Domain Specific - a Binary Decision?

Bernhard Merkle
SICK AG
bernhard.merkle@gmail.com

Markus Voelter
Independent/itemis
voelter@acm.org
Context & Problem
Viewpoints suitable abstractions and notations for each
Viewpoints

Integrated
via
symbolic
references
and
seamless
transitions
Viewpoints

Domain Specific

custom
purpose-built

create/include
Viewpoints

Domain Specific

Custom Notations

real business expert integration
Viewpoints

General Purpose

predefined library

configure
Viewpoints

General Purpose

Domain Specific

LEGO Robot Control
Viewpoints

General Purpose

Components

State Machines

Sensor Access

Domain Specific

LEGO Robot Control
Different Worlds
Programming Tools ≠ Modeling Tools
Different Worlds

Modeling Tool

!=

Modeling Tool
From a given set of configuration options you select a subset. Constraints between configuration options limits valid combinations.
You define a language that can be used to define a basically an unlimited number of variants.

You then define a sentence in that language that describes a particular variant.
Configuration

Feature Models

Construction

Continuum

Domain Specific Languages
We don’t want to model, program!
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
We don’t want to model, we want to program!

And always: precise and tool processable
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,
Modular Language

Like frameworks and libraries, but with syntax and IDE support
3 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

... 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
released in
Q3 2009

currently
1.5
Open Source under

Apache 2.0
Example Scenario
Two Classes in Embedded Systems Development

(Yes, this is a slight simplification)
Incremental Extension of C
Incremental Extension of Components

Tasks

State Machines

Physical Units

Special Data Types
Incremental Extension of syntactically and semantically integrated C
Incremental Extension of extensible with domain-specific constructs (DSLs)
Demo
The End.

Bernhard Merkle  
SICK AG  
bernhard.merkle@gmail.com

Markus Voelter  
Independent/itemis  
voelter@acm.org
Language Extension and Composition with Language Workbenches

The End.