Modernizing Domain-Specific Languages with XMLText and IntellEdit

Patrick Neubauer*, Robert Bill*, and Manuel Wimmer*
*Business Informatics Group, TU Wien, Austria
{neubauer, bill, wimmer}@big.tuwien.ac.at

***Abstract*—The necessity of software evolution caused by novel requirements is often triggered alongside the advancement of underlying languages and tools. Although modern language workbenches decrease the opportunity cost of creating new language implementations, they do not offer automated and complete integration of existing languages. Moreover, they still require complex language engineering skills and extensive manual implementation effort to suit the expectations of domain experts, e.g., in terms of editor capabilities. In this work we present XMLIntellEdit—a framework for evolving domain-specific languages by automating the generation of modernized languages offering advanced editing capabilities, such as extended validation, content-assist, and quick fix solutions. Our approach builds on techniques from Model-Driven Engineering and Search-based Software Engineering research. Initial results indicate that XML Schema definitions containing restrictions can be applied for the automated generation of advanced editing facilities.**

***Index Terms*—Domain Specific Languages, Model Driven Engineering, Search-based Software Engineering, Advanced Editor Support, XML**

I. Introduction

Software has the ability to enable or even advance changes in technology, economy, society, and humanity. To reflect these changes and endure relevancy to stakeholder, software is destined to evolve [1]. However, the increasing number of new maintenance and requirement requests often outgrows the ability of software languages. Therefore, reengineering, and evolution of software often involves shifting to different software languages and their accompanying tools, which offer the basis to meet new requirements, such as performance, maintainability, reliability, and usability. Language workbenches—a product of language-oriented programming—are environments designed to support the construction of domain-specific (modeling) languages (DS(M)Ls) offering an opportunity to meet not only current maintenance and requirement requests but also the ability to address future requests that current languages are unable to meet. Language workbenches promise to guide domain experts towards directly carrying out development effort by enabling the creation of editors customized for user interaction. Thus, on one hand, productivity of development is improved by better suitable editors and closer relationships with domain experts [2]. On the other hand, the application of language workbenches for the evolution of existing languages, e.g., as realized by XML technologies, as well as the creation of sophisticated editors, still involves expert skills of language engineers and considerable effort both in the initial creation and maintenance of the modernized language. Consequently, the evolution of languages to productivity-enhancing and domain expert-focused languages is limited to developers owning both language engineering expertise as well as domain-specific knowledge.

In an effort to address limitations imposed by language evolution, we present an approach for the automated modernization of XML Schema (XSD)-based languages based on model-driven engineering (MDE) as well as search-based software engineering (SBSE) techniques and exploits extended language definitions for the creation of advanced domain-specific language editors.

II. The Approach

The goal of *XMLIntellEdit*[1] is to (i) extend our language modernization approach for XSD-based languages, i.e., XMLText [3], by exploiting XSD restrictions for the generation of formal constraints and hence enable to (ii) apply the Intelligent Domain-Specific Editing (IntellEdit) [4] framework for the generation of advanced editing facilities.

XMLText [3] has been created to tackle the shortcomings of XML, which has been designed as a machine-processible format following a fixed angle-bracket syntax that presents itself as complex and verbose in terms of human-comprehension and therefore impedes maintainability [5]. In addition, it automates the complex, error-prone, and time-consuming task of manually creating DSML implementations enhancing usability and comprehensibility. XMLText realizes the modernization of XSD-based languages with modelware and grammarware [6] by means of (i) transforming XSD-based

[1]A ready-to-use virtual machine image and Eclipse instance of the *XMLIntellEdit* framework as well as application examples can be retrieved online from http://xmlintelledit.big.tuwien.ac.at.

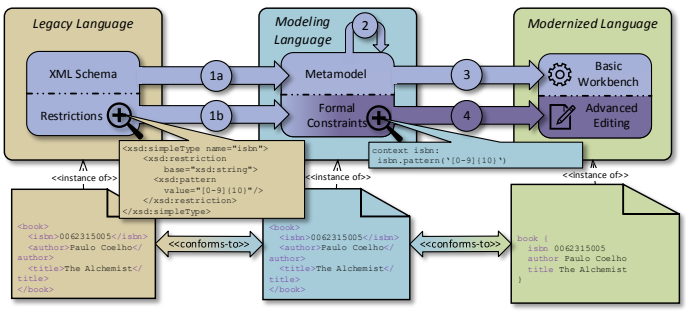

Fig. 1: Language Modernization with *XMLIntellEdit*.

languages to *Metamodel*s (cf. 1a in Fig. 1), (*ii*) adapting metamodels to facilitate the production of effective language grammars (cf. 2 in Fig. 1), (*iii*) generating *Modernized Language*s from metamodels (cf. 3 in Fig. 1) in terms of language grammars and *Basic Workbench*es, and (*iv*) providing round-trip transformations for conforming language instances.

IntellEdit [4] represents a framework for the automated generation of consistency-achieving DSML editors by overcoming the limitations of modern language workbenches in preserving instance consistency of elaborated language-specific constraints. Hence, IntellEdit defeats the necessity to manually implement *Advanced Editing* capabilities for such constraints by automating their generation (cf. 4 in Fig. 1) and therefore supports editor users in comprehending and resolving consistency violations. In detail, IntellEdit-supported DSML implementations offer extended validation, content-assist, and quick fix solutions to help domain experts in achieving and retaining consistency. Moreover, the IntellEdit plugin, which runs alongside IntellEdit-supported DSML editors, applies SBSE techniques and a three-stage neighborhood search-process for seeking constraint repair solutions that are eventually displayed as quick fix solutions to the editor user.

However, to enable the generation of IntellEdit-supported DSMLs, we need to fulfill the requirement of providing languages containing formal constraints. Hence, to realize XMLIntellEdit, we extend XMLText by transforming *Restrictions* imposed on data types of the *Legacy Language* to *Formal Constraints* of the *Modeling Language* (cf. 1b in Fig. 1). Finally, XMLIntellEdit supplies these artifacts to IntellEdit to generate a *Modernized Language* providing advanced editing facilities.

III. Application Example

The simplified language depicted in Fig. 1 is used to represent libraries of books. Books have a title, author, and an International Standard Book Number (ISBN). The data type of ISBNs is represented by a string, which is restricted by the pattern "`[0-9]{10}`", i.e., allowing digits with values from zero to nine to occur exactly ten times. In case an invalid ISBN is supplied, the *IntellEdit* runtime-plugin looks for fitting solutions to replace the invalid ISBN and displays highly-ranked quick fix solutions to be applied by the editor user. In our example, supplying "`006-2-31-500-5`" as ISBN results in a quick fix solution suggesting to replace the String by "`0062315005`", i.e., a string without hyphen characters.

IV. Related Work

Fouad et al. [7] present a set of rules to transform XML constraining facets to OCL expressions. Differences in our approach include the application of a DOM XML parser to find constraining facets and establish OCL expressions instead of model transformations. Moreover, they do not consider other parts of the language, such as structure, and hence do not create a complete language but only a list of OCL expressions. Eysholdt and Rupprecht report [8] on the modernization of a XML/UML-based legacy software to Xtext/GMF due to the inefficiency of XML in terms of verbose syntax and lack of tool support. 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. Egyed et al. and Reder et al. [9], [10] present an approach to assist editor users in fixing inconsistencies in UML models by generating a set of concrete changes as well as their impact on consistency rules. Their scalability results indicate that tree data structures may be highly appropriate for the representation of repairs. We perform a similar search for basic repairs, such as reference insertions, but differ in the search for complete repairs by performing global search processes.

V. Conclusion and Future Work

In this work, we presented an approach towards addressing challenges imposed by maintenance and requirement requests on domain-specific languages. The automation of complex language engineering tasks enables the construction of domain expert-oriented editors. Thus, domain experts employing the modernized language, which results from applying our modernization approach, may unlock productivity gains by accelerated composition, modification, and apprehension of modernized instances as well as comprehension and resolution of consistency violations through extended validation and automated repair solutions. Our initial results indicate that XSD-based languages, which contain restrictions, are applicable for the automated generation of formal constraints and advanced editing facilities. Future work includes the evaluation of our approach based on industrially employed DSMLs as well as professional domain experts for gathering empirical data on performance, scalability, and usability of both modernization procedure as well as generated DSML editors.

References

[1] T. Mens, Y. Guéhéneuc, J. Fernández-Ramil, and M. D'Hondt, "Guest editors' introduction: Software evolution," *IEEE Software*, vol. 27, no. 4, pp. 22–25, 2010.

[2] M. Fowler, *Domain-specific languages*. Pearson Education, 2010.

[3] P. Neubauer, A. Bergmayr, T. Mayerhofer, J. Troya, and M. Wimmer, "XMLText: From XML Schema to Xtext," in *Proceedings of SLE*, 2015, pp. 71–76.

[4] P. Neubauer, R. Bill, T. Mayerhofer, and M. Wimmer, "Automated Generation of Consistency-Achieving Model Editors," in *Proceedings of SANER, 2017*, 2017.

[5] G. J. Badros, "JavaML: A Markup Language for Java Source Code," *Computer Networks*, vol. 33, no. 1, pp. 159–177, 2000.

[6] P. Klint, R. Lämmel, and C. Verhoef, "Toward an engineering discipline for grammarware," *ACM Trans. Softw. Eng. Methodol.*, vol. 14, no. 3, pp. 331–380, 2005.

[7] T. Fouad and B. Mohamed, "Transforming XML schema constraining facets and XML queries to object constraint language (OCL)." *Journal of Theoretical & Applied Information Technology*, vol. 87, no. 3, 2016.

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

[9] A. Egyed, E. Letier, and A. Finkelstein, "Generating and evaluating choices for fixing inconsistencies in UML design models," in *Proceedings of ASE*, 2008, pp. 99–108.

[10] A. Reder and A. Egyed, "Computing repair trees for resolving inconsistencies in design models," in *Proceedings of ASE*, 2012, pp. 220–229.