Advanced SW Engineering: Domain-specific Languages

Uwe Zdun

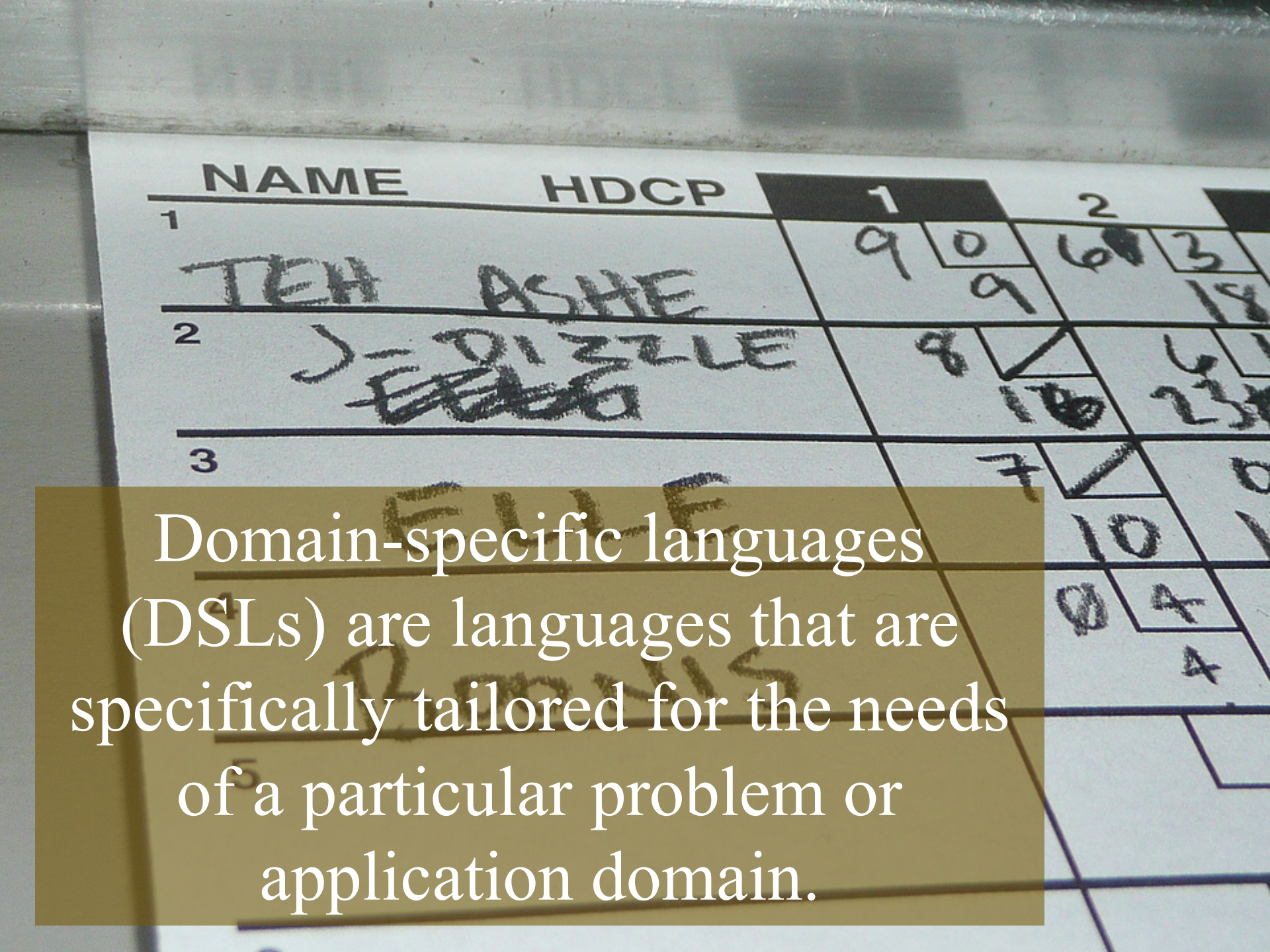
Software Architecture

Faculty of Computer Science

University of Vienna

http://cs.univie.ac.at/swa

DSL



Domain-specific languages (DSLs) are languages that are specifically tailored for the needs of a particular problem or application domain.

On Using DSLs

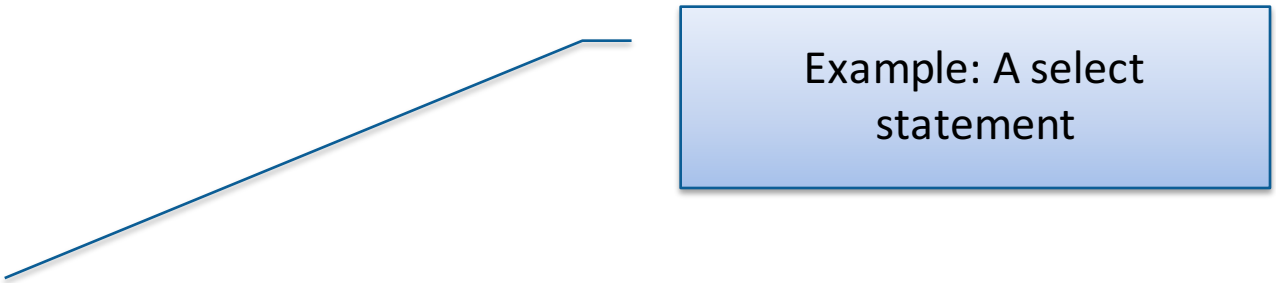
- DSLs promise that domain experts themselves can understand, validate, modify, test, and sometimes even develop DSL programs
- DSLs can help to enable involving domain experts more closely in the software design activities
- DSLs receive a constantly growing attention in recent years

SOME DSL EXAMPLES

Example: Language for Dispatching Incoming HTTP Requests

From:
http://www.eclipse.org/Xtext/7languagesDoc.html#httpRouting

Example: SQL




```
SELECT LAT_N, CITY, TEMP_F
FROM STATS, STATION
WHERE MONTH = 7 AND STATS.ID = STATION.ID
ORDER BY TEMP_F;
```

Example: Regular Expressions

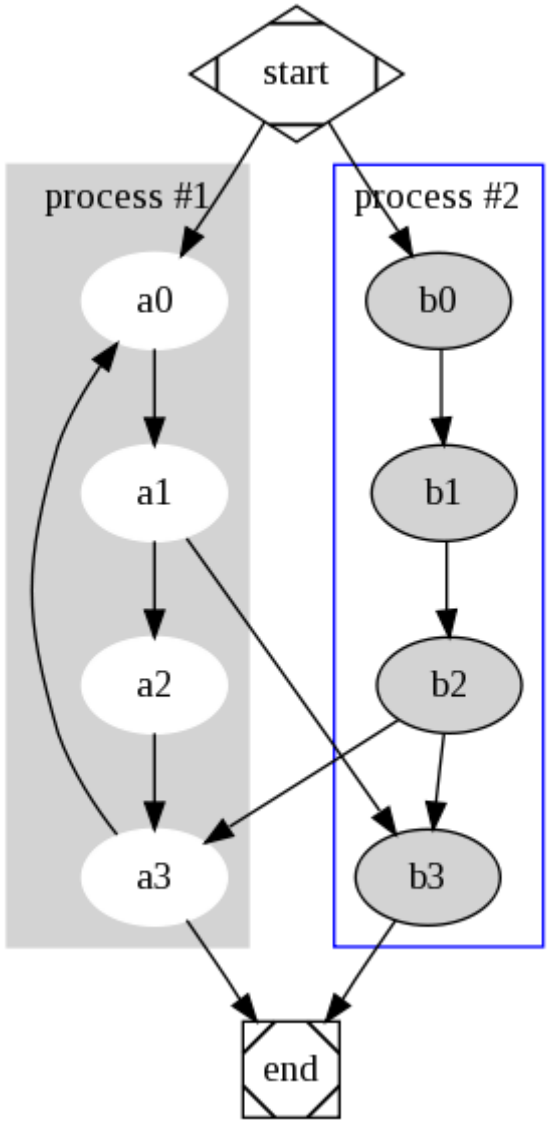
```
set str 66.70.7.154

regexp "(\[0-9]{1,3})\.(\[0-9]{1,3})\.(\[0-9]{1,3})\.(\[0-9]{1,3})" \
        $str all first second third fourth

puts "$all \n $first \n $second \n $third \n $fourth \n"
```

Regular expression languages are often embedded in other programming languages (here: Tcl)

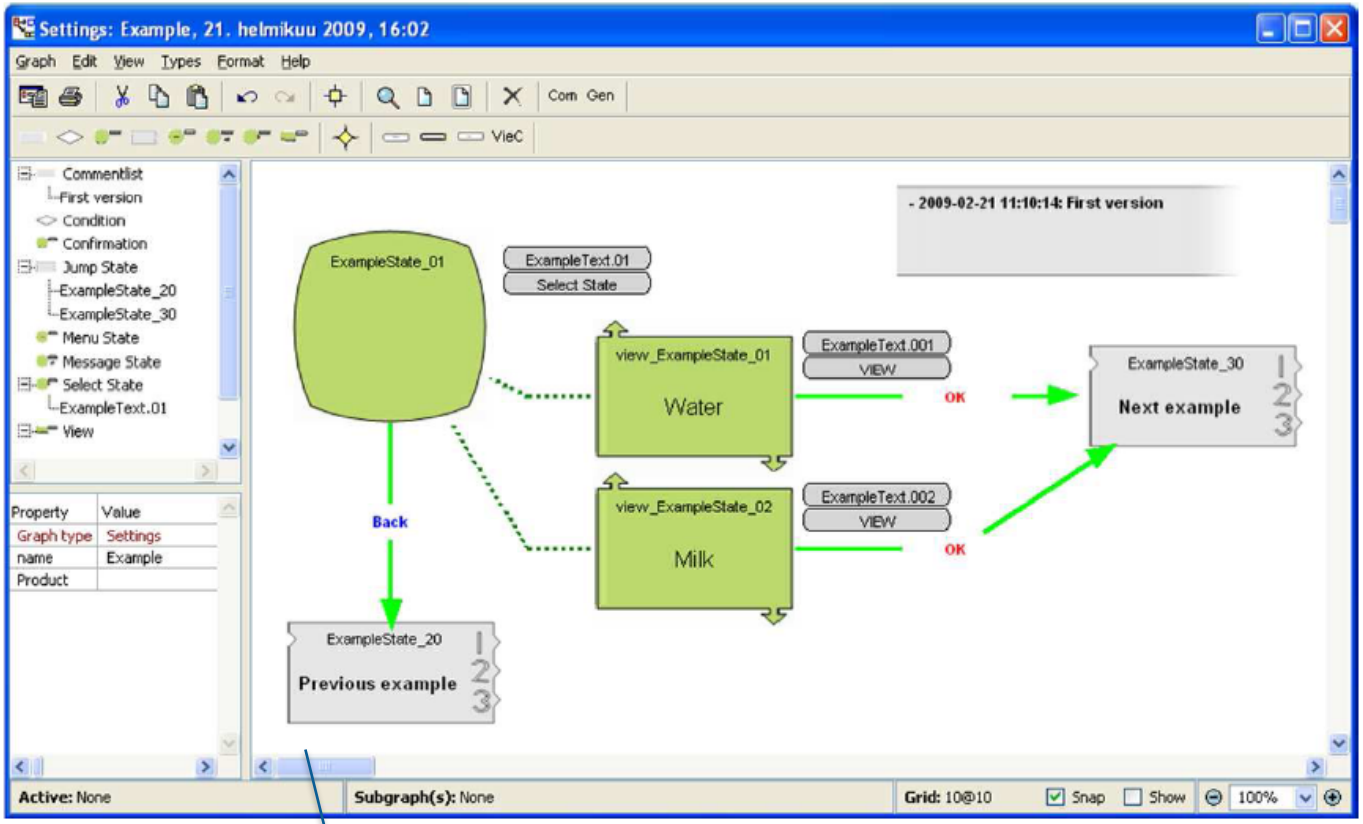
Example: A regexp to check an IP address for correct syntax

Example: Graphviz

```
digraph G {
        subgraph cluster_0 {
                style=filled;
                color=lightgrey;
                node [style=filled,
                  color=white];
                a0 -> a1 -> a2 -> a3;
                label = "process #1";
        }

        subgraph cluster_1 {
                node [style=filled];
                b0 -> b1 -> b2 -> b3;
                label = "process #2";
                color=blue
        }
        start -> a0;
        start -> b0;
        a1 -> b3;
        b2 -> a3;
        a3 -> a0;
        a3 -> end;
        b3 -> end;

        start [shape=Mdiamond];
        end [shape=Msquare];
}
```



Example: Graphical DSL for UI application modeling



Modeling UI states

For products such as

MetaEdit+ Example from:
http://www.dsmforum.org/events/DSM09/Papers/Karna.pdf

Example: DSL for Embedded System Control

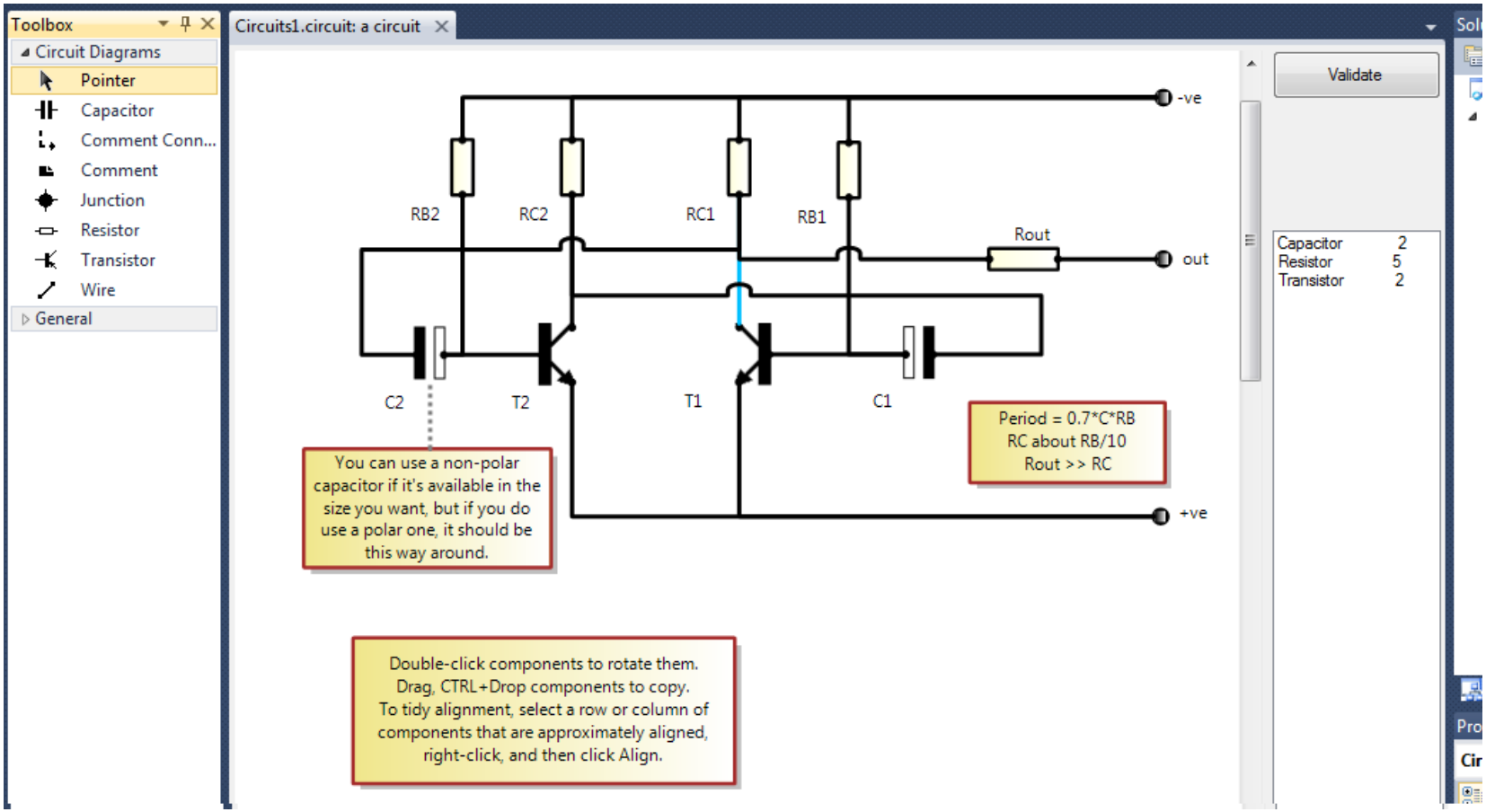
```
doc This module represents the code for the line follower lego robot. It has a coupl
module main imports OsekKernel, EcAPI, BitLevelUtilies {

  constant int WHITE = 500;

  constant int BLACK = 700;

  constant int SLOW = 20;

  constant int FAST = 40;

  doc Statemachine to manage the
  statemachine linefollower {
    event initialized;
    initial state initializing {
      initialized [true] -> runni
    }
    state running {

    }
  }

  initialize {
    ecrobot_set_light_sensor_act
    event linefollower:initializ
  }
```

```
doc This is the cyclic task that is called every 1ms to do the actual control of the
task run cyclic prio = 2 every = 2 {
  stateswitch linefollower
    state running
      int32 light = 0;
      light = ecrobot_get_light_sensor(SENSOR_PORT_T::NXT_PORT_S1);
      if ( light < ( WHITE + BLACK ) / 2 ) {
        updateMotorSettings(SLOW, FAST);
      } else {
        updateMotorSettings(FAST, SLOW);
      }
    default
      <noop>;
}

doc This procedure actually configures the motors
void updateMotorSettings( int left, int right ) {
  nxt_motor_set_speed(MOTOR_PORT_T::NXT_PORT_C, l
  nxt_motor_set_speed(MOTOR_PORT_T::NXT_PORT_B, r
}
```

Modeling a line follower Lego robot

From: http://mbeddr.wordpress.com/examples/

Example: Graphical DSL for Drawing Electronic Circuit Diagrams



From: Microsoft Visualization and Modeling SDK (DSL Tools). http://code.msdn.microsoft.com/Visualization-Modeling-SDK-763778e8

DSL CONCEPTS

3GL vs. DSL

3GL	DSL
Examples: C, C#, Java, Perl, Python, Ruby, or Tcl	Examples: Regular Expressions, CSS, Graphviz, make, SQL
General Purpose Programming Language	Special Purpose Programming Language
Can be applied to arbitrary problem domains	Suitable only for specific problem domain
Turing complete	Particularly expressive and easy to use in one problem domain

DSL Abstraction Levels

- DSLs can be designed and used on different abstraction levels
  - from DSLs for technical tasks
  - to DSLs for tasks on the business-level
- DSLs can also be defined for nontechnical stakeholders, such as business analysts or biologists, for example

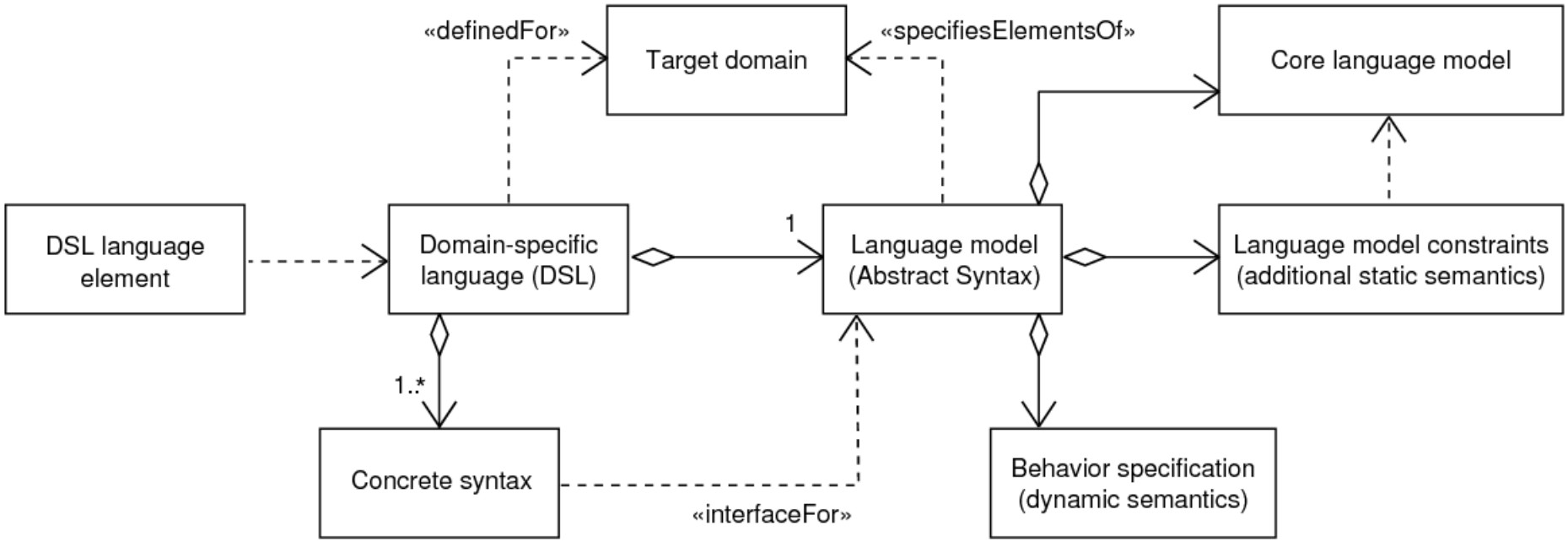
Why are DSLs useful?

- Aspects/Concerns/Viewpoints of a system can be modeled formally, independent of others
- Formal models can be processed automatically by tools (analysis/check, transformation, generation/Interpretation)
- Technology independence of implementation code
- More efficient development
- More reusable systems
- Better software quality

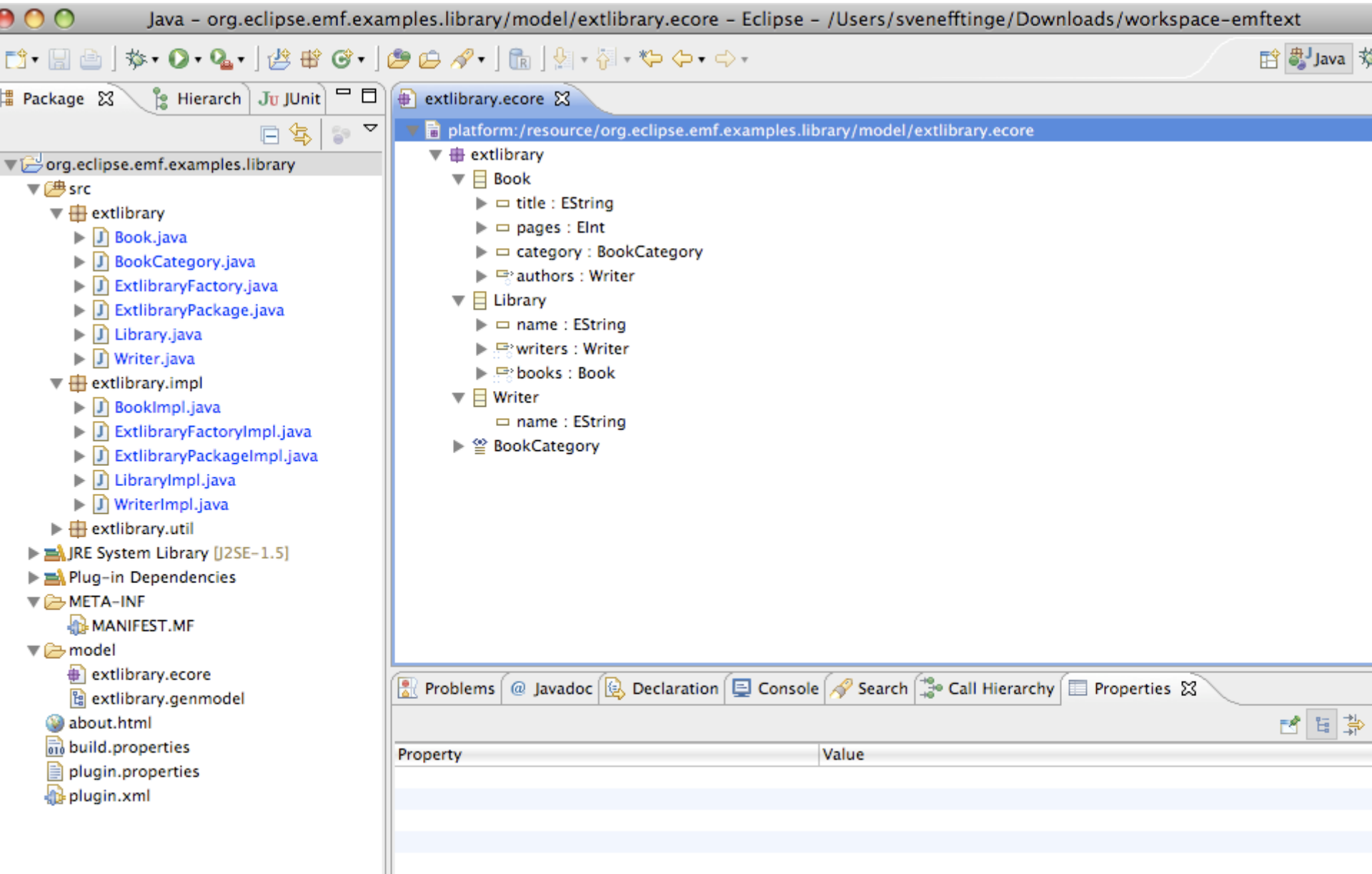
When not to use DSL based software development?

- Effort: Often the development of a DSL does not pay off with a single use, but rather with the second or third use of the DSL
- Language Integration / Learning: Having to deal with multiple languages and their integration
- Reinventing the wheel (e.g. rebuilding a general purpose language implementation)

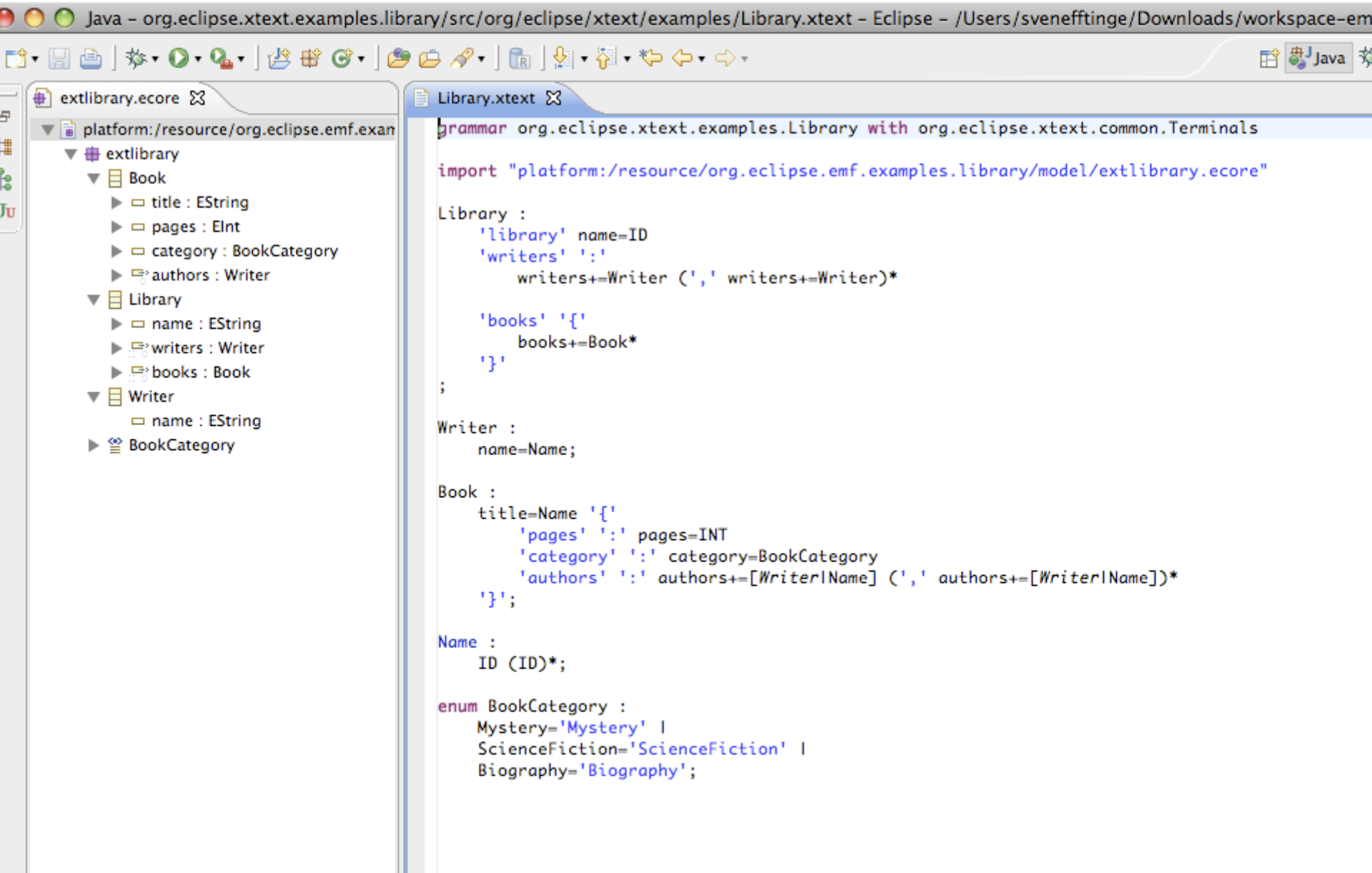
Main Concepts (Artifacts) in DSL Development



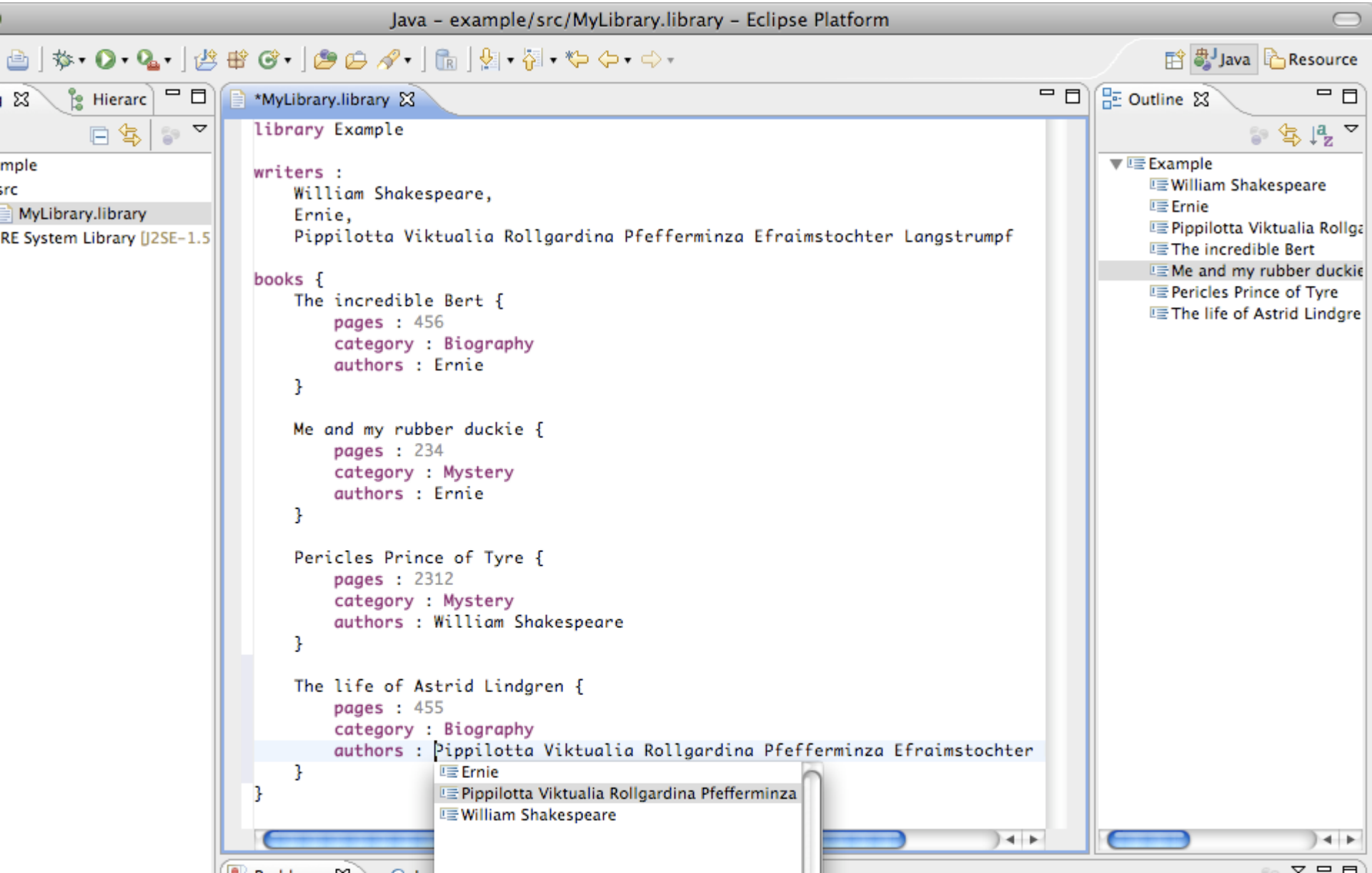
Xtext Example: Language Model in EMF



Xtext Example: Abstract Syntax (Grammar)



Xtext Example: Concrete Syntax

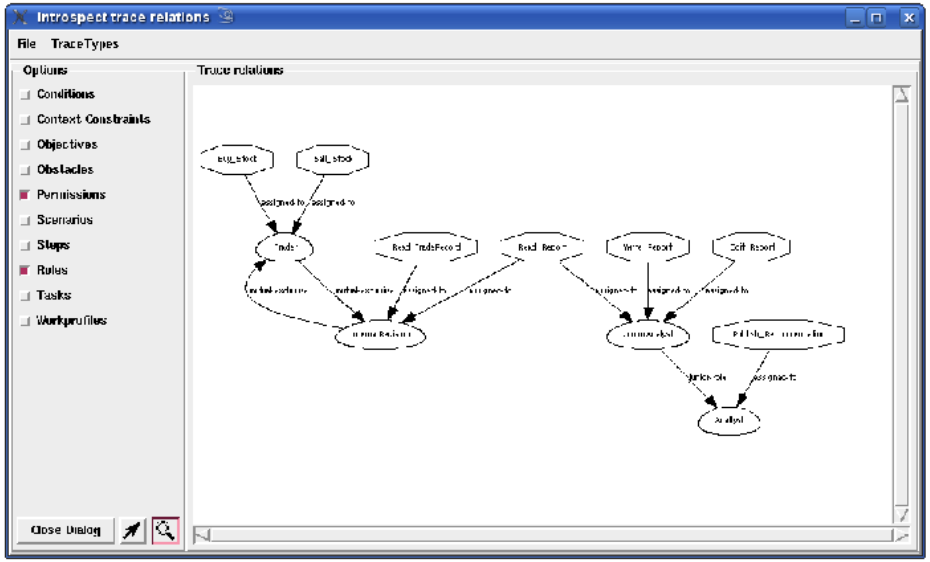


Example: Different concrete syntaxes of a DSL for role-based access control

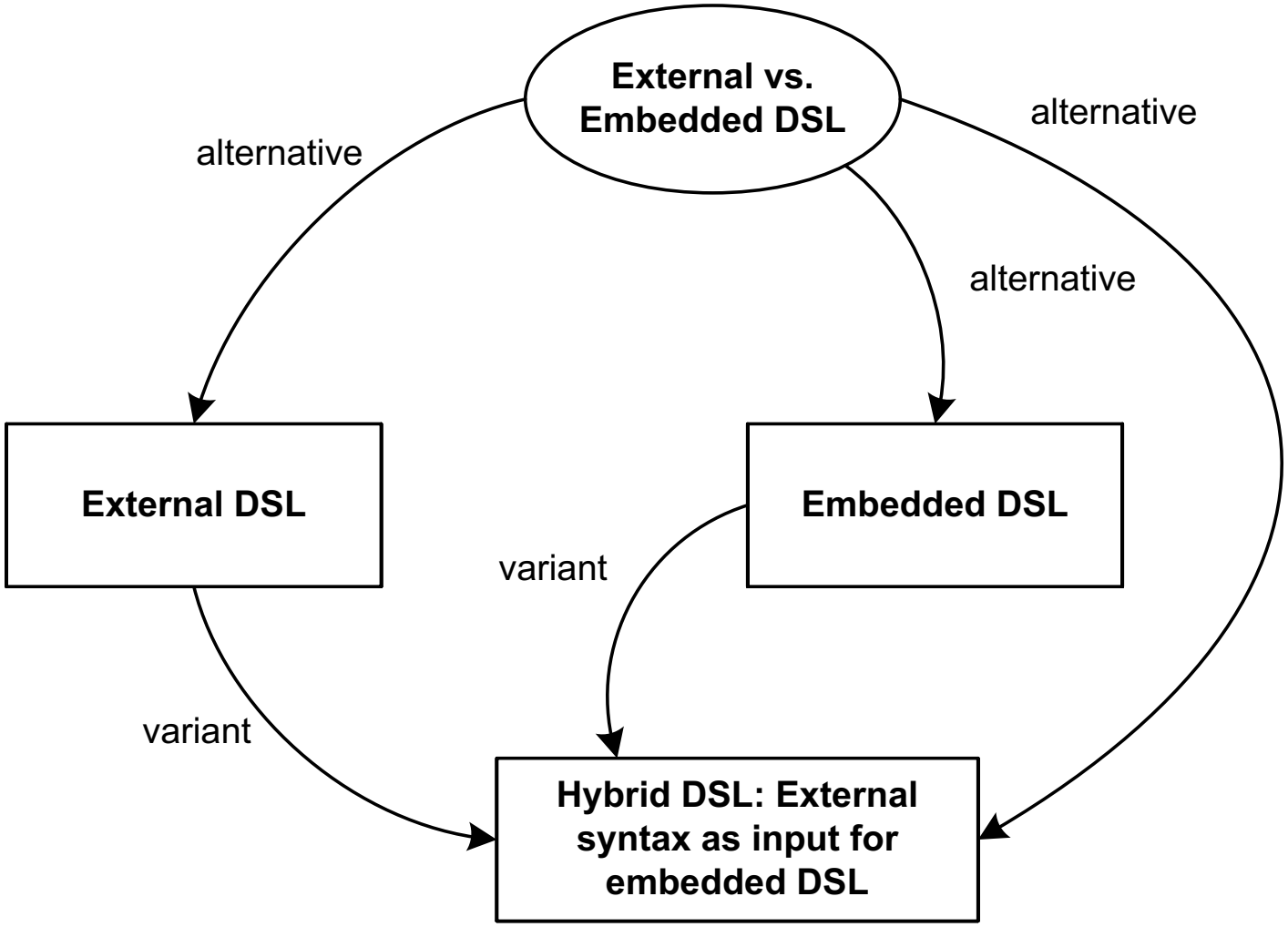
```
<policySpec>
 <subject name="Sarah">
   <role>Analyst</role>
   <role>Trader</role>
 </subject>

 <subject name="Sophie">
   <role>InternalRevision</role>
 </subject>

 <role name="Analyst">
   <permission>Publish_Recommendation</permission>
   <junior>JuniorAnalyst</junior>
 </role>

 <role name="JuniorAnalyst">
   <permission>Read_Report</permission>
   <permission>Edit_Report</permission>
 </role>

 <role name="Trader">
   <permission>Buy_Stock</permission>
   <permission>Sell_Stock</permission>
   <ssd>InternalRevision</ssd>
 </role>

 <role name="InternalRevision">
   <permission>Read_TradeRecord</permission>
   <permission>Read_Report</permission>
   <ssd>Trader</ssd>
 </role>

 <permission name="Buy_Stock">
   <operation>buy</operation>
   <object>stock</object>
 </permission>

 <permission name="Read_TradeRecord">
   <operation>read</operation>
   <object>traderecord</object>
 </permission>

 ...

 <permission name="Write_Report">
   <operation>write</operation>
   <object>report</object>
 </permission>
</policySpec>
```

```
Role create Analyst
Role create JuniorAnalyst
Analyst addJuniorRole JuniorAnalyst
Role create InternalRevision
Role create Trader
...
Permission create Read_Report -operation read -object report
Permission create Write_Report -operation write -object report
Permission create Edit_Report -operation edit -object report
...
JuniorAnalyst assignPermission Read_Report
JuniorAnalyst assignPermission Edit_Report
...
Subject create Sarah
Sarah assignRole Analyst
Sarah assignRole Trader

Subject create Sophie
Sophie assignRole InternalRevision
...
```

Textual concrete syntax for developers (embedded RBAC DSL)



External vs. Embedded DSL



Example: External DSL Definition: An Xtext Grammar

```
Statemachine :
  'events'
     (events+=Event)*
  'end'
  'commands'
     (commands+=Command)*
  'end'
  (states+=State)*;

Event :
  (resetting?='resetting')? name=ID code=ID;

Command :
  name=ID code=ID;

State :
  'state' name=ID
     ('actions' '{' (actions+=[Command])+ '}')?
     (transitions+=Transition)*
  'end';

Transition :
  event=[Event] '=>' state=[State];
```

Example: Embedded DSL – Ruby concrete syntax for Recipe DSL

```
recipe "PBJ Sandwich"

ingredients "two slices of bread",
            "one heaping tablespoon of peanut butter",
            "one teaspoon of jam"

instructions "spread peanut butter...",
             "spread jam...",
             "place other slice..."

servings 1
prep_time "2 minutes"
```

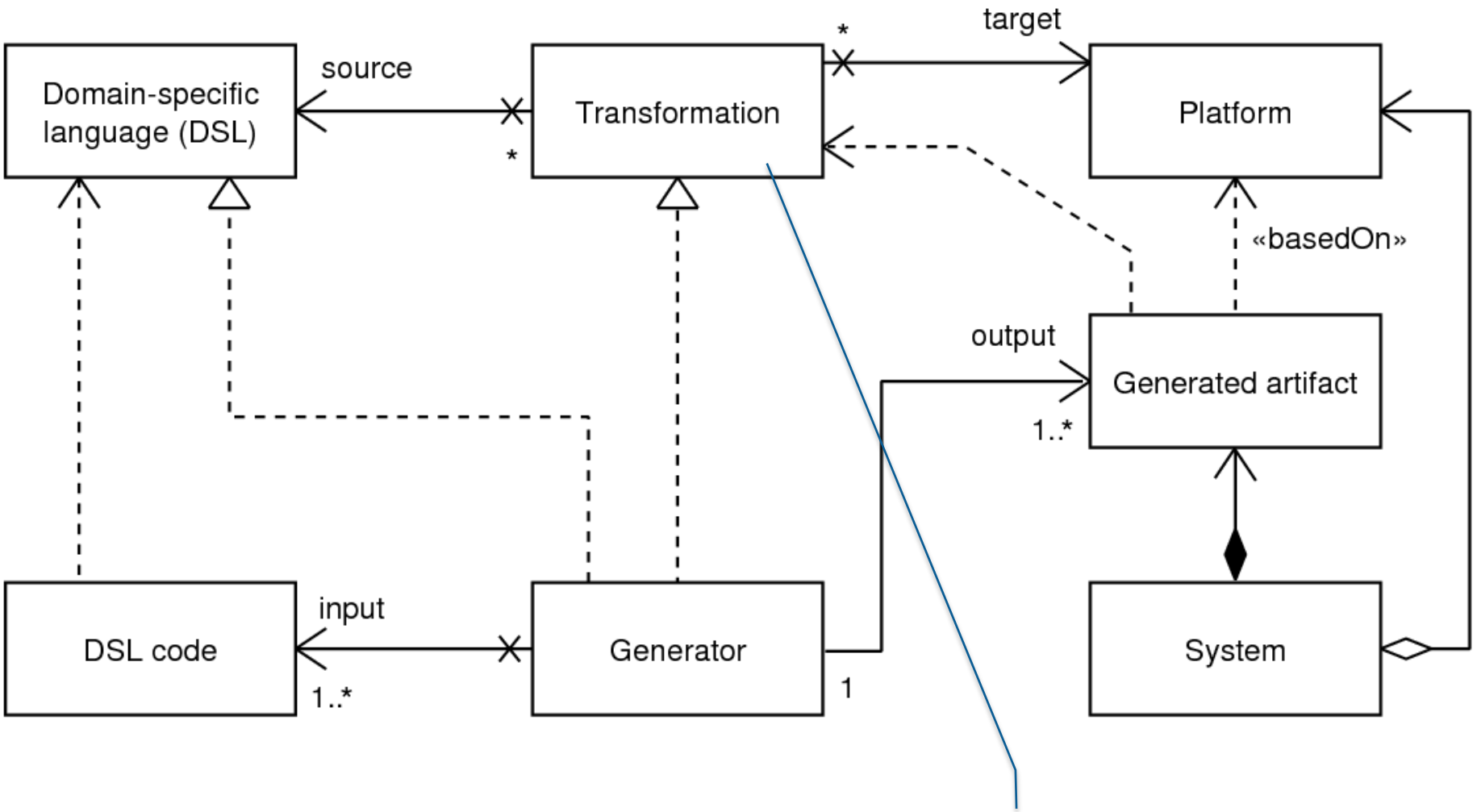
Fluent Interfaces

- A **fluent interface** is an implementation of an object oriented API that aims to provide for more readable code
- A step towards embedded DSLs are fluent interfaces

```
private void makeFluent(Customer customer) {
      customer.newOrder()
      .with(6, "TAL")
      .with(5, "HPK").skippable()
      .with(3, "LGV")
      .priorityRush();
}
```

Example from:
http://martinfowler.com/bliki/FluentInterface.html

DSL Transformation Concepts



Includes code generation, interpretation of DSL code, byte code compilation of DSL code, model-to-model transformations, etc.

Relevant Literature and Sources

- M. Völter. DSL Engineering - Designing, Implementing and Using Domain-Specific Languages. http://dslbook.org/, 2013.
- Reusable Architectural Decisions for DSL Design: Foundational Decisions in DSL Development. Zdun, Uwe and Strembeck, M. In: Proceedings of 14th European Conference on Pattern Languages of Programs (EuroPLoP 2009), Irsee, Germany (2009). http://cs.univie.ac.at/research/research-groups/software-architecture/publikation/infpub/2327/
- M. Voelter. Textuelle DSLs mit Eclipse Xtext. prio.conference, 2008. http://www.voelter.de/data/presentations/TextualDSLs.pdf

Many thanks for your attention!

Uwe Zdun

Software Architecture
Faculty of Computer Science
University of Vienna
http://cs.univie.ac.at/swa