Using Domain Specific Languages for Product Line Engineering

SPLC 2009 Tutorial

Markus Voelter
Independent/Itemis
www.voelter.de
voelter@acm.org

Variability
Configuration
Customization
MDD Intro

MDD Tooling
Model Variability
Transformation Var.
Summary & Wrapup
Variability

... differences among products in PL
Variation Point

... a point where a variation can occur
... must be bound for each product
... bind when?
... bind how?

Binding Time

<table>
<thead>
<tr>
<th></th>
<th>flexibility</th>
<th>performance</th>
<th>code size</th>
<th>complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>source time</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>compile time</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>link time</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>load time</td>
<td>++</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>run time</td>
<td>+++</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
</tbody>
</table>
Variability Mechanisms

Removal

... optionally take away from overall whole

Challenge:
overall whole can get big and unwieldy
Variability Mechanisms
Injection

... optionally add to minimal core

Challenge:
how to point into the core and add something to it
**Variability Mechanisms**

**Parametrization**

... define values for predefined params

---

**Challenge:**

Types for parameters can be non-trivial (DSLs)
Configuration vs. Customization

Variability

Configuration

Customization

MDD Tooling

Model Variability

Transformation Var.

MDD Intro

Summary & Wrapup
Configuration

... selecting options
... setting param values

Feature Models
Robin DR-400
An aircraft with a low wing, piston engine and made of metal, wood and cloth.

Airbus A 320
An aircraft with low wing, jet engine(s) and made of metal.
Configuration Feature Models

Schleicher ASW 27
An aircraft with shoulder wing, no engine and made of plastic

Configuration Feature Models

...
Customization

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

Customization
Languages
Customization Languages

```java
Data {
  data Patient : 
    name : String
    birthDate : String
    address : String
    phone : String
    doctor : String
}

Data {
  data Order : 
    item : String
    quantity : Integer
    price : Float
}

Data {
  data Form : 
    field : String
    value : String
}
```

```java
records {
  record Patient {
    p : Patient
    1 : p.first
    2 : p.second
    3 : p.third
  }
  record Address {
    a : Address
    1 : a.street
    2 : a.city
    3 : a.zip
  }
  record Order {
    o : Order
    1 : o.item
    2 : o.quantity
    3 : o.price
  }
}
```
Customization
Languages

Variability
Configuration
Customization
MDD Tooling
Model Variability
Transformation Var.
MDD Intro
Summary & Wrapup
programming

started

close to the hardware

abstractions

∼

computing

chips

abstractions

∼

computing

bits
abstractions
\sim computing

Java
SQL abstractions computing?
general purpose

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

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)

automated!

GPL Program

Generation
Transformation
Compilation

Interpretation
Automation
faster, deterministic
Increased Quality
well defined structures allthrough 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

generator "optimizes away"
Capture
Implementation Strategy
in the generators

Everything is a model
including for example hardware (some) hardware
EMF

Ecore meta meta model

+ Editing
Transactions
Validation
Query
Distribution/Persistence

EMF

Metamodel As Tree
can also be edited as UML-like diagram
EMF

Constraints

with OCL and dialects

GMF

Graphical Box/Line editors based on EMF
TMF / Xtext
Building Textual Editors

TMF / Xtext
Building Textual Editors
M2M

Model-to-Model Transformations
INRIA’s ATL
QVT
Xtend

M2T

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

Model-to-Text Transformations
Extensions to modularize complex expressions

openArchitectureWare
One Stop Toolkit for DSLs + X
Version 5.0 is current
Lively ecosystem of tools and extensions
Proven track record in various domains & project contexts
Stable, productive and helpful developer, support and user communities

Integration with Eclipse:
  Part of Eclipse, Working Group
  Uses EMF as a basis
  Graphical editors based on GMF
  All editors and tooling based on Eclipse
Specify Grammar
Antlr Grammar and Parser is generated from this specification

Generated Metamodel
Specify Constraints

Generated Editor
Generated Editor

**Code Completion**

Generated Editor

**Syntax Coloring**

Custom Keyword Coloring
Generated Editor

Realtime Constraint Validation

Generated Editor

Customizable Outlines
Generated Editor

Code Folding

Generated Editor

Goto Definition
Find References
Cross-File References
Model as EMF
Generated Editor

Xtext Overview
Building a sample textual DSL and code generator for a simple domain using Eclipse TMF/openArchitectureWare
Another Tool...?

also do...

IntelliJ IDEA
Resharper

24.06.2009
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 DSLs that **reuse** parts of BaseLanguage
Language Extension Example

Old

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

New

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

Structure ◆ Editor ◆ Typesystem ◆ Generator
Language Extension Example

Result behaves like a native base language construct
Language Extension Example

Result behaves like a native base language construct

Language Extension Example

Translated to regular Java code based on the generator
Example Languages

UI Language

```
component MenusLanguage

column

label

help

label

help

label

help

label

help

end

include

include

include

include

end
```

Example Languages

HTML Templates

```
<html>
<head>

<title>Example Languages</title>

</head>

<body>

<h1>Example Languages</h1>

<h2>UI Language</h2>

<pre>
component MenusLanguage

column

label

help

label

help

label

help

label

help

end

include

include

include

include

end
</pre>

<h2>HTML Templates</h2>

```

<!-- Template Strings

String message, boolean hideHeader, boolean useLast:

is not refordable. Only one root element allowed for refordable template.

<% variables %>

root links: << root links >> << root template name >>

<% explicit js import %>

if (!hideHeader && !useLast) {

  open class: right-margin-space

  include visible

  "message" Overlink(more: message.url)!, bold: true

} /

open class: right-margin-space

include visible

"message": Data: message.created.

//

if (message.updated) {

  open class: right-margin-space

  include visible

  "message": Data: message.updated.

} /

</body>

</html>
```
Example Languages

Persistent Classes

```java
public persistent class Form extends <name> implements <name> {  
  features
  save changes history if: false
  save changes history callback: no callback
  version mismatch resolution: default
  invariant: no invariant

  static fields <<
  public simple string name opt;
  public unordered child 7Thread(0..n) threads opt;
  public unordered bidirectional association List[0..n] subscribers update(client);
  public unordered bidirectional association List[0..n] watchers update(client);

  public Form(string name, User creator) {
    this.name = name;
    this.watchers.add(creator);
  }

  << destructor >>
```
Two Levels

∼ problem space vs. software space

Problem Space:
Configuration

Software Space:
Customization
Two Levels Removal

```c
#if defined (ACE_HAS_TLI)
    static ssize_t t_snd_n (
        ACE_HANDLE handle,
        const void *buf,
        size_t len,
        int flags,
        ACE_Time_Value *timeout = 0,
        size_t *bytes_transferred = 0);
#endif /* ACE_HAS_TLI */
```

Model-Based Implementation

... customization in problem space
... Problem-Space DSL
Model-Based Implementation

Diagram showing the relationship between different components and their implementation artefacts.
MDSD - Thumbnail

Model → Transformation → Implementation Artefacts

more abstract
less detailed

less abstract
more detailed

MD-PLE - Thumbnail

Transformation

○ = Variation Point

fewer!
Two Levels Removal

```
component DelayCalculator {
    provides default: IDelayCalculator
    requires screens[0..n]: IInfoScreen
    provides mon: IMonitoring feature monitoring
}
```
Two Levels Removal

namespace monitoringStuff feature monitoring {

  component MonitoringConsole {
    requires devices: [*]: IMonitor
  }

  instance monitor: MonitoringConsole

  dynamic connect monitor.devices query {
    type = IMonitor
  }

}

Two Levels Removal
Two Levels Injection

```cpp
namespace monitoring {
    component MonitoringConsole {
        instance monitor; ...
        dynamic connect monitor.devices ...
    }

    aspect (*) component {
        provides mon: IMonitoring
    }
}
```

Two Levels Injection

```cpp
component DelayCalculator {
    ...
}
component AircraftModule {
    ...
}
component InfoScreen {
    ...
}

aspect (*) component {
    provides mon: IMonitoring
}
component DelayCalculator {
    ...
    provides mon: IMonitoring
}
component AircraftModule {
    ...
    provides mon: IMonitoring
}
component InfoScreen {
    ...
    provides mon: IMonitoring
}
```
Two Levels Removal Injection

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

Manual Code Variability

public class LightDriverImplementation extends LightDriverImplBase {
    @Override
    protected String getIdInternal() {
        return getConfigParamValueForId();
    }
    ..
    //~# dimmableLights
    @Override
    protected int setLightLevelInternal(int level) {
        state().setEffectiveLightLevel(level);
        return level;
    }
    //~# dimmableLights
}
Extending the sample DSL to include feature-based variability connected to an external feature model
MD-PLE – Thumbnail II

Transformation

○ = Variation Point

more options
Transformation Variability

create System transformPs2Cbd( Building building ):
  ~
  hasFeature("burglarAlarm") ? { handleBurglarAlarm() -> this } : this;

handleBurglarAlarm( System this ):
  let conf = createBurglarConfig(); {
    configurations.add( conf ) ->
      ...
      conf.connectors.add( connectSimToPanel( createSimulatorInstance(),
                                  createControlPanelInstance() ) ) ->
      hasFeature( "siren" ) ? conf.addAlarmDevice("AlarmSiren") : null ->
      hasFeature( "bell" ) ? conf.addAlarmDevice("AlarmBell") : null ->
      hasFeature( "light" ) ? conf.addAlarmDevice("AlarmLight") : null
};
Generator Variability

Introducing Variability into the Code Generator built before
Variability
Configuration
Customization
MDD Intro

MDD Tooling
Model Variability
Transformation Var.

Summary & Wrapup

DSLs can be used to effectively describe customization variab.

Transformation and Generation can be used to map PS to SS

Configuration and Customization can be sensibly combined

Various Tools are available,
http://eclipse.org/modeling
http://dslvariantmanagement.googlecode.com/
Using
Domain Specific Languages
for
Product Line Engineering

THE END.
Thank you.
Questions?

Markus Voelter
Independent/Itemis
www.voelter.de
voelter@acm.org

THE END.