

Die Entwicklung eines Parsers für eine domänenspezifische Sprache

Studienarbeit
WS 08/09

Friedrich-Schiller-Universität Jena
Fakultät für Mathematik und Informatik



betreut von:

Dipl.-Inf. Christian Schachtzabel
Dipl.-Inf. Christoph Henniger

vorgelegt von:

Steffen Gemein (87896)
steffen.gemein@uni-jena.de

Jena, den 14. März 2009

Inhaltsverzeichnis

1	Einleitung	4
2	Grundkonzept	5
3	Die Jenaer Entity Language	6
4	Das Metamodell	7
4.1	JelModel	7
4.2	JelDatatype	8
4.3	JelUserDatatype	9
4.4	JelEnum	10
4.5	JelLiteral	11
4.6	JelDbType	11
4.7	JelPackage	12
4.8	JelEntity	13
4.9	JelInheritanceStrategy	15
4.10	JelStructuralFeature	16
4.11	JelAttribute	17
4.12	JelReference	17
4.13	JelAssociation	19
4.14	JelUniqueGroup	19
4.15	JelHistoryGroup	20
4.16	JelHistory	20
4.17	JelHistoryType	21
5	Der Parser	22
5.1	Die Funktionsweise eines Parsers	22
5.2	Grammatik und Regeln	22
5.3	Regeln mit Actions	23
5.4	Vom Parser-Baum zum Syntaxbaum	27
5.5	Semantikprüfung und Erstellen des Metamodells	30
6	Der visuelle Editor	35
7	Anhang	36
7.1	Screenshots entity-parser	36
7.2	Screenshots entity-editor	38
7.3	Klassendiagramme	40
7.4	Syntaxdiagramme für die Jenaer Entity Language	43
7.5	ANTLR-Grammatik für die Jenaer Entity Language	48

Abbildungsverzeichnis

1	Parser und Metamodell gliedern sich im Gesamtkonzept ein	5
2	Parser-Baum einer einfachen Package-Definition	27
3	Syntaxbaum einer einfachen Package-Definition	28
4	Syntaxbaum für das Model der Entity-Sprache	29
5	Syntaxbaum für eine Entity der Entity-Sprache	29
6	Syntaxbaum für einen Aufzähltypen der Entity-Sprache	30
7	Eingabedialog des Parsers	36
8	Ausgabedialog des Parsers	37
9	Quelltext-Ansicht des Editors	38
10	Design-Ansicht des Editors	39
11	statische Sicht des Metamodells	40
12	statische Sicht des Parsers	41
13	Beispielserweiterung des Modells durch implementierte Fabrik	42

Tabellenverzeichnis

1	Vordefinierte Standarddatentypen	8
2	EBNF Grammatik-Subregeln	24
3	Klassen des Projekts	31
4	Dateien die von ANTLR generiert werden	31

Quelltextverzeichnis

1	Regel für Packages in der Entity-Sprache	23
2	Bezeichnerdefinition in der Entity-Sprache	23
3	Token für Bezeichner in der Entity-Sprache	25
4	Eine Datei bindet eine andere Datei ein	25
5	Die Datei wird von anderen Dateien eingebunden	25
6	Token für Include in der Entity-Sprache	25
7	Kommentare in der Entity-Sprache	26
8	Token für Kommentare in der Entity-Sprache	27
9	Einfache Package-Definition	27
10	Package-Regel mit Überschreibregeln für den Syntaxbaum	28
11	Param-Regeln der ANTLR-Grammatik	29
12	Einbinden der generierten ANTLR-Klassen	32
13	HashMaps um 2. Baumdurchlauf zu umgehen	32
14	Vererbung der Entities herstellen	33
15	Einfaches Benutzen des Parsers	33
16	Erweiterung des Metamodells durch abstrakte Fabrik	33
17	EntityLang.g	48

1 Einleitung

Allein mit den Mitteln einer allgemeinen Programmiersprache lassen sich nicht immer passende Abstraktionen zur Beschreibung verschiedener Sachverhalte finden. In diesen Fällen greift man auf domänenspezifische Sprachen (engl. domain specific language, kurz DSL) zurück. Eine domänenspezifische Sprache ist eine zielplattformunabhängige, formale Sprache, die für ein bestimmtes Problemfeld entworfen wurde und dort Anwendung findet. Mit dieser Sprache lassen sich Probleme eines bestimmten Anwendungsbereiches (Domäne) detailliert und zielgerichtet beschreiben. DSLs haben gegenüber allgemeinen Programmiersprachen den Vorteil, dass der Sprachumfang für ihre Domäne angepasst ist. Das bedeutet, dass mit einer domänenspezifischen Sprache nicht mehr und nicht weniger als das Problemfeld beschrieben werden kann.

Die DSL wird nach der Modellierungsphase generativ oder interpretativ auf eine Zielplattform abgebildet. Dazu ist es notwendig, den Quelltext im domänenspezifischen Format zu parsen (engl. für analysieren), um die Semantik der Eingabe zu erschließen.

In diesem Dokument wird beschrieben, wie ein Parser für eine domänenspezifische Sprache - die Jenaer Entity Language - entwickelt wird. Das Dokument ist in mehrere Kapitel untergliedert, welche unabhängig voneinander gelesen werden können. Sollte es doch notwendig sein, an bestimmten Textabschnitten einen anderen Abschnitt zu lesen, so befinden sich Verweise an den entsprechenden Stellen.

Das Kapitel 2 gibt einen Überblick über dieses Projekt und dessen Verwendungsmöglichkeiten. Es bietet sich an, dieses Kapitel als Einführung zu lesen.

Im Kapitel 3 wird die Jenaer Entity Language kurz vorgestellt und erläutert. Es werden die Möglichkeiten und Grenzen der Sprache aufgezeigt.

Für Anwender des Metamodells und der Entity-Sprache ist Kapitel 4 interessant. Hier werden die Java-Klassen, welche die Semantik des Entity-Modells halten, referenzartig erläutert. Ebenfalls hier zu finden, sind einzelne Codebeispiele, um die Syntax der Jenaer Entity Language zu klären.

Entwickler der Entity-Sprache oder des Code-Generators werden sich für das Kapitel 5 interessieren. Hier ist beschrieben, wie die Eingabe analysiert und in die Klassen des Metamodells transferiert wird. Das Kapitel beschreibt, wie der Parser funktioniert und erweitert werden kann.

Das abschliessende Kapitel 6 gibt einen Ausblick auf zukünftige Einsatzmöglichkeiten des Parsers.

7 Anhang

7.1 Screenshots entity-parser

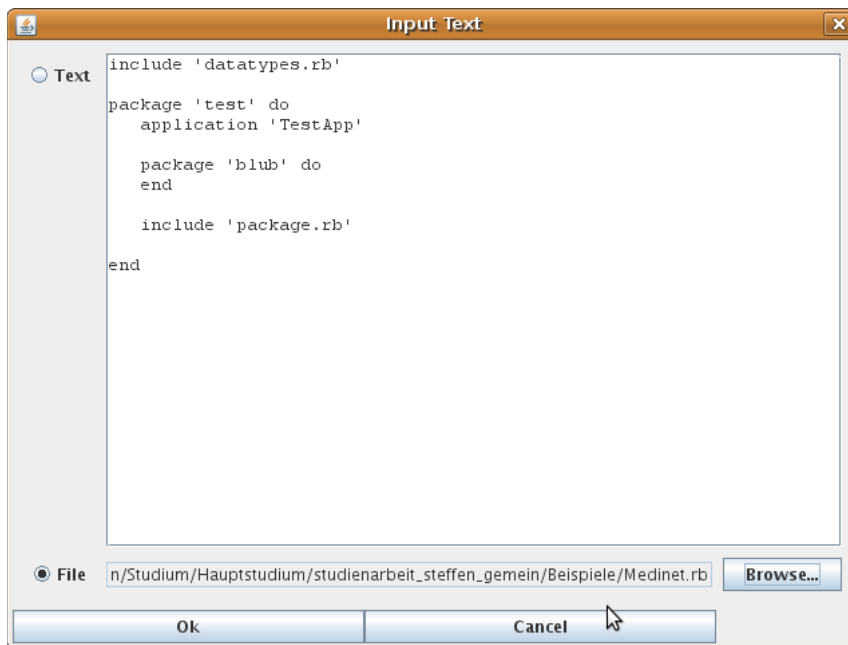


Abbildung 7: Eingabedialog des Parsers

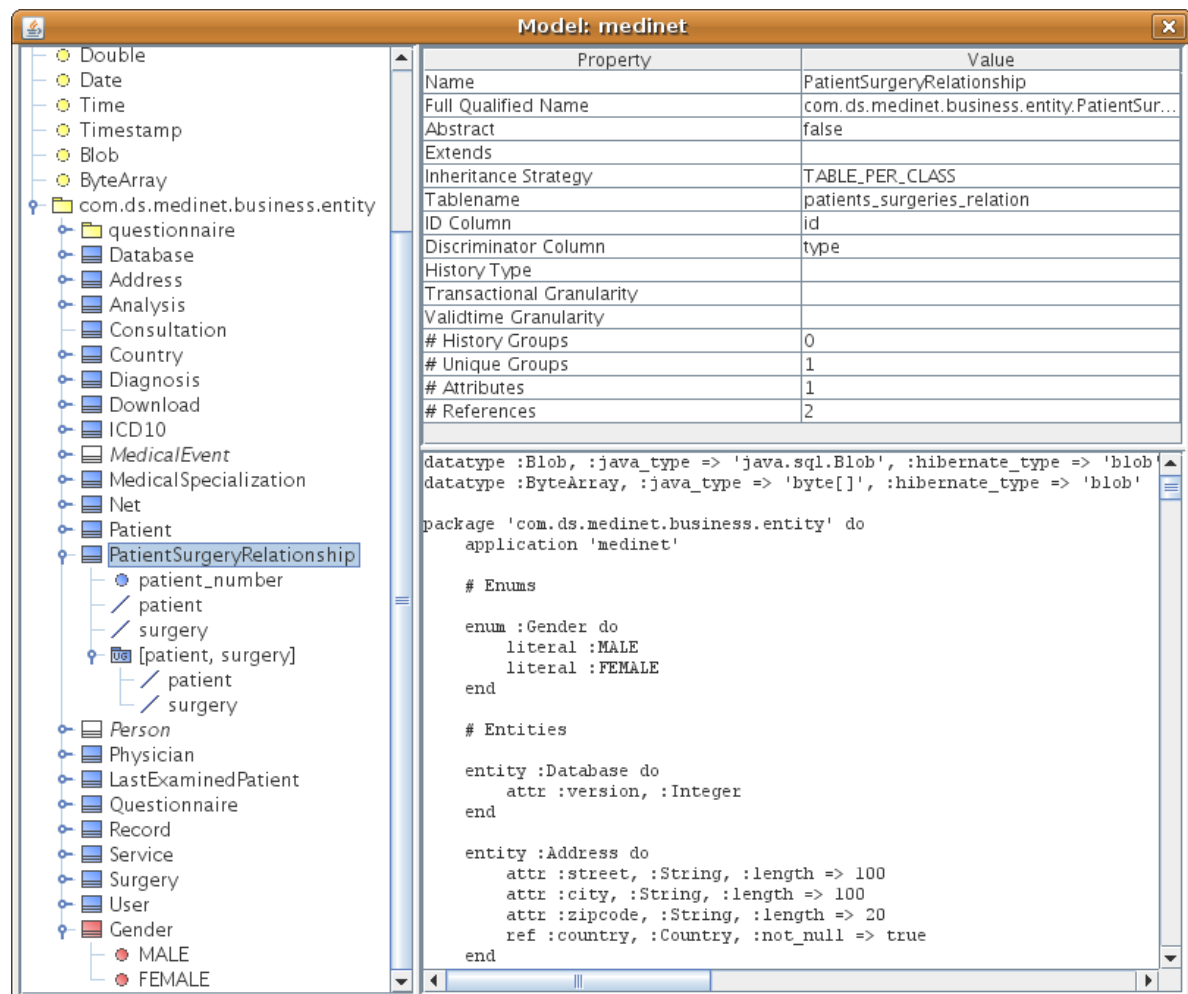


Abbildung 8: Ausgabedialog des Parsers

7.2 Screenshots entity-editor

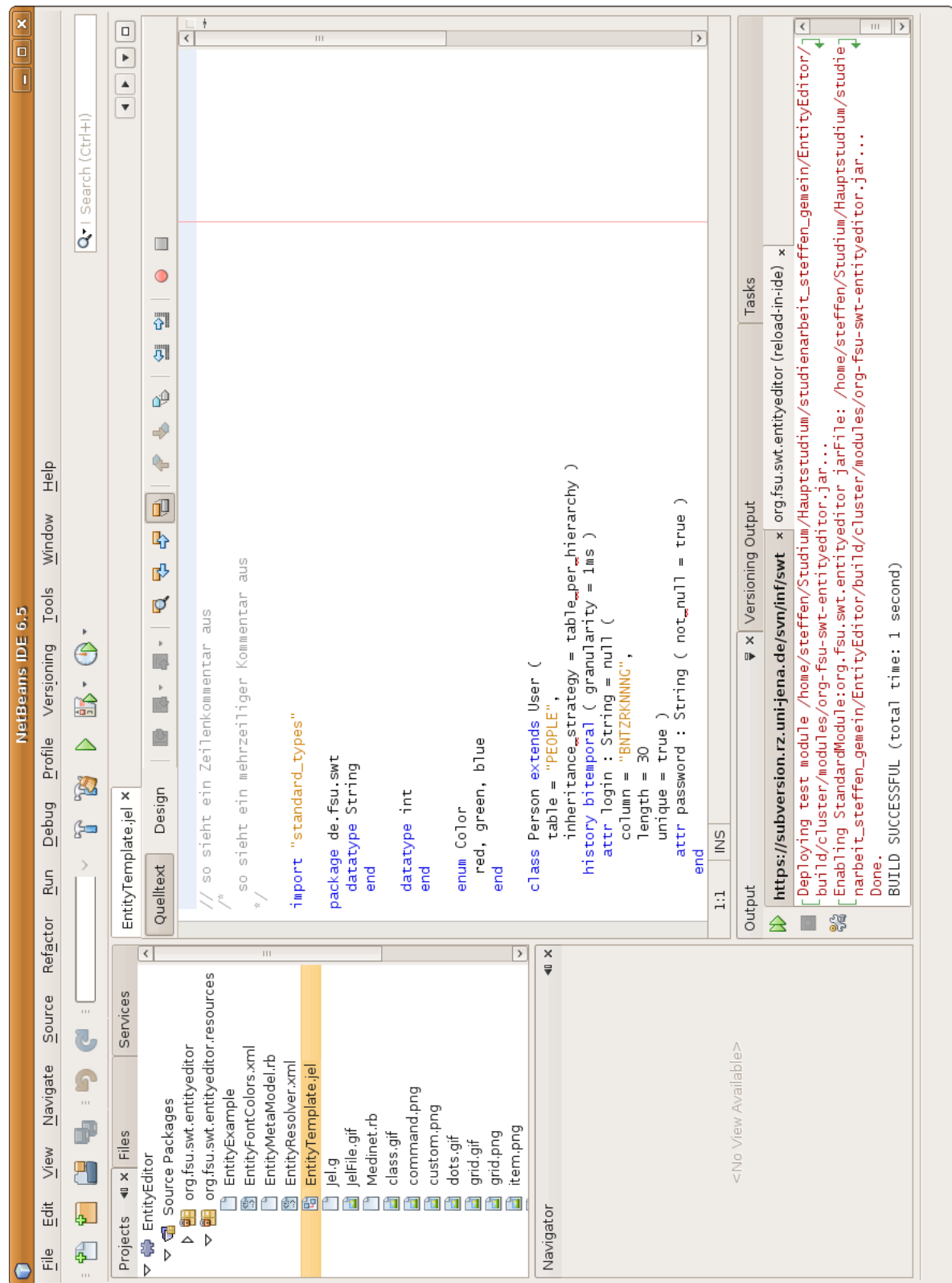


Abbildung 9: Quelltext-Ansicht des Editors

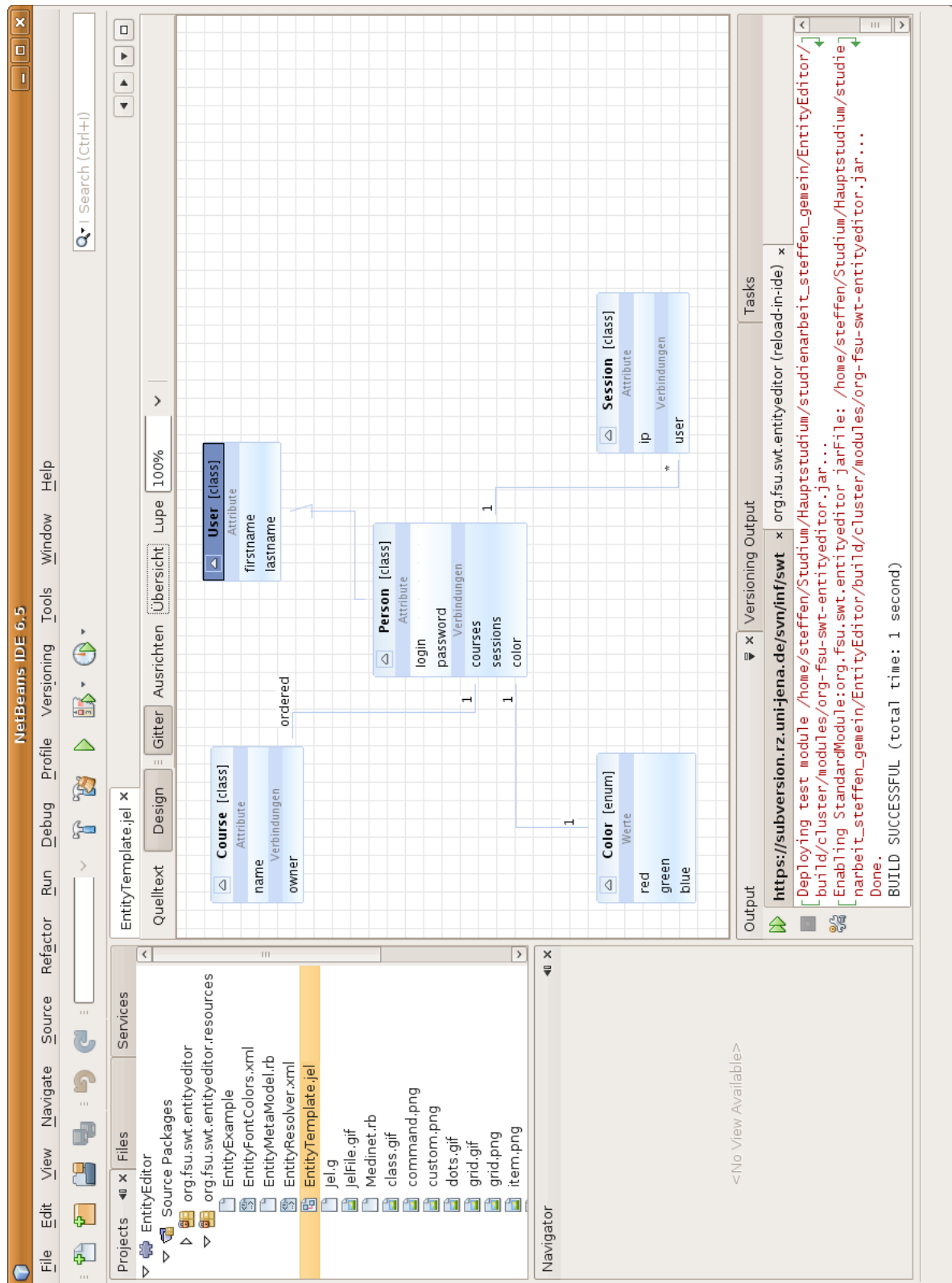


Abbildung 10: Design-Ansicht des Editors

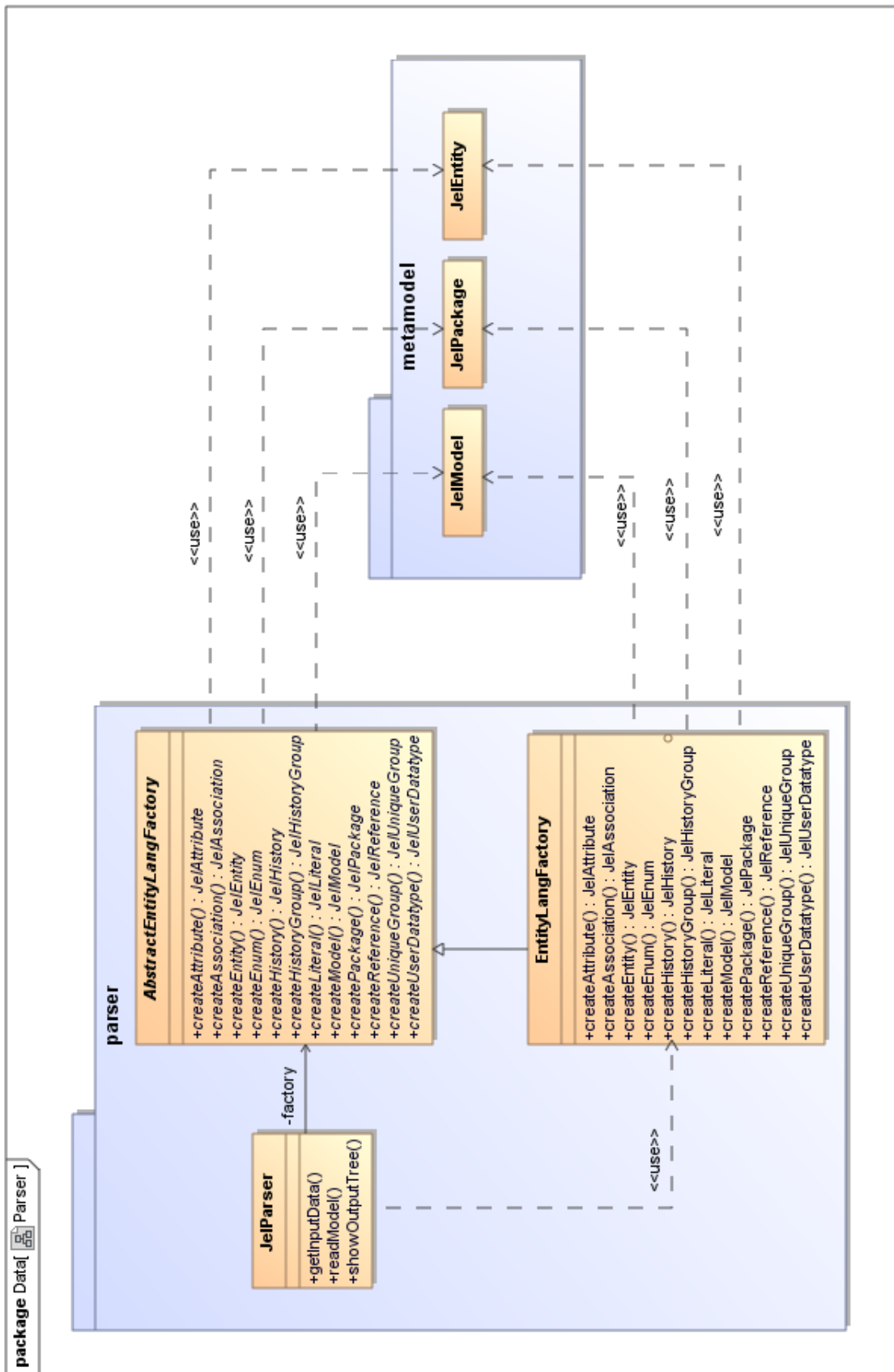


Abbildung 12: statische Sicht des Parsers

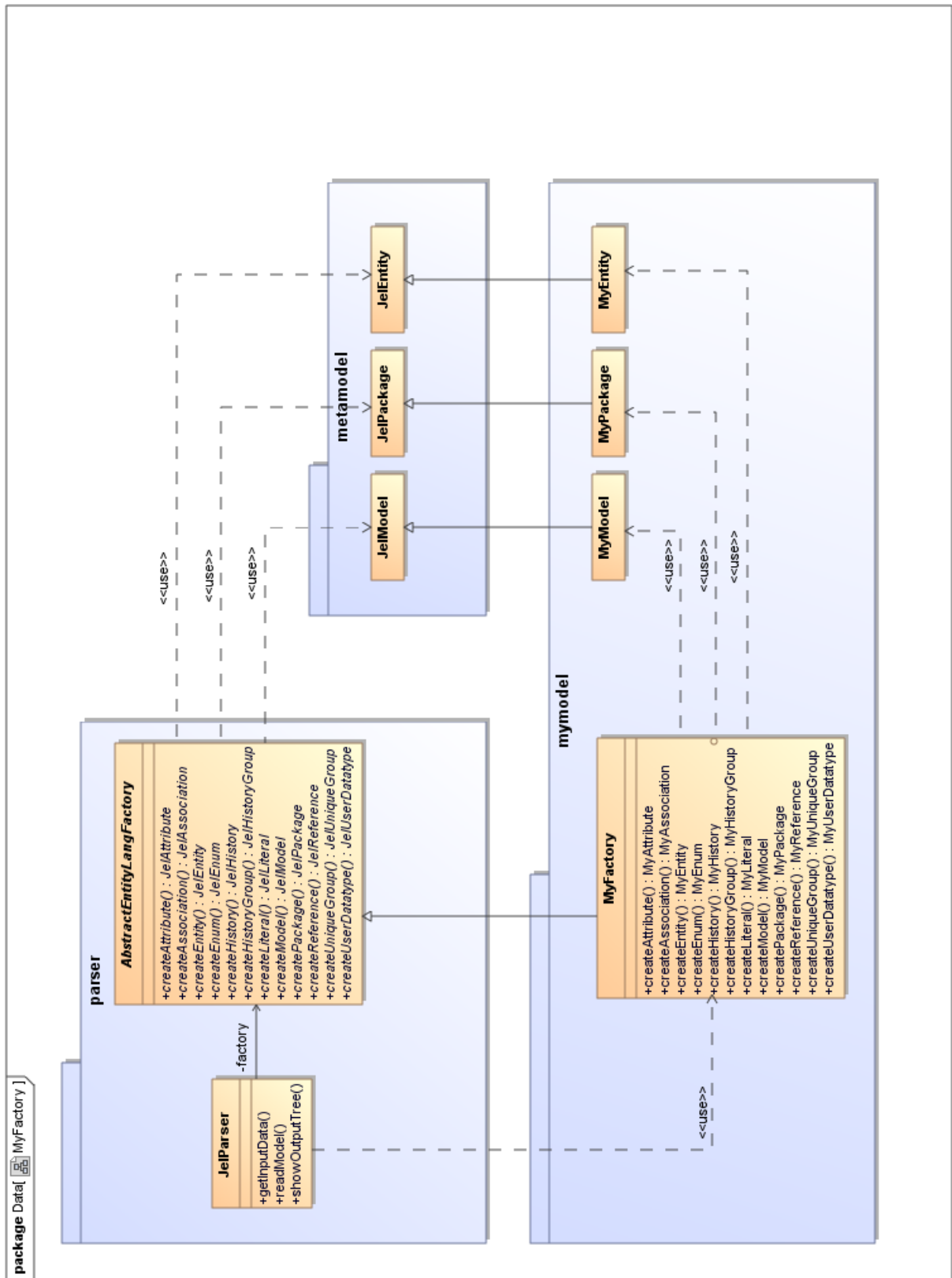
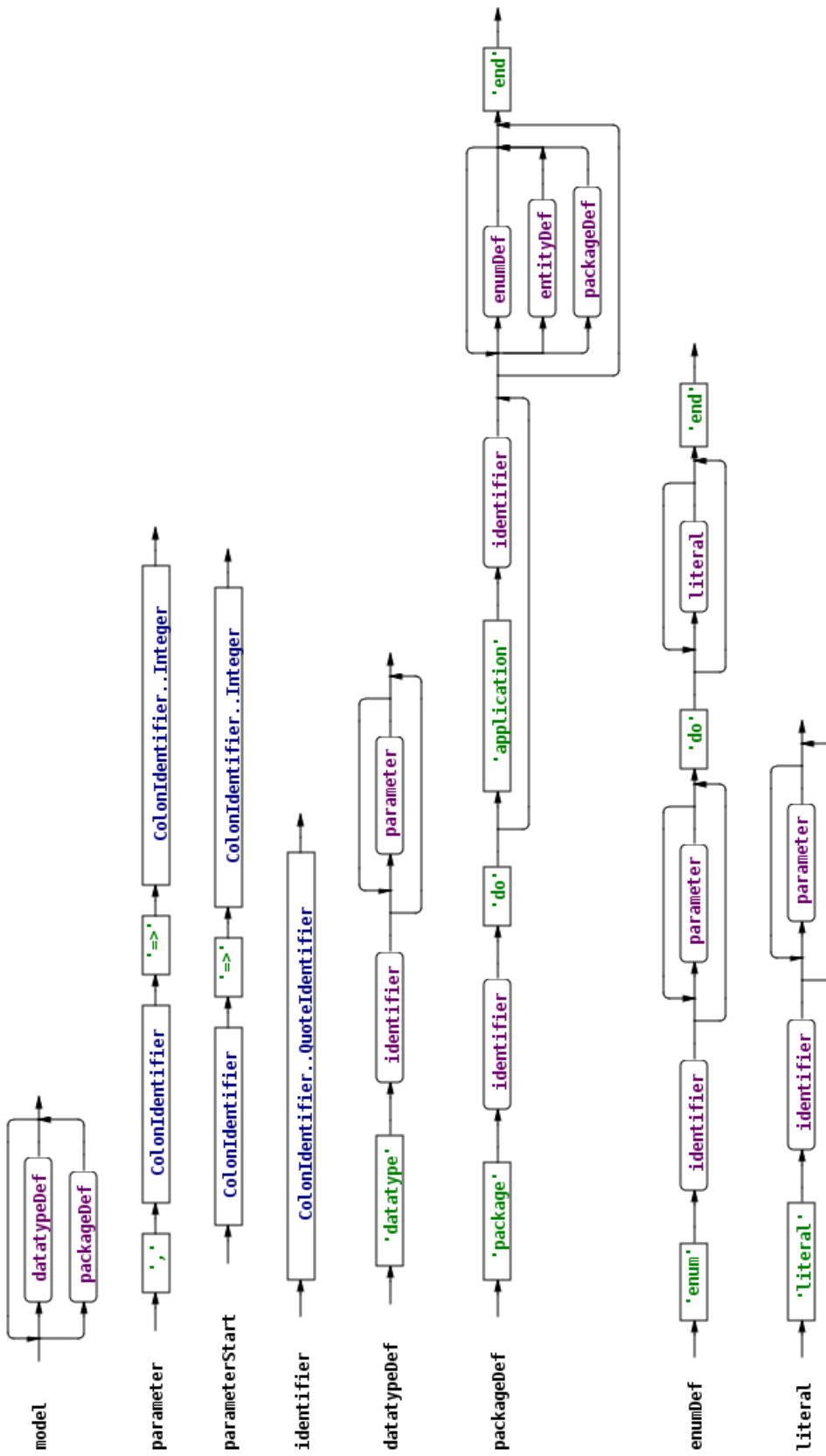
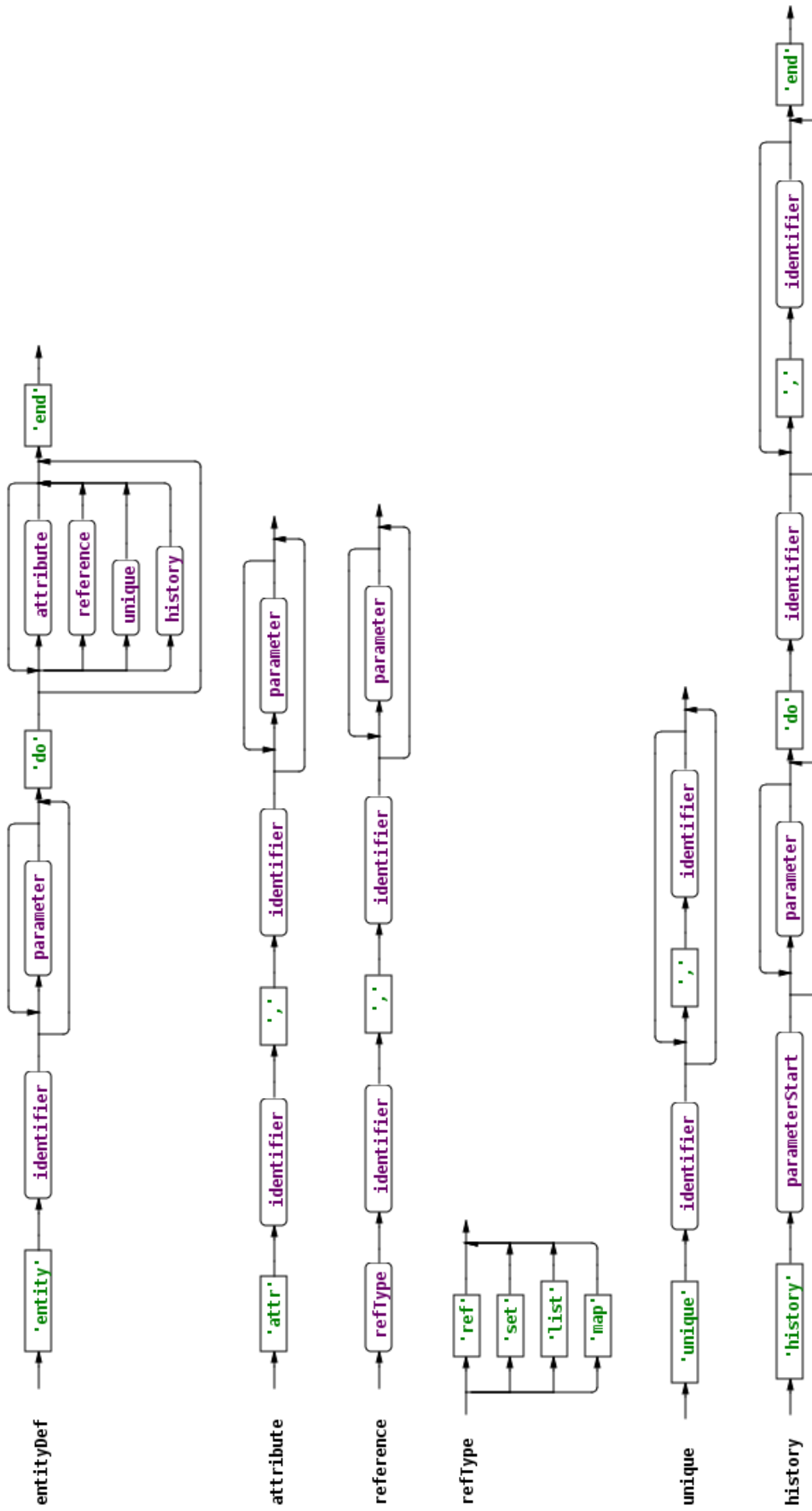
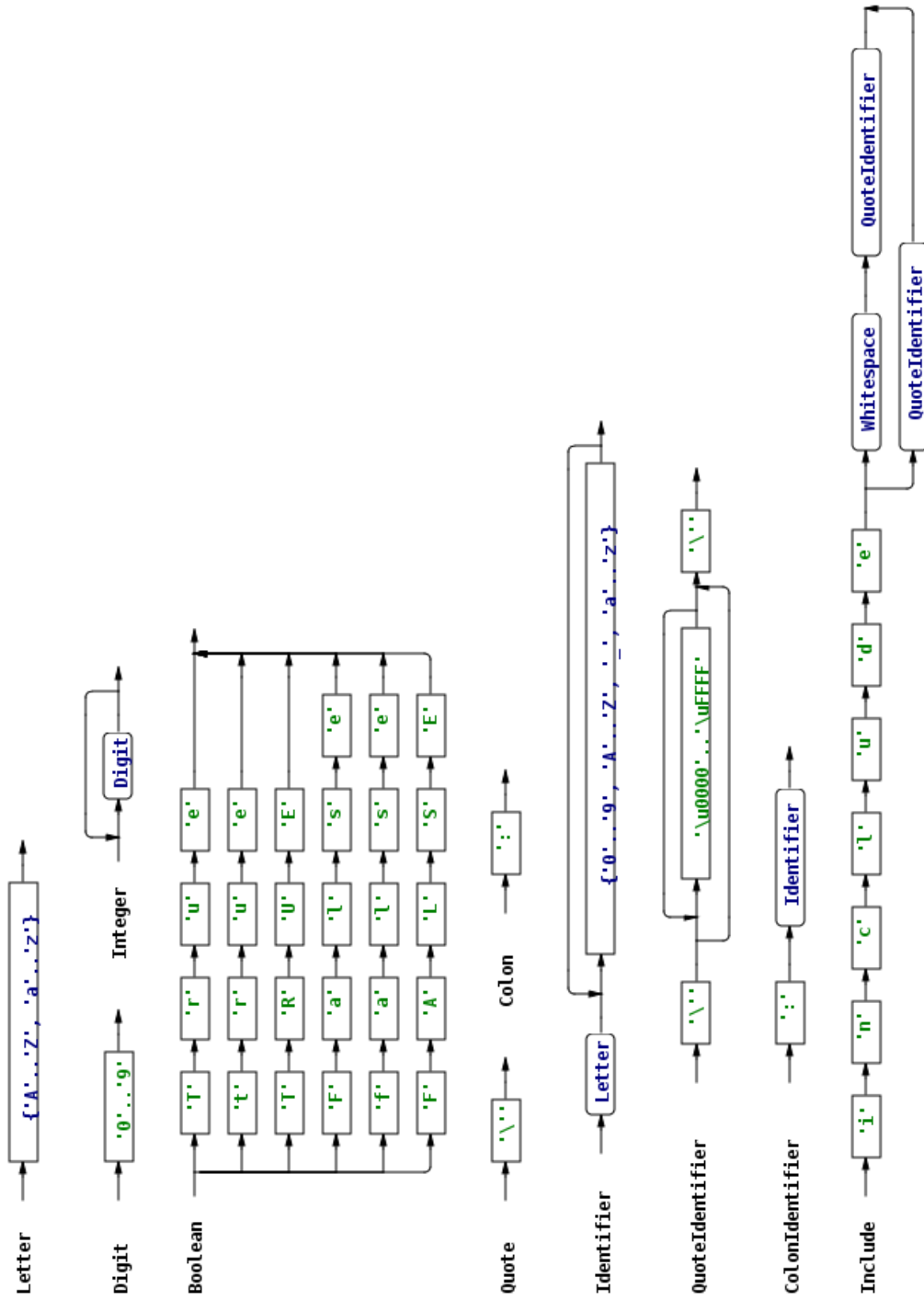


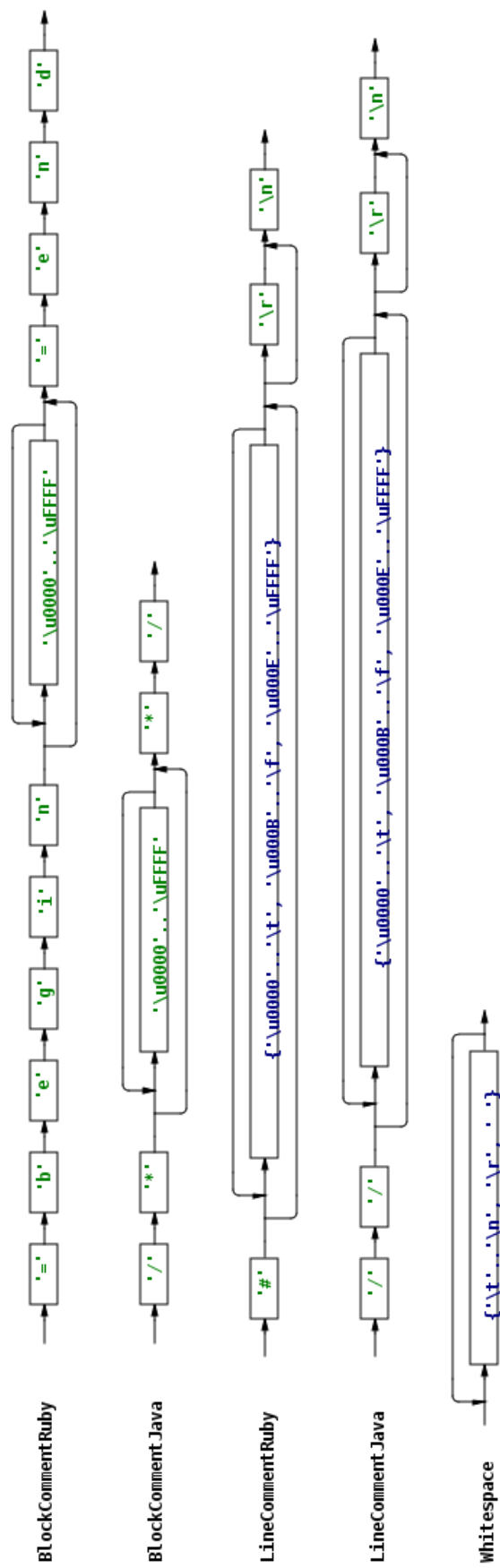
Abbildung 13: Beispielerweiterung des Modells durch implementierte Fabrik

7.4 Syntaxdiagramme für die Jenaer Entity Language









7.5 ANTLR-Grammatik für die Jenaer Entity Language

```
1  /*
2  * Grammar Definition for ANTLR v.3
3  * JenaerEntityLanguage Version 2.0
4  * Author: Steffen Gemein
5  * Last Modified: 16.01.2009
6  */
7
8  grammar EntityLang;
9
10 options {
11     output=AST;
12     ASTLabelType=CommonTree;
13 }
14
15 // virtual nodes
16 tokens {
17     NAME;
18     PARAM;
19     DATATYPE;
20     PACKAGE;
21     APPLICATION;
22     ENUM;
23     LITERAL;
24     ENTITY;
25     ATTR;
26     REF;
27     SET;
28     LIST;
29     MAP;
30     TARGET;
31     UNIQUE;
32     HISTORY;
33 }
34
35 // package declarations for generated Java files
36 @parser::header {
37     package org.fsu.swt.entitylang.parser;
38 }
39 @lexer::header {
40     package org.fsu.swt.entitylang.parser;
41 }
42
43 @lexer::members {
44     class SaveStruct {
45         SaveStruct(CharStream input){
46             this.input = input;
47             this.marker = input.mark();
48         }
49         public CharStream input;
50         public int marker;
51     }
52
53     Stack<SaveStruct> includes = new Stack<SaveStruct>();
54
55     public Token nextToken() {
56         Token token = super.nextToken();
57
58         if(token==Token.EOF_TOKEN && !includes.empty()){
59             SaveStruct ss = includes.pop();
60             setCharStream(ss.input);
61             input.rewind(ss.marker);
62             token = this.nextToken();
63         }
64
65         if(((CommonToken)token).getStartIndex() < 0)
66             token = this.nextToken();
67
68         return token;
69     }
70 }
```

```

69     }
70 }
71
72 model
73     : (datatypeDef | packageDef)+
74     ;
75
76 parameter
77     : ','! ColonIdentifier '=>'! (QuoteIdentifier | ColonIdentifier | Boolean | Integer)
78     ;
79
80 parameterStart
81     : ColonIdentifier '=>'! (QuoteIdentifier | ColonIdentifier | Boolean | Integer)
82     ;
83
84 identifier
85     : QuoteIdentifier
86     | ColonIdentifier
87     ;
88
89 // Datatype
90
91 datatypeDef
92     : 'datatype' identifier parameter*
93     -> ^(DATATYPE ^(NAME identifier)
94         ^(PARAM parameter)* )
95     ;
96
97 // Package
98
99 packageDef
100    : 'package' identifier 'do'
101    ('application' identifier)?
102    (packageDef | entityDef | enumDef)*
103    'end'
104    -> ^(PACKAGE ^(NAME identifier)
105        ^(APPLICATION identifier)?
106        packageDef* entityDef* enumDef*)
107    ;
108
109 // Enum
110
111 enumDef
112     : 'enum' identifier parameter* 'do'
113     literal*
114     'end'
115     -> ^(ENUM ^(NAME identifier)
116         ^(PARAM parameter)*
117         ^(LITERAL literal)* )
118     ;
119
120
121 literal
122     : 'literal' identifier parameter*
123     -> ^(NAME identifier)
124     ^(PARAM parameter)*
125     ;
126
127 // Entity
128
129 entityDef
130     : 'entity' identifier parameter* 'do'
131     (attribute | reference | unique | history)*
132     'end'
133     -> ^(ENTITY ^(NAME identifier)
134         ^(PARAM parameter)*
135         ^(ATTR attribute)*
136         reference*
137         ^(UNIQUE unique)*
138         ^(HISTORY history)* )

```

```

139 | ;
140 |
141 | attribute
142 | : 'attr' identifier ',' identifier parameter*
143 |   -> ^(NAME identifier)
144 |       ^(DATATYPE identifier)
145 |       ^(PARAM parameter)*
146 | ;
147 |
148 | reference
149 | : refType identifier ',' identifier parameter*
150 |   -> ^(refType ^(NAME identifier)
151 |       ^(TARGET identifier)
152 |       ^(PARAM parameter)* )
153 | ;
154 |
155 | refType
156 | : ('ref ') -> ^(REF)
157 | | ('set ') -> ^(SET)
158 | | ('list ') -> ^(LIST)
159 | | ('map ') -> ^(MAP)
160 | ;
161 |
162 | unique
163 | : 'unique' identifier (',' identifier)*
164 |   -> ^(identifier)*
165 | ;
166 |
167 | history
168 | : 'history' parameterStart parameter* 'do'
169 |   identifier (',' identifier)*
170 |   'end'
171 |   -> ^(PARAM parameterStart)
172 |       ^(PARAM parameter)*
173 |       ^(ATTR identifier)*
174 | ;
175 |
176 | // Identifier
177 |
178 | fragment Quote
179 | : '\''
180 | ;
181 |
182 | fragment Colon
183 | : ':'
184 | ;
185 |
186 | fragment Letter
187 | : ('a'..'z'|'A'..'Z')
188 | ;
189 |
190 | fragment Digit
191 | : '0'..'9'
192 | ;
193 |
194 | Boolean
195 | : 'True' | 'true' | 'TRUE'
196 | | 'False' | 'false' | 'FALSE'
197 | ;
198 |
199 | Integer
200 | : Digit+
201 | ;
202 |
203 | Identifier
204 | : Letter (Letter | Digit | '_' )+
205 | ;
206 |
207 | QuoteIdentifier
208 | : '\'' (options {greedy=false;} : . )* '\''

```

```
209     {setText(getText().substring(1, getText().length()-1));}
210     ;
211
212 ColonIdentifier
213     : ':' Identifier
214     {setText(getText().substring(1, getText().length()));}
215     ;
216
217 // Include
218
219 Include
220     : 'include' (Whitespace)? file=QuoteIdentifier {
221         String name = file.getText();
222         name = name.substring(1,name.length()-1);
223         try {
224             SaveStruct ss = new SaveStruct(input);
225             includes.push(ss);
226             setCharStream(new ANTLRFileStream(name));
227             reset();
228
229             } catch(Exception ex) { throw new Error("Cannot open file " + name); }
230
231     }
232     ;
233
234 // Comment
235
236 BlockCommentRuby
237     : '=begin' ( options {greedy=false;} : . )* '=end' {$channel=HIDDEN;}
238     ;
239
240 BlockCommentJava
241     : '/*' ( options {greedy=false;} : . )* '*/' {$channel=HIDDEN;}
242     ;
243
244 LineCommentRuby
245     : '#' ~('\n'|\r)* '\r'? '\n' {$channel=HIDDEN;}
246     ;
247
248 LineCommentJava
249     : '/' ~('\n'|\r)* '\r'? '\n' {$channel=HIDDEN;}
250     ;
251
252 Whitespace
253     : (' |\t|\n|\r')+ {$channel=HIDDEN;}
254     ;
```

Listing 17: EntityLang.g

Literatur

- [1] Böck, Heiko: NetBeans Platform 6. Rich-Client-Entwicklung mit Java. Galileo Press. Bonn 2008.
- [2] Boudreau, Tim; Tulach, Jaroslav; Wielenga, Geertjan: RichClient Programming. Plugging into the NetBeans IDE. Pearson Education, Inc. Massachusetts 2007.
- [3] Casey, John u.a.: Better Builds with Maven. The How-to Guide for Maven 2.0. Version 1.7.0. Exist Global, 2008.
- [4] Parr, Terence: The Definitive ANTLR Reference Guide. Building Domain-Specific Languages. 1. Auflage. Pragmatic Programmers, 2007.