XMLText: From XML Schema to Xtext

Patrick Neubauer, Alexander Bergmayr, Tanja Mayerhofer,
Javier Troya, and Manuel Wimmer

Business Informatics Group
Vienna University of Technology, Austria
{neubauer,bergmayr,mayerhofer,troya,wimmer}@big.tuwien.ac.at

Abstract

A multitude of Domain-Specific Languages (DSLs) have been implemented with XML Schemas. While such DSLs are well adopted and flexible, they miss modern DSL editor functionality. Moreover, since XML is primarily designed as a machine-processible format, artifacts defined with XML-based DSLs lack comprehensibility and, therefore, maintainability. In order to tackle these shortcomings, we propose a bridge between the XML Schema Definition (XSD) language and text-based metamodeling languages. This bridge exploits existing seams between the technical spaces XMLware, modelware, and grammarware as well as closes identified gaps. The resulting approach is able to generate Xtext-based editors from XSDs providing powerful editor functionality, customization options for the textual concrete syntax style, and round-trip transformations enabling the exchange of data between the involved technical spaces.

We evaluate our approach by a case study on TOSCA, which is an XML-based standard for defining Cloud deployments. The results show that our approach enables bridging XMLware with modelware and grammarware in several ways going beyond existing approaches and allows the automated generation of editors that are at least equivalent to editors manually built for XML-based languages.

Categories and Subject Descriptors D.2.6 [*Programming Environments*]: Programmer workbench; I.7.2 [*Document Preparation*]: Markup languages

Keywords DSL, Language Engineering, Markup Language, Language Modernization, XSD, Xtext

General Terms Algorithms, Languages

CONF 'yy, Month d–d, 20yy, City, ST, Country.

http://dx.doi.org/10.1145/nnnnnnn.nnnnnnn

1. Introduction

XML has been primarily designed as a machine-processible format following the fixed angle-bracket syntax. While for prominent XML-based languages, such as OASIS's TOSCA [8], advanced editors have been handcrafted, for others, like Artificial Intelligence Markup Language (AIML) [19], no dedicated editor is available. In the latter case, language users are bound to the angle-bracket syntax that is verbose and complex in terms of human-comprehension and therefore impedes maintainability [3].

Tackling these major limitations requires breaking out of inflexible XML syntax by providing support to construct a fully-customizable concrete syntax and language workbench. While state-of-the-art Model-Driven Language Engineering (MDLE) frameworks, such as Xtext [9], allow the development of Domain-Specific Modeling Languages (DSMLs) as well as accompanying customized concrete syntax and rich language workbenches, manually recreating existing XML-based languages with such frameworks is a complex, error-prone, and time-consuming task requiring language-engineering skills. Additionally, modeling languages that ought to replace XML-based languages leave behind backward-compatibility issues with the usually comprehensive set of applications built for the XML-based language.

To overcome these issues, our approach—the Model-Driven Language Modernization (MDLM) approach which is instantiated through the XML to Xtext (XMLText) framework[1]—facilitates the modernization of XSD-based languages with modelware and grammarware [14] by (i) transforming existing XSD-based languages to metamodels, (ii) adapting those metamodels to facilitate the production of effective language grammars, (iii) generating both customized language grammars and workbenches from the adapted metamodels, and (iv) enabling round-trip transformations between the original XSD-based language and the modernized DSML by generic serializers and parsers.

[1] Access to XMLText is provided on the paper's website at `http://xmltext.big.tuwien.ac.at`.

By supporting round-trip transformations, our framework inherently merges benefits of both XMLware and grammarware, namely machine-processibility and re-use of extensive XMLware applications of the former and high-customizability, enabling to target human-comprehensibility and therefore maintainability, of the latter. XMLText is evaluated based on a reproduction study on the XML-based language TOSCA, in particular, the framework's ability to produce complete DSMLs from XML Schemas.

The remainder of this paper is organized as follows. Section 2 provides an overview of gaps between XMLware, modelware, and grammarware as exposed by our case study. Section 3 introduces the MDLM approach as well as the XMLText framework. Section 4 evaluates the findings based on a reproductive study concerning an industrial strength language. Finally, Section 5 discusses related work before Section 6 concludes with a perspective on future work.

2. Gaps between XMLware, Modelware, and Grammarware

MDLE frameworks like Xtext accelerate the development of DSMLs and DSML environments to a great extent. They cover all aspects of a textual language infrastructure, including the default generation of a lexer, parser, as well as an editor featuring rich editing capabilities, such as, syntax highlighting, error indication, and content assisting. At the same time, they provide language engineers with the power to completely customize the look-and-feel, i.e., the textual concrete syntax, of DSMLs and therewith to construct DSMLs tailored to optimize human-comprehensibility—a customization not possible in XML due to its fixed concrete syntax.

For the purpose of modernizing XSD-based languages by transforming them to metamodel-based DSLs, we seek for a fully automated approach that produces from a given XSD a language grammar that fulfills our needs for human-comprehensibility. As we describe in the next paragraphs, we build upon existing tools that are integrated in EMF, namely the *EMF XSD Importer* and the *Xtext Grammar Generator*. First, the *EMF XSD Importer* is employed to produce an *Ecore Metamodel* from an existing *XML Schema*. Secondly, the *Ecore Metamodel* is used as input for the *Xtext Grammar Generator*, which transforms it to a corresponding *Xtext Grammar* as well as to a corresponding *Xtext Workbench*. However, our investigations have shown that chaining these tools together (into what is from now on referred to as *Default Transformation Chain*) leaves many gaps between the technical spaces of XMLware, modelware, and grammware open.

In the next paragraphs we dig into specific gaps discovered through a case study on the TOSCA cloud topology and orchestration language. Version 1.0 of the TOSCA XSD[2] contains 791 lines of code, 99 complex types, 11 simple types, 54 global types, 10 wildcards, 2 abstract types, and 2 global elements. Table 1 depicts an overview of *XML Schema* concepts currently processed by the *Default Transformation Chain* focusing on language concepts used in typical TOSCA instances. The column "Supported" denotes whether a particular *XML Schema* concept can be transformed to a DSML grammar rule able to represent the original *XML Schema* concept. In the following we summarize the concepts which are currently unsupported by the *Default Transformation Chain*.

XML Schema concepts	Definition	Supported	Notes
Element	xs:element	✓	Grammar rule is created
Attribute	xs:attribute	✓	Feature in grammar rule is created
Containment	(through nesting)	✓	Grammar rule is created and rule call stated
Mixed content	mixed="true"	✗	Ecore feature map is neglected in grammar generation
Wildcard	xs:any, xs:anyAttribute	✗	Ecore feature map is neglected in grammar generation
Restriction	xs:restriction	✗	Different interpretations
Data type	type="xs:string", type="..."	✗	Placeholder terminal and a to-do comment replace data types
Identifier and identifier reference	type="xs:ID", type="xs:IDREF"	✗	Placeholder terminal replaces identifier value

Table 1. Overview of *XML Schema* language concepts and their support by the *Default Transformation Chain*

Gap 1: Mixed Content and Wildcards. *XML Schema* allows to define mixed complex type elements, i.e., allowing character data to appear within the body of the element. Furthermore, the use of `xs:any` (cf. line 4 in Listing 1), i.e., a wildcard element, allows to specify any type of markup content in XML instances (cf. line 3-4 in Listing 2). The *EMF XSD Importer* translates such types to metaclasses containing feature maps that represent ambiguous language concepts whose handling is delegated to the underlying parser and serializer implementations. However, since the *Xtext Grammar Generator* neglects the occurrence of such implicitly modeled language concepts, they become unavailable at grammar level as well as on instance level.

```
1 <xs:element name="Properties" minOccurs="0">
2   <xs:complexType>
3     <xs:sequence>
4       <xs:any namespace="##other" processContents="
            ↪lax"/>
5     </xs:sequence>
6   </xs:complexType>
7 </xs:element>
```

Listing 1. TOSCA XML Schema (excerpt)

Gap 2: Data Types and Restrictions. The W3C Recommendation on *XML Schema* data types [6] describes a set of built-in data types for different kinds of data, such as numbers, dates, strings, identifiers, and references. The *EMF XSD Importer* successfully transforms *XML Schema* data types to custom data types mapped to Java types at metamodel level. However, the *Xtext Grammar Generator* transforms any metamodel data type to a placeholder terminal symbol replacing the actual data type. Therefore,

[2] `http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/schemas/TOSCA-v1.0.xsd`.

the *Xtext Grammar* created by the *Default Transformation Chain* does not allow to construct instances able to store values for variables of any kind of data type. Moreover, this limitation also impacts *XML Schema* restrictions. For example, in case of a restricted XSD attribute of type `xs:string`, even if we correct the type definition of the resulting grammar attribute to the `STRING` terminal rule provided by Xtext, the attribute created in the Xtext grammar is interpreted differently: a String in *XML Schema* and Ecore is interpreted by excluding its surrounding quotes and in Xtext it is interpreted by including its surrounding quotes.

Gap 3: Identifiers and Identifier References. The *EMF XSD Importer* transforms attributes of type `xs:ID` (cf. `id` attributes in Listing 2) to Ecore attributes of type `java.lang.String` having set the `ID` property to `true`. Attributes of type `xs:IDREF` are also transformed to Ecore attributes instead of references. Hence, even if the gap related to data types is closed, the *Xtext Grammar Generator* still handles `xs:IDREF` equally to attributes – capable of holding primitive values not referring to other elements.

```
<nodeTemplate id="ApacheWebServer" type="
    ↪ApacheWebServerType" name="Apache␣Web␣Server">
  <properties id="ApacheWebServerProperties">
    <numCpus>1</numCpus>
    <memory>1024</memory>
  </properties>
</nodeTemplate>
```

Listing 2. TOSCA Moodle XML instance (excerpt)

Gap 4: Customizing Concrete Syntax. XML has been primarily designed as a machine-processible format composed of immutable concrete syntax. Therefore, users of XML-based languages are bound to angle-bracket syntax that is described as verbose and complex in terms of human-comprehension and therefore impedes maintainability [3].

3. XMLText

Our approach proposes bridging *XMLware*, *Modelware*, and *Grammarware*. Therefore, our goal is to provide a framework that automatically modernizes XSD-based languages to metamodel-based languages providing flexible syntax, rich language workbenches, and model-based techniques such as transformation, validation, and code generation. We achieve this goal by improving the transformations of the *Default Transformation Chain* as well as introducing new transformations that overcome the issues discussed in Section 2. Figure 1 depicts a conceptual overview of our XML to Xtext (XMLText) framework. Details are discussed in the following subsections.

Like in the *Default Transformation Chain*, the first step is to transform a given *XML Schema* to an *Ecore Metamodel* by employing the *EMF XSD Importer* (1). In order to tackle the issue of feature maps causing the production of empty grammar rules, we adapt the *Ecore Metamodel* by replacing feature maps with generic concrete constructs (cf. (2) in Figure 1). Next, the adapted metamodel is used as

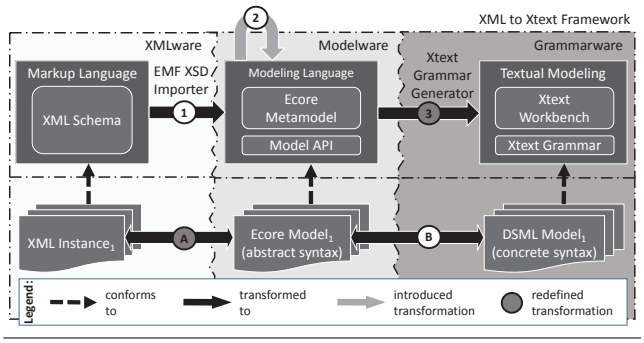

Figure 1. Overview of the XMLText framework

input for generating the *Xtext Grammar*. However, to store actual values for attributes we enhance the *Xtext Grammar Generator* (cf. (3) in Figure 1) by creating, importing, and referencing a library of data types. Moreover, we enable the automated customization of the textual concrete syntax of the target DSML by providing a configurable grammar rule template.

For the adaptions of the *Ecore Metamodel* introduced by the XMLText framework, it is necessary to customize existing transformations (cf. (A) in Figure 1) to act upon them on instance level. Therefore, we customize (i) the deserializer that reads *XML Instance*s and creates in-memory *Ecore Model* representations conforming to the adapted *Ecore Metamodel* and (ii) the serializer that stores *Ecore Model*s as *XML Instance*s. As a result of keeping the *Xtext Grammar* coupled to the *Ecore Metamodel*, we are able to reuse the existing transformation (B). With the introduction of transformation (2) and the adaption of the transformations (3) and (A), our XMLText framework overcomes limitations of existing bridges between *XMLware* and *Grammarware* and thus allows an improved automated modernization of XML-based languages to metamodel-based DSMLs. In the following, we detail these transformations. Listing 3 shows the result of applying the XMLText framework on the exemplary XML-based language instance used in Listing 2.

```
TNodeTemplate ApacheWebServer {
  name: "Apache␣Web␣Server"
  type: ApacheWebServerType
  Properties ApacheWebServerProperties {
      NumCpus: "1"
      Memory: "1024"
  }
}
```

Listing 3. TOSCA Moodle DSML model (excerpt)

3.1 Mixed Content and Wildcards

The definition of mixed content as well as wildcards causes the *EMF XSD Importer* to create an attribute of type `EFeatureMapEntry`. However, since the *Xtext Grammar Generator* neglects feature maps, such XSD concepts cannot be represented with the resulting *Xtext Grammar*. To successfully cope with the occurrence of feature maps, we replace them with generic concrete constructs for which grammar

rules are generated (cf. ② in Figure 1). As shown in Figure 2, `AnyGenericConstruct` is an abstract class extended by `AnyGenericElement` and `AnyGenericText`. Therefore, a wildcard XML tag is represented by `AnyGenericElement` and the text before or after an XML tag is represented by `AnyGenericText`, thus, allowing mixed content. While the former represents the notion of wildcards in terms of `xs:any`, the latter allows representing mixed content appearing either prior or after an XML tag. The successful application of this solution is depicted by lines 5-6 of Listing 3.

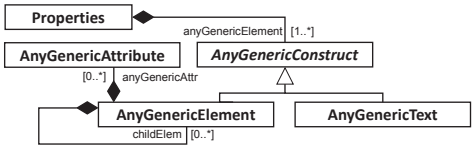

Figure 2. Explicit modeling structures replacing feature maps

3.2 Data Types

As mentioned earlier, the *Xtext Grammar Generator* does not create rules for metamodel data types. Therefore, both the specification of terminal rules as well as calls to these rules are missing. To overcome this limitation, we first constructed the *Xtext Data Type Library* defining terminal rules for built-in XSD data types and secondly adapted transformation ③ in Figure 1, such that these terminal rules are used in the final language grammar.

3.3 Identifiers and Identifier References

In order to tackle the gap associated with identifiers and identifier references, transformation ② replaces model attributes of type `xs:IDREF` with references to the metaclass `EObject`, which is the uppermost class in the Ecore hierarchy. While such a reference allows objects to reference any kind of object, an `xs:IDREF` attribute can only reference elements having an `xs:ID` attribute. Thus, transformation ③ introduces a necessary refinement. In particular, for an attribute of type `xs:IDREF`, we generate a grammar rule, allowing only to refer to objects with `xs:ID` attribute. Therefore, we define a new terminal rule (cf. example in Listing 4). Hereby, the subsequently generated editor provides support for referencing objects through content assist.

```
1 SourceElementType returns SourceElementType
2   referencingAttribute=[ecore::EObject|IDREF])?;
3
4 IDREF returns ecore::EString:
5   ID;
```

Listing 4. Xtext grammar for Identifiers and Identifier References

3.4 Customizing Concrete Syntax

To overcome the major limitation imposed by inflexible XML syntax, we provide an approach to construct a customizable concrete syntax enabling the specification of human-comprehensible and therefore increasingly maintainable instances. Usually, changing the concrete syntax of a DSML requires either to manually adapt the associated language grammar or the *Xtext Grammar Generator* transformations. The XMLText framework introduces a template mechanism that allows to specify customizable template files defining the concrete syntax of the target language. For example, as depicted by Listing 5, `VariableValueSpecification-TerminalSymbol` determines the terminal symbol used in the language grammar to specify a variable's value.

```
1 InterPackageReferenceTerminalSymbol = '.'
2 VariableValueSpecificationTerminalSymbol = ':'
3 PropertyMemberOpenTerminalSymbol = '{'
4 PropertyMemberCloseTerminalSymbol = '}'
```

Listing 5. Concrete syntax customization template file (excerpt)

4. Evaluation

In the evaluation we aimed at answering the following research question (*RQ*): Is the DSML generated by XMLText, i.e., $TOSCA_{XMLText}$, more complete as the DSML produced by the *Default Transformation Chain*, i.e., $TOSCA_{DTC}$ as well as available hand-crafted DSLs?

Evaluation Procedure. First we operate the *Default Transformation Chain* with the TOSCA XSD version 1.0 to generate $TOSCA_{DTC}$. Secondly, we employ the XMLText framework with the same TOSCA XSD to produce $TOSCA_{XMLText}$. Third, we gather language concepts and features appearing in the TOSCA XSD-conforming Moodle reference example [4], i.e., a complete definition of topology and orchestration details for an open source course management system, and correlate them with language concepts and features available in (i) $TOSCA_{DTC}$, (ii) *Cloudify DSL*—the only available textual DSL based on the TOSCA standard—, and (iii) $TOSCA_{XMLText}$.

Due to the fact that these languages have been implemented using different approaches, a common way of comparing them in terms of their completeness to a common language specification—the TOSCA standard [8] XSD version 1.0—is established. Therefore, our comparison takes into account language concepts and features occurring in the TOSCA XSD as well as in the individual language implementations. In particular, to analyze the language concepts of the *Cloudify DSL*, for which no language grammar is provided, we examine its language parser[3] on the existence of language concepts and features as defined in the TOSCA XSD. Furthermore, the unit of analysis in $TOSCA_{DTC}$ as well as $TOSCA_{XMLText}$ is represented by the DSML grammar.

Results. Table 2 depicts TOSCA language concepts and features employed in the Moodle example and their availability in $TOSCA_{DTC}$, *Cloudify DSL*, and $TOSCA_{XMLText}$. In total the Moodle example uses 19 different language con-

[3] The *Cloudify DSL* parser (version from April 1, 2015) examined during this evaluation can be retrieved online at `https://goo.gl/JzPL7U`.

cepts and 35 features defined in the TOSCA XSD. When looking for the availability of the combination of language concepts and features in the different languages we found that (i) *TOSCA$_{DTC}$* contains 17%, (ii) *Cloudify DSL* accommodates 37%, and (iii) *TOSCA$_{XMLText}$* encloses 98% of the TOSCA standard concepts and features found in the TOSCA XSD-conforming Moodle instance.

	Moodle Example	TOSCA$_{DTC}$	Cloudify DSL	TOSCA$_{XMLText}$
TOSCA Concepts	19	2 (~11%)	11 (~58%)	19 (100%)
TOSCA Features	35	7 (20%)	9 (~26%)	34 (~97%)
TOSCA Combined	54	9 (~17%)	20 (~37%)	53 (~98%)

Table 2. Availability of TOSCA standard concepts and features in different languages based on the Moodle example

In summary, we conclude that (i) the language grammar of *TOSCA$_{DTC}$* is missing essential concepts, such as, nodes and relationships, and is therefore not sufficient to represent the Moodle example. Furthermore, (ii) while the *Cloudify DSL* parser contains more language concepts and features as available in the *TOSCA$_{DTC}$*, it is still missing certain concepts, such as, requirements and capabilities. Moreover, for some missing concepts, such as `TDefinitions`, features are scattered throughout different language concepts in the *Cloudify DSL*. For example, their parser rule `models.Plan` contains policies and relationships that are originally located in `TDefinitions`. Therefore, the *Cloudify DSL* does not fully conform to the TOSCA standard and hence requires the user to map TOSCA XSD-conforming instances to *Cloudify DSL*-conforming instances. Finally, (iii) *TOSCA$_{XMLText}$* allows to represent almost entirely the same information as depicted in the Moodle example. In more detail, *TOSCA$_{XMLText}$* is missing the representation of the `xmlns` feature which is represented in the root element of the metamodel. Therefore, except for the occurrence of `xmlns`, the XMLText framework is able to perform round-trip transformations between TOSCA XSD-conforming XML instances and modernized TOSCA DSML-conforming models facilitating the re-use of existing XMLware applications as well as advanced capabilities of modern DSMLs.

Threats to Validity. We identified three threats of validity: (i) misinterpretation of language concepts and features due to their naming differences in the *Cloudify DSL* and the TOSCA standard, (ii) consideration of a subset of the TOSCA language represented by the TOSCA Moodle example, i.e., representing a subset of possible TOSCA language concepts and features, and (iii) the consideration of TOSCA as a representative for an XSD-based language, i.e., considering only a subset of all possible XML Schema language concepts and features. As a countermeasure to (i), we studied both the language concepts and features appearing in the *Cloudify DSL* language parser as well as in the *Cloudify DSL* language documentation. In order to act upon (ii), we identified that several TOSCA-based examples can act as a countermeasure. However, due to the lack of available TOSCA-based open source examples, we did not act upon it. Although, to encounter (iii) we selected TOSCA because it is a relatively complex language which poses several challenges when turning it into a modern textual DSML, we cannot claim any results outside of the TOSCA language.

5. Related Work

On a general level, we apply the ModelGen operator of Atzeni et al. [2]. This operator defines a general pattern which uses bridges on the meta-language level to derive transformations on the language level and instance level. This pattern also fits our architecture as presented in Figure 1. Traditionally, this pattern is proposed and used in the database field for schema-independent transformations, but it is of course also applicable in language engineering.

With respect to the complete transformation chain proposed in this paper, there exist a set of related approaches which cover certain aspects of this chain by focusing on the transitions between the involved technical spaces: (i) bridges between XMLware and modelware and (ii) bridges between modelware and grammarware. To the best of our knowledge, there exists only one approach [10] to bridge XMLware and grammarware directly which focus on XSD and Xtext. But of course, there are other efforts in different contexts for bridging XSD and BNF-like languages such as it is done in the context of grammar hunting [21].

XMLware and Modelware Several approaches for realizing either forward engineering from modelware to XMLware [5, 7] or reverse engineering from XMLware to modelware [16, 18] exist. In previous work [18], we presented an approach for generating MOF-based metamodels from DTDs. Our work presented in this paper differs from the previous approach in several ways, e.g., XSDs are used instead of DTDs and the full transition to textual modeling languages is done instead of stopping with the creation of the language's abstract syntax.

Modelware and Grammarware There exist grammar-driven approaches [1, 15, 20] in which metamodels are generated out of existing grammar definitions. In addition, metamodel-driven approaches generate grammars out of existing metamodels [11, 17] or link metamodels with grammars [13]. Especially, EMFText [12] seems to be an interesting alternative to Xtext used in this paper, as there is also the possibility to define several concrete textual syntaxes for one metamodel. As opposed to our work, the user of these approaches has to define its own transformation rules between either individual grammar rules or terminal rules and metamodel elements instead of relying on a generic and automated transformation of XSDs as proposed in this work.

XMLware and Grammarware Eysholdt and Rupprecht present a report [10] on the migration of a modeling environment from XML/UML to Xtext/GMF. Due to inefficiency of XML in terms of verbose syntax and lack of tool support

they perform the modernization of a legacy modeling environment. In detail, they start by creating Ecore metamodels from XSDs, then perform changes as well as customizations of Xtext features, and finally end up with a modernized language and workbench. Compared to our approach, they manually perform metamodel changes as well as customizations of Xtext features instead of building a generic and automated transformation chain.

6. Conclusion and Future Work

In this work we aimed at highlighting currently existing limitations in bridging XML-based languages with textual modeling languages and overcoming them by means of several improvements. This includes dealing with specific XSD concepts, namely mixed content, wildcards, restrictions, identifier and identifier references, and data types, as well as concrete syntax customization. The main principle that guided our solution was to represent each XSD concept explicitly in the modelware technical space. By this, important characteristics of XML such as being able to represent semi-structured data is now also better reflected in the corresponding textual modeling languages.

The proposed improvements have been bundled into the XMLText framework as well as evaluated regarding completeness. In particular, the evaluation has been carried out based on the OASIS TOSCA standard. The evaluation results indicate that the proposed XMLText framework significantly improves over existing solutions and generates a textual modeling language for TOSCA that is more complete than the currently available hand-crafted *Cloudify DSL*.

With respect to future work, first, to fully exploit the benefits of modernizing XSD-based languages with modeling languages, we strive to extend the current framework by addressing currently unresolved challenges as well as eventually arising challenges when conducting further case studies. In detail, we plan to conduct case studies based on different examples of the TOSCA language as well as other XSD-based languages covering different sets of XML Schema concepts. Secondly, we plan to quantify the actual impact of modernizing XSD-based languages by conducting user studies focusing on human-comprehension.

Acknowledgments

This work is funded by the European Commission under ICT Policy Support Programme, grant no. 317859 and by the Christian Doppler Forschungsgesellschaft and the BMWFW, Austria.

References

[1] M. Alanen and I. Porres. A Relation Between Context-Free Grammars and Meta Object Facility Metamodels. Technical report, Turku Centre for Computer Science, 2003.

[2] P. Atzeni, P. Cappellari, and P. A. Bernstein. ModelGen: Model Independent Schema Translation. In *Proc. of ICDE*, pages 1111–1112, 2005.

[3] G. J. Badros. JavaML: A Markup Language for Java Source Code. *Computer Networks*, 33(1):159–177, 2000.

[4] T. Binz, U. Breitenbücher, F. Haupt, O. Kopp, F. Leymann, A. Nowak, and S. Wagner. *OpenTOSCA – a runtime for TOSCA-based cloud applications*. Springer, 2013.

[5] L. Bird, A. Goodchild, and T. Halpin. Object Role Modelling and XML-Schema. In *Proc. of ER*, pages 309–322. Springer, 2000.

[6] P. V. Biron and A. Malhotra. XML schema part 2: Datatypes second edition. W3C recommendation, W3C, Oct. 2004. http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/.

[7] R. Conrad, D. Scheffner, and J. C. Freytag. XML Conceptual Modeling using UML. In *Proc. of ER*, pages 558–571. Springer, 2000.

[8] Derek Palma, Thomas Spatzier. Topology and Orchestration Specification for Cloud Applications Version 1.0, 2013.

[9] M. Eysholdt and H. Behrens. Xtext: Implement your Language Faster than the Quick and Dirty Way. In *Companion Proc. of OOPSLA*, pages 307–309. ACM, 2010.

[10] M. Eysholdt and J. Rupprecht. Migrating a Large Modeling Environment from XML/UML to Xtext/GMF. In *Companion Proc. of OOPSLA*, pages 97–104. ACM, 2010.

[11] F. Heidenreich, J. Johannes, S. Karol, M. Seifert, and C. Wende. Derivation and Refinement of Textual Syntax for Models. In *Proc. of ECMDA-FA*, pages 114–129. Springer.

[12] F. Heidenreich, J. Johannes, S. Karol, M. Seifert, and C. Wende. Model-Based Language Engineering with EMFText. In *Proc. of GTTSE*, pages 322–345. Springer, 2011.

[13] F. Jouault, J. Bézivin, and I. Kurtev. TCS: a DSL for the specification of textual concrete syntaxes in model engineering. In *Proc. of GPCE*, pages 249–254. ACM, 2006.

[14] P. Klint, R. Lämmel, and C. Verhoef. Toward an engineering discipline for grammarware. *ACM Trans. Softw. Eng. Methodol.*, 14(3):331–380, 2005.

[15] A. Kunert. Semi-Automatic Generation of Metamodels and Models from Grammars and Programs. *Electronic Notes in Theoretical Computer Science*, 211:111–119, 2008.

[16] M. Mani, D. Lee, and R. R. Muntz. Semantic Data Modeling using XML Schemas. In *Proc. of ER*, pages 149–163. Springer, 2001.

[17] P.-A. Muller and M. Hassenforder. HUTN as a bridge between modelware and grammarware-an experience report. In *Proc. of WISME Workshop*, 2005.

[18] A. Schauerhuber, M. Wimmer, E. Kapsammer, W. Schwinger, and W. Retschitzegger. Bridging WebML to model-driven engineering: from document type definitions to meta object facility. *IET Software*, 1(3):81–97, 2007.

[19] R. S. Wallace. *The anatomy of ALICE*. Springer, 2009.

[20] M. Wimmer and G. Kramler. Bridging Grammarware and Modelware. In *Proc. of Satellite Events at MoDELS*, pages 159–168. Springer, 2006.

[21] V. Zaytsev. Grammar Zoo: A corpus of experimental grammarware. *Sci. Comput. Program.*, 98:28–51, 2015.