The State of the Art in External DSLs

WGT 2011

Markus Voelter
Independent/itemis
voelter@acm.org

1 A little History
2 DSLs
3 Example: Fountains
4 Programming vs. Modeling
5 Projectional Editing
6 Example: Embedded Development
A little History

programming
started
close to the hardware

abstractions
¬computing

chips
abstractions
~computing?
Java

abstractions
~computing?
SQL
general purpose

domain specific
tailor made
effective++
specialized, limited
used by experts
together with other specialized tools

Domain Specific Languages
A DSL is a **focussed, processable language** for describing a specific **concern** when building a system in a specific **domain**. The **abstractions** and **notations** used are natural/suitable for the **stakeholders** who specify that particular concern.
DSL Program (aka Model)

map

automated!

GPL Program

map

Generation
Transformation
Compilation

Interpretation
Activities

Analysing Domains
Defining Languages
Adapting/Selecting
Building Editors
Transforming Models
Building Generators
Building Frameworks
Analysing Domains
Defining Languages
Adapting/Selecting
Building Editors
Transforming Models
Building Generators
Building Frameworks
... and using all of that to build apps

Example 1:
Fountain Control
Hardware Structure

```
feature BasicOnePump
    pump compartment ccl
    static compressor c1
feature AtLeastOneZone extends BasicOnePump
    water compartment comp1
    pumped by c1
    compartment levelsensor ct_f1
    light l_f1
feature[f] SuperPowerCompartment
    water compartment adds to f
    superPowerMode
feature WithAlarm
    level alarm a1
fountain StdFountain extends AtLeastOneZone
```
 Behaviour

```c
pumping program P1 for AtLeastOneDone + WithAlarm + 
       SuperPowerCompartment[f=comp1] { 
    parameter defaultWaterLevel : int
    parameter superWaterLevel: int
    event superPowerTimeout

    init { 
        set comp1->targetHeight = defaultWaterLevel
    }

    start: 
        on (comp1->needsPower == true) && !(comp1->isPumping) { 
            do comp1->pumpOn
        }
        on comp1->enough { 
            do comp1->pumpOff
        }
        on comp1.superPumping->turnedOn { 
            set comp1->targetHeight = superWaterLevel
            raise event superPowerTimeout after 20
        }
        on comp1.superPumping->turnedOff or superPowerTimeout { 
            set comp1->targetHeight = defaultWaterLevel
        }
    }
```

Plus:

In-IDE Simulator
Unit Test Support
Eclipse Modeling
Eclipse Xtext

Programming vs. Modeling
Different Worlds
Programming Tools ≠ Modeling Tools
Different Worlds
Mix Models and Programs
AST Navigation & Query
Code Constraints

Why
the difference?
History?

Modeling

... (Mostly) Textual Notations
... Concrete Syntax
... Storage
... (Fancy) ASCII Editors
... Read-Only Visualizations

Programming
### Modeling

- (Mostly) Graphical Notations
- Abstract Syntax Storage
- Projecting Editors
- Different editable views for model

### Programming

- (Mostly) Textual Notations
- Concrete Syntax Storage
- (Fancy) ASCII Editors
- Read-Only Visualizations

---

**Why the difference?**
It is time for ...

... a Different Perspective
Programming the way we do Modeling?

Modeling the way we do Programming?

Modeling = Programming
Programming = Modeling
We don’t want to model, we want to program!

... at different levels of abstraction
... from different viewpoints
... integrated!
We don’t want to model, we want to
program!

... with different degrees of
domain-specificity
... with suitable notations
... with suitable expressiveness

And always:
precise and tool processable
Programming Languages are not MODULAR enough.

Programming Languages are not COMPOSABLE enough.
Programming Languages are not CONFIGURABLE enough.

Programming Languages are not ADAPTABLE enough.
Programming Language Syntax is not flexible enough.

Big Language?

with many first class concepts!
Small Language?

with a **few, orthogonal** and **powerful** concepts

Modular Language

with many **optional, composable** concepts
Modular Language

Like frameworks and libraries, but with syntax and IDE support

Projectional Editing
Parser-based

**text**

... to tree

... to text

Projectional

**tree**

... to text-lookalike (editor)

... to other trees ... [*]

... to text
Programming as Modeling

... (Mostly) Graphical Notations
... Abstract Syntax Storage
... Projecting Editors
... Different editable views for model

Programming as Modeling

... (Mostly) Graphical Any kind of Notations
... Abstract Syntax Storage
... Projecting Editors
... Different editable views for model
Language Composition

There’s no parsing.
Unique Language Element Identity.
Unlimited language composition.

Flexible Notations

Textual like ASCII
Graphical box & line
Semi-Graphical mathematical

treated the same can be mixed
Automatic IDE Extension

Tool support is inherent for languages build with projectional tools.

Language definition implies IDE definition.

Multiple Notations

... for the same concepts

e.g. in different contexts or for different tasks
Partial Projections

... different views
... for different roles/people
... only a particular variant

Live Programs

think: spreadsheet

a change to one part of program can lead to (dependent) changes in other parts
Tree Editing

... is different from editing text
... try to make it feel like text
... takes some getting used to

but: for more flexible notations
a more general editing paradigm
is needed

Infrastructure Integration

... storage is not text
... diff/merge must be in tool
... existing text tools don’t work
Proprietary Tools

... no standards

... no interop
Example 2: Embedded Systems
EXAMPLE CASE

SOLUTION

```python
# This module represents the code for the line follower Lego robot. It has a couple
# modules main imports CashKernel, KAUT, BitLevelUtilties

constant int WHITE = 500;
constant int BLACK = 700;
constant int SLOW = 20;
constant int FAST = 40;

def StateMachine to manage the:
stateMachine linefollower:
    event initialized:
    initial state initializing []
    initialized [true] -> running
    state running []

initialize

robot_set_light_sensor_actu
event linefollower:initialized

```
exported interface MotorControl {
    void stop( );
    void setLeftSpeed( int8 speed );
    void setRightSpeed( int8 speed );
}

exported component Motors {
    provides motorControl : MotorControl;
}

exported component implementation MotorsNXT : Motors {

    procedure void motorControl.stop( ) {
        nxt_motor_set_speed(MOTOR_PORT_V::NXT_PORT_B, 0, 1);
        nxt_motor_set_speed(MOTOR_PORT_V::NXT_PORT_C, 0, 1);
    }

    procedure void motorControl.setLeftSpeed( int8 speed ) {
        nxt_motor_set_speed(MOTOR_PORT_V::NXT_PORT_C, speed, 1);
    }

    procedure void motorControl.setRightSpeed( int8 speed ) {
        nxt_motor_set_speed(MOTOR_PORT_V::NXT_PORT_B, speed, 1);
    }

}

module impl imports <<imports>> {

    int speed( int val ) {
        return 2 * val;
    }

    robot script stopAndGo {
        block main on bump block retreat on bump <no bumpReaction>
            step
                accelerate to 0 - 30 within 2000
                drive on for 2000
                decelerate to 0 within 1000
                step
                accelerate to speed(25) within 3000
                drive on for 2000
                turn left for 2000
                block driveMore on bump <no bumpReaction>
                    accelerate to 80 within 2000
                    turn right for 3000
                    decelerate to 0 within 3000
                    step
                }

}
JetBrains MPS

coordinates
web www.voelter.de
e-mail voelter@acm.org
twitter markusvoelter
xing http://www.xing.com/profile/Markus_Voelter
linkedin http://www.linkedin.com/pub/0/377/a31

THE END.