Markus Voelter
Indepenent/itemis
voelter@acm.org
http://www.voelter.de

Using Domain Specific Languages in Product Line Engineering

Contents:
1 PLE Intro
2 Classifying Variability
3 Configuration
4 Customization
5 Binding Time
6 Architecture DSLs
7 Business Domain DSLs
8 Configuration for Customization
9 Variability in Generators
10 Summing Up
Systematically analyse, design and build families of products, not single systems.
Identify Commonalities

Manage Differences
This Talk

- Product Management/Requirements
- Architecture/Implementation
- Process
- Organization

Domain Eng‘g

... analyse domain
... build common artifacts
... define/build way to build the products
(i.e. build/setup tools)
App Eng‘g

... analyze product req’s
... use domain artifacts
... to build products
... evolve domain artifacts

Problem Space
- Domain
- Product Management
- Business

Software Space
- Realization
- Technology
- Architects/Developers
what

how

Automated!

what

how
Variation Point

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

Variability Mechanisms
Removal

... optionally take away from overall whole
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
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
Variability Mechanisms

**Parametrization**

... define values for predefined params

**Challenge:**

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

Variability

Guidance, Efficiency ↔ Complexity, Flexibility

Routine Configuration Ė Creative Construction

Configuration Parameters Ė Feature-Model Based Configuration Ė Graph-Like Languages

Property Files Ė Tabular Configurations

Wizards Ė Manual Programming

Configuration

… selecting options … setting param values
Customization

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

Configuration
Configuration

... selecting options
... setting param values

Configuration

Feature Models
Configuration

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
Configuration

Feature Models

Connector
  
Paradigm
  
Client/Server
  
  Synchronous
  
  Asynchronous
  
  Timeout
    [incomplete]
  
  Polling
    [incomplete]
  
  Blocking
    [incomplete]

Message-based
  
  Sender
    [incomplete]
  
  Receiver
  
  Pull
  
  Push
  
  Queued
    [incomplete]
  
  Non-queued

Technology
  
  CAN
  
  Local

Customization
Customization

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

Sidebar

Domain-Specific Languages
programming
started
close to the hardware

abstractions
~computing

chips

abstractions
~computing

bits
abstractions
≈ computing

abstractions
≈ computing?
abstractions
~computing?
general purpose
domain specific

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

map
End of Sidebar

Domain-Specific Languages

Customization Languages
5 Binding Times for Customization

### 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>
Binding Time for DSLs?
Interpretation vs. Generation

Binding Time Generation

- Text Editor
- Textual Source Code
- Parser
- AST (Abstract Syntax Tree)
- Graphical Editor
- Interpreted
- Code
- Generator
- Compiled
- Code

Interpreted code can be easily inspected.

Generation

resulting code can be easily inspected
Interpretation vs. Generation

Generation
resulting code can be easily debugged

Generation
resulting code can be optimized and more efficient
Generation

Templates can be derived from existing code

Generation

work around limitations of target language
Interpretation vs. Generation

**Generation**

no changes to target environment

(leaves no trace)

---

**Generation**

reuse runtime infrastructure

(garbage collection, monitoring...)

---
Interpretation vs. Generation

Interpretation
- faster turnaround
- no regeneration
- test
- build
- deploy

Interpretation
- for platform independence
- an interpreter might be less porting effort
Interpretation vs. Generation

**Combinations**

- **Generation**
  - Java
  - DSL

- **Interpretation**
  - Bytecode
  - XML

- **Interpretation**
  - Java VM

- **Generation**
  - JIT-Compiler

Platform

- **Generated Application**
  - Domain Frameworks
  - Libraries
  - Middleware
  - Drivers
  - Operating System
Rich Platform

Platform

Grown with the DSL!

Domain Frameworks
Libraries

Rich Platform

Platform
Extreme Case

Generated Application populates Domain Frameworks
Architecture

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

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

Express the applications with the language.
subsystem the.worid.occasions (  
    declare:  
        immutable type ProblemReport (  
            problem: string  
            severity: int  
        )
    )

    interface Radio (  
        import m.TileMap goofy (): void
    )

    interface Press (  
        broadcast: string | string  
    )

    private:
        component Source (  
            provide source: Press
        )

    )

subsystem the.woon.occasions (  
    uses the.worid.occasions  
    private:
        component Armstrong (  
            task: string describing msg|press|startup
            priority: Radio  
        )

)
http://eclipse.org/modeling

**xtext** Specify Grammar
Antlr **Grammar** and **Parser** is generated from this specification

---

**Gen. Metamodel**
xtext Constraints

```java
import org.eclipse.xtext.typescore.type((((((java.lang.Class.forName("com.xtext.editor.model:Extensions"))))).getPackage().getName()) + "::Extensions")

@Constraint(extend="com.xtext.editor.model:Extensions")
public class FlightCheck extends Extensions
{
  @Extension(extends="com.xtext.editor.model:Extensions")
  public class FlightRules
  {
    public static final String FLIGHTCHECKER = "FlightChecker";
    public static final String FLIGHTRULES = "FlightRules";
    public static final String FLIGHTCHECKER_RULES = "FlightCheckerRules";
    public static final String FLIGHTCHECKER_RULES_TYPE = "FlightCheckerRulesType";
    public static final String FLIGHTCHECKER_RULES_TYPE_NAME = "FlightCheckerRulesTypeName";
    public static final String FLIGHTCHECKER_RULES_TYPE_FIELD = "FlightCheckerRulesTypeField";
    public static final String FLIGHTCHECKER_RULES_TYPE_FIELD_NAME = "FlightCheckerRulesTypeFieldName";
    public static final String FLIGHTCHECKER_RULES_TYPE_FIELD_TYPE = "FlightCheckerRulesTypeFieldType";
    public static final String FLIGHTCHECKER_RULES_TYPE_FIELD_TYPE_NAME = "FlightCheckerRulesTypeFieldTypeName";
    public static final String FLIGHTCHECKER_RULES_TYPE_FIELD_TYPE_FIELD = "FlightCheckerRulesTypeFieldTypeField";
    public static final String FLIGHTCHECKER_RULES_TYPE_FIELD_TYPE_FIELD_NAME = "FlightCheckerRulesTypeFieldTypeFieldName";
    public static final String FLIGHTCHECKER_RULES_TYPE_FIELD_TYPE_FIELD_TYPE = "FlightCheckerRulesTypeFieldTypeFieldType";
    public static final String FLIGHTCHECKER_RULES_TYPE_FIELD_TYPE_FIELD_TYPE_NAME = "FlightCheckerRulesTypeFieldTypeFieldTypeName";
    public static final String FLIGHTCHECKER_RULES_TYPE_FIELD_TYPE_FIELD_FIELD = "FlightCheckerRulesTypeFieldTypeFieldField";
    public static final String FLIGHTCHECKER_RULES_TYPE_FIELD_TYPE_FIELD_FIELD_NAME = "FlightCheckerRulesTypeFieldTypeFieldFieldName";
    public static final String FLIGHTCHECKER_RULES_TYPE_FIELD_TYPE_FIELD_FIELD_TYPE = "FlightCheckerRulesTypeFieldTypeFieldFieldType";
    public static final String FLIGHTCHECKER_RULES_TYPE_FIELD_TYPE_FIELD_FIELD_TYPE_NAME = "FlightCheckerRulesTypeFieldTypeFieldFieldTypeName";
    public static final String FLIGHTCHECKER_RULES_TYPE_FIELD_TYPE_FIELD_FIELD_TYPE_FIELD = "FlightCheckerRulesTypeFieldTypeFieldFieldTypeField";
    public static final String FLIGHTCHECKER_RULES_TYPE_FIELD_TYPE_FIELD_FIELD_TYPE_FIELD_NAME = "FlightCheckerRulesTypeFieldTypeFieldFieldTypeFieldName";
    public static final String FLIGHTCHECKER_RULES_TYPE_FIELD_TYPE_FIELD_FIELD_TYPE_FIELD_TYPE = "FlightCheckerRulesTypeFieldTypeFieldFieldTypeFieldType";
    public static final String FLIGHTCHECKER_RULES_TYPE_FIELD_TYPE_FIELD_FIELD_TYPE_FIELD_TYPE_NAME = "FlightCheckerRulesTypeFieldTypeFieldFieldTypeFieldTypeName";
    public static final String FLIGHTCHECKER_RULES_TYPE_FIELD_TYPE_FIELD_FIELD_TYPE_FIELD_FIELD = "FlightCheckerRulesTypeFieldTypeFieldFieldTypeFieldField";
    public static final String FLIGHTCHECKER_RULES_TYPE_FIELD_TYPE_FIELD_FIELD_TYPE_FIELD_FIELD_NAME = "FlightCheckerRulesTypeFieldTypeFieldFieldTypeFieldFieldName";
    public static final String FLIGHTCHECKER_RULES_TYPE_FIELD_TYPE_FIELD_FIELD_TYPE_FIELD_FIELD_TYPE = "FlightCheckerRulesTypeFieldTypeFieldFieldTypeFieldFieldType";
    public static final String FLIGHTCHECKER_RULES_TYPE_FIELD_TYPE_FIELD_FIELD_TYPE_FIELD_TYPE_NAME = "FlightCheckerRulesTypeFieldTypeFieldFieldTypeFieldFieldName";
    public static final String FLIGHTCHECKER_RULES_TYPE_FIELD_TYPE_FIELD_FIELD_TYPE_FIELD_FIELD_TYPE_FIELD = "FlightCheckerRulesTypeFieldTypeFieldFieldTypeFieldFieldField";
    public static final String FLIGHTCHECKER_RULES_TYPE_FIELD_TYPE_FIELD_FIELD_TYPE_FIELD_FIELD_TYPE_NAME = "FlightCheckerRulesTypeFieldTypeFieldFieldTypeFieldFieldName";
    public static final String FLIGHTCHECKER_RULES_TYPE_FIELD_TYPE_FIELD_FIELD_TYPE_FIELD_FIELD_TYPE_FIELD = "FlightCheckerRulesTypeFieldTypeFieldFieldTypeFieldFieldFieldField";
    public static final String FLIGHTCHECKER_RULES_TYPE_FIELD_TYPE_FIELD_FIELD_TYPE_FIELD_FIELD_TYPE_NAME = "FlightCheckerRulesTypeFieldTypeFieldFieldTypeFieldFieldName";
    public static final String FLIGHTCHECKER_RULES_TYPE_FIELD_TYPE_FIELD_FIELD_TYPE_FIELD_FIELD_TYPE_FIELD = "FlightCheckerRulesTypeFieldTypeFieldFieldTypeFieldFieldFieldFieldField";
    public static final String FLIGHTCHECKER_RULES_TYPE_FIELD_TYPE_FIELD_FIELD_TYPE_FIELD_FIELD_TYPE_NAME = "FlightCheckerRulesTypeFieldTypeFieldFieldTypeFieldFieldName";
    public static final String FLIGHTCHECKER_RULES_TYPE_FIELD_TYPE_FIELD_FIELD_TYPE_FIELD_FIELD_TYPE_FIELD = "FlightCheckerRulesTypeFieldTypeFieldFieldTypeFieldFieldFieldFieldFieldField";
    public static final String FLIGHTCHECKER_RULES_TYPE_FIELD_TYPE_FIELD_FIELD_TYPE_FIELD_FIELD_TYPE_NAME = "FlightCheckerRulesTypeFieldTypeFieldFieldTypeFieldFieldName";
    public static final String FLIGHTCHECKER_RULES_TYPE_FIELD_TYPE_FIELD_FIELD_TYPE_FIELD_FIELD_TYPE_FIELD = "FlightCheckerRulesTypeFieldTypeFieldFieldTypeFieldFieldFieldFieldFieldFieldField";
    public static final String FLIGHTCHECKER_RULES_TYPE_FIELD_TYPE_FIELD_FIELD_TYPE_FIELD_FIELD_TYPE_NAME = "FlightCheckerRulesTypeFieldTypeFieldFieldTypeFieldFieldName";
    public static final String FLIGHTCHECKER_RULES_TYPE_FIELD_TYPE_FIELD_FIELD>Type: FlightChecker
    
    
    
    
    ```
```
xtext Generated Editor

Code Completion

xtext Generated Editor

Syntax Coloring
Custom Keyword Coloring
Realtime Constraint Validation

Customizable Outlines
Code Folding

Goto Definition
Find References
Cross-File References
Model as EMF
Jetbrains‘
Meta Programming System

Language Extension Example

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

New
ReadWriteLock l = ...
lock (l) {
  //code
}

Language Extension Example
Result behaves like a native base language construct

```java
public class JniClass extends base implements base {    
   <static fields>
   private final lock
   <getters/ters>    
   public JniClass() {    
      lock(this.unlock);  
      SharedResource.instance().doSomething();
   }
   
   private lock getlock() {    
      return this.lock;
   }
   <static methods>
   <static inner classifiers>
}
```
Language Extension Example

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;
    }

    // Other methods...
}
```

Example Languages

UI Language
Example Languages

HTML Templates

```html
<!-- template: MessageHeader (Message message, boolean hidePrefer, boolean unreadLast) -->

is not refreshable: Only one root element allowed for refreshable template.

<!-- variables -->

<rs:link><rs:link><rs:template uri> <!-- explicit to import -->

if (!hidePrefer) {<rs:template uri>}

<open class:right-margin-open>

[todo include visible=

and !DestLink(inter: message.prior(), hold: true)]
</open>

</open class:right-margin-open>

[not include visible=

and !DestLink(inter: message.created)]
</open>

if (null === message.updated) {

<open class:right-margin-open>

[not include visible=

and !DestLink(inter: message.updated)]
</open>

}
```

Example Languages

Persistent Classes

```java
public persistent class Forum 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 op[+];
public unordered child JThread[3..9] threads op[+];
public unordered bidirectional association Users[2..n] subscribers op[+];
public unordered bidirectional association Users[2..n] watchers op[+];

public Forum(string name, User creator) {
    this.name = name;
    this.threads = create();
}

<~ destructor >>
```
Domain Users Programming? Precision $\neq$ Programming

Programming = Precision + X
Precision

- Scientists
- Insurance Mathematicians
- Logisticians
- Medical Doctors
If Domain Users don’t get it, it might hint at a problem with the language! … or the documentation!

Creating the Language vs. Using the Language
Domain Users vs. Experts

Creating: Domain Expert

Using: Domain User

Creating: Domain Expert

... senior
... complete
... big picture
... deep
... precise, formal
... guru
Using: Domain User
... not senior
... narrower
... shallow

Can Domain Users **understand** what the Experts put into the language?

Verify early and often!
TOOLING

Intentional Software
Domain Workbench
Pension Workbench Example

Text Editing Domain

Pension Workbench Example

Insurance Mathematics Domain
Pension Workbench Example

Symbolically integrated

Niche DSL

Business Domain 1 DSL

Business Domain 2 DSL

Architecture DSL

Code + other target platform artifacts
Cascading Niche DSL

MODELS READ ONLY!

Architecture DSL

M2M

Business Domain 1 DSL

M2M

Business Domain 2 DSL

M2M

Code + other target platform artifacts

Cascading

Niche DSL

M2M

Business Domain 1 DSL

M2M

Annotation Model

M2T

Architecture DSL

M2M

Code + other target platform artifacts
Configuration and Customization

In-Language
Varying models
DSLs as Parameters
The language provides **explicit** support!

(we can learn that from OO)

---

**Specialization**

overriding, overwriting

**Leaving Holes**

for variant to fill in

**Inject Stuff**

several places at a time?
Reusable and Variations

- Specialization
  - Inheritance
- Leaving Holes
  - Template Method Pattern
- Inject Stuff
  - Aspect Orientation

Configuration and Customization

- In-Language
- Varying models
- DSLs as Parameters
Two Levels
~ problem space vs. software space

Problem Space: Configuration
Software Space: Customization

Two Levels
~ problem space vs. software space
# 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 */
```

---

**Two Levels Removal**

```
<entity>
  Party
  name: String
</entity>

<dependentOb>
  Phone
  number: int
  regionCode: int
  countryCode: int
</dependentOb>

<entity>
  Address
  city: String
  state: String
  zip: String
  street: String
</entity>

<entity>
  Persistence
  Party
  NeedsState
  Multiple Addresses
</entity>

<entity>
  XML
  Hibernate
  JDO
</entity>

International Phone  
LocalPhone

Phone
```
Two Levels Removal

```java
component DelayCalculator {
    provides default: IDelayCalculator
    requires screens[0..n]: IIInfoScreen
    provides mon: IMonitoring feature monitoring
}
```

Two Levels Removal

```java
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

```csharp
namespace monitoring {
    component MonitoringConsole ...
    instance monitor: ...
    dynamic connect monitor.devices ...

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

```
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
    }
}
```
Looks just like “normal” MPS Java code with state machine extensions

But contains “Feature Markup” that can be shown optionally

Feature Expressions are “real code”, not comments
This is the code for the pedestrian configuration:

```
public staticmachine class TrafficLights extends Component implements Component {
    // Static fields
    <static initializers>
    event q; { };
    event stop; { };
    state stop {
        call this.transition(true);
    }
    state q {
        call this.transition(true);
        stop -> stop
        call this.transition(false);
    }
    private boolean green;
    private boolean red;
    <properties>
    <initializers>
    public TrafficLights() {
        <constructors>
    }
    protected void setGreen(boolean b) {
        this.red = b;
    }
    protected void setGreen(boolean b) {
        this.green = b;
        for (int i = 0; i < 10; i++) {
            if (i == 7) {
                System.out.println("Green light on!");
            }
        }
    }
    }
```

This is the code for the car configuration:

```
public staticmachine class TrafficLights extends Component implements Component {
    // Static fields
    <static initializers>
    event q; { };
    event stop; { };
    state stop {
        call this.transition(true);
        green = true
        call this.transition(false);
    }
    state transitioning {
        call this.transition(true);
        stop -> stop
        call this.transition(false);
    }
    state q {
        call this.transition(true);
        stop -> stop
        call this.transition(false);
    }
    private boolean green;
    private boolean red;
    private boolean yellow;
    <properties>
    <initializers>
    public TrafficLights() {
        <constructors>
    }
    protected void setRed(boolean b) {
        this.red = b;
    }
    protected void setGreen(boolean b) {
        this.green = b;
    }
    }
```
In-Language Varying models
DSLs as Parameters

Feature Model w/ Parameters

... parameters with simple types
... complex types: DSL/metamodel
Variability in Generators

Model-Based Implementation

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

Model

Impl. Artifact 1

Impl. Artifact 2

Problem Space
Domain Requirements

Domain Engineering

Application Engineering

Software Space

Core Assets

Product

Domain
Requirements

Product
Requirements

Formal Domain Model

Formal Domain MetaModel

M

Formal Solution Space

Formal Solution Space MetaModel

M

...
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

Summing Up
THE END.

Markus Voelter
Independent/itemis
voelter@acm.org
http://www.voelter.de