Transition from EBNF to Xtext

Jianan Yue

State Key Laboratory for Novel Software Technology, Nanjing University
Department of Computer Science & Technology, Nanjing University
210023 Nanjing, China
b111220168@smail.nju.edu.cn

Abstract. Xtext is a framework for developing programming languages and domain specific languages (DSLs). Xtext contains a language infrastructure including parsers, compiler and interpreter. In recent years, it has been applied to develop various DSLs. To benefit from Xtext, in certain cases, it is needed to transform an Extend Backus Naur Form (EBNF) based language to the Xtext grammar. For example, the well-known UML profile MARTE has a BNF-based Value Specification Language (VSL). It is needed to transform MARTE VSL in EBNF to a MARTE VSL Xtext grammar for the purpose of enabling tool integration. In this paper, as the first step towards a fully automated transformation from EBNF to Xtext, a number of transformation rules are defined. The ultimate objective of the project is to provide developers a fully featured and customizable Eclipse-based IDE for developing DSLs using EBNF grammar or enabling tool integration by providing transformation from EBNF to Xtext.

Keywords: Xtext, EBNF, meta-model, DSL, Transformation

1 Introduction and Background

Xtext[1] gains increasing popularity as a framework to developing Domain Specific Languages (DSLs)[2]. It generates not only a parser, but also the meta-model for the grammar and an Eclipse-based IDE integration. Users in specific fields can generate their own DSL efficiently.

While Model Driven Engineering (MDE)[3] techniques have been developed and applied widely, a growing number of DSLs in different grammar forms (i.e., model form and text form) was proposed and applied in many fields. Unlike traditional programming language, language tools such as Xtext and EMFText[4] are model-driven in the sense that they link text patterns with model patterns. However, many DSLs define the grammar of text and model separately, particularly using the EBNF[5] text form to describe the text grammar and managing the meta-model to construct the modeling grammar. As the result of such a fashion of separation of concerns, it is needed in practice to translate a EBNF-like style to the Xtext grammar form in order to use Xtext. For instance, Architecture Analysis & Design Language (AADL)[6] defines its grammar in text based on the EBNF pattern. If developers want to support AADL by Xtext, they need

to translate the original definition to the Xtext form grammar. Another example is that the well-known MARTE[7] profile defines the Value Specification Language (VSL), which is BNF-based. There is therefore a need to transform the MARTE BNF into a MARTE Xtext grammar.

Few resources [8] have been identified to contain discussions on generating EBNF descriptions for Xtext-based DSLs for the purpose of understanding and using the Xtext-based DSLs, since EBNF is considered more familiar to users, compared to Xtext. We did not find any resource or related work on the transition from EBNF to Xtext, except [9], where only the motivation of the transition was described.

The transition method follows certain rules depending on the EBNF grammar pattern. In this proposal, several common patterns are presented and transformed to Xtext grammar sentences. After all Xtext codes are generated, we can use Xtext to get corresponding meta-model and DSL's IDE. As part of the future work, we will automate the transition.

2 Relationship between EBNF and Xtext's Grammar

First, the notations used in EBNF can all be translated to corresponding Xtext notations, as shown in Table 1. In EBNF, the terminals are strings quoted by a pair of double quotation marks. While single quotation marks can be used in Xtext, the terminals in Xtext can also be defined by user using terminal rules.

Table 1. Notations in EBNF and Xtext

Usage	EBNF Notation	Xtext Notation
Definition	=	:
Concatenation	,	(Directly concatenate without notation)
Termination	;	;
Alternation	|	|
Option	[...]	(...)?
Repetition	{...}	(...)*
Grouping	(...)	(...)
Terminal String	"..."	'...'
Comment	(*...*)	'/*'->'*/'
Exception	-	!

In addition, Xtext has more notations. The notation '(...)+' can describe the cardinalities one or more. The operator '..' can describe character ranges such as *('0'..'9')*, which could define the common terminal *INT*.

Xtext has a special kind of rules that return the enumeration strings: Enum Rules. The same pattern can appear in EBNF, but EBNF does not offer simiar kinds of rules. Thus, when an enumeration pattern is detected in EBNF, it is transferred into Enum Rules in Xtext.

A parser rule is regarded as a tree of non-terminal and terminal tokens. In EBNF, the rules are simply described as the combination of non-terminals, terminals or both. However, Xtext has an even stronger representation that can relate to semantics of a meta-model. In Xtext, we can use the assignment notations '=' and '+=' (multi-valued feature) to assign a feature to a non-terminal object.

Though left recurrences are allowed in EBNF rules, they cannot occur in the Xtext input since Xtext leverages the ANother Tool for Language Recognition (ANTLR)[10], which implements an LL parser[11]. Nonetheless, a required transition can eliminate the left recurrence in EBNF and form reasonable Xtext rules.

In summary, for as an arbitrary EBNF rule, there exists a corresponding Xtext format description. Therefore, the transition from EBNF to Xtext is possible.

3 EBNF to Xtext Transition Processes

To implement the transition from EBNF to Xtext, certain procedures are required. As shown in Fig. 1, before we get the Xtext description structure, we have to make decisions according to the certain patterns of a EBNF rule and take some actions accordingly as highlighted (yellow boxes) in the figure. In this section, the four different actions highlighted in the yellow rectangles are discussed respectively.

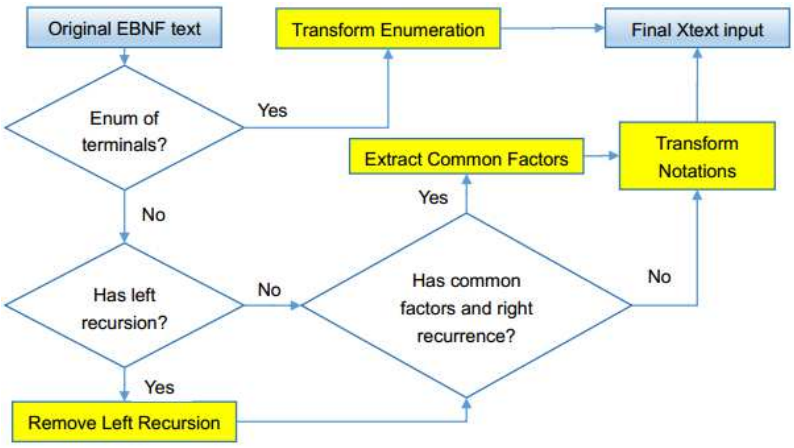

Fig. 1. Transition processes from EBNF to Xtext

Transform Enumeration
For a EBNF pattern with one terminal in every alternation:

1. Add *Enum* in the left of the defined nonterminal;
2. Assign new meta-model features to every alternations in Xtext.

As shown in Row 2 of Table 2, the nonterminal *Operation* is changed to *Enum Operation*, *ADD* and *SUB* are two features of the object *Operation*.

Remove Left Recursion
If a left recursion occurs in a EBNF rule:

1. Replace the left recursion by a new nonterminal in corresponding Xtext input;
2. Add the new defined nonterminal as a new alternation in Xtext rule;
3. Transport alternations that do not have the left recursion in EBFN rule to the new defined nonterminal.

The left recursion A in Column 1 Row 3 of Table 2 is replaced by *AtomA*.

Extract Common Factors
For a right recursive EBNF grammar pattern like A=B |BA, if B is not a simple nonterminal or terminal:

1. Extract the common factors B as a new Xtext grammar rule;
2. The original rule is changed to (*newid* += B)+, where *newid* is a multi-valued feature of object A in meta-model.

In Row 4 of Table 2, the common factors B“:”C can be extracted as a new rule.

Transform Notations
As shown in the last row of Table 2, for a general EBNF pattern without recurrence and enumeration, the transition rule is:

1. Change notations to the comparable ones in Xtext as described in the above section;
2. For every nonterminal in the right-hand of the definition, add an identifier and an assignment notation in the left of the nonterminal to assign a feature to the currently produced object of corresponding meta-model.

Table 2. Transition actions from EBNF to Xtext

EBNF Format	Xtext Format
$Operation =$ “add” | “sub”;	$enum\ Operation:\ ADD =$ ‘add' | $SUB =$ ‘sub';
$A =\ A$“+”“a” | “b”	$A:\ AtomA \mid AtomA$ ‘+’ ‘a';
	$AtomA:$ ‘b';
$A = B$“:”$C \mid B$“:”$C\ A$;	$A:\ (multiValue+ = CommonA)+$;
	$CommonA:\ b = B$‘:’ $c = C$;
$A =$ [“b”]“&&”C;	$A:$ (‘b')? ‘&&’ $c = C$;

4 Case Study: A Complete Transition Instance

This section illustrates the complete transition from EBNF rules to the corresponding Xtext rules, as shown in Fig. 2, of a declarative query language named JIns[12].

EBNF rules

```
S = "select" RS "from" SP ["for" "all" DS] "where" CE
RS = Id":"T|Id":"T RS
DS = RS "satisfying" CE
T = "interface"|"class"|"method"|"object"|"statement"
SP = "repository" STRING
CE = CE"&&"CE|CE"||"CE|"("CE")"|"!"CE|ID"."Att"="STRING
|ID"."Att"="ID"."Att|ID Rel ID;
Att = "name"|"modifier"|"type"|"returnType"|"paramsType"
Rel = "extends"|"implements"|"in"|"call"|"use"
```

```
S: 'select'targetIds=RS'from'sp=SP('for''all'ds=DS)?'where'ce=CE;
RS: (ids+=Identifier)+;
Identifier: name=ID ':' type=T;
enum T: INTERFACE='interface'|CLASS='class'|METHOD='method'
        |OBJECT='object'|STATEMENT='statement';
SP: 'repository' (url=STRING);
DS: allRs=RS 'satisfying' allCe=CE;
CE:AtomCE|AtomCE'&&'sub=CE|AtomCE'||'sub=CE|'('sub=CE')'|'!'sub=CE;
AtomCE: leftId=ID '.' leftAtt=Att '=' cond=STRING
        |leftId=ID '.' leftAtt=Att '=' rightId=ID '.' rightAtt=Att
        |leftId=ID rel=Rel rightId=ID;
enum Att: NAME='name'|MODIFIER='modifier'|TYPE='type'
        |RETURNTYPE='returnType'|PARAMSTYPE='paramsType';
enum Rel: EXTENDS='extends'|IMPLEMENT='implement'|IN='in'|CALL='call'|USE='use';
```

Fig. 2. The result of a transition from EBNF to XText.

5 Conclusion

To better benefit from Xtext, in some contexts, it is needed to transform an Extend Backus Naur Form (EBNF) based language to the Xtext grammar. Therefore, in this paper, we proposed four processes of transition from EBNF to Xtext. As a future work, we will develop a tool to support the automated transition and conduct case studies to evaluate the proposed approach.

Acknowledgement This work is supported by the National Natural Science Foundation of China (No.61472180), and by the Jiangsu Province Research Foundation(BK20141322).

References

1. Xtext, http://www.eclipse.org/Xtext/
2. Fowler, M.: Domain Specific Language. Addison-Wesley Professional, 2010.
3. Schmidt, D. C. Guest Editor's Introduction: Model-Driven Engineering. IEEE Computer 39: 25-31, 2006.
4. EMFText, http://www.emftext.org/index.php/EMFText
5. Extended Backus-Naur Form, http://en.wikipedia.org/wiki/Extended_Backus-Naur_Form
6. SAE Aerospace. SAE AS5506A: Architecture Analysis and Design Language (AADL). Version 2.0, 2009.
7. The UML Profile for MARTE: Modeling and Analysis of Real-Time and Embedded Systems, http://www.omgmarte.org/
8. Generate EBNF-Descriptions for Xtext-based DSL, http://xtexterience.wordpress.com/2011/05/13/an-ebnf-grammar-in-xtext/
9. Xtext, BNF, Marte and MSCs, http://5ise.quanxinquanyi.de/2010/03/12/xtext-bnf-marte-and-mscs/
10. ANTLR, http://en.wikipedia.org/wiki/ANTLR
11. LL parser, http://en.wikipedia.org/wiki/LL_parser
12. Zhang, T., Zheng, X., Zhang, Y., Zhao, J. and Li, X. A declarative approach for Java code instrumentation. Software Quality Journal: 1-28, 2013.