Variability in PLE

... differences among products in PL
Variability Mechanisms
Removal
... optionally take away from overall whole

Variability Mechanisms
Injection
... optionally add to

Variability Mechanisms
Parametrization
... define values for predefined params

Configuration vs. Customization
Variability

Configuration
... selecting options ... setting param values

Configuration Feature Models
Customization

... „real languages“
... instantiation
... connections

Languages

Customization

PLE is also about...

Process
Organization
People
Product Mgt.

not today!
Domain Specific Languages

Focus of this talk!

programming started close to the hardware

abstractions ~ computing

abstractions ~ computing
abstractions computing?

Java

abstractions computing?

SQL

?
general purpose

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

effectiveness?

map
As you understand and develop your Platform Architecture...

Develop a language to express it!
Language resembles architectural concepts...

Express the applications with the language.

Another Story...
Architecture As Language
Modeling and Programming

Compare

Modeling

Programming

Flexible!

Domain Specific Notations and Abstractions

Limited!

Frameworks
Libraries
(Fluent) APIs
Graphical Textual Forms Tables
Flexible! Limited!
Textual Trees

Customize Generator or Interpreter
Flexible! Limited!
Reflection Meta Programs Open Compilers

Define custom Query or Navigate or Transform
Flexible! Limited!
AST APIs Static Analysis Regex

Custom Validation or Error Checks
Flexible! Limited!
IDE plugins Static Analysis Open Compilers

Different Representations and Projections
Flexible! Limited!
Text is Text Code Folding Tree Views Visualizations

Mixing and Composing Languages
Flexible? Limited!
Python-to-C-like Internal DSLs Embed-As-String Specific: LINQ
Mature!Brittle!ScalableUsableIDE Support

TestingDebugging

UsableBrittle!

Mature!

Modeling Tools...!?Brittle!

Mature!

for some tools...

BranchingVersioning

Diff, Merge

Jobs done.

some people doubt that... everybody agrees...

Gets theJob Done!

Different WorldsProgramming Tools != Modeling Tools
Different Worlds
Modeling Tool
≠ Modeling Tool

Different Worlds
Mix Models and Programs
AST Navigation & Query

Different Worlds
Mix Models and Programs
AST Navigation & Query
Integration of 3GL code

Different Worlds
Mix Models and Programs
AST Navigation & Query
Integration of 3GL code
Code Constraints
Why the difference?

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 Programming == Modeling

We don’t want to model, we want to 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

Language Workbench
(Martin Fowler)
Language Workbench (Martin Fowler)

Freely define languages and integrate them.

Freely define persistent abstract representation?

Language Workbench (Martin Fowler)

language ::= schema + editors + generators

Language Workbench (Martin Fowler)

use projectional editing?

Language Workbench (Martin Fowler)

persist incomplete or contradictory information

Language Workbench (Martin Fowler)

use powerful editing, testing, refactoring, debugging, groupware

Language Workbench (Martin Fowler)

language definition implies IDE definition
Language Workbench
(Martin Fowler)
support for "classical" programming and modeling

Syntax primarily textual

Syntax primarily textual
with more symbols
think: mathematics

Syntax primarily textual
sometimes box&line style

Syntax primarily textual
sophisticated visualizations
Modeling as Programming

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

Parser-based
text
... to tree
... to text

Custom Syntax
Graphical Textual Symbolic++

IDE Support
Teamwork Debugging Custom Editors
Complete Symbolic Integration
- Goto Def
- Find Refs
- Refactoring

Integrates with Current Dev Infrastructure
- Version Mgt
- Diff/Merge
- Cmd Line Tools

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
Eclipse Xtext
Building Textual Editors

Xtext: Specify Grammar

Xtext: Gen. Meta Model

Xtext: Constraints
Xtext: Generated Editor

- Code Completion
- Syntax Coloring
- Custom Keyword Coloring
- Realtime Constraint Validation
- Customizable Outlines
- Code Folding
Building DSLs with Eclipse Xtext
Programming as Modeling

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

Projectional

tree
... to text-lookalike (editor)
... to other trees ... [*]
... to text

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 built 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

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
useful e.g. for tests running in the workbench

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 out-of-the-box interop

Jetbrains’ Meta Programming System

Build new standalone DSLs

released in:
Q3 2009
licensed under:
Apache 2.0
Build new **standalone** DSLs
Build DSLs that **reuse** parts of other languages

(MPS comes with **BaseLanguage**) **extend** base language

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**

Language Extension Example

Old

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

New

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

Language Extension Example

Old

```java
lock(l) {
    //code
}
```

New

```java
lock(l) {
    //code
}
```
Language Extension Example

Result behaves like a native base language construct

Translated to regular Java code based on the generator

```java
package jaxdemo.sandbox.sandbox;
import java.util.concurrent.locks.Lock;
public class DemoClass {
    private Lock lock;
    public DemoClass() {
        try {
            this.getLock().lock();
            SharedResouce.instance().doSomething();
        } finally {
            this.getLock().unlock();
        }
    }
    private Lock getLock() {
        return this.lock;
    }
}
```
DEMO II

Extending Java with JetBrains MPS

Example Languages

UI Language

Example Languages

HTML Templates

Example Languages

Persistent Classes
Pension Workbench Example

Insurance Mathematics Domain

Pension Rules Domain w/ tests

Pension Workbench Example

All in one Document

Pension Workbench Example

Symbolically integrated

Expressing Configurability
Configuration on model level
- fewer variation points
- less complex
- "expanded" via generator

More abstract
- less detailed

Implementation Artefacts
- less abstract
- more detailed

Transformation
- Variation Point

Configuration on model level
- fewer variation points
- less complex
- "expanded" via generator

Transformation
- Variation Point

Negative Variability:
Conditionally taking something away

Feature Models
component DelayCalculator {
  provides default: IDelayCalculator
  requires screens[0..n]: IInfoScreen
  provides mon: IMonitoring feature monitoring
}

namespace monitoringStuff feature monitoring {
  component MonitoringConsole {
    requires devices[*]: IMonitor
  }
  instance monitor: MonitoringConsole
  dynamic connect monitor.devices query {
    type = IMonitor
  }
}

Positive Variability:
Conditionally adding something to a minimal core

Aspects
namespace monitoring {
    component MonitoringConsole {
        instance monitor: ...
        dynamic connect monitor.devices ...
    }
    aspect (*) component {
        provides mon: IMonitoring
    }
}

component DelayCalculator {
    ...
}
component AircraftModule {
    ...
}
component InfoScreen {
    ...
}

component DelayCalculator {
    ...
} provides mon: IMonitoring
component AircraftModule {
    ...
} provides mon: IMonitoring
component InfoScreen {
    ...
} provides mon: IMonitoring

aspect (*) <type>
all instances of type
aspect (tag=bla) <type>
all instances with tag bla
aspect (name=S*) <type>
all instances whose name starts with S

Weaver is **generic**: works with all (container) model elements

AO + Features
Adding Variability and connectivity to a feature model to an Xtext DSL

A statemachine

Actually, several...!

Variant for Pedestrians
Variant for Cars

DEMO IV

Expressing Variability with MPS

Facilities to express Variability are completely independent of the target language!

Facilities to express Variability are completely independent of the target language! which leads to...
Viewpoints
Business

custom
purpose-built
create/include

Viewpoints
Business

Custom
Notations
real
business
expert
integration

Viewpoints
Technical

predefined
library
configure

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

Not a new idea...
Growing A Language
(Guy L. Steele)

Adding matrices to C in an embedded environment.

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

```c
qmabxint10b(x1) ModelMatrix = par * x_kappa * \theta \cdot ...

vector<int10b> modellvector =...#\variance

vector<op10b> gaussiextl = qmabxint10b \wedge \text{var} = \text{var}

// gaussiextl... \alpha \cdot \beta \cdot \gamma \cdot \delta ...

return \beta;

vector<int10b> modellvector =

int16 det(qmabxint10b A)

return \sum_{i=0}^{n}(\text{sym}(i) \prod_{j=0}^{n} a_{i,j});
```

Better Solution

```
qmabxint10b(x1) ModelMatrix = par * x_kappa * \theta \cdot ...

vector<int10b> modellvector =...#\variance

vector<op10b> gaussiextl = qmabxint10b \wedge \text{var} = \text{var}

// gaussiextl... \alpha \cdot \beta \cdot \gamma \cdot \delta ...

return \beta;

vector<int10b> modellvector =

int16 det(qmabxint10b A)

return \sum_{i=0}^{n}(\text{sym}(i) \prod_{j=0}^{n} a_{i,j});
```

**Better Solution**

**generic matrix and vector types**

**Better Solution**

**real matrix and vector literals**
Better Solution

syntax highlights for vectors and matrices

Better Solution

operator overloading

Better Solution

operator overloading

Better Solution

math notation

Better Solution

a separate language module

used only by those who really need it
In addition: PLE Variability

```cpp
typedef int64_t elementType;

vector<elementType> tmp1;

elementType someOperation(
    vector<elementType> & tmp1,
    elementType x,
    elementType y
)
{
    // some code
    return 0;
}
```

In addition: PLE Variability

```cpp
typedef int64_t elementType;

vector<elementType> tmp1;

elementType someOperation(
    vector<elementType> & tmp1,
    elementType x,
    elementType y
)
{
    // some code
    return 0;
}
```

In addition: PLE Variability

```cpp
typedef int64_t elementType;

vector<elementType> tmp1;

elementType someOperation(
    vector<elementType> & tmp1,
    elementType x,
    elementType y
)
{
    // some code
    return 0;
}
```

In addition: PLE Variability

```cpp
typedef int64_t elementType;

vector<elementType> tmp1;

elementType someOperation(
    vector<elementType> & tmp1,
    elementType x,
    elementType y
)
{
    // some code
    return 0;
}
```
DEMO V

Language Modularity with MPS
Summary

Usefulness of DSLs
Usable tools ✓
Variability in Models ✓
Modular Languages ✓

Usefulness of DSLs
Usable tools ✓
Variability in Models ✓
Modular Languages ✓
Usefulness of DSLs ✓
Usable tools ✓
Variability in Models ✓
Modular Languages ✓