Programming and Modeling
Two Worlds?
Markus Voelter
Independent/itemis
voelter@acm.org
Programming Languages are not expressive enough.

Programming Languages are not high-level enough.
Programming Languages are not abstract enough.

Programming Languages are not domain-specific enough.

Programming Languages are not enough.

Programming Languages Formats

Programming Languages Frameworks are not enough.
So?

Modeling
... Higher Level
... Domain Specific Concepts & Notations
... Code Generation Interpretation
Solves the Problem!

DSL

general purpose

domain specific

tailor made
effective++
specialized, limited
used by experts
together with other specialized tools
But:

Different Worlds
Programming Tools ≠ Modeling Tools

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?
History?

Why the difference?

It is time for ...

... a Different Perspective

Modeling = Programming
Programming = Modeling

modeling == programming
modeling == programming

... at different levels of abstraction
... from different viewpoints
... integrated!

modeling == programming

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

modeling == programming

And always:
precise and tool processable

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

language ::= schema + editors + generators

Language Workbench (Martin Fowler)

use persistent abstract representation

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 and "classical" modeling

Language Workbench (Martin Fowler)

language definition implies IDE definition
Syntax primarily textual with more symbols think: mathematics

When a graphical notation is better, you can visualize.

Syntax primarily textual

Syntax primarily textual sometimes box&line style
Available Tooling

Eclipse
Xtext

Modeling as Programming

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

Custom Syntax

Graphical
Textual
Symbolic++

IDE Support

Teamwork
Debugging
Custom Editors

Complete Symbolic Integration

Goto Def
Find Refs
Refactoring

Infrastructure Integration

... storage not text
... diff/merge with existing tools
... existing text work well!
Language Composition

Grammar composition with traditional parsers is tough!
More advanced parsers currently research

Limited to Unicode

how to handle non-character symbols

Graphics != Text

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

http://eclipse.org/modeling
http://eclipse.org/xtext
Xtext: Specify Grammar

Xtext: Generated Editor

Xtext: Generated Editor

Xtext: Generated Editor

Xtext: Generated Editor

Xtext: Generated Editor

Xtext: Generated Editor
Code Folding

Goto Definition
Find References
Cross-File References
Model as EMF

DEMO I
Building DSLs with Eclipse Xtext

Available Tooling
Jetbrains’ Meta Programming System
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.

Multiple Notations

... for the same concepts, e.g. in different contexts or for different tasks.

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.
Apache 2.0 licensed under Apache 2.0 released in Q2 2009 currently 1.1 RC1

Build new standalone DSLs
Build DSLs that reuse parts of other languages

Build DSLs that reuse parts of other languages

Build DSLs that reuse parts of other languages

Build new standalone DSLs
Build DSLs that reuse parts of other languages

(MPS comes with BaseLanguage)
extend base language

Language Extension Example

Java++

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
Java

```java
ReadWriteLock l = ...
l.readLock().lock();
try {
    //code
} finally {
    l.readLock().unlock();
}
```

Java + Extension

```java
ReadWriteLock l = ...
lock (l) {
    //code
}
```

Language Extension Example

Old

```java
ReadWriteLock l = ...
l.readLock().lock();
try {
    //code
} finally {
    l.readLock().unlock();
}
```

New

```java
ReadWriteLock l = ...
lock (l) {
    //code
}
```

Structure

- Editor
- Typesystem
- Generator

Structure

- Editor
- Typesystem
- Generator
Language Extension Example
Result behaves like a native base language construct

Language Extension Example
Translated to regular Java code based on the generator

DEMO II
Building DSLs with JetBrains MPS
A vision for Programming

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.
with many first class concepts!

Small Language?
with a few, orthogonal and powerful concepts

Modular Language
like frameworks and libraries,

Modular Language
like frameworks and libraries,
but with syntax and IDE support
Not a new idea...

Growing A Language

(Guy L. Steele)
Example Languages
Adding matrices to C in an embedded environment.

(Seemingly) Simple Example

Currently:

1. Declare Data Structures in XML
2. Generate Headers
3. Implement manually in C

Currently:

Matrices not supported in XML format and generator

Currently:

**Tool team**

- would have to do the extension
- a lot of work
- busy
- one central tool

Currently:

No real compiler support in the resulting C code
- type checks
- operator overloading
- generics (matrix<int>)
- matrix syntax?
Better Solution

```
# Example code

from BetterSolution import *

# Adjacency matrix
adj_matrix = [[0, 1, 0], [1, 0, 1], [0, 1, 0]]

# Vector
vec = [1, 2, 3]

# Matrix operations
result = vec @ adj_matrix
```

Better Solution

- **generic matrix and vector types**
- **real matrix and vector literals**
- **syntax highlights for vectors and matrices**
- **operator overloading**
Better Solution

```cpp
qmatrsvextlb[LxL] MModelMatr = pow + rmatrsgn + e Q == 0

vector*entlsb(const dim) modelentlsb = temp;

vector*entlsb gausserfolgel(qmatrsvextlb), vector*entlsb i { }
  // gausserfolgel... A * A + b = x * A + b
  return f;

vector*entlsb = MModelMatr * modelentlsb;
```

Better Solution

```cpp
qmatrsvextlb[LxL] MModelMatr = pow + rmatrsgn + e Q == 0

vector*entlsb(const dim) modelentlsb = temp;

vector*entlsb gausserfolgel(qmatrsvextlb), vector*entlsb i { }
  // gausserfolgel... A * A + b = x * A + b
  return f;

vector*entlsb = MModelMatr * modelentlsb;
```

Better Solution

```cpp
qmatrsvextlb[LxL] MModelMatr = pow + rmatrsgn + e Q == 0

vector*entlsb(const dim) modelentlsb = temp;

vector*entlsb gausserfolgel(qmatrsvextlb), vector*entlsb i { }
  // gausserfolgel... A * A + b = x * A + b
  return f;

vector*entlsb = MModelMatr * modelentlsb;
```

Available Tooling

Intentional Software Domain Workbench
Commercial Product
Eval available upon request

Pension Workbench Example
Text Editing Domain

Pension Workbench Example
Insurance Mathematics Domain

Pension Workbench Example
Pension Contract Rules Domain

Pension Workbench Example
All in one Document
Pension Workbench Example

Symbolically integrated

THE END.

THE END.