Ina Schaefer, Dimitris Karagiannis, Andreas Vogelsang, Daniel Mendez (Hrsg.): Modellierung 2018,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2017 1

Taming the Software Development Complexity with Domain Specific Languages

Experiences from Deploying MPS-based DSLs for Computed Tomography Scanners at Siemens Healthineers

Daniel Ratiu[1] Holger Nehls[2] Jochen Michel[3]

Abstract: Modern computed tomography (CT) scanners are highly complex and flexible devices. This versatility is realized with a multitude of interconnected parameters and rules which are defined by domain experts in so-called scanner model specifications distributed over almost one hundred documents. The primarily used tools to write these documents (e.g. MS Word, MS Excel) are domain agnostic and they support only plain natural-language for the specification. Consequently, maintaining a valid scanner specification is a tedious, error-prone and therefore expensive process. To tackle the complexity of scanners parameters specifications, over the last two years we developed and deployed an eco-system of domain specific languages (DSLs) and associated tooling, covering a central portion of the scanner domain. The languages are developed using the Jetbrains' MPS language workbench. In this paper, we present our experiences with developing our language eco-system. We briefly describe the language architecture, the design and development process that led us there, and discuss variation points of our approach and present in more detail a set of lessons learnt and best practices.

Keywords: domain specific languages, industrial experience, Jetbrains' Meta-Programming System

1 Introduction

Non-invasive imaging is one of the most important improvements in medicine and enables doctors to diagnose and heal diseases that are not visible without having insights into the human body. Modern Computed Tomography (CT) scanners are highly complex machines, enabling radiologists to perform examinations of patients, like trauma scans, evaluation of neurological abnormalities, detection of tumors or diagnostic of heart diseases. While the x-ray beam rotates around the patient, the detector measures the attenuation, which represents the composition of the scanned object. Based on this volume data, it is possible to reconstruct slice images and calculate 3D visualisations of the human body and organs. This data is the basis for applications that support the radiologist in the diagnose process.

CT scanners are perfect examples of software intensive cyber-physical systems – a large amount of software enable the realization of complex use-cases and the interaction with

[1] Siemens Corporate Technology, Munich, daniel.ratiu@siemens.com
[2] Siemens Healthineers, Forchheim, holger.nehls@siemens-healthineers.com
[3] Siemens Healthineers, Forchheim, jochen.michel@siemens-healthineers.com

the real world. The system depends on a wide set of parameters that represent quantities from the physical world, such as dose parameters, geometric properties and special scanner capabilities. Besides the program code per se, the complexity of software is also due to the big variability space defined by these configuration parameters. Valid combinations of parameters reflect physical capabilities of the devices and the desired clinical cases to be performed. A central challenge that the scanner development teams need to address is to keep the parameters consistent for a wide variety of clinical cases, on different hardware and across product lines. Inconsistencies of the parameters configurations can lead to bad imaging or even damages to the CT scanners.

Traditionally (Figure 1-up), experts from the CT domain (e.g. physicists) define possible parameter configurations using tables in MS Excel and MS Word documents. These documents are written in plain natural language and have a very weakly defined structure. Quality assurance of these specification documents is realized exclusively through manual reviews. Once the valid configurations are defined, a semi-automatic process involving manual transformations and different scripts is employed to generate configuration files in XML format which can be loaded by the scanner software. This manual process of creating the specifications is slow, prone to inconsistencies and reaches its limits due to the complexity of modern CT scanners.

The use of domain specific modeling tools drastically increases the development productivity and quality: on the productivity side domain experts benefit from a higher level of abstraction and higher automation; on the quality side they benefit from modeling guidance and advanced consistency checks. To cope with the complexity of the CT domain, we have built a set of

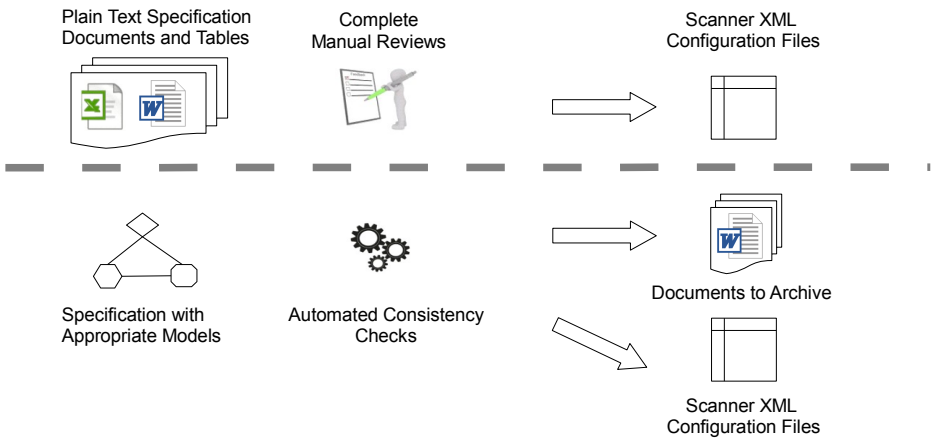

Fig. 1: Specify parameters and their relations using plain text and Excel tables requires big review effort for quality assurance and fragile semi-automatic generation of XML files (top); Model-driven specification of parameters enables deep and automatic consistency checks and automatic generation of XML-based configuration files to be loaded on the server (bottom).

domain specific languages and tooling (in the following called *Scanner-DSL*) which we use to specify the configurations of parameters of CTs. These rich models allow a wide range of consistency checks and automatic generation of configuration files in XMLs format and which are subsequently loaded in CT scanners (Figure 1-bottom). Besides XML, for process compliance reasons, we also generate PDF documents which are subsequently archived.

Contributions: In this paper we present our experiences with developing an eco-system of domain specific modeling languages over the last two years. Model-driven development approaches are widely used within Siemens Healtineers. However, the Scanner-DSL project is the first big project based on language engineering technologies. Thereby, besides presenting the tooling per se, this paper also describes our approach on technology transfer and adoption of domain specific modeling approaches in an industrial setting. Last but not least, based on our experience, we derive a set of lessons learnt and open challenges which need to be addressed for a broader adoption of the technology.

Structure: In Section 2, we give a brief overview of the technologies used and the developed DSLs. In Section 3 we present the development process and the major phases of our project. In Section 4 we discuss variation points of the approach and present our lessons learnt. In Section 6 we conclude the paper and give an outlook on future work.

2 Scanner-DSL

Scanner-DSL is an eco-system of languages and tooling built with Jetbrains' MPS[4] language workbench and the extensions offered by mbeddr-platform[5]. In Figure 2 we illustrate the architecture of our tool.

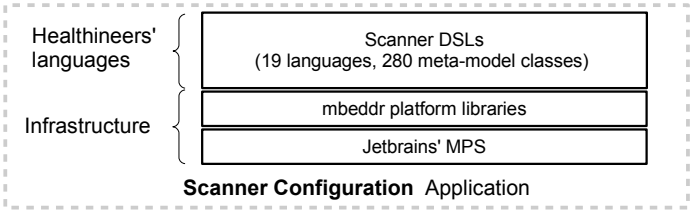

Fig. 2: Scanner-DSL architecture at a glance: the application, built around a set of domain specific languages, is based on Jetbrain's MPS and the extensions provided by the mbeddr-platform.

For reasons of brevity, we will not describe MPS or the mbeddr-platform libraries in detail; we refer the reader to [Ca14], [Vo13] or [Vo16]. However, in order to make this paper self-contained, we will briefly present in the following two subsections the major features

[4] https://www.jetbrains.com/mps/
[5] http://mbeddr.com/platform.html

of MPS and of mbeddr-platform which we used in our project. In Section 2.3 we briefly describe our tool.

2.1 Jetbrain's MPS

Jetbrains' Meta-Programming System[6] is an open-source language workbench which offers comprehensive support for all concerns of the development of DSLs and associated tooling. In MPS, a language implementation consists of several *language aspects* – e.g. structure, concrete syntax, constraints, type system, transformations, interpreters, or debuggers. MPS ships with a set of dedicated DSLs for implementing each language aspect.

Editors. MPS, at its core, features a projectional editor to display models. Projectional editors do not use parsers; instead, they render, or project, a program's abstract syntax tree (AST) in a notation defined by the language developer. Language engineers can choose to use specific notations appropriate for the business domain they address – e.g. plain text, forms, tables, diagrams, mathematical formula or trees.

Context Sensitive Constraints. MPS guides the language users (i.e. domain experts) towards building models in two ways: 1) *constructively* by preventing the definition of invalid models up-front using an advanced set of scopes and constraints; and 2) *analytically* by allowing the definition of advanced checks in the IDE. The constructive way is using the projectional nature of MPS directly – the users are allowed to enter only valid content. Further constraints are essentially implemented as if-statements that check some property of the AST and report errors if invalid code is detected.

Generators. MPS generators usually work as a chain of model-to-model transformations where domain-specific ASTs get enriched by platform-specific information with each transformation step. Eventually, a chain reaches the target language and a model-to-text transformation produces an output text file. Whilst MPS generators are not bound to this design, it is the most common use case since it allows for a very modular approach to combine and interchange transformation steps.

Tooling via IDE Extensions MPS also allows the definition of IDE extensions such as new menus or views; language engineers use these extensions extensively for building domain specific tooling or to integrate external tools. The IDE extensions are implemented via regular Java/Swing programs and a couple of MPS-specific extension points.

[6] `https://www.jetbrains.com/mps/`

Foundational Support for Model-driven Development MPS offers comprehensive support for the entire life-cycle of model based development with domain specific languages: integrating with different version control systems, merging and diffing at model level, testing of different aspects of the language definition, refactorings of languages and models, migration of models when languages evolve.

2.2 mbeddr Platform

Besides the set of DSLs and language definitions aspects shipped with MPS, we made use of an additional set of libraries for developing new languages. These libraries are offered by the *mbeddr platform* [mbe15] provide additional language definition aspects and DSLs for defining special editors. We have made use of tabular and mathematical notations [VL14], grammar cells for the definition of consistent editors and language documentation aspect.

2.3 Scanner DSL

In Figure 3 we present a screenshot of our tool which contains examples of three models, each built with a different DSL: a model which describes the available parameters with their set of possible values (top-left) and two models describing possible valid combinations of these parameters. Our models make use of textual, mathematical and tabular notations. On the bottom-right we illustrate errors caused by a failed consistency check.

Our eco-system of DSLs contains 19 languages providing 264 concepts (i.e. meta-model classes), with 57 properties and 264 relations between them. We implemented 59 constraints and scoping rules which restrict constructively the building of semantically flawed models and 112 more complex consistency checks. For quality assurance of these languages, we have extensively used the testing infrastructure of MPS. We created ca. 60 test-cases with more than 450 assertions. The generators are tested by comparing the generated artifacts with a manually reviewed baseline.

Usage Our tool is currently used in production to model three existing scanners. The initial modeling was done by one of the members of the language engineering team. We were constantly in touch with our domain experts and future users of the tool. At the end of 2017, the user models have 104 parameters, 1111 composite rules containing 5553 atomic rules distributed across more than 200 tabular rules. These models have been recently taken over and enhanced by three domain experts. Our domain specific tool went into production several months ago and is used today full-time by two users (both non-computer scientists).

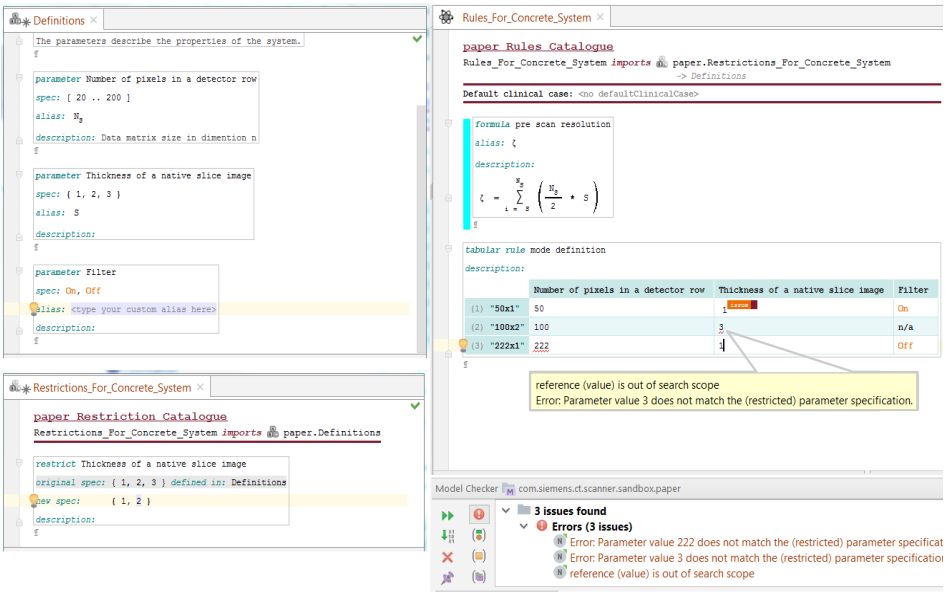

Fig. 3: Examples of models built with three different DSLs. The models are deeply integrated with each other which enables complex consistency checks.

3 Development Process

In the following we present three highlights of our project concerning the development process: the phases of our project, enabling continuous integration of languages and models and involving domain experts.

3.1 Project Phases

Our development process varied substantially within the last two years of the project and can be divided into four phases. In Table 1 we present an overview over these phases, their duration and the number of persons involved.

Phase 1: Ramping-up During the first 6 months of the project, until the team got confident with the technology, we had a ramping-up phase. It was characterized by the exploration of the MPS technology stack and rapid prototyping of relevant use-cases in order to understand the limitations. The main goal was to produce enough functionality to convince the other teams about the meaningfulness of the modeling approach. During the ramping-up phase most of the development took place during several hackathons, each of them being three days long.

Phase 2: Initial development After the ramping-up phase, several team members got confident with the MPS technology and the team allocated half a person for the development. Soon after, the first interns joined the team and the development got a higher dynamics also in-between the hackathons. The hackathons were used to explore advanced features of MPS and to solve more complex problems.

Phase 3: Mature project After one year, the development was accelerated in order to synchronize with the planned deadlines. Our team grew further and this lead to a significant increase of the code-base and functionality. We integrated our project in the existing continuous integration infrastructure and increased the coverage of our tests.

Phase 4: Production After one and half years since project start, our system went into production and the size and the number of user-models describing parameters configurations started to grow rapidly. The first domain experts started to use our DSLs and the development of languages and user models got different dynamics. This lead to the necessity to decouple the life-cycles of language development and the use of languages for developing CT specifications.

Phase	Duration (in months)	Team-size (#developers / #students)
Ramping-up	6	0.5
Initial devel.	6	0.8 / 1
Mature project	9	1.2 / 2
Production	5	2.8 / 2

Tab. 1: The dynamics of our project changed substantially between phases: we started with a very small team and and once the value of the technology had been proven, the team was increased in size and contributes directly to productive software development projects.

3.2 Continuous Integration

Both the language engineering and the domain experts teams are distributed: language development happens in Forchheim, Cologne and Munich. The domain experts work distributed from Germany and China. Starting relatively early in the development of the Scanner-DSLs, we introduced continuous integration for the development of DSLs. After reaching the first functional milestone, we deployed the languages and tooling as a standalone application to be used by domain experts. In Figure 4 we illustrate these two delivery pipelines: the domain experts use releases of the Scanner-DSL tooling for building their models. Artifacts (like configurations as XML) are generated automatically from the models and integrated within the continuous integration pipeline for our scanner software.

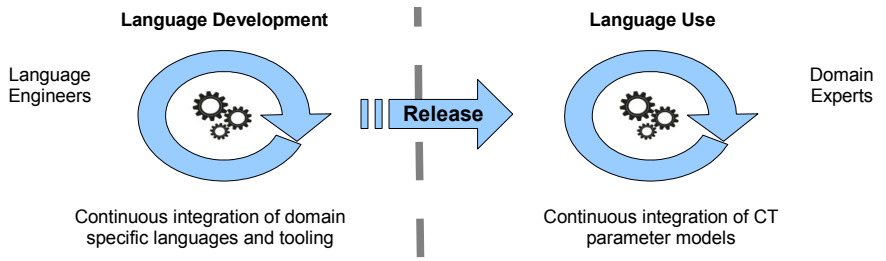

Fig. 4: The continuous integration is performed both for the language development and for the language use. When the eco-system of DSLs reach a new baseline, we make a release of the Scanner-DSL tooling and the domain experts use this release for their development of parameters configurations for the CT scanners.

3.3 Involving Domain Experts

The DSLs we develop capture the semantics of the CT scanner domain in an explicit and precise manner. The language development process went closely together with a knowledge engineering process inside the organization. At any point in time, at least one member of the language development team had several years of experience with the scanner software development and thereby in-depth domain knowledge.

During creation of languages, we had to make the domain knowledge explicit (i.e. choose which concepts and relations among them are captured as first class language constructs, and which constraints need to be implemented). We identified edge-cases for which several domain experts needed to be involved in discussions. Many times these experts had a slightly different view over their domain and they needed to agree upon how the domain looks like.

4 Discussion and Lessons Learnt

In this section we discuss important variation points of our approach and present our lessons learnt and open challenges.

4.1 Discussion

On Projectional Editing One of the most important distinguishing feature of MPS is the projectional editing. Being a projectional editor, MPS does not feel like text editors when users edit their models. In order to increase the fluency of models' creation and modification, MPS allows advanced customization of editor actions – e.g. what happens when the user presses ”backspace” in a certain editor cell, or how are linear sequences of lexical items are transformed into models. We have invested some effort to make the editing experience as

intuitive as possible. At the same time this means that the users, who know and sometimes expect fundamental features of Excel or Word, need to understand that the focus of the new tool (from their point of view) is not to mirror known textual editor features, but to model the semantics of the computed tomography scanner data. The initial feedback from our users with respect to the usability of the editor is positive – the users immediately understood that they are not editing ”simple” text but rich models and thereby they calibrated their expectations.

On MPS's Extensibility The seamless extension capabilities of MPS with additional language definition aspects is a key feature which we made use of in our project. Besides the standard aspects shipped with MPS, as presented in Section 2, we have intensively used the ”mbeddr-platform” libraries featured as part of mbeddr. Dependencies among different extensions need to be managed appropriately such that the deployed Rich Client Platform (RCP) can be built in a meaningful manner.

On Language Evolution and Models Migration on Multiple Branches MPS provides out-of-the-box advanced support for evolving DSLs and migrate the models to the new versions of the DSLs. We have used these features intensively to perform agile language development. However, if the models are built on different branches then they also need to be migrated to new language versions individually. Comparing (or merging) these branches after migrations have been performed on them proves to be challenging.

4.2 Lessons Learnt

DSL Development and Domain Engineering go Hand-in-hand. In addition to the DSL development itself, substantial effort has been involved in domain engineering. The knowledge of domain experts (i.e. domain concepts, their relations and constraints on valid combinations) at a certain point in time was formalized in the DSL. The DSLs help us better understand, manage and consolidate the knowledge in our organization. Once initial versions of the DSL was built, it was subsequently piloted to model different aspects of the scanner domain and by doing this we identified improvements of DSL.

Along with the development of DSLs the team went through a learning process about the domain – we continuously got feedback from domain experts – and ca. 10-15 persons are aware about different details of our DSLs and continuously validate what we are developing.

Continuously Demonstrate the Added Value from Early Stages Initial experience with creating the user models was very useful to convince the stakeholders about the value of the model-based approach. We were able to detect several inconsistencies in the original

data which passed through multiple-review sessions and this was a strong argument for semantically rich models. The possibility to generate XML configuration files from models quickly and in a fully automated way served as additional argument for our approach.

MPS Enables Highly Efficient Development of DSLs The infrastructure provided by MPS and mbeddr-platform allowed us to develop the languages in a highly efficient manner. After a few hours of development, we could get a baseline for languages and tooling which can be used as input to engage in discussions with domain experts. This baseline is then subject to iterative improvements, each iteration consisting often only of several hours of development.

Need for Support for the Entire Life-cycle of DSLs The support for entire life-cycle of DSL engineering and development offered by MPS proved to be essential - starting from the DSLs used to define different language aspects, with testing, refactoring and support for continuous integration.

Configuration Management The DSL development team is distributed between Forchheim, Cologne and Munich. The team of domain experts using the DSLs is distributed as well between Germany and China. The support offered by MPS for advanced versioning and merging both for language development as well as for the language use is of high importance for the adoption.

Semantic Richness Besides the definition of appropriate language constructs and constraints which prevent up-front building meaningless models, we have implemented a rich set of consistency and plausibility checks on the scanner models. These have proven to be highly useful and are appreciated by domain experts since they get feedback immediately in the IDE (e.g. in case consistency is violated). Corrective actions can be taken immediately before errors are discovered later in process or even introduced into the production.

Testing and quality assurance We have developed a comprehensive test-suite for testing the context-sensitive constraints and the generators. The checking rules are developed test-driven; the generator for XML artifacts is validated by using a baseline test method and reaches a block coverage of about 97% (measured at Java level using the *EMMA*[7] code coverage tool). Each test is performed latest on our build server, triggered by every commit. These tests proved to be highly useful whenever we evolved the DSLs or migrated them to a new version of MPS. Overall, we estimate that the effort spent on writing unit test was 30% of the total development effort. The integration of testing in our continuous integration framework accounted for further 10% of the effort. We feel however that the testing capabilities of MPS could be enhanced towards support for ”end-to-end” testing.

[7] `http://emma.sourceforge.net/`

5 Related Work

[Vo17] presents lessons learnt from developing mbeddr, an open-source stack of domain specific languages built on top of C using Jetbrains' MPS. mbeddr is one of the biggest DSLs based projects involving 10+ person years of development effort. This is the closest work on experience with instantiating DSLs technology with Jetbrains' MPS. Compared to [Vo17], this paper presents experiences and particularities with transfering the language engineering technologies into industrial context and the entire life-cycle (i.e. from domain engineering to supporting domain experts in using the tooling) of deploying DSLs.

[MPP14] presents experiences and challenges with domain specific modeling in industrial automation domain. [To16, TK16] describes experiences with introducing MetaEdit+ language workbench in industrial context to create domain specific modeling languages. The experienced presented by Tolvanen and colleagues are very much similar to our experiences: support for the entire life-cycle of languages and models is needed, domain specific modeling and domain specific tooling drastically increase the productivity of the software development. Compared to these works, our experiences in this paper are based on a single, medium sized language engineering project in the healthcare domain. We also describe our approach to transferring language engineering technology in industrial practice.

6 Conclusions

In this paper we presented the first results from a two years endeavor on deploying domain specific languages to describe parameters and their configurations for CT scanners. This is only a first step to a holistic model based approach tailored to the needs of Siemens Healthineers Computed Tomography. The efforts reported here are part of a longer term project aimed at increasing the automation of the development of Computed Tomography scanner software. This is only the initial baseline representing one specification document – we plan to extend the languages and models for up to other 100 specifications successively. In parallel, the development of new innovative systems continues and new areas of applicability can be anticipated.

Acknowledgements. We would like to thank Robert Walter[8] for the discussions and feedback on this paper.

References

[Ca14] Campagne, Fabien: The MPS Language Workbench. CreateSpace Publishing, 2014.

[mbe15] mbeddr Platform. `http://mbeddr.com/platform.html`, 2015. Accessed: 2017-12-15.

[8] Independent Consultant, Gleueler Str. 179, 50931 Köln, info@digital-ember.com

[MPP14] Moser, Michael; Pfeiffer, Michael; Pichler, Josef: Domain-specific Modeling in Industrial Automation: Challenges and Experiences. In: Proceedings of the 1st International Workshop on Modern Software Engineering Methods for Industrial Automation. 2014.

[TK16] Tolvanen, Juha-Pekka; Kelly, Steven: Model-Driven Development Challenges and Solutions - Experiences with Domain-Specific Modelling in Industry. In: Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development - Volume 1: Ind Track MODELSWARD. 2016.

[To16] Tolvanen, Juha-Pekka: MetaEdit+ for Collaborative Language Engineering and Language Use (Tool Demo). In: Proceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering. 2016.

[VL14] Voelter, Markus; Lisson, Sascha: Supporting Diverse Notations in MPS Projectional Editor. In: Workshop on The Globalization of Modeling Languages, co-located with MODELS. S. 7–16, 2014.

[Vo13] Voelter, Markus; Benz, Sebastian; Dietrich, Christian; Engelmann, Birgit; Helander, Mats; Kats, Lennart; Visser, Eelco; Wachsmuth, Guido: DSL Engineering. dslbook.org, 2013.

[Vo16] Voelter, Markus; Szabó, Tamás; Lisson, Sascha; Kolb, Bernd; Erdweg, Sebastian; Berger, Thorsten: Efficient development of consistent projectional editors using grammar cells. In: Proceedings of the International Conference on Software Language Engineering. 2016.

[Vo17] Voelter, Markus; Kolb, Bernd; Szabó, Tamás; Ratiu, Daniel; van Deursen, Arie: Lessons learned from developing mbeddr: a case study in language engineering with MPS. Software & Systems Modeling, 2017.