Domain Specific Languages
Examples, Tools, Practices

Markus Voelter
independent/itemis

Introduction

Abstraction is a conceptual process by which higher, more conceptual concepts are derived from the usage and classification of literal (i.e. "real" or "concrete") concepts.

Wikipedia
Abstractions may be formed by reducing the information content of a concept or an observable phenomenon, typically to retain only information which is relevant for a particular purpose.

Wikipedia

[...] to understand and solve problems and communicate their solutions with the computer in some particular computer language. It allows programmers to separate concepts from implementation, so that they do not depend on software or hardware.

Wikipedia

One Language to Rule Them All?

What is a good Language? ... for a given purpose?
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)

GPL Program

automated!

Generation
Transformation
Compilation

Interpretation

Analysing Domains
Defining Languages
Adapting/Selecting
Building Editors
Transforming Models
Building Generators
Building Frameworks
Activities

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

internal vs. external

customization vs. configuration

graphical vs. textual
Example 1: Embedded Protocol Handler

Examples

Component Specification

Message Format Definition

Testing
Example 2:
Pension Fund Specification

Example 4:
Alarm System Menus
Menu Structure

Software Components

Example 5: OSGi-based System

Component Specification

Test Case Modeling
Example 8:

Miscellaneous

A DSL Hearing Aid Configuration

A DSL for Refrigerator Configuration

SOLUTION

BPEL Designer
Block Diagrams

PLC Programming

State Charts

Domain Specific Languages
Examples, Tools, Practices

Concepts & Terms

Metamodel
(Abstract Syntax)
Metamodel (Abstract Syntax)
Concrete Syntax (Notation)

Static Semantics (Constraints)
Dynamic Semantics

Domain Specific Languages
Examples, Tools, Practices

Benefits
Automation
faster, deterministic

Increased Quality
well defined structures all through the system

Meaningful Validation
more semantics in the model

Capture
Domain Knowledge
formalized into languages and models

Suitable Notations
textual, graphical, tabular

Technology Independence
generate "technology glue code"
Abstraction w/o Runtime Overhead

Capture Implementation Strategy

in the generators

Domain Specific Languages
Examples, Tools, Practices

Tooling

Tooling!

Tooling!

Editor
Tooling!
Editor, Debugger

Tooling!
Editor, Debugger, Testing

Tooling!
Editor, Debugger, Testing, Groupware

Tooling!
Editor, Debugger, Testing, Groupware, Scalable

Tooling!
Editor, Debugger, Testing, Groupware, Scalable, „All in Eclipse“
Tools Tooling

Language Definition Tools
abstract syntax, concrete syntax, constraints

Editor Frameworks
Transformation Languages
Code Generation Tools

Domain Specific Languages
Examples, Tools, Practices

Tooling: Eclipse Xtext

Xtext
Building Textual Editors
Constraints
expressed in Java -or- with OCL and dialects

Xtext Typesystem F/W
Typing Rules For Expr's

M2M
Model-to-Model Transformations
INRIA's ATL QVT Xtend

M2T
Model-to-Text Transformations
JET: Java Emitter Templates Xpand: oAW’s template engine

GMF
Graphical Box/Line editors based on EMF

EMF
Ecore meta meta model +
Editing Transactions Validation Query Distribution/Persistence
DEMO

Building DSLs with Eclipse Xtext

Domain Specific Languages
Examples, Tools, Practices

Programming vs. Modeling
programming? vs. modelling?

different? the same?

Different Worlds
Programming Tools $\neq$ Modeling Tools

Different Worlds
Modeling Tool $\neq$ Modeling Tool

Different Worlds
Mix Models and Programs
Different Worlds
Mix Models and Programs
AST Navigation & Query

Different Worlds
Mix Models and Programs
AST Navigation & Query
Code Constraints

Modeling

Programming

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

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

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

... a Different Perspective

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**

**Domain Specific Languages**

Examples, Tools, Practices

**Modular Languages**

**Big Language?**

with **many** first class concepts!

**Small Language?**

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

Like frameworks and libraries,

with many optional, composable concepts

Modular Language

Like frameworks and libraries,

but with syntax and IDE support

Viewpoints

suitable abstractions and notations for each

Viewpoints

Integrated via symbolic references and seamless transitions
**Viewpoints**

**General Purpose**
- predefined library
- configure

**Domain Specific**
- custom purpose-built
- create/include

**Custom Notations**
- real business expert integration

**LEGO Robot Control Components**
- State Machines
- Sensor Access
- LEGO Robot Control
Not a new idea...

Growing A Language
(Guy L. Steele)

Growing A Language
(Guy L. Steele)
stoneage-OOPSLA talk,
get it on Google Video!

Language Workbench
(Martin Fowler)

Language Workbench
(Martin Fowler)
Freely define languages and integrate them
Language Workbench
(Martin Fowler)

use persistent abstract representation

language ::= schema + editors + generators

Language Workbench
(Martin Fowler)

projectional editing

Language Workbench
(Martin Fowler)

persist incomplete or contradictory information

Language Workbench
(Martin Fowler)

powerful editing testing refactoring debugging groupware

Language Workbench
(Martin Fowler)

support for "classical" programming "classical" and modeling

Language Workbench
(Martin Fowler)

language definition implies IDE definition
Not a new idea…

Not a new idea…

But

NOW!

a practical reality :-)

Domain Specific Languages
Examples, Tools, Practices

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

Language Composition

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

Flexible Notations

Textual
like ASCII

Graphical
box & line

Semi-Graphical
mathematical

Automatic IDE Extension

Tool support is inherent for languages build with projectional tools

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

Storage != Schema

... store arbitrary meta data
change log
conflicting information
variability annotations
... independent of language schema!
... „aspects“, overlay

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
Domain Specific Languages
Examples, Tools, Practices

Tooling: MPS

also do...

IntelliJ IDEA
Resharper

released in
Q3 2009
currently
1.5
April 2011
2.0

Open Source
licensed under
Apache 2.0

& me
& JetBrains
& itemis
Build new standalone DSLs
Build DSLs that reuse parts of other languages

(MPS comes with BaseLanguage)
extend base language

build DSLs that reuse parts of BaseLanguage
Domain Specific Languages
Examples, Tools, Practices

Demo Scenario

C is **good**
- fast
- flexible
- wide-spread
- multi-platform

C is **bad**
- limited abstractions
- not domain-specific
- „dangerous“ preprocessor
- not extensible

**Modeling**
compensates for C’s problems

**Modeling**
compensates for C’s problems, specifically
- good abstractions
- domain-specific
- extensible

**Modeling**
compensates for C’s problems, but:
- not extensible
- tool/tool integration
- tool/code integration
- process integration
Two Classes

(Yes, this is a slight simplification)

Plan Oriented
Top Down
Big Systems
Big Companies

Modeling

Grown
Bottom Up
Small/Medium Systems
Smaller Companies

C Code

Two Classes

... and good luck with mixing the two!

mbeddr.com
Modular Embedded Language

Incremental Extension
of C
Incremental Extension of Components
Tasks
State Machines
Physical Units
Special Data Types

Incremental Extension of syntactically and semantically integrated

Incremental Extension of extensible with domain-specific constructs (DSLs)

SOLUTION

```c
// This module represents the code for the line follower NIK robot. It has a small
module main imports (Continuous, Real, Int, Boolean) { }

constant int INIT = 0;
constant int FWD = 1;
constant int REV = 2;
constant int SNAP = 3;

state machine { }
    state (INIT) { }
    state (FWD) { }
    state (REV) { }
    state (SNAP) { }

    transitions { }
        event lightSensor: moveLightSensor; action { } state (FWD) { }
        event lightSensor: moveLightSensor; action { } state (SNAP) { }

    output { }
        state (SNAP) { }

    input { }
        state (SNAP) { }

    constants { }
        state (SNAP) { }
}

SOLUTION
```
module imports 

int speci int val [] 

return 2 * val;

robot script stopAndGo
block main on long block external on long -longReaction-
step
accelerate to 0 - 30 within 3000
drive on for 3000
accelerate to 0 within 1000
stop
accelerate to 300 - 60 within 3000
drive on for 3000
turn left for 3000
block drive on long -longReaction-
accelerate to 30 within 500
turn right for 3000
decelerate to 0 within 3000
step

SOLUTION

Imported Requirements

SOLUTION

Program Code with Annotations (green)

SOLUTION

Selecting from the Requirements

SOLUTION

Find Usages of Requirements

SOLUTION

over black aircraft

\[
\begin{align*}
v & : \text{double } [\text{m/s}] / \text{current aircraft speed} \\
A & : \text{double } [\text{m}] / \text{area of the wing} \\
\alpha & : \text{double } [\text{deg}] / \text{angle of attack} \\
c_w & : \text{double } [\text{deg}] / \text{Widerstandswinkel} \\
\alpha & : \text{double } [\text{deg}] / \text{Number of Wings} \\
\end{align*}
\]
Extending Languages With MPS

Domain Specific Languages
Examples, Tools, Practices

Tool: Spoofax
Traditionally:
Language ::= Token* | Rule*;

Traditionally:
Language ::= Token* | Rule*;
Token ::= Character*;

Traditionally:
Language ::= Token* | Rule*;
Token ::= Character*;
Rule ::= Token* | Rule;

Traditionally:
Language ::= Token* | Rule*;

2 Phases:
1. Global Tokenization (aka Scanning)
2. Grammar Recognition (aka Parsing)

What if!
combine languages
same seq<Char> as
different tokens?

Ambiguous!
Not Parsable.
Bad Luck.
Modern:
Language ::= Rule*;
Rule ::= Char* | Rule*;

1 Phase:
Grammar Recognition directly on char*

Scannerless Parsing
No Ambiguity when combining languages
Always Parsable.
(well, almost. There are some corner cases...)

Custom Syntax
IDE Support
Graphical Textual Symbolic++
Teamwork Debugging Custom Editors

Complete Symbolic Integration
Limited to Unicode
Goto Def Find Refs Refactoring
how to handle non-character symbols
Graphics != Text

two worlds...
separate editors
... per syntax/viewpoint
... models can still be ref integrated

SDF: Declarative Syntax Definition

Stratego: Rewriting Language
Recap: What did we do?

Domain Specific Languages
Examples, Tools, Practices

Abstraction

Make Explicit

Formalize

Notation
Notation, Notation, Notation

Graphical vs. Textual

DSL Semantics

Viewpoints

Partitioning

Evolution
The Fallacy of Generic Languages

Learn from 3GLs

Tooling Matters!

Interpretation vs. Generation

Rich Domain Specific Platform

Integrating Generated and Manually Written Code
Model-2-Model To Simplify Generators

Don’t forget Testing

Iterate!

Documentation is still necessary

Domain Users Programming?