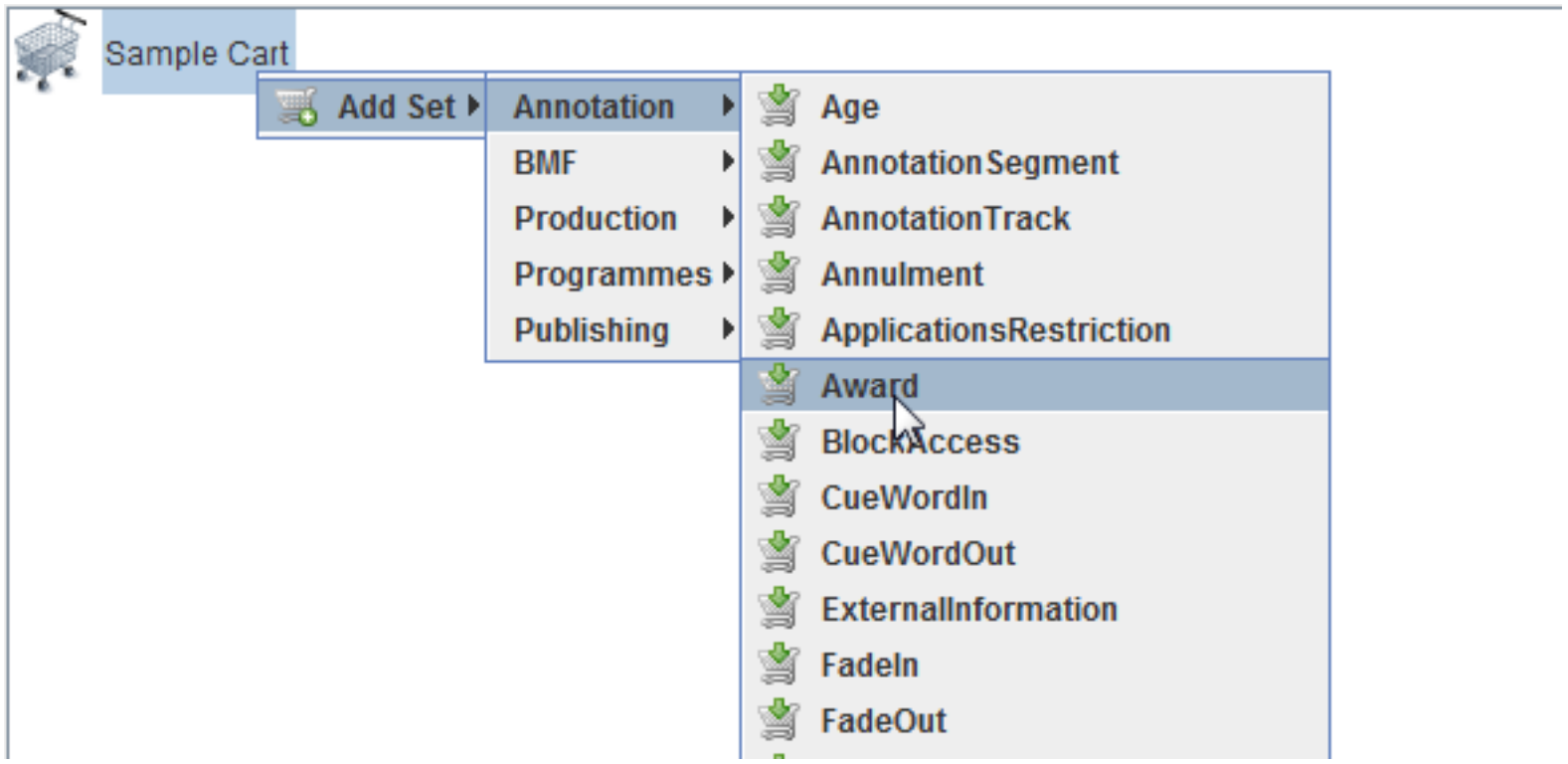
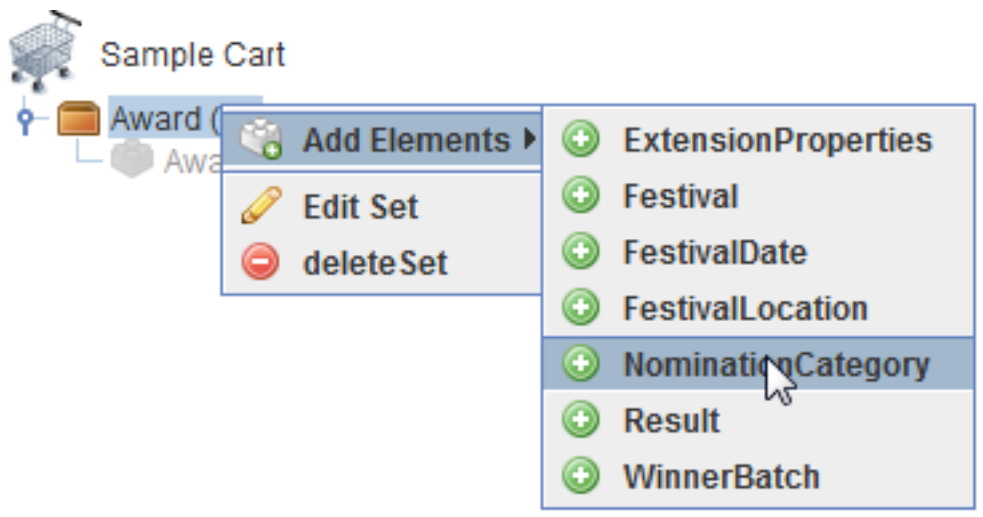


Figure 4: The shopping mall user interface Adding metadata elements to metadata set

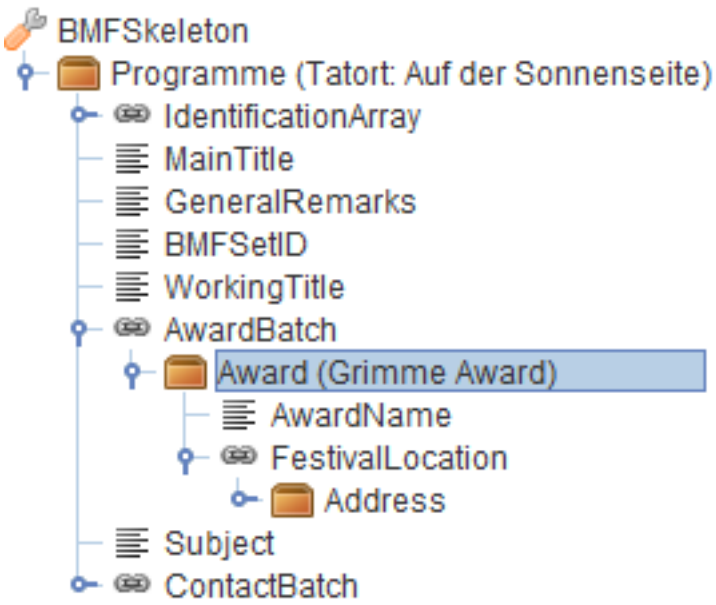


The shopping-mall interface has been adopted for uses beyond the task of instance generation. The collecting of metadata sets and metadata elements into a shopping cart provides a new way of browsing through the domain model, notably advancing ease of access to the model even for the modeling experts at IRT.

We are now turning our attention to the third instance generator component, the building site. This component is able to open a shopping cart that has previously been saved in the shopping mall. It tries then to build a BMF instance from the cart's content, following user hints that might also be stored in

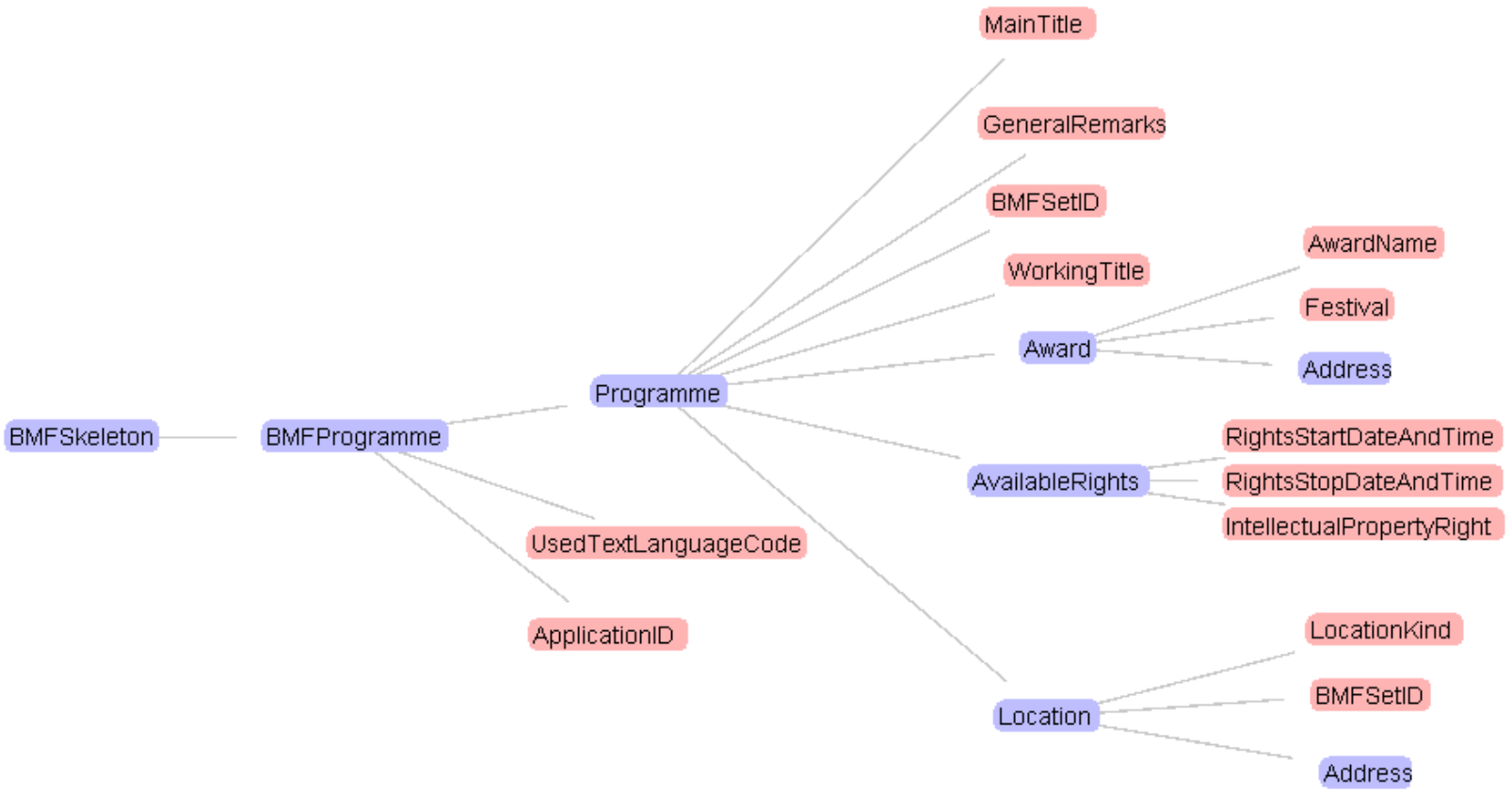
the cart (see Figure 5). Afterwards, it offers a graphical user interface that lets users edit BMF instances with the help of context-sensitive menues that also are created dynamically and indirectly, by querying the domain model through the model advisor.

Figure 5: The building site user interface Displaying an automatically generated BMF instance



Once more, the objects that the user is dealing with, in this case metadata set and metadata element instances, are presented directly as domain objects, independently of implementation technology. The target representation of an XML instance that conforms to the BMF XML Schema is created by an export modul. This divide-and-conquer strategy pays itself off when further output formats need to be supported. As a case in point, the building site component offers alternative export as a GraphML document that can then be visualized with the helpf of the prefuse toolkit; see Figure 6. This kind of visualisation proves to be instrumental in understanding the inter-dependencies of model elements in BMF, again on a conceptual domain level.

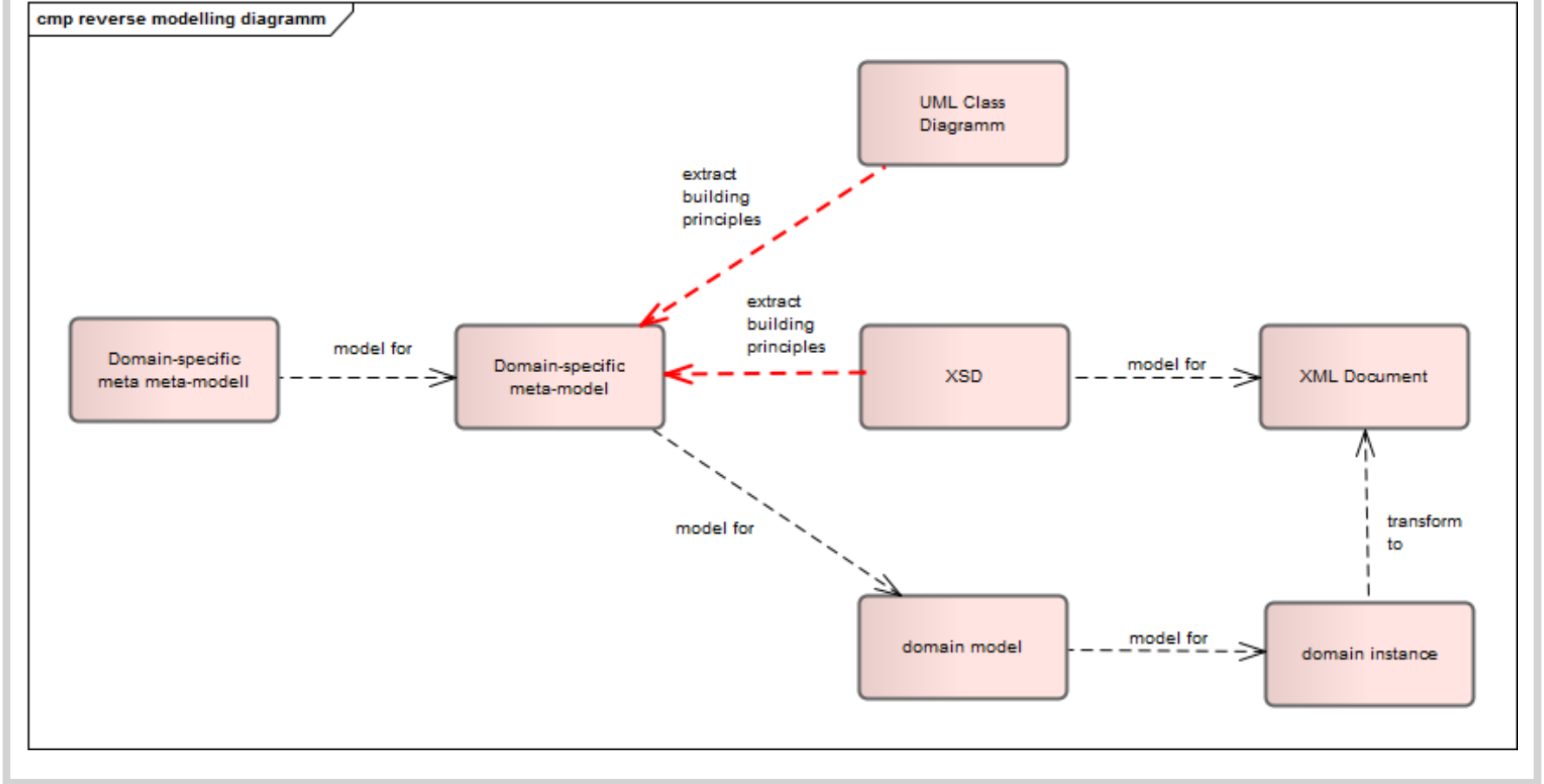
Figure 6: Visualizing a BMF instance in GraphML format



Conclusion

In this section, we have presented the method of reverse modeling, as summarized in Figure 7. Working backwards from pre-existing model incarnations that only implicitly define the domain model, we have identified the domain-specific building principles of the implied domain model, thus coming up with a domain-specific meta-model, from which an explicit domain model can be derived by instantiation.

Figure 7: Reverse modeling



We have also demonstrated how we can follow a divide-and-conquer strategy and leverage the domain-specific meta-model for the BMF instance generator, so that the two main project requirements are met. Consistently reflecting the domain-specific meta-model in the system design and the user interface also leads to the unexpected but highly welcome benefit that also domain experts who are not conversant in XML technology can work easily and productively with the BMF instance generator.

It is also worth mentioning that reverse modeling and the resulting domain-specific meta-model raised the level of understanding of the BMF data model. They made a very sophisticated data model more usable and manageable. We wish to emphasise that usability is not only an important factor in the design of user interfaces. It is also a decisive condition for the adoption of a new data model. In our experience, the focussing on the domain aspect through reverse modeling has made BMF more accessible to domain users, especially to those who do not have knowledge in modeling technologies such as UML or XML.

Reflection: Models, meta-models and domains

As the name suggests, reverse modeling constructs a domain model working backwards from one or more legacy incarnations of a pre-existing model, that may be defined only implicitly through implementation languages such as XML Schema. We have found in the BMF instance generator project that the real value of reverse modeling lies not so much in the conceptual model itself but in the domain-specific building principles that allow conceptual entities to be composed into larger entities. Such building principles are customarily considered to be part of the meta-model. This raises the question how domain-specific building principles AKA constructors on the meta-model level are best handled when modeling a domain.

By way of an example, let us suppose that we wish to model some entity as a rigid record structure that calls for a number of named sub-entities, exactly one for each name from a pre-defined list, without restricting the order. In the case that the sub-entities are each simply typed, such record structures correspond precisely to tables in first normal form in the relational data model.

Naturally, such a model can be expressed in most modeling languages or meta-models, including UML,

XML Schema, and Relax NG, and so can a more flexible record structure that allows sub-entities to be optional, or an even more rigid record structure that imposes order on the sub-entities. The point is that generic meta-models such as UML or XML Schema differentiate between these variants of record structures only implicitly, for example by consistently affixing specific occurence indicators to sub-entity declarations. In a way, they simulate a specific variant of record structures with their generic operators without making the common building principles behind them explicit by at least naming or ideally formally defining them.

The common ground between two record structures, one with sub-entities A, B and C and the other with sub-entities D an E is the fact that they *are* record structures. What is missing in meta-models such as UML, XML Schema or Relax NG for this scenario is the ability to explicitly provide a building principle for record structures that can be named in a domain-specific way and that can be parameterized by sub-entities such as A, B and C in one case and D and E in the other, similarly to parameterized types or classes in programming languages.

In contrast, in a meta-model that is customized to a domain that exhibits rigid record structures we envision a custom operator that directly and explicitly constructs such a record structure from its ingredients. In another scenario that displays more flexible record structures, again there would be a custom operator that caters to flexible record structures. And in a scenario that exhibits both rigid and flexible record structures, there would, of course, be two custom operators, one for each type of record structure. We require that in the meta-model we can customize operators: We wish to name them and to customize their sub-components as fits the domain.

In the BMF meta-model, we make use of a set operator who assembles named members that each can be either of simple type or of set type. We have customized it by naming the set operator “metadata set” and its members “metadata elements”. The BMF model instantiates the meta-model by recursively giving names to metadata sets and their members and by also stating members' types.

We argue in this paper that, compared to generic meta-models, domain-specific meta-models that directly and explicity express domain-specific building rules may give rise to more concise and less complex models that better capture the essentials of a domain and better support system development.

Translating XML-encoded printer data for invoices into semantically meaningful XML

In this section, we briefly present a second case study even though it is not as fully worked as the BMF study. Yet we feel that it helps to prove the point of the paper.

In his Diploma Thesis [D10] project, Tamer Demirel tackles the problem of up-converting XML-encoded printer data for invoices into semantically meaningful XML. The XML-encoded printer data are gathered from PDF versions of the invoices with the help of publicly available XML printer drivers, with the XML data representing the invoice data at page level, in terms of positions, font information and text strings. The target XML documents need to conform to an XML Schema for invoices.

Related invoice data from a single source share a common layout that facilitates analysis and offers the opportunity to up-translate the low-level printer data into higher-level, semantically meaningful XML data. An XSLT transformation program can be controlled by configuration data that express the common layout. In his Diploma Thesis project, Tamer Demirel designs and implements an Excel tool that enables clerical workers to capture the common layout of related invoice data. These data are saved

in XML format and then transformed, via XSLT, into an XSLT program that up-translates any invoice data instances that conform to the common layout from the printer data into established workflows “on the side”, without interfering with non-XML practices.

How is all this related to the topic of this paper, reverse modeling? The link is in the XML Schema for the target invoice data. This schema models a flexible record structure similarly to the one we have presented in the reflection section of this paper, with the twist that some field values are not of primitive but of tabular type. Tables generalize flexible record structures by allowing for repeated sets of values. This description gives rise to a meta-model for invoice data that structurally happens to be quite similiar to the meta-model for BMF data that we have introduced earlier. Note, though, that the meta-model is customized to the business domain in its use of terminology.

In Tamer Demirel's Diploma Thesis project, the meta-model serves a double purpose: First, it structures system design and algorithms for data analysis by dividing all tasks into the two separate concersn of record structures and tables. Second, it makes the work resilient against change in the schema, since it can be adapted to any schema that follows the meta-model.

Patterns as a source for meta-models

In the previous two sections, we propose two custom meta-models for domain models and demonstrate how to leverage them for two specific publishing applications. In this section, we take a more general view and survey patterns that have evolved in the conceptual modeling of documents and data and that implicitly suggest sound meta-modeling constructs. Taken together, these three sections bring us one step closer to our superior goal to develop a meta-meta-modeling facility whose instances are custom meta-models for domain document and data models. They accomplish that by identifying a core set of elementary constructors that a meta-meta-modeling facility should provide.

Maler and El Andaloussi in their early seminal work on document modeling [MA95] propose to classify document constituents into one of four categories, which we call metadata, organizational items, information items and information snippets. Document constituents in any of the four categories exhibit specific structures which today would be called patterns.

Metadata are commonly blocked together and associated with the document as a whole or with its major divisions, but may also be associated with more fine-grained document constituents. As the name “metadata” implies, they hold information about a document constituent rather than being part of it. Typical high-level metadata are author, publisher, publication date and so on as standardized by the Dublin Core initiative and others. Examples for low-level metadata are the height, depth or format of a picture. Metadata are typically structured as the flexible type of records that we have used above as an example in the reflection section above.

Organizational items structure a document into high-level units. They typically form a hierarchy, of which each level is organized as a sequence of specific and often repeatable items. Typical organizational items are books with frontmatter, a number of chapters and backmatter, of which each chapter is organised into a title, a number of introductory paragraphs and a number of sections.

Information items are smaller units of discourse that can be semantically understood out of context, such as paragraphs, lists, or quotations. Characteristically, an organizational item of the lowest level will be allowed to contain an arbitrary number of information items whose type may be freely chosen from a repertoire.

Information items may be shallowly organized into sub-items, as a list is organised into list items, but will eventually contain just text, possibly mixed with the smallest and lowest type of document constituents, namely information snippets.

Finally, information snippets are small units of information that normally cannot be semantically interpreted out of context. Typical information snippets are emphasized phrases, cross references and technical terms. Characteristically, information snippets may contain text and possibly further information snippets that are freely chosen from some repertoire.

A custom meta-model could allow modelers to classify document constituents into any of the four categories and to further constrain their type in accordance with the typical structures of their category. This could be another use for HyTime's concept of architectural forms [DD84].

Speaking of information snippets, their characteristic recursive structure of stretches of text interspersed with lower-level information snippets are aptly described by the data modeling pattern [GHJV95] called *Composite*, as has been pointed out in a 2007 Extreme Markup Languages Paper [BST07]. A Composite constructor would be a valuable tool in a meta-meta-model facility.

Some work has been done on implementational patterns that are applicable to models written in XML Schema [KS06,L10,S04]. Examples are the patterns Russion Doll, Venetian Blind, Salami Slice and Garden of Eden, the four possibilities that arise when combining local or global element declarations with local or global type definitions, respectively. In further studies, we are going to discuss, if any of these patterns can be usefully “lifted” from the implementational to a conceptual level. We will also mine the literature on data modeling, particularly the foundational book by Simsion [S07] for patterns in data modeling that also apply to document modeling.

Conclusion and further work

A number of modeling languages or meta-models have been used in the XML context, mostly with the goal of integrating XML Schema as an implementation technology into the development of XML-based systems. Nečaský [N06] surveys conceptual modeling languages that extend the entity-relationship model. Bernauer et al. [BKK04] survey methods of integrating XML Schema into modeling with UML. The conference series Extreme Markup Languages and Balisage offer a number of papers in the area [KH00,PB09,B09], the latter introducing a subset of Guizzardi's [G05] Unified Foundational Ontology as a conceptual modeling language on a layer of abstraction above UML. All these languages are of a generic nature; that is, their meta-models only offer generic constructors that cannot be customized to the domain.

We find it beneficial to put modeling into a larger context, taking up ideas from domain-driven design. We put forward the notion of domain-specific meta-models that capture building principles of domains. This raises the larger research question of a meta-meta-modeling facility, instances of which are domain-specific meta-models. We have identified potential constructors of a meta-meta-modeling facility by doing reverse modeling in two case studies and by studying patterns in conceptual models that can be abstracted into constructors.

Benefits of domain-specific meta-models are: Clarity of unterstanding, reduction of complexity, support of system design and guidance for user interfaces.

In our work at EPT, we will futher explore customized meta-models. We intend to build (or even

generate) editors for domain-specific meta-models, for example for forms documents. And we will formalize meta-models (that is, define a meta-meta-model), probably using abstract state machines [BS03] as a technology-neutral specification language.

Acknowledgments

We extend our heartfelt thanks to BMF project leader Andreas Ebner as well as to Rico Zimmermann und Christoph Nufer at Institut für Rundfunktechnik for expertise, commitment and support.

We also gladly acknowledge the long-standing collaboration with Werner Simon and Thorsten Reinheimer of ExxTainer AG, who suggested and sponsored Tamer Demirel's Diploma Thesis project. We appreciate their vision regarding XML technology in business and their competent and engaged supervision of students.

Dedication

The first author of this paper, Anne Brüggemann-Klein, dedicates the paper to colleague and friend Professor Derick Wood, PhD, on occasion of his 70th birthday this year. Derick has been a most influentical scientific mentor to me. I have learned from him much that I know about the English language and scientific writing, and I enjoyed our long-standing and quite successful research collaboration. I also appreciate that through Derick and his wife, Mary Chen, I got a bit of an inside view of Canada and Hongkong. Thank you!

References

[B09] B.T. Bauman: *Prying Apart Semantics and Implementation: Generating XML Schemata directly from ontologically sound conceptual models*. Balisage 2009. Available from http://www.balisage.net/Proceedings/. doi:10.4242/BalisageVol3.Bauman01

[BKK04] M. Bernauer,G. Kappel, G. Kramler: *Representing XML Schema in UML—A Comparison of Approaches*. Technical Report Business Infomatics Group at TU Wien 2004. Available from http://www.big.tuwien.ac.at/research/publications/2004/0304.pdf.

[BS03] E. Börger and R. Stärk: *Abstract State Machines. A Method for High-Level System Design and Analysis*. Springer-Verlag 2003.

[BST07] 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/.

[ES08] 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.

[CSF00] 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. doi:10.1007/3-540-45393-8_40

[D10] T. Demirel. *Grundsatzuntersuchung zur Übernahme von nicht-semantischen XML-Druckerdaten in semantisches XML auf Basis eines XSD-Schemas*. Diploma Thesis, Technische Universität München, 2010.

[DD84] S.J. DeRose and D.G. Durand: *Making Hypermedia Work: A User's Guide to HyTime*. Kluwer Academic Publishers 1984.

[E04] E. Evans: *Domain-Driven Design. Tackling Complexity in the Heart of Software*. Addison-Wesley 2004.

[EE04] R. Eckstein, S. Eckstein. *XML und Datenmodellierung*. DPunkt-Verlag 2004.

[G05] G. Guizzardi: *Ontological Foundations for Structural Conceptual Models*. Ph.D. Thesis, University of Twente, The Netherlands, 2005. Available from https://doc.telin.nl/dsweb/Get/Document-55835/Ontological%20Foundations%20for%20Structural%20Conceptual%20Models.pdf.

[GHJV95] E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns*. Addison-Wesley 1995.

[KH00] 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/.

[KS06] A. Khan, M. Sum. *Introducing Design Patterns in XML Schemata*. Sun Developer Network 2006.

[L10] T. Lainevool. *Develop Effective XML Documents Using Structural Design Patterns*. http://www.LainevoolXMLPatterns.com/.

[MA95] E. Maler, J. El Andaloussi. *Developing SGML DTDs: From Text to Model to Markup*. Prentice Hall 1995.

[MSUW04] S.J. Mellor, K. Scott, A. Uhl, D. Weise: *MDA Distilled*. Addison-Wesley 2004.

[N06] M. Nečaský: *Conceptual Modeling for XML: A Survey*. Proceedings of the Dateso Annual International Workshop on Databases, Texts, Specifications and Objects 2006. Available from http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-176/

[P08] D. Pagano. *Modeling and Defining XML Applications with UML and XML Schema*. Diploma Thesis, Technische Universität München, 2008.

[PB09] D. Pagano and A. Brüggemann-Klein. *Engineering Document Applications — From UML Models to XML Schemas*. Balisage 2009. Available from http://www.balisage.net/Proceedings/. doi:10.4242/BalisageVol3.Bruggemann-Klein01.

[S04] D. Stephenson. *XML Schema Best Practices*. HP Dev Resource 2004. http://devresource.hp.com/drc/resources/vdVlistXMLSchemaBestPractices.jsp.

[S07] G. Simsion: *Data Modeling Theory and Practice*. Technics Publications 2007.

[SMPTE01] Society of Motion Picture and Television Engineers: SMPTE 335M-2001- Metadata Dictionary Structure. SMPTE Standard for Telivision, 2001.

[SMPTE03] Society of Motion Picture and Television Engineers: SMPTE 395M-2003 - Metadata Groups Registry Structure. SMPTE Standard for Telivision, 2003.

[SMPTE08] Society of Motion Picture and Television Engineers: RP210-11-2008 - Data Element Dictionary. SMPTE Recommended Practice, 2008.

[T09] A. Tai: *Requirements analysis, conception and implementation of a BMF-Generator on the basis of the BMF-XML Schema*. Master's Thesis. Technische Universität München, 2009.

[WM07a] E. Wilde und F. Michel: *SPath: a path language for XML schema*. In: WWW '07: Proceedings of the 16th international conference on World Wide Web, S. 1343–1344. ACM, 2007. doi:10.1145/1242572.1242842.

[WM07b] E. Wilde und F. Michel: *SPath: A Path Language for XML Schema*. In: Paper 2007-001 . School of Information, 2007. Available from http://dret.net/netdret/docs/wilde-irep07-001-spath.pdf.

[WM07c] E. Wilde und F. Michel: *XML-based XML schema access*. In: WWW '07: Proceedings of the 16th international conference on World Wide Web, S. 1351–1352. ACM, 2007. doi:10.1145/1242572.1242846.