Domain-Specific Languages (DSLs)
motivation, concepts, examples

Thorsten Berger
Associate Professor
thorsten.berger@chalmers.se
architecture of a compiler
a perspective from systems engineering

- UML models
- C code
  ```c
  #if CONFIG_JFFS2_FS_DEBUG > 0
  /* Enable "paranoia" checks and dumps */
  #define JFFS2_DBG_PARANOIA_CHECKS
  #define JFFS2_DBG_DUMPS
  ...
  #ifdef CONFIG_JFFS2_ZLIB
  jffs2_zlib_init();
  #endif
  ```
- hybrid models
- OS generation
- calibration
- HW/SW mapping
Manual process to ensure coherence between views
HISTORY OF LANGUAGES
continuous abstractions

language

machine language

Assembler

Algol, Fortran, LISP

C, Perl, Pascal

C++, Java, C#

concepts

objekt-orientation: classes, objects

modularization, functions

conditionals, loops

jumps, arithmetic expressions

count, read register, I/O

(C) Thorsten Berger, Chalmers | University of Gothenburg
general-purpose tools
domain-specific tools

focus on the domain!

tailored towards domain requirements
effective within the domain
both specialized and limited
used by domain experts
use domain-specific languages!

Expressiveness:

- **General-Purpose Language (GPL)**
  - Assembler
  - Algol, Fortran, LISP
  - C, Perl, Pascal
  - C++, Java, C#

Concepts:

- arbitrary domain concepts
- objekt-orientation: classes, objects
- modularization, functions
- conditionals, loops
- jumps, arithmetic expressions
- count, read register, I/O
domain-specific language

advantages

less expressive
separates domain-related and infrastructure code
improves communication with domain experts and customers

DSLs exist for a long time, such as:

- regular expressions: \[-+]?[0-9]*\.?[0-9]+\]
- SQL: `SELECT ... FROM ... WHERE ...`
- CSS/HTML: `b{ color: #926C41 }`

recent improvements in language technology help

to quickly create your own DSL
language workbenches instead of CASE tools
model-transformation languages

disadvantage: tools are still somewhat complex
kinds of DSLs

- configuration tool
- library in a programming language
- wizard
- textual DSL
- visual DSL
EXAMPLES
INTERNAL (EMBEDDED) DSLS
UI programming in Java

Java

```java
Display display = Display.getDefault();
Shell shell = new Shell(display);
shell.setText("SWT");
shell.setLayout(new FillLayout());
Label label = new Label(composite, SWT.NONE);
label.setText("Hello World!");
shell.pack();
shell.open();
while(!display.isDisposed()){  
   if(!display.readAndDispatch()){  
       display.sleep();
   }
}
display.dispose();
```

JRuby with Glimmer DSL

```ruby
shell {
   text "SWT"
   label {
       text "Hello World!"
   }
}
```
object Lunar extends Baysick {
    def main(args: Array[String]) = {
        10 PRINT "Welcome to Baysick Lunar Lander v0.9"
        20 LET ('dist := 100)
        30 LET ('v := 1)
        40 LET ('fuel := 1000)
        50 LET ('mass := 1000)

        60 PRINT "You are drifting towards the moon."
        70 PRINT "You must decide how much fuel to burn."
        80 PRINT "To accelerate enter a positive number"
        90 PRINT "To decelerate a negative"

        100 PRINT "Distance " % 'dist % "km, " % "Velocity"
        110 INPUT 'burn
        120 IF ABS('burn) <= 'fuel THEN 150
        130 PRINT "You don't have that much fuel"
        140 GOTO 100
        150 LET ('v := 'v + 'burn * 10 / ('fuel + 'mass))
        160 LET ('fuel := 'fuel - ABS('burn))
        170 LET ('dist := 'dist - 'v)
        180 IF 'dist > 0 THEN 100
        190 PRINT "You have hit the surface"
        200 IF 'v < 3 THEN 240
        210 PRINT "Hit surface too fast (" % 'v % ")km/s"
        220 PRINT "You Crashed!"
        230 GOTO 250
        240 PRINT "Well done"

        250 END
    }
}

https://www.scala-lang.org/old/node/1403
EXTERNAL GRAPHICAL DSLS
Automate (Android App)
Lego Mindstorms Robots
Sound Tracker:
Every time a sound is detected, this box makes NAO look toward the sound source.
EXTERNAL TEXTUAL DSLS
message Person {
    enum PhType { MOBILE = 0; WORK = 1; }
    message PhoneNo {
        required string no = 1;
        optional PhType type = 2 [default = MOBILE];
    }
    required string name = 1;
    repeated PhoneNo phone = 2;
}
developed in the MDE course

```json
{
    "Lego": "Star",
    "Length": 20,
    "Width": 20,
    "Bricks": [{
        "Brick": "Wars",
        "Width": [4],
        "Length": [2]
    }, {
        "Brick": "Trek",
        "Width": [2],
        "Length": [2]
    }]
}
```
AdjLegoSystem {
    thickness 20
    finalBrick Pizza
abstractLegobrick {
    RoundedBrick Pizza{
        roundedSide ALL
        sizeproperties {
            int length = 7,
            int width = 7
        }
    },
    SlicedBrick Slice {
        portions 3
        brick Pizza
    }
}
}
AdjLegoSystem {
    thickness 7
    finalBrick Boomerang
    abstractlegobrick {
        RoundedBrick Frisbee{
            roundedSide RIGHT
            sizeproperties {
                int length = 4,
                int width = 2
            }
        },

        SquareBrick Stick {
            sizeproperties {
                int length = 4,
                int width = 2
            }
        },

        Combination Boomerang {
            mainSide LEFT
            position 3
            main Frisbee
            secondary Stick
        }
    }
}
dimensions 10 x 10;
"2x4": 2 x 4;
"4x2": 4 x 2;
"1x8": 1 x (2 * "4x4".width);
"4x4": "2x4".height x (2 * "2x4".width);
"Composite Brick 1": "2x4" <- "4x4" TOP: LEFT 1 <- LEFT 1;
"Composite Brick 2": "2x4" <- "4x2" BOTTOM: LEFT 1 <- LEFT 4;
"Composite Brick 3": "1x8" <- "4x2" RIGHT: TOP 1 <- BOTTOM 2;
brick smiley

```
o o o o o o o,
o_ _ o o _ _ o,
o_ _ o o _ _ o,
o_ _ o o _ _ o,
o o o o o o o,
o _ o o _ _ o,
o _ _ _ _ _ _ o,
o o o o o o o,
```

abstract brick a

```
o o o,
o o o,
o o o,
o o o,
```

abstract brick b

```
_ o o,
_ o o,
_ o o,
```

combo T a over b
Autonomous Driving, Parking DSL
Controlling a NAO Robot
kinds of DSLs

comprehensible
good for nonprogrammers
layout conveys information
layout tedious to use
layout tedious to implement
syntax scalability issues
examples: BPMN, class diagrams, feature models, Yahoo pipes, MIT scratch

implemented as a separate language; parsed, translated/interpreted
good control over syntax, semantics, tooling
costlier to implement
eexample: Google protocol buffers

clear order of reading
efficient editing
easy to incorporate expressions
popular among programmers
cheaper to implement, good tools exist
harder to read, esp. for non-programmers
complex dependencies hide in indirections
examples: Google protocol buffers, Rails active record

a library within a host language
host must be flexible syntactically and support meta-programming features:
scala, ruby, python, smalltalk, C++, haskell
cheap to implement, reuse from host
limited control of syntax, semantics, tooling
eexample: Rails active record
BUILDING DSLS
language workbenches

language workbench: tool for creating and using (domain-specific) languages

early workbenches (textual)
   SEM, MetaPlex, Metaview, QuickSpec, MetaEdit
workbenches for graphical languages
   MetaEdit+, DOME, GME
workbenches for textual languages
   Centaur, Synthesizer, ASF+SDF Meta-Environment, Gem-Mex/Montages,
   LRC, Lisa, JastAdd, Rascal, Spoofax, Xtext
workbenches for projectional languages
   JetBrains MPS, Intentional Domain Workbench, …

in-class demo

We will implement a language (incl. abstract syntax and a textual and graphical concrete syntax) for expressing simple graphs.

Using the following technological space:
meta-model (abstract syntax)

how to build a modelling languages?
we *model* the modelling language
result is a meta-model:

![UML Diagram]

(C) Thorsten Berger, Chalmers | University of Gothenburg
meta-model: definition

A meta-model is a model that precisely defines the parts and rules needed to create valid models.

Parts: domain concepts (model elements)

Defines a languages’ abstract syntax: elements and their relations independent of the representation

Mapped to:
- Concrete syntax: representation of models (instances), e.g., within an editor
- Static semantics: semantics evaluable without executing/interpreting the model (using constraints)
- Dynamic semantics: what the model means or expresses

(C) Thorsten Berger, Chalmers | University of Gothenburg
meta-model: lego

Meta-Model
Building Rules

Building Block

connected with

Model
Lego

Real world
House

Examples from Steffen Becker, Model-Driven Software Development, Univ. of Paderborn

Images: pixelquelle.de
**meta-model: languages**

**Meta-Model**
- Building Rules

**Model**
- A natural description
  
  "A nice brown and white coloured house in the middle of the black forest"

**Real world**
- House

Examples from Steffen Becker, Model-Driven Software Development, Univ. of Paderborn

Images: pixelquelle.de
a well-known DSL

Music notation

Metamodel

Model

Music sheet

examples from Steffen Becker, Model-Driven Software Development, Univ. of Paderborn
meta-model levels

a meta-model is *also a model*
how is the meta-model then being modelled?
using a meta-meta-model! 😊

this goes (theoretically) infinite or ends if the model is *self-describing*

typically called *bootstrapping* (your favorite Java compiler is likely implemented in Java itself)
abstract vs. concrete syntax

abstract

<table>
<thead>
<tr>
<th>Name:Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributeName=&quot;Name&quot;</td>
</tr>
<tr>
<td>type=&quot;String&quot;</td>
</tr>
</tbody>
</table>

Lecturer:Class

<table>
<thead>
<tr>
<th>className=&quot;Lecturer&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>modelName=&quot;Lecturer&quot;</td>
</tr>
</tbody>
</table>

Concrete

Lecturer

<table>
<thead>
<tr>
<th>Name:String</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Class Lecturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>String Name;</td>
</tr>
</tbody>
</table>

modelclass Lecturer

<table>
<thead>
<tr>
<th>attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name : Type String</td>
</tr>
</tbody>
</table>

Compare to the AST concept in compiler construction!

elements from Steffen Becker, Model-Driven Software Development, Univ. of Paderborn
abstract vs. concrete syntax

abstract syntax uses the meta-model concepts to represent the models

concrete syntaxes can choose any kind of representation plus a mapping
abstract vs. concrete syntax

A mapping maps elements of the concrete syntax to the meta-model elements.

```
<table>
<thead>
<tr>
<th>aClass</th>
<th>&lt;&lt;mapped to&gt;&gt;</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>aAttribute</td>
<td>&lt;&lt;mapped to&gt;&gt;</td>
<td>*Name:String</td>
</tr>
<tr>
<td></td>
<td>&lt;&lt;mapped to&gt;&gt;</td>
<td>Attribute</td>
</tr>
</tbody>
</table>
```

Concrete UML Syntax

Abstract UML Syntax

Examples from Steffen Becker, Model-Driven Software Development, Univ. of Paderborn
abstract vs. concrete syntax

A mapping maps elements of the concrete syntax to the meta-model elements

```xml
<class
  className="Lecturer">
  <attribute
    attributeName="Name"
    type="String"/>
</class>
```

there is a standard defining the mapping from (MOF) models to XML: XMI (XML Metadata Interchange)

examples from Steffen Becker, Model-Driven Software Development, Univ. of Paderborn
example walkthrough again

EXTERNAL DSL (TEXTUAL)
coffee machine
coffee machine
input / output

\[ \Sigma = \{ \text{coin, timeout, tea, coffee, break} \} \]
\[ \Gamma = \{ \text{Tea served, enjoy!}, \text{Machine is broken.}, \ldots \} \]
int current = INITIAL;
while (true) {
    String input = scanner.nextLine();
    switch (current) {
    case INITIAL:
        switch (input) {
            case "coin":
                System.out.println ("What drink do you want?");
                current = SELECTION; break;
            case "break":
                System.out.println ("Machine is broken");
                current = FAILURE; break;
        } break;
    case SELECTION:
        switch (input) {
            ...
        }
    }
}

(C) Thorsten Berger, Chalmers | University of Gothenburg
other implementation options

switch pattern
  simple, fast
  cluttered

state pattern [Gamma et al. 95, Johnson and Zweig 91]
  comprehensible (esp. for hierarchical models)
  OO overhead (polymorphism)

interprete a data structure at runtime [Pinter and Majzik 03, Zündorf 02]
  few code, very memory-efficient
  interpreter overhead

(C) Thorsten Berger, Chalmers | University of Gothenburg
use existing tools?

IBM Rational Rhapsody

Papyrus

Mathworks Simulink
let’s build our own DSL

domain-specific language (DSL)

programming language (GPL)

```java
int current = INITIAL;
while (true) {
    String input = scanner.nextLine();
    switch (current) {
        case INITIAL:
            switch (input) {
                case "coin":
                    System.out.println("What drink do you want?");
                    current = SELECTION; break;
                case "break":
                    System.out.println("Machine is broken");
                    current = FAILURE; break;
            }
            break;
        case SELECTION:
            switch (input) {
                ... code generation ...
                case "break":
                    System.out.println("Machine is broken");
                    current = FAILURE; break;
            }
            break;
    }
}
```

graphical DSL

textual DSL

DSL ... Domain-Specific Language
GPL ... General-Purpose Language

(C) Thorsten Berger, Chalmers | University of Gothenburg
automata model in concrete textual syntax

```plaintext
machine coffeeMachine [
  initial ^initial
  state ^initial [
    on input "coin" output "what drink do you want?" and go to selection
    on input "break" output "machine is broken" and go to failure
  ]
  state selection [
    on input "tea" output "serving tea" and go to makingTea
    on input "coffee" output "serving coffee" and go to makingCoffee
    on input "timeout" output "coin returned; insert coin" and go to ^initial
    on input "break" output "machine is broken!" and go to failure
  ]
  state makingCoffee [
    on input "done" output "coffee served. Enjoy!" and go to ^initial
    on input "break" output "machine is broken!" and go to failure
  ]
  state makingTea [
    on input "done" output "tea served. Enjoy!" and go to ^initial
    on input "break" output "machine is broken!" and go to failure
  ]
  state failure
]
```

editor with:
auto-completion,
syntax- and
reference-checks

(C) Thorsten Berger, Chalmers | University of Gothenburg
two external DSL tactics

ANTLR supports many target platforms. There exists a tool like that for any serious language technical space
ANTLR can do a bit more than the figure would suggest (AST classes and construction)
there exist workbenches for visual languages (GMF, Graphiti, Microsoft DSL Tools, ...)
implementation of graphical DSLs only differs from textual in the syntax/editor aspect
the third tactic: do everything manually is left only to the insane ...
syntax

**xtext**
(language workbench)

abstract syntax
(meta model, e.g., class diagram)

concrete syntax
(grammar, EBNF-like)

automatic generation
syntax

abstract syntax  concrete syntax

FiniteStateMachine:
  'machine' name=ID
  (['' initial' initial=[State]
   (states+=State)*
  ']')?;

State:
  'state' name=ID (['' leavingTransitions+=Transition* ''])?;

Transition:
  'on' 'input' input=STRING ('output' output=STRING)?
  'and' 'go' 'to' target=[State];
model-driven development

domain-specific language (DSL)
in concrete and abstract syntax

programming language (GPL)

```java
int current = INITIAL;
while (true) {
    String input = scanner.nextLine();
    switch (current) {
        case INITIAL:
            switch (input) {
                case "coin":
                    System.out.println ("What drink do you want?");
                    current = SELECTION;
                    break;
                case "break":
                    System.out.println ("Machine is broken");
                    current = FAILURE;
                    break;
            }
            break;
        case SELECTION:
            switch (input) {
            }
            break;
        case FAILURE:
            break;
    }
}
```

code generation (M2T)
def static compileToJava(FiniteStateMachine it) {
    var int i = -1
    ...

    Scanner scanner = new Scanner(System.in);
    current = «initial.name.toUpperCase»;
    while (true) {
        print ("[" + stateNames[current] + "]");
        print (("What is the next event? available: " + availableInputs[current]));
        String input = scanner.nextLine();
        switch (current) {
            «FOR state : states»
            case «state.name.toUpperCase»:
                switch (input) {
                    «FOR t : state.leavingTransitions»
                    case "«t.input»":
                        println ("machine says: «t.output»");
                        current = «t.target.name.toUpperCase»;
                        break;
                    «ENDFOR»
                }
            break;
            «ENDFOR»
        }}}}

def static compileToDot(FiniteStateMachine it) {
    ...

    digraph "«it.name»" {
        _init -> «it.initial.name»;
        «FOR state : states»
            «FOR t : state.leavingTransitions»
                «case "«t.input»":»
                    println ("machine says: «t.output»");
                    current = «t.target.name.toUpperCase»;
                    break;
            «ENDFOR»
        break;
        «ENDFOR»
    }}}}

(C) Thorsten Berger, Chalmers | University of Gothenburg
overview

grammar of our automata DSL

meta model og out automata DSL

Xtend grammar

Java grammar

automata model (text)

automata model (objects)

M2T

Java code

instanceOf

instanceOf

instanceOf

instanceOf

(C) Thorsten Berger, Chalmers | University of Gothenburg
machine coffeeMachine [  
    initial ^initial  
    state ^initial [  
        on input "coin" output "what drink do you want?" and go to selection  
        on input "break" output "machine is broken!" and go to failure  
    ]  
    state selection [  
        on input "tea" output "serving tea" and go to makingTea  
        on input "coffee" output "serving coffee" and go to makingCoffee  
        on input "timeout" output "coin returned; insert coin" and go to ^initial  
        on input "break" output "machine is broken!" and go to failure  
    ]  
    state makingCoffee [  
        on input "done" output "coffee served. Enjoy!" and go to ^initial  
        on input "break" output "machine is broken!" and go to failure  
    ]  
]
GRAPHICAL SYNTAX
creation of a graphical syntax

Steps
  Define graphical elements
  Develop editing tool
  Implement mapping to meta-model

Typical questions
  Where do we store layout information?
  Do we support partial views?
  Auto layouts?

Graphical Editing Framework (GEF)
  Eclipse Plugin
  Framework for Diagram Editors
  used in IBM Rational Software Architect and Borland Together
Example: Mapping an end state…

![Diagram of a model with layers labeled Oval, Stack Layout, Label, and End, leading to an end state label]
graphical syntax

http://www.eclipse.org/sirius/gallery.html
projectional language workbenchs for

LANGUAGE-ORIENTED PROGRAMMING
projectional editing

parser-based editing

projectional editing
(a.k.a., syntax-directed editing, structured editing)

no grammars or parsers involved

Concrete Syntax

Abstract Syntax Tree

Concrete Syntax

Abstract Syntax Tree
advantage: language composition

separate files

L2 \rightarrow L1

type system
transformation
constraints

in one file

type system
transformation
constraints
syntax
IDE
advantage: flexible notations

regular code/text

mathematical

tables

graphical
advantage: flexible notations

regular code/text

```c
// [ A documentation comment with references ]
to @arg(data) and @arg(dataLen)
void aSummingFunction(int8[] data, int8 dataLen) {
    int16 sum;
    for (int8 i = 0; i < dataLen; i++) {
        sum += data[i];
    }
} aSummingFunction (function)
```

mathematical

```latex
double midnight2(int32 a, int32 b, int32 c) {
    \[ -b + \sqrt{b^2 - \sum_{i=1}^{4} a_i c_i} \]
    \[ \frac{2}{a} \]
} midnight2 (function)
```

tables

```c
int16 decide(int8 spd, int8 alt) {
    return spd > 0 spd > 100 otherwise 0;
    | alt < 0 | 1 1 |
    | alt == 0 | 10 20 |
    | alt > 0 | 30 40 |
    | alt > 100 | 50 60 |
} decide (function)
```

graphical

```
Cst.Customer
    cust 1

Contract
    starts: date
    ends: date

Tariff
    attributes
    trf 1
```
Jetbrains Meta Programming System (MPS)

MPS is a language workbench
(a tool for defining, composing, and using languages)
MPS language workbench
MPS language ecosystems

- 5+ base languages
- 50+ extensions to C
- 10+ extensions to requirements lang.

slide credits: Markus Völter
in-depth?

Kursplan för DAT240 - Model-driven engineering

Model-driven engineering

Kursplanen fastställd 2013-02-03

Årskurs: MSOSF

7.8 Hägersten

Svårgrenader: TH, FM, Fys, Tre, Undersöknad

Utbildningsnivå: Master, informationsprogramkomp. och Teknik

Huvudområde: Datavetenskap och informationsstegy

Institution: TH, CMK

Undervisningsmetod: Projektsarbete

Sätt för utvärdering: 1a

Max antal deltagare: 30

Innehåll:

Kursmål

C110 Värdetill, 3,0 hp

C119 Present 4,0 hp

P410 Projekt 4,0 hp

P411 Projekt 4,0 hp

P412 Projekt 4,0 hp

Examen datum

20 Maj 2013

Sedan februari

(Nedanstöpt med kurs 2 lekär

Program

MSOSF SOFTWARE ENGINEERING AND TECHNOLOGY - UTSEKUNG OCH IMPLEMENTERING AV HJÄRVARA, MASTERPROGRAM, ÅKURS 1 (OBLIGATORISK VAL),

Eksamenskriterier:

Thorsten Berger

Kursansvarig: Thorsten Berger

Behörigheter:

För kurser på avancerat nivå gäller samma grundläggande och ansvarsfulla behörigheter som till det kursgivande programmet. (Vilken kursen på avancerad nivå minskats

Sedan februari

Curskemiska lernet

Begränsningar:

Sedan februari

Många kurskemiska

(C) Thorsten Berger, Chalmers | University of Gothenburg

73
questions?

Domain-Specific Languages

Thorsten Berger, Assoc. Prof.
<thorsten.berger@chalmers.se>