Generating Metamodels from Grammars by Chaining Translational and By-Example Techniques*

Alexander Bergmayr and Manuel Wimmer

Business Informatics Group, Vienna University of Technology, Austria
lastname@big.tuwien.ac.at

Abstract. Bridging grammarware and modelware is still challenging, though often required as a prerequisite for several model-driven engineering scenarios. For instance, in model-driven reverse engineering, program code has to be lifted to the model level before model-driven techniques are applicable. Manually building metamodels based on given grammars introduces a significant overhead and may lead to inconsistencies between the resulting metamodels and the grammars, especially when dealing with large languages.
In previous work, we have investigated a purely translational approach that is able to semi-automatically generate metamodels from grammars by utilizing user input. In this work, we aim to provide a higher degree of automation by combining the translational approach with by-example techniques to reduce the manual effort. In particular, knowledge is derived from concrete programs to further refine the produced metamodels. We demonstrate a transformation chain that combines translational and by-example techniques to produce Ecore-based metamodels from EBNF-based grammars.

1 Introduction

In the last decade, much effort has been invested to bridge the gap between GrammarWare (GW) [22] and ModelWare (MW) [24], thereby gaining advantages from both areas by transferring artifacts and techniques between these technical spaces [18]. Approaches for transforming grammars into metamodels [1,23,31], programs into models [10], and APIs into models [9] have been presented. Furthermore, the development of concrete textual syntaxes for (modeling) languages has been proposed [12,14,19,27]. Thus, the current research landscape offers several concepts, techniques, and tools to switch back and forth between the two technical spaces. However, efficiently establishing bridges between them at the meta-level remains still a major challenge, in particular when artifacts from the GW technical space have to be shifted to the MW technical space.

* This work is co-funded by the European Commission under the ICT Policy Support Programme, grant no. 317859.

Challenge. We explored such scenarios in the ARTIST project[1] [5], which aims at modernizing legacy code by a migration towards cloud environments. Model-Driven Reverse Engineering (MDRE) offers useful features for such scenarios, even though these features are employable only if some prerequisites are fulfilled in the MW technical space, which includes the existence of (*i*) a metamodel that corresponds to the given grammar of the program code, (*ii*) a parser, i.e., a text-to-model transformation, to build models for programs, and (*iii*) a printer, i.e., a model-to-text transformation, to produce programs from models if forward engineering or round-trip engineering needs to be supported. These artifacts facilitate efficiently switching between both technical spaces. Clearly, certain essential properties have to be fulfilled in this respect. First, the produced metamodel has to be consistent with the grammar, i.e., a valid sentence in the grammar needs to be expressible as a valid model of the metamodel and vice versa. Second, the text-to-model and model-to-text transformations have to be information-preserving. The development of such bridges between GW and MW, even if guided by tool support, still involves several manual tasks that require extensive effort from a user perspective, especially when the rich features of current metamodeling languages of the MW technical space should be exploited.

Contribution. To reduce the manual effort and the heavy involvement of users in the development of bridges between GW and MW, we present in this paper a transformation chain that semi-automatically generates Ecore-based metamodels from given EBNF grammars [16]. The transformation chain glues together several transformations that serve as translators to generate an initial version of a metamodel. To further refine such an initial metamodel version, we apply a by-example approach where the examples are programs expressed in the grammar for which a metamodel is generated. This second step of the transformation chain is novel in this context and essential to reduce the manual effort of the metamodel generation process and at the same time to achieve high quality metamodels as a result of this process. For example, EBNF provides only limited capabilities to express typed cross-references between production rules. In fact, such information is not covered by EBNF grammars, though indeed required for the generation of an appropriate metamodel. This kind of extra information is in our metamodel generation process inferred by applying a by-example approach, which is inspired from existing work on bottom-up (meta)modeling [2,25], on grammar, schema, metamodel, and ontology recovery [7,11,17], and on solving Model-Driven Engineering (MDE) problems by exploiting by-example techniques, cf. [3,13,21,28,30,32] for concrete approaches and [20] for a survey.

Structure. In Section 2, we briefly describe the background of our work and introduce a motivating example that shows one particular challenge when moving from GW to MW, namely how to infer types of cross-references that are not represented in EBNF-based grammars. In Section 3, we present our transformation chain and demonstrate its application on the motivating example by relying on EBNF in the GW technical space, and Xtext[2] and Ecore in the MW

[1] `http://www.artist-project.eu`
[2] `http://www.eclipse.org/Xtext`

technical space. In this context, the application of Xtext as a mediator in the transformation chain is novel. Xtext appears to be useful because it provides features of both technical spaces. We conclude with an outlook towards an extensive evaluation of our approach in Section 4.

2 Prerequisites and Motivating Example

With today's language development tools and established model transformation techniques, metamodels and corresponding parsers/printers can be automatically generated from grammars defined with meta languages residing in the MW technical space. Exploiting this automatic generation for producing metamodels also from EBNF-based grammars, i.e., grammar definitions residing in the GW technical space, relies on two assumptions: (*i*) the heterogeneities of the meta-languages used in GW and MW can be resolved on a syntactical level, and (*ii*) the information needed to produce high quality metamodels is provided by artifacts of the GW technical space. We assume in our transformation chain EBNF as the meta-language in the GW context, and Xtext and Ecore as the meta-languages in the MW context.

Xtext is a language for defining textual syntaxes of languages and allows the generation of Ecore-based metamodels and corresponding parsers/printers. Hence, Xtext and Ecore are well integrated with each other. As a result, the challenge is to bridge EBNF with one of the two candidates. Xtext seems to be the preferable choice as an intermediate format, thereby avoiding to lose the concrete syntax elements typically expressed in an EBNF grammar because Ecore is capable of representing elements from an abstract syntax perspective, only. Thus, Xtext may serve as a mediator between EBNF and Ecore as it is a hybrid approach, comprising technological features and meta-language concepts from both technical spaces.

2.1 Motivating Example: The MiniJava Grammar

Let us take MiniJava[3] as a concrete example language that is expressed in EBNF and for which we want to generate a metamodel. Furthermore, for the purpose of this paper, let us focus on one concrete production rule from the MiniJava grammar, as shown in Listing 1.1.

Listing 1.1: Production rule *ClassDeclaration* in EBNF

```
ClassDeclaration = "class", Identifier, [ "extends", Identifier ];
Identifier = {'a' | 'b' | ...}
```

Straightforwardly translating the *ClassDeclaration* production rule expressed in EBNF to a parsing rule expressed in Xtext would result in the Xtext-based grammar given in Listing 1.2.

[3] MiniJava is a small language related to Java that is mainly used for teaching purposes. More information on the language may be found at: `http://cs.fit.edu/~ryan/cse4251/mini_java_grammar.html`

Listing 1.2: Parser rule *ClassDeclaration* in Xtext

```
ClassDeclaration: "class" name=IDENTIFIER ("extends" superClass=
    ↪IDENTIFIER)?;
terminal IDENTIFIER: ('a' | 'b' | ...)*;
```

Besides some small syntactical differences, we are able to express the *ClassDeclaration* rule in both meta-languages in a similar way. However, Xtext provides several additional features compared to EBNF. These features facilitate the generation of meaningful Ecore-based metamodels from Xtext-based grammars. In particular, Xtext allows us to distinguish between different kinds of rule calls. In the context of our example, the first rule call is used to express the name of a declared MiniJava class while the second rule call actually represents a cross-reference to the class, which serves as super class. If we do not distinguish between these two different intentions of the rule calls, we would end up with just having String-typed attributes in the corresponding Ecore-based metaclasses generated from the Xtext grammar instead of having typed references as well. Figure 1(a) depicts the metaclass corresponding to the *ClassDeclaration* rule in Listing 1.2. However, what we actually would like to achieve is the metaclass as illustrated in Figure 1(b) because it provides a typed reference for expressing inheritance relationships between MiniJava classes.

Fig. 1: Derived and desired Ecore-based metamodels for the production rule *ClassDeclaration*

When taking a closer look at our *ClassDeclaration* rule in Listing 1.2, a better solution is to make use of Xtext's support for cross-references and exploit built-in terminals provided in terms of a library. Listing 1.3 depicts the required Xtext grammar to obtain the desired metaclass in Figure 1(b).

Listing 1.3: Desired parser rule *ClassDeclartion* in Xtext

```
ClassDeclaration: "class" name=ID ("extends" superClass=[
    ↪ClassDeclaration])?;
```

This example shows that with a straightforward translation of EBNF-based grammars to Xtext-based grammars, we may not obtain the metamodel which we would expect from the perspective of the MW technical space. Thus, additional effort is required to produce meaningful metamodels in the expected quality. In the following, we present a concrete transformation chain that is able to generate metamodels comprising typed cross-references. The type information is inferred from example programs expressed in terms of the original grammar.

3 GW2MW Transformation Chain By-Example

Figure 2 provides a conceptual overview of our transformation chain and highlights the prerequisite steps for the automatic metamodel generation: EBNF grammars need to be expressed in terms of Xtext (Section 3.1), the heterogeneities between EBNF and Xtext need be resolved (Section 3.2), and additional information is needed from example programs to exploit the rich features of meta-languages used in the MW technical space (Section 3.3).

To allow for applying MDE techniques in our transformation chain, the first step ① is to transform a given EBNF-based grammar into a model-based representation. For this step, we need a preparation step ⓪ that relies on the formalisation of the EBNF language in Xtext, i.e., an Xtext-based grammar is developed for EBNF. By this, we reach temporarily the M4 layer in the metamodeling stack by describing a meta-language with a different meta-language. Having the Xtext-based grammar for EBNF allows to derive an Ecore-based metamodel for EBNF and a parser for injecting grammars expressed in EBNF into the MW technical space.

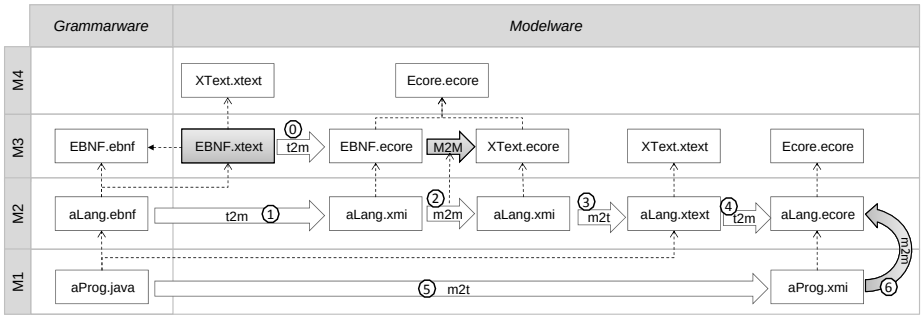

Fig. 2: GW2MW Transformation Chain at a Glance

Now we are able to apply MDE techniques to actually shift a given language definition represented as a model towards Xtext ② by applying a model transformation we developed in ATL between the metamodels of EBNF and Xtext. By relying on the printer capabilities provided by Xtext, the generation of an Xtext-based grammar for a given language definition is achieved ③. This also eliminates the M4 meta-level. The generation of an Ecore-based metamodel for the language definition expressed as an Xtext-based grammar ④ is out-of-the-box provided by Xtext. Furthermore, the Xtext-based representation of the language definition allows to exploit further the capabilities of Xtext, namely to parse programs that conform to the given language ⑤. The result of our transformation chain, i.e., the Ecore-based metamodel and the models representing the programs, is fully operable, though the quality of the generated metamodels requires improvement ⑥ to render them useful for future MDE tasks that rely on these metamodels. A by-example approach is the technique of choice to achieve such improvements in this paper. In particular, information is derived in this step from programs that is missing in the grammar.

3.1 Expressing EBNF Grammars in Xtext

To reach the MW technical space for the given artifacts in the GW technical space, our current approach is to express EBNF in Xtext[4] to enable the parsing of language definitions expressed in EBNF as well as to parse programs conforming to this language definitions.

To automatically generate the Xtext grammar in Listing 1.3 from the EBNF grammar given in Listing 1.1, the actual intention behind the *Identifier* needs to be expressed and the type of the reference is required to be known if the intention refers to a cross-reference, as discussed in the motivating example. Therefore, we introduced dedicated annotations into our Xtext-based EBNF grammar to ensure an effective transformation of the intention behind rule calls. Listing 1.4 shows the annotated version of the *ClassDeclaration* production rule. The user has to mark the rule calls that refer to the provision of the identifier values for a language concept (cf. @*id* annotation) as well as for rule calls that actually represent references to other elements (cf. @*idref* annotation).

Listing 1.4: Annotated production rule *ClassDeclartion* in EBNF

```
ClassDeclaration = "class", @id Identifier, [ "extends", @idref(
    ↪superClass) Identifier ];
Identifier = {'a' | 'b' | ...};
```

Although with this approach the dedicated rules in the model transformation to generate the expected result can be triggered, the type of the *superClass* reference remains generic. While the annotations may be manually introduced even in large grammars with reasonable effort, the decision for an adequate type appears to be challenging in general when just reasoning on the grammars without taking a look on a larger example base. Here our design rationale is that information that may be inferred by the user in a local context, e.g., by looking at one particular production rule, is provided by the user, while information that requires reasoning on a global scope is derived automatically by the help of analyzing examples.

3.2 Transforming EBNF Grammars to Xtext

On a general level, concepts of EBNF straightforwardly map to concepts of Xtext. As Xtext provides a richer set of concepts compared to EBNF, the model transformation between them needs to resolve certain heterogeneities. From a technical perspective, the model transformation addresses features of Xtext that are required to obtain an operable parser/printer. For example, identifiers are often only informally specified by EBNF grammars. Xtext provides built-in terminals that already fulfill the typical requirements. Perhaps, even more important, the model transformation needs to deal with conceptual aspects as well. Besides the difficulties that may arise with LL-based parsing, effectively dealing with our annotations in the EBNF grammar falls in this category. Listing 1.5

[4] An Xtext-based EBNF implementation can be found under `http://xtexterience.wordpress.com/2011/05/13/an-ebnf-grammar-in-xtext`

shows the required adaptations in the EBNF grammar for providing annotations in language definitions.

Listing 1.5: Annotation support for EBNF

```
Expression_Rule_Reference returns Expression:
    ({Expression_Rule_Reference} (((idRef ?= "@idref") ("("
        ↪refName=NAME ")")?) | (idName?="@id"))? rule=[
        ↪ProductionRule|NAME]);
```

Rule calls annotated with @*id* are transformed to "name" attributes while @*idref* leads to a more complex output in the sense that a common production rule is temporarily introduced as a default candidate for the reference end-point.

Listing 1.6 shows the automatically generated Xtext grammar for the *ClassDeclaration* parsing rule. Compared to Listing 1.3, the type of *superClass* is not yet *ClassDeclaration* as this information cannot be inferred solely from the given grammar. Instead, *CommonReferenceRule* serves as intermediate placeholder by generalizing all other existing production rules in the grammar. Ideally, example programs allow us to subsequently infer the most specific types as depicted in the refined version of the grammar in Listing 1.3.

Listing 1.6: Automatically generated parser rule *ClassDeclartion* in Xtext

```
ClassDeclaration: "class" name=ID ("extends" superClass=[
    ↪CommonReferenceRule])?;
CommonReferenceRule: ClassDeclaration | ... | ... ;
```

3.3 By-Example Refinement of Xtext-based Grammars and Ecore-based Metamodels

Inspired by existing work on bottom-up (meta)modeling [2,25] and on grammar, schema, metamodel, and ontology recovery [7,11,17], we provide an additional refinement step for the initially produced metamodels. In fact, although a grammar of the language is available, we still miss some important information to produce the desired metamodel that is not given by the EBNF-based grammars. One aspect we focus on in this paper is the typing of the cross-references that we explored particulary challenging from a users perspective when dealing with large grammars allowing complex links between different kind of elements.

By having the initially produced Xtext grammar as shown in Listing 1.6, we already have the possibility to parse the programs into models. In these models, the concrete links between model elements are available that can be analyzed to produce the information to further refine the generated cross-references of the Xtext grammar and in the corresponding Ecore metamodel.

In the by-example based refinement step, we apply a *select/infer/adapt* process that—as the name already suggests—selects the language definition parts that may need further refinements, infers additional information from the examples, and adapts the language definition part by incorporating the newly inferred information.

For our concrete task of inferring the exact types, i.e., the most specific types, of cross-references, we show the used algorithm of the by-example reasoning

process in Listing 1.7 using OCL-like pseudo code notation. As input we need the metamodel to be refined (the refined version of it is the output of the algorithm) and the example base represented by a set of models that are produced from a set of programs by the parser generated from our initial metamodel.

Listing 1.7: By-example type refinement of cross-references

```
var mm : Metamodel; -- input/output
var mb : Set{Model}; -- input
-- selection phase
Set{Reference} refsToAdapt = mm.getCrossReferences();
refsToAdapt -> foreach(ref |
    -- inference phase
    Set{Link} links = mb -> collect(m|m.getLinksOfType(ref)) ->
        ↪flatten();
    Set{Class} classes = links -> collect(l|l.getTargetType()) ->
        ↪asSet();
    -- adaptation phase
    if classes.size() = 1 then ref.setTargetType(classes.first());
    else if ... endif endif
)
```

Due to space limitations, we only show one concrete case of the type refinement process. First, we collect in the selection phase all cross-references available in the metamodel (because all of them are only typed to *CommonReferenceRule* in the initial version of the metamodel). For the inference phase, we iterate over this set of references and for each reference we query the instantiated links from all models to collect the types of the target objects referenced by the links. By converting this collection to a set, we get the information that we need for the adaptation phase. Here we only discuss one possible case, namely if all referenced objects are of the same type, i.e., the set has only one entry, that results in a simple adaptation by just setting the type of the reference under study to the inferred type. In cases where objects are referenced that are of different types, more sophisticated adaptations are necessary, e.g., the introduction of new classes that act as super classes for the inferred types, because in Ecore only one type is allowed for each reference. Coming back to our motivating example, when executing this algorithm on a set of example models, we are able to transform the Xtext grammar shown in Listing 1.6 into the Xtext grammar of Listing 1.3. From the latter, the meta-class as shown in Figure 1(b) can be automatically derived that represents the expected metamodel definition for the MiniJava grammar excerpt.

4 Outlook

With the implementation of our transformation chain and the experimentation with the MiniJava grammar, we achieved initial evidence that metamodels can be automatically generated from grammars and refined by applying by-example based techniques. By having the latter step, we are able to infer additional knowledge about the language that is not directly presented in the grammar definitions but in other tools residing in the GW technical space such as compilers. Furthermore, the assumption of having enough examples to provide meaningful

refinements of metamodels seems valid, because the trigger for producing the metamodels is the existence of GW artifacts that should be transferred to MW.

On the basis of our results, we plan an extensive evaluation of our approach with grammars from well-established languages such as Java or C# as next steps. In particular, different variants of a Java metamodel [8,15] are already available. The idea is to use them as a reference for evaluating our results from the transformation chain. Large open-source code bases provide excellent input for inferring potential refinements of the generated metamodels. We may then exploit existing work for type-safe restructuring on the meta-level [6] to turn such refinements into concrete model transformations that automate this task. By applying such model transformations, the hope is to significantly improve the quality of the generated metamodels. Architectural metrics, such as, mean features per classifiers, mean inheritance hierarchy depth, and understandability [4,26] may serve as reference for proofing actual improvements. The quality of the generated metamodels seems to be crucial for their use in real MDE scenarios where queries and transformations are formulated against the metamodels. In this paper, we discussed an important aspect, namely computing the exact types of cross-references, but several other aspects that may be enhanced by applying by-example based reasoning remain as future challenges. Furthermore, one refinement for our inferencing algorithm would be necessary for dealing with non-global unique identifiers, such as having a field declaration and a method declaration with the same name. A promising approach in this context is to incorporate Java binding rules as presented in [29].

References

1. Alanen, M., Porres, I.: A Relation Between Context-Free Grammars and Meta Object Facility Metamodels. Tech. rep., Turku Centre for Computer Science (2003)
2. Bagheri, H., Sullivan, K.: Bottom-up model-driven development. In: ICSE. pp. 1221–1224 (2013)
3. Balogh, Z., Varró, D.: Model transformation by example using inductive logic programming. SoSym 8(3), 347–364 (2009)
4. Bansiya, J., Davis, C.G.: A Hierarchical Model for Object-Oriented Design Quality Assessment. TSE 28(1), 4–17 (2002)
5. Bergmayr, A., Bruneliere, H., Cánovas Izquierdo, J.L., Gorroñogoitia, J., Kousiouris, G., Kyriazis, D., Langer, P., Menychtas, A., Orue-Echevarria Arrieta, L., Pezuela, C., Wimmer, M.: Migrating Legacy Software to the Cloud with ARTIST. In: CSMR. pp. 465–468 (2013)
6. Bergmayr, A., Wimmer, M., Retschitzegger, W., Zdun, U.: Taking the Pick out of the Bunch - Type-Safe Shrinking of Metamodels. In: SE. pp. 85–98 (2013)
7. Bex, G.J., Neven, F., Vansummeren, S.: Inferring XML Schema Definitions from XML Data. In: VLDB. pp. 998–1009 (2007)
8. Bruneliere, H., Cabot, J., Jouault, F., Madiot, F.: MoDisco: A Generic and Extensible Framework for Model Driven Reverse Engineering. In: ASE. pp. 173–174 (2010)
9. Cánovas Izquierdo, J.L., Jouault, F., Cabot, J., García Molina, J.: API2MoL: Automating the building of bridges between APIs and Model-Driven Engineering. IST 54(3), 257–273 (2012)

10. Cánovas Izquierdo, J.L., Sánchez Cuadrado, J., Garcıa Molina, J.: Gra2MoL: A Domain Specific Transformation Language for Bridging Grammarware to Modelware in Software Modernization. In: Workshop on Model-Driven Software Evolution (2008)
11. Drumond, L., Girardi, R.: A Survey of Ontology Learning Procedures. In: WONTO (2008)
12. Efftinge, S., Voelter, M.: oAW xText - A framework for textual DSLs. In: Eclipse Summit Europe Conference (2006)
13. García-Magariño, I., Gómez-Sanz, J.J., Fuentes-Fernández, R.: Model transformation by-example: An algorithm for generating many-to-many transformation rules in several model transformation languages. In: ICMT. pp. 52–66 (2009)
14. Heidenreich, F., Johannes, J., Karol, S., Seifert, M., Wende, C.: Model-Based Language Engineering with EMFText. In: GTTSE. pp. 322–345 (2011)
15. Heidenreich, F., Johannes, J., Seifert, M., Wende, C.: Closing the Gap between Modelling and Java. In: SLE, pp. 374–383 (2010)
16. ISO: ISO/IEC 14977:1996(E), Information technology - Syntactic metalanguage - Extended BNF (1996)
17. Javed, F., Mernik, M., Gray, J., Bryant, B.R.: MARS: A metamodel recovery system using grammar inference. IST 50(9-10), 948–968 (2008)
18. Jézéquel, J.M., Combemale, B., Derrien, S., Guy, C., Rajopadhye, S.: Bridging the Chasm between MDE and the World of Compilation. SoSym 11(4), 581–597 (2012)
19. Jouault, F., Bézivin, J., Kurtev, I.: TCS: a DSL for the specification of textual concrete syntaxes in model engineering. In: GPCE (2006)
20. Kappel, G., Langer, P., Retschitzegger, W., Schwinger, W., Wimmer, M.: Model transformation by-example: A survey of the first wave. In: Conceptual Modelling and Its Theoretical Foundations. pp. 197–215 (2012)
21. Kessentini, M., Sahraoui, H.A., Boukadoum, M., Omar, O.B.: Search-based model transformation by example. SoSym 11(2), 209–226 (2012)
22. Klint, P., Lämmel, R., Verhoef, C.: Toward an engineering discipline for grammarware. ACM Trans. Softw. Eng. Methodol. 14(3), 331–380 (2005)
23. Kunert, A.: Semi-automatic Generation of Metamodels and Models From Grammars and Programs. Electr. Notes Theor. Comput. Sci. 211, 111–119 (2008)
24. Kurtev, I., Bézivin, J., Akşit, M.: Technological spaces: An initial appraisal. In: CoopIS (2002)
25. de Lara, J., Guerra, E., Sánchez Cuadrado, J.: Abstracting Modelling Languages: A Reutilization Approach. In: CAiSE. pp. 127–143 (2012)
26. Ma, H., Shao, W., Zhang, L., Ma, Z., Jiang, Y.: Applying OO Metrics to Assess UML Meta-models. In: UML. pp. 12–26 (2004)
27. Muller, P.A., Hassenforder, M.: HUTN as a bridge between modelware and grammarware-an experience report. In: WISME Workshop (2005)
28. Saada, H., Dolques, X., Huchard, M., Nebut, C., Sahraoui, H.A.: Generation of operational transformation rules from examples of model transformations. In: MoDELS. pp. 546–561 (2012)
29. Schäfer, M., Thies, A., Steimann, F., Tip, F.: A Comprehensive Approach to Naming and Accessibility in Refactoring Java Programs. TSE 38(6), 1233–1257 (2012)
30. Sun, Y., White, J., Gray, J.: Model transformation by demonstration. In: MoDELS. pp. 712–726 (2009)
31. Wimmer, M., Kramler, G.: Bridging Grammarware and Modelware. In: MODELS Satellite Events. pp. 159–168 (2005)
32. Wimmer, M., Strommer, M., Kargl, H., Kramler, G.: Towards model transformation generation by-example. In: HICSS (2007)