The Language Evolver -

A Tool for the Evolutionary Development of Domain Specific Languages

Elmar Jürgens
Markus Pizka

{juergens,pizka}@in.tum.de

Outline

- Motivation
- Language Evolver Concepts
- Demonstration (Powerpoint)

Domain Specific Languages (DSLs)

- Problem specific notations and abstractions
- More precise / concise than general purpose languages ⇨ Increased Development Productivity

DSL-Evolution:

- Many domains undergo continuous change
- Better domain understanding leads to new abstractions and notations

Change to domain / perception ⇨ Change to DSL

Change Impact

- Existing Words must be migrated
- Processing Tools (Compiler) must be adapted

Language Evolver (Lever)

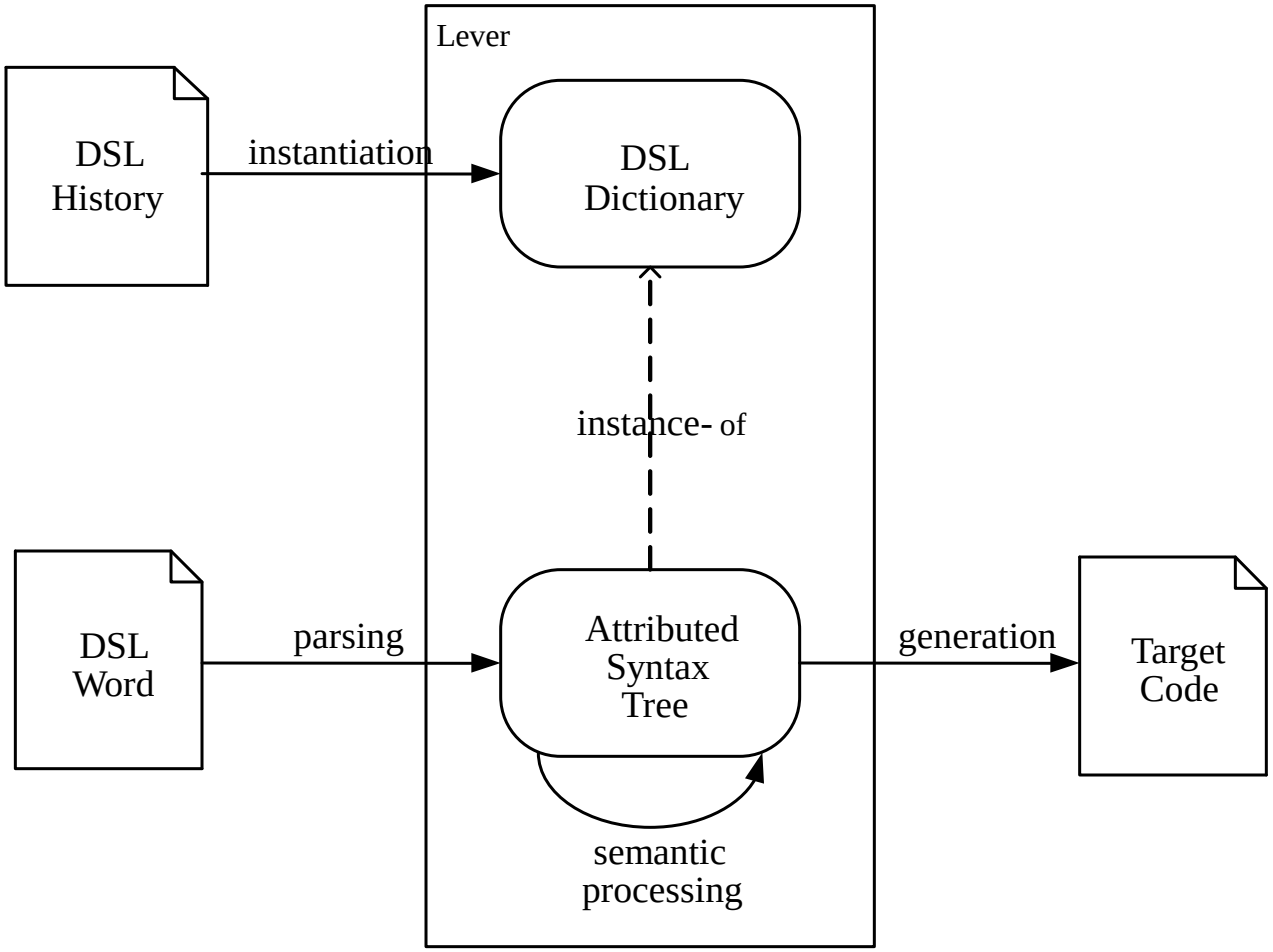
Approach

- Evolution operations on DSL Specifications
- Automation of
  - Adaptation of DSL Compiler
  - Migration of existing Words

DSL History

- Evolution operations
- Specification of the first DSL version
- Changes between consecutive versions

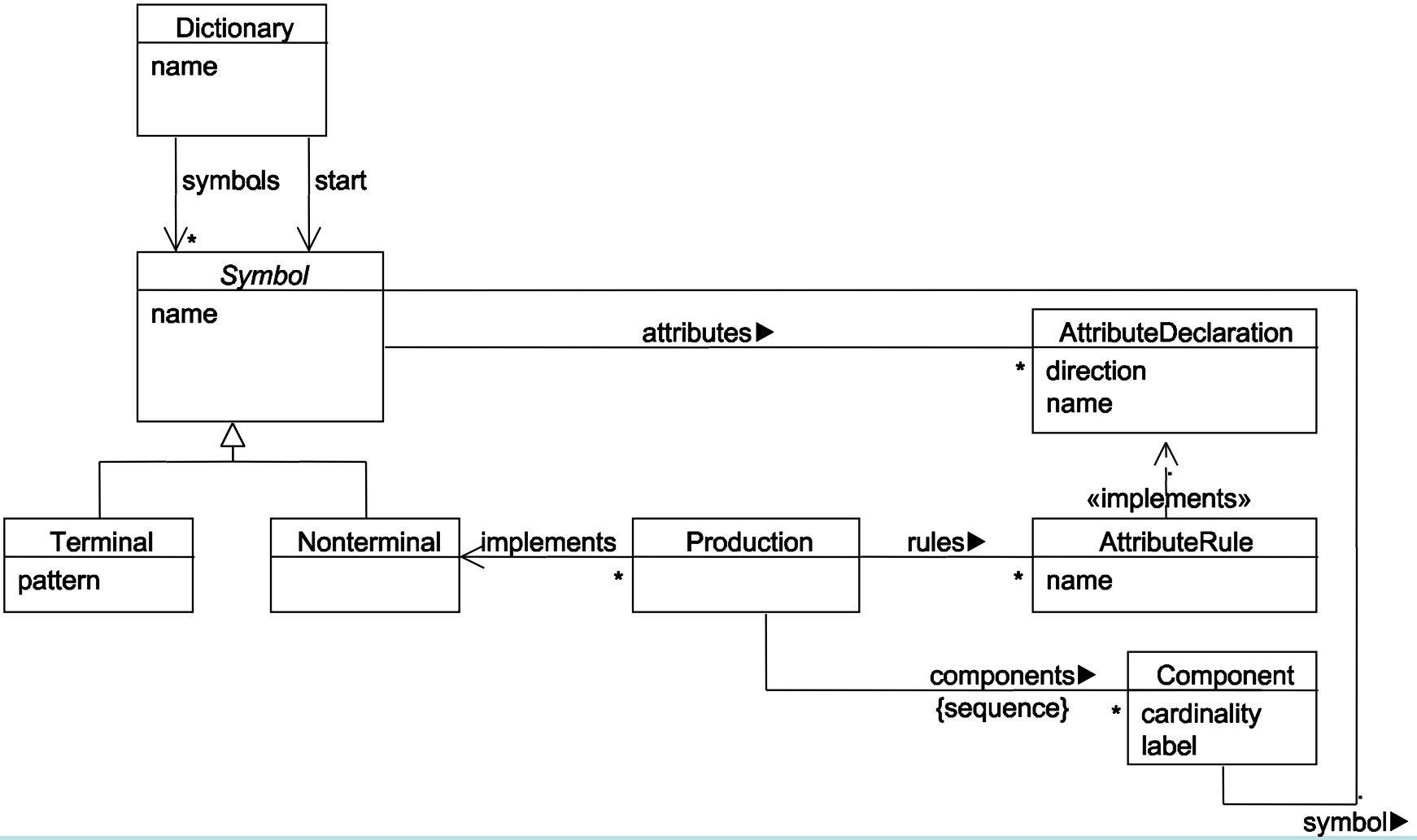
Language Evolver - Overview



DSL Dictionary

- Complete DSL Specification
  - concrete Syntax
  - abstract Syntax
  - static Semantics
  - translational Semantics
- Modularized along language elements
- Explicit, mutable
- Object-oriented Attribute Grammars

DSL Dictionary Metamodel



Semantic Rules

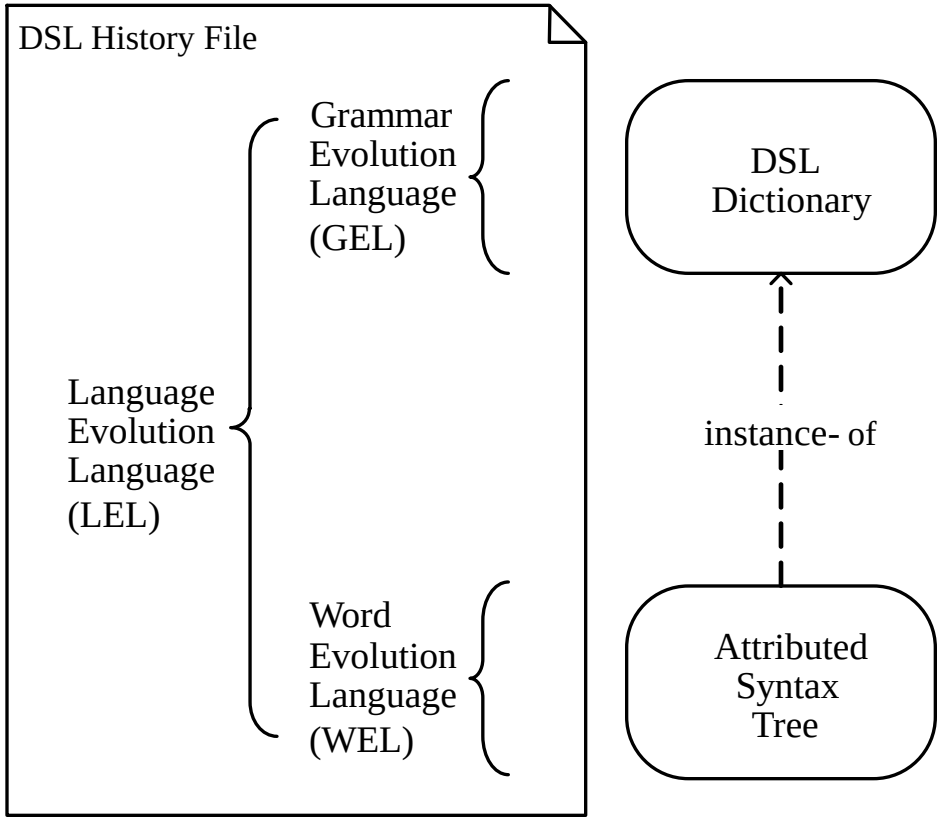
Implementation Languages

- Jython for static Semantics
- Velocity Templates for translational Semantics

XPath expressions for access to syntax tree

- local and remote tree nodes
- textual and attribute values
- Can be validated statically

DSL History



GEL

- DSL Dictionary manipulation
- Complete

LEL = GEL ◦ WEL

- Coupled evolution operations

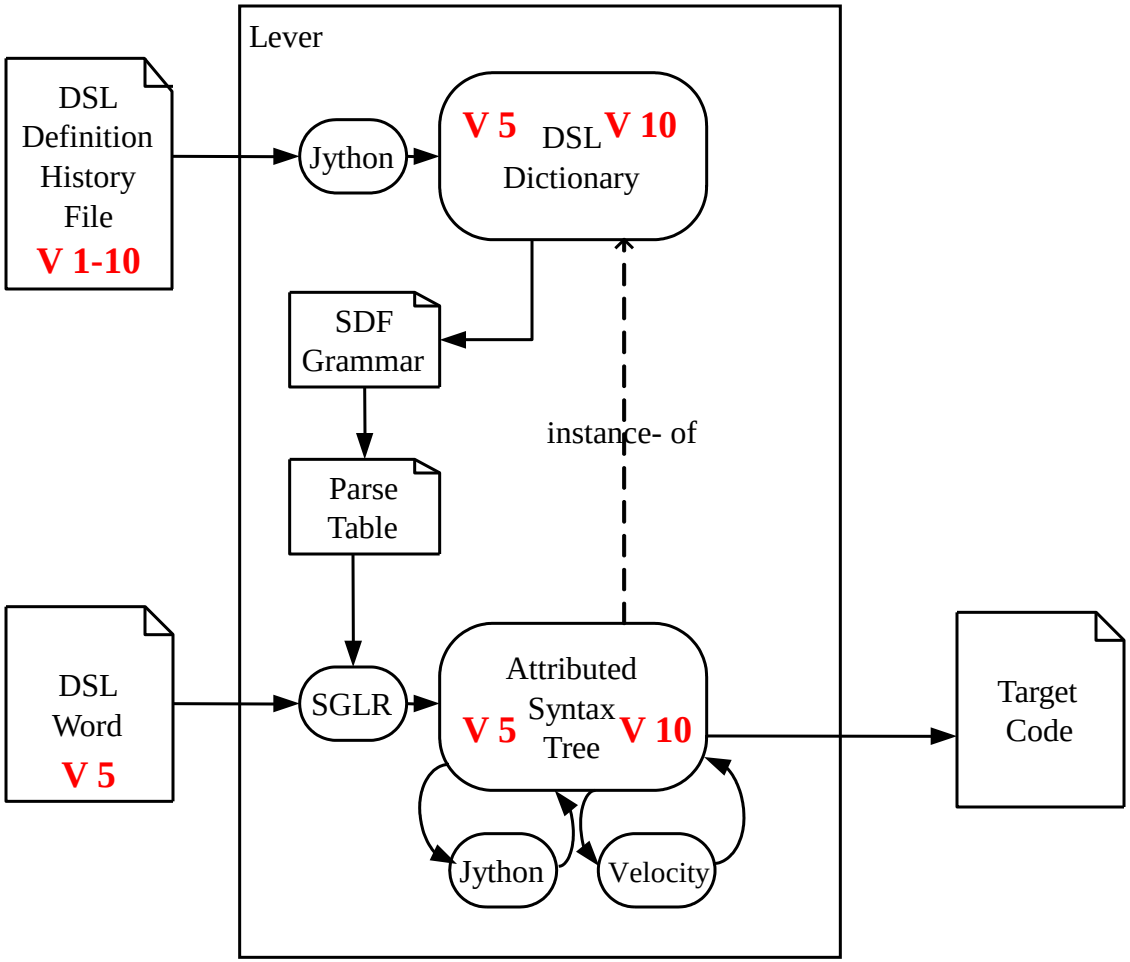
⇒ Higher level of abstraction

⇒ Reuse

WEL

- Transformation of syntax tree
- complete, set-oriented

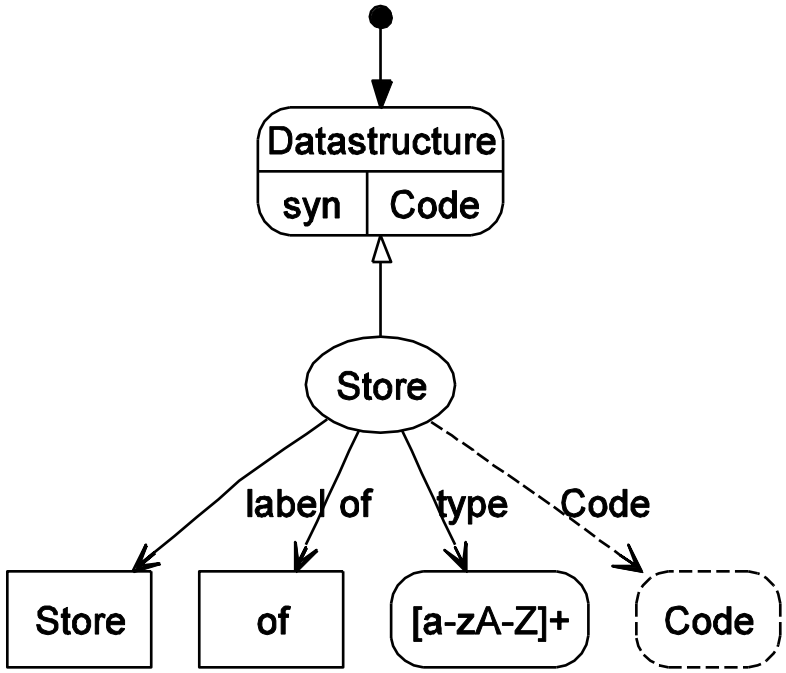
Compilation Process

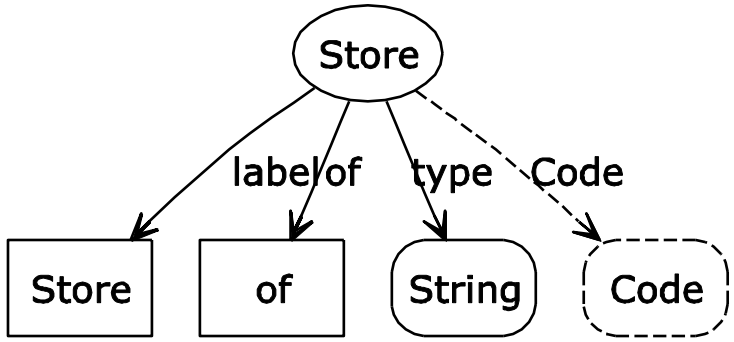


Demonstration

- DSL for generation of typed data structures
- Development in three exemplary evolution steps

Version 1






```
Store of String
```

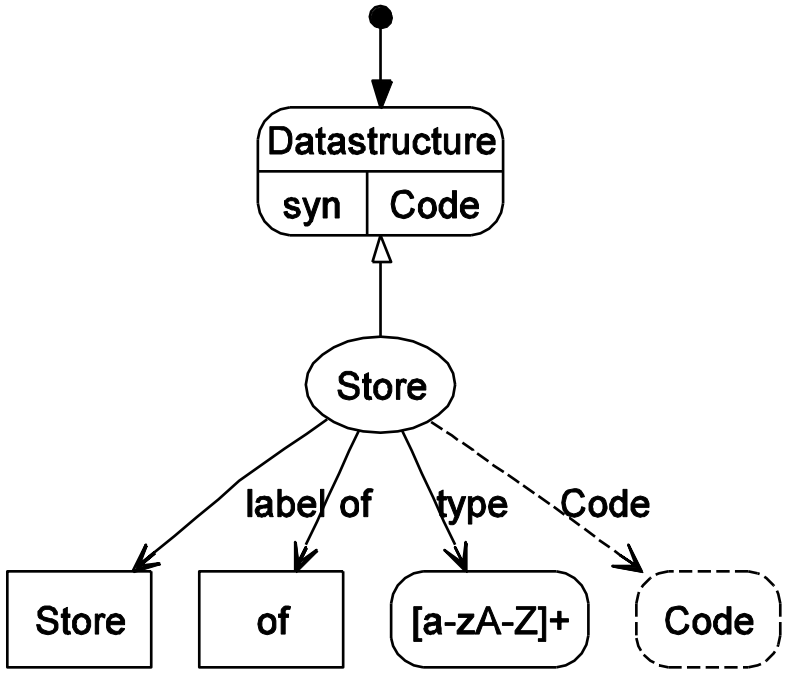
GEL Statements for first Version

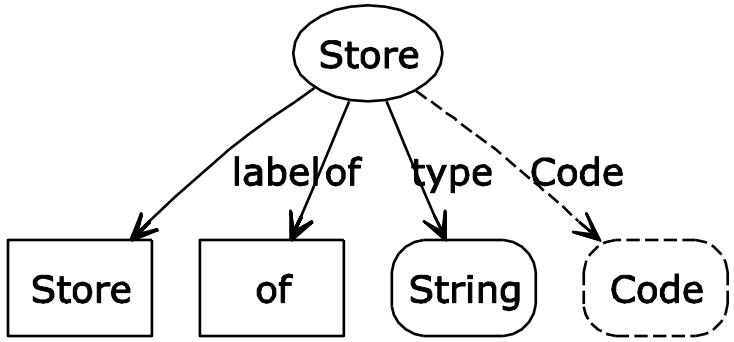
```
g_create_sort("Datastructure");
g_create_attribute("Datastructure", "Code", "syn");
g_set_start_symbol("Datastructure");

s = g_create_production("Store", "Datastructure");
g_append_literal("label", "Store", prod=s);
g_append_literal("of", "of", prod=s);
g_append_terminal("type", "[a-zA-Z]+", prod=s);
g_set_velocity_attribute_rule("Code", file("Store.Code.vtl.v1"),
    prod="Store");
```

Semantic Rule for Code

```
#set($Type = «type»)

/** Typed Linked List for $Type objects. */
public class SimpleList {

    private ListElement head = null;
    class ListElement{

        private ListElement next = null;
        private $Type content = null;

        public ListElement($Type content) {
                this.content = content;
        }
…
```

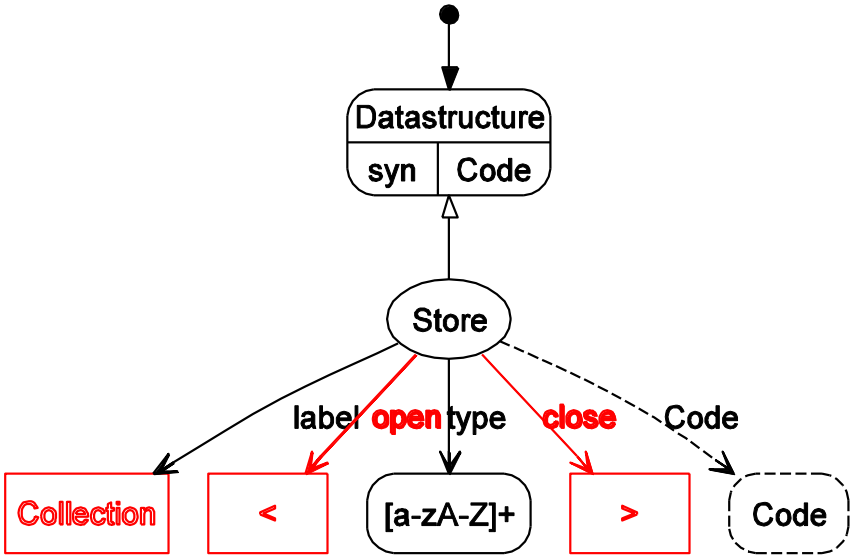
Version 1

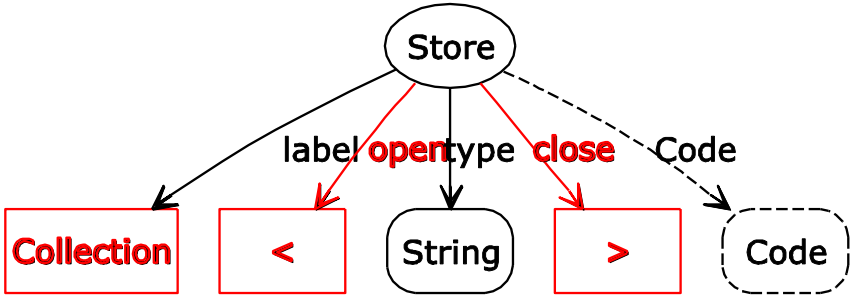





```
Store of String

Collection <String>
```

V2: Change to Concrete Syntax



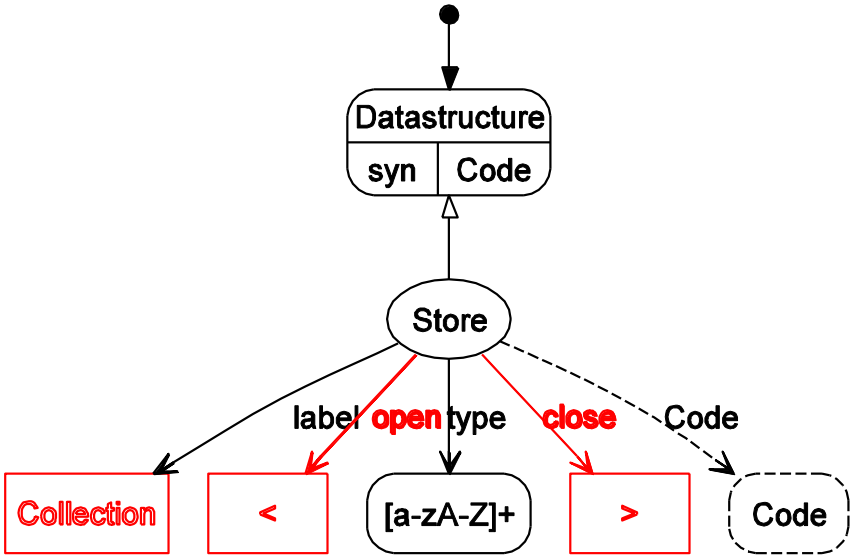


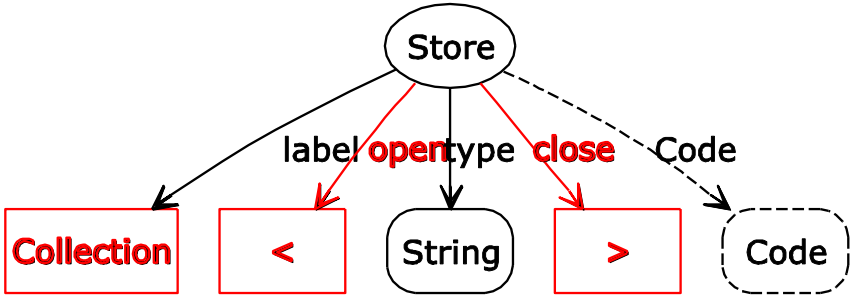

```
Collection <String>
```

LEL statements for version 2

```
l_change_literal("label", "Collection", prod="Store");
l_remove_literal("of", prod="Store");
l_insert_literal_before("open", "<", "type", prod="Store");
l_insert_literal_behind("close", ">", "type", prod="Store");
```

V2: Change to Concrete Syntax

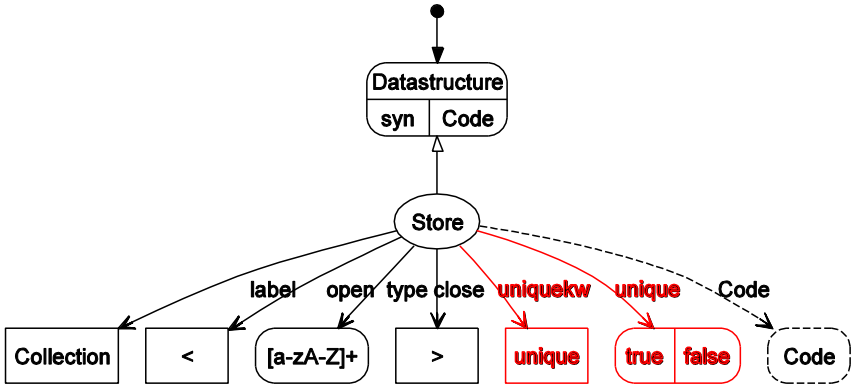


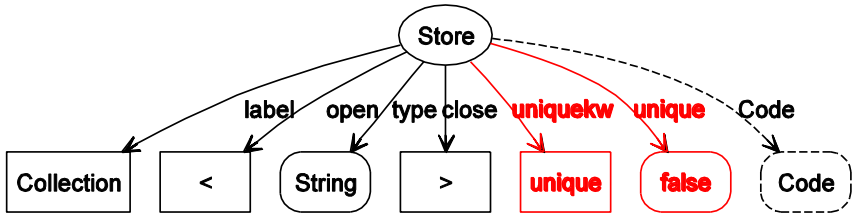



```
Collection <String>
```

```
Collection <String> unique false
```

V3: Language Extension



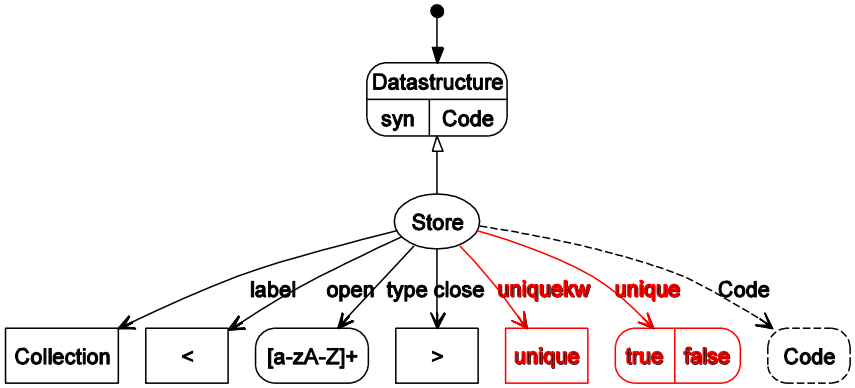


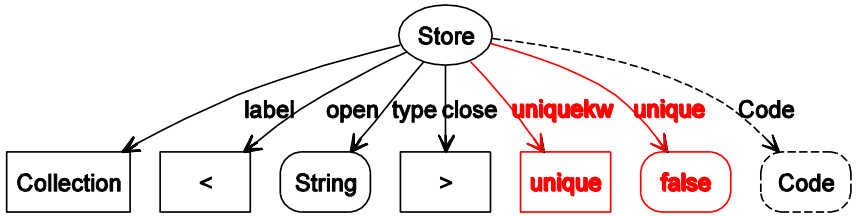

```
Collection <String> unique false
```

Evolution Ops. for Version 3

```
l_insert_literal_behind("uniquekw", "unique", "close", prod="Store");
l_insert_terminal_behind("unique", "true|false", "false", "uniquekw",
    prod="Store");

g_set_velocity_attribute_rule("Code", file("Store.Code.vtl.v3"),
    prod="Store");
```

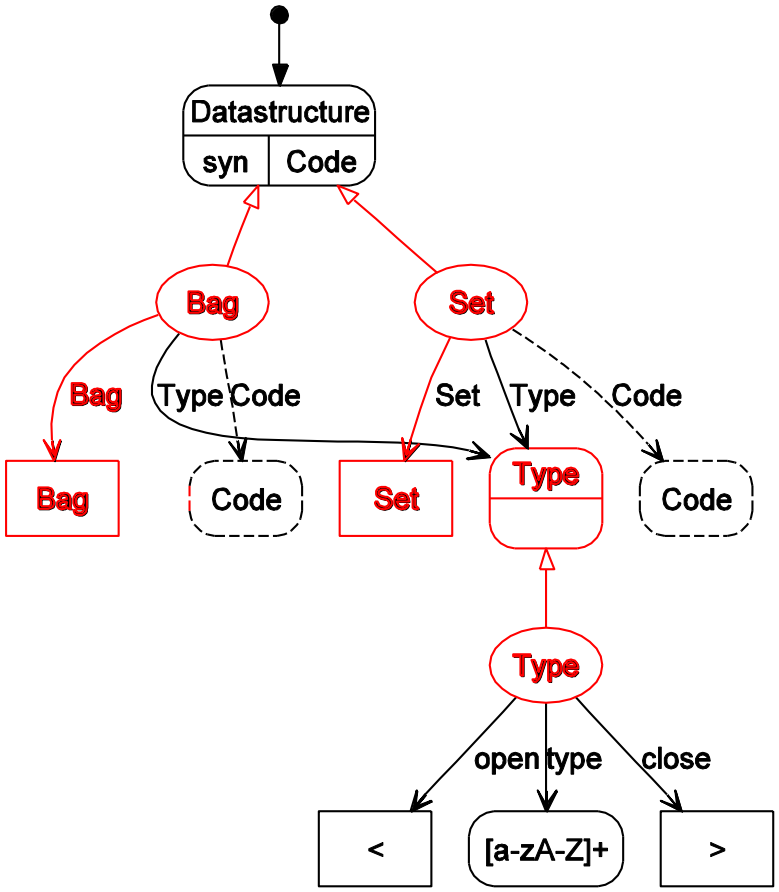
V3: Language Extension

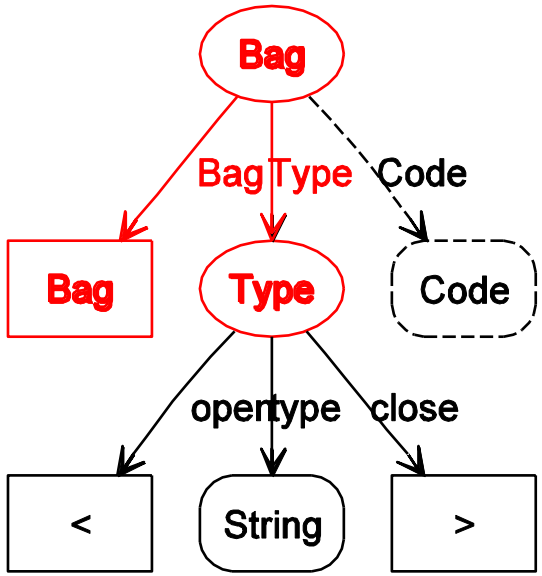





```
Collection <String> unique false

Bag <String>
```

V4: Restructuring






```
Bag <String>
```

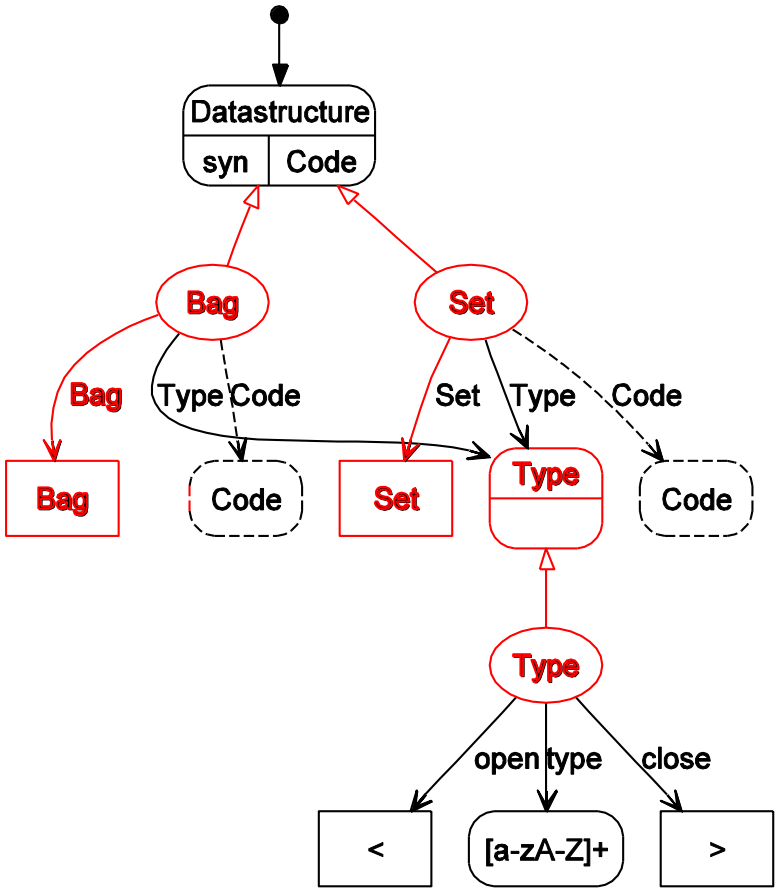
Evolution Ops for Version 4 (1)

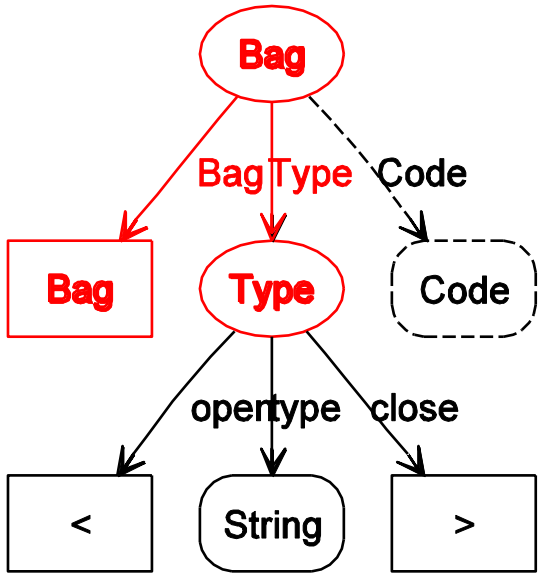
```
l_encapsulate("Type", "Store", "open", "type", "close");
g_set_velocity_attribute_rule("Code", "", prod="Store");

g_create_production("Bag", "Datastructure");
g_append_literal("Bag", prod="Bag");
g_append_nonterminal("Type", "Type", prod="Bag");
g_set_velocity_attribute_rule("Code", file("Bag.Code.vtl.v4"),
    prod="Bag");

g_create_production("Set", "Datastructure");
g_append_literal("Set", prod="Set");
g_append_nonterminal("Type", "Type", prod="Set");
g_set_velocity_attribute_rule("Code", file("Set.Code.vtl.v4"),
    prod="Set");

g_del_production("Store");
```

Evolution Ops. for Version 4

```
bags = select_set("//.[Schema='Datastructures.Store'][unique='false']");
for bag in bags:
    bag.setSchema(g_get_production("Bag"))
    w_append_leaf("Bag", "Bag", bag);
```

V4: Restructuring






```
Bag <String>
```

Summary

Evolution operations automate

- Migration of existing words
- Adaptation of the Compiler

Future Work

- Visual DSLs
- Adaptation of further tools (Pretty Printer, Syntax Highlighting, Code Completion, …)
- More advanced attribute grammar formalism

Thank you!

Questions?

Contact
{juergens,pizka}@in.tum.de