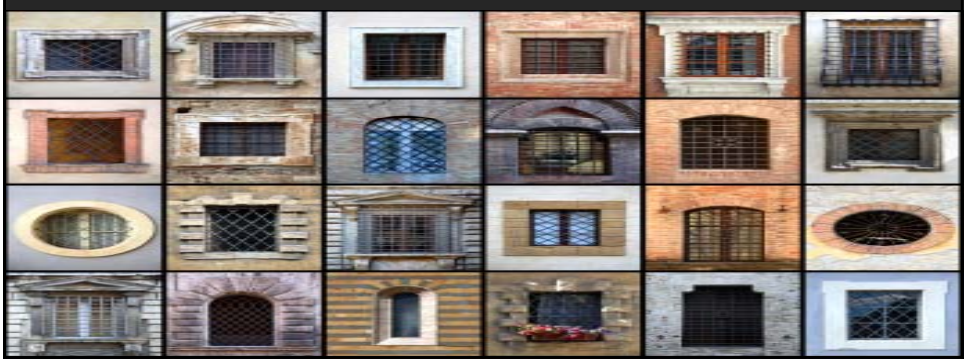
Using Domain Specific Languages for Product Line Engineering

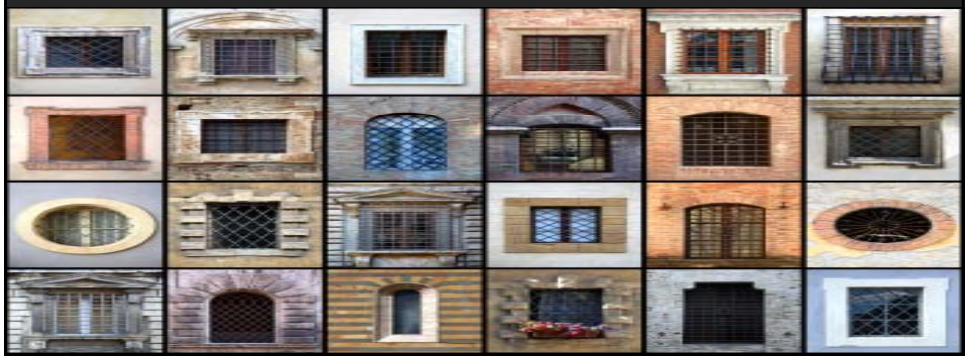
SPLC 2009 Tutorial

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

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



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



Variability

… differences among products in PL

Variation Point

… a point where a variation can occur

… must be bound for each product

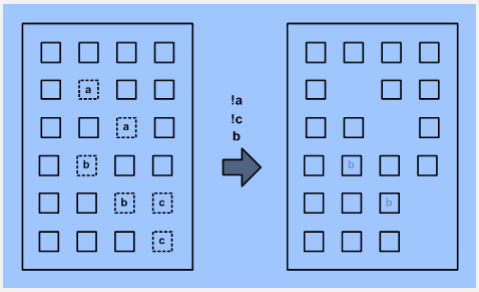
… bind when?
… bind how?

Binding Time

	flexibility	performance	code size	complexity
source time	-	+	+	-
compile time	+	+	+	-
link time	+	+	+	-
load time	++	+	+	+
run time	+++	-	-	+

Variability Mechanisms
Removal

… optionally take away
from overall whole



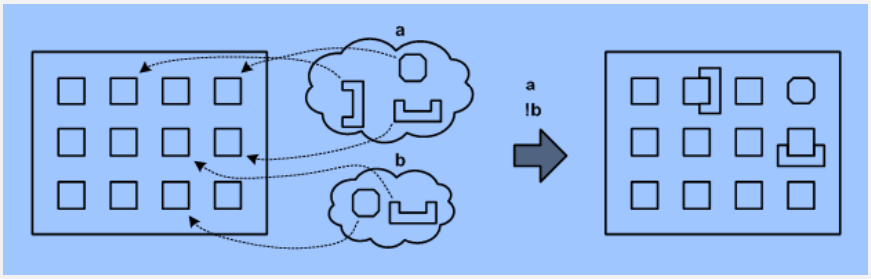
Variability Mechanisms
Removal

… optionally take away
from overall whole

Challenge:
overall whole can
get big and unwieldy

Variability Mechanisms
Injection

… optionally add to minimal core



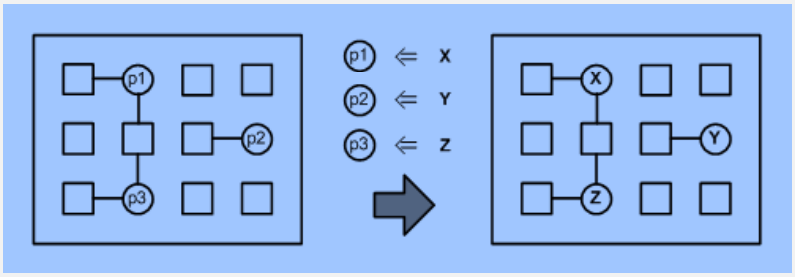
Variability Mechanisms
Injection

… 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

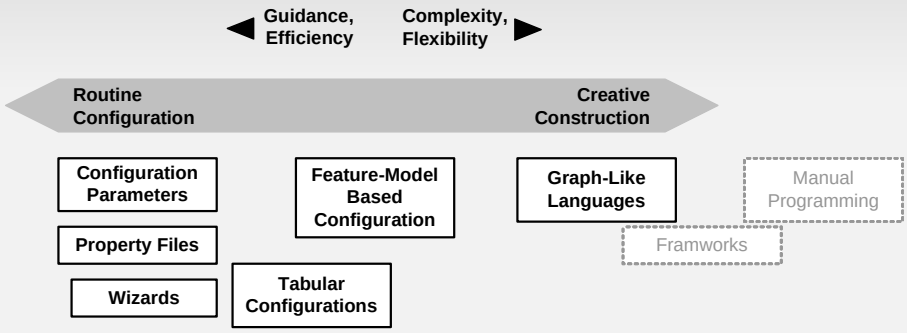


Variability Mechanisms
Parametrization

… define values for predefined params

Challenge:
types for parameters can be non trivial (DSLs)

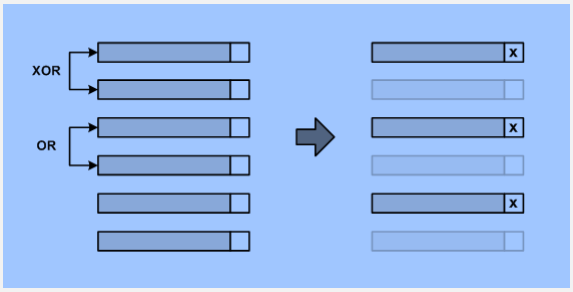
Configuration vs. Customization
Variability



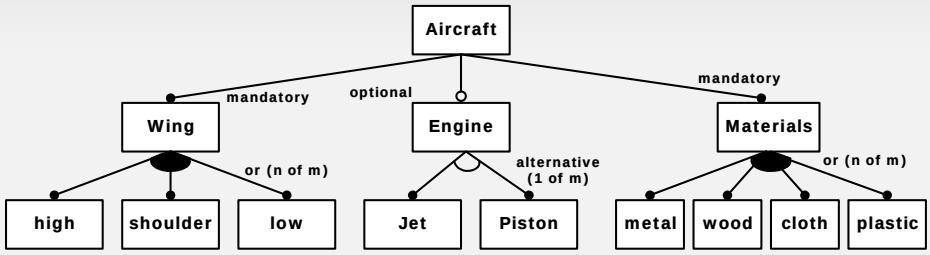
Variability
MDD Tooling
Configuration
Model Variability
Customization
Transformation Var.
MDD Intro
Summary & Wrapup

Configuration

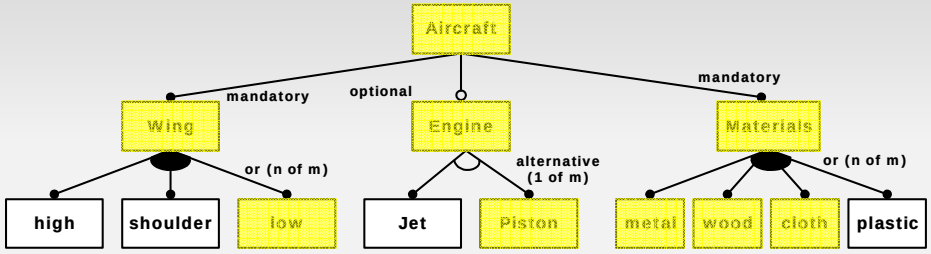
… selecting options
… setting param values



Configuration
Feature Models

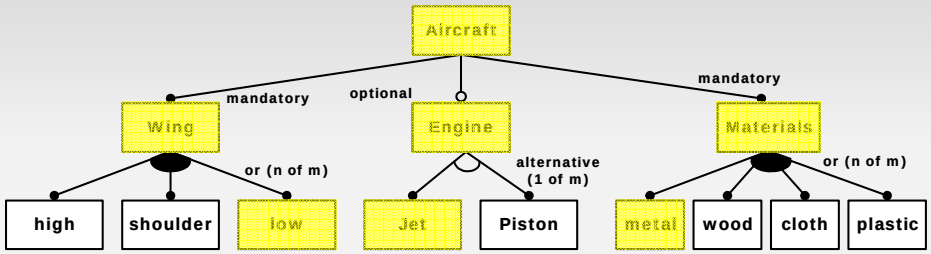


Configuration
Feature Models



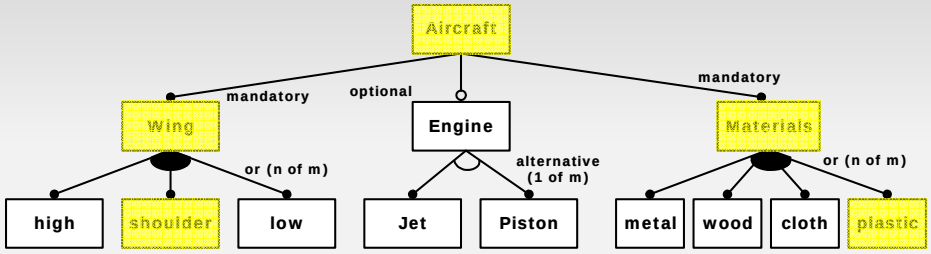
Robin DR-400
An aircraft with a low wing, piston engine and made of metal, wood and cloth

Configuration
Feature Models



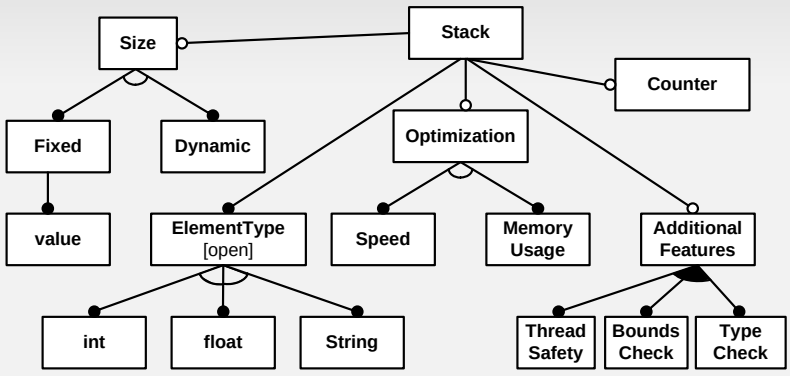
Airbus A 320
An aircraft with low wing, jet engine(s) and made of metal.

Configuration
Feature Models

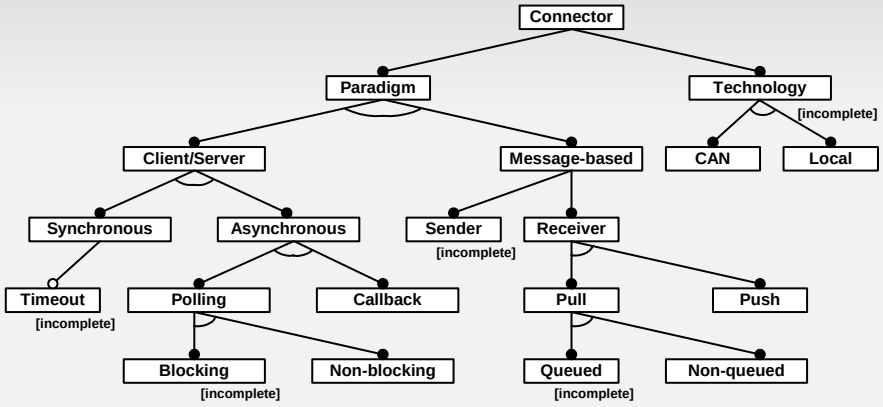


Schleicher ASW 27
An aircraft with shoulder wing, no engine and made of plastic

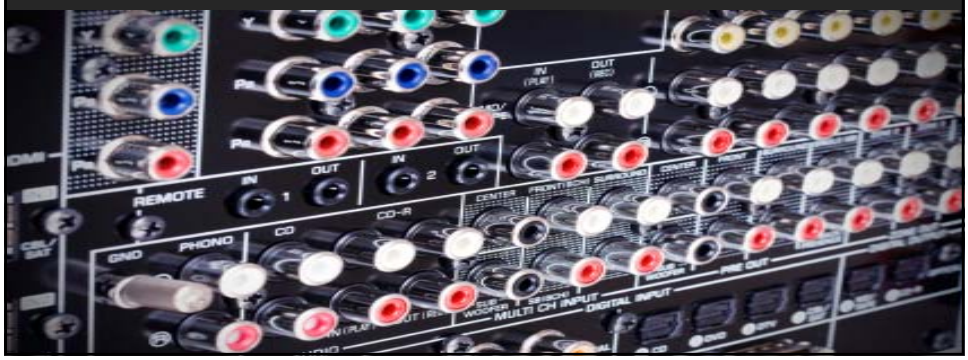
Configuration
Feature Models



Configuration
Feature Models

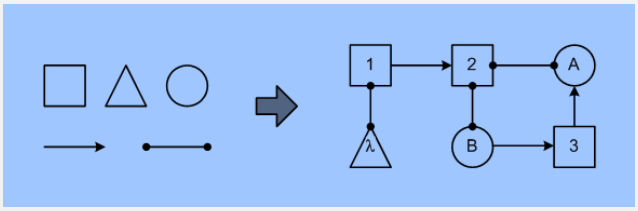


Variability
MDD Tooling
Configuration
Model Variability
Customization
Transformation Var.
MDD Intro
Summary & Wrapup



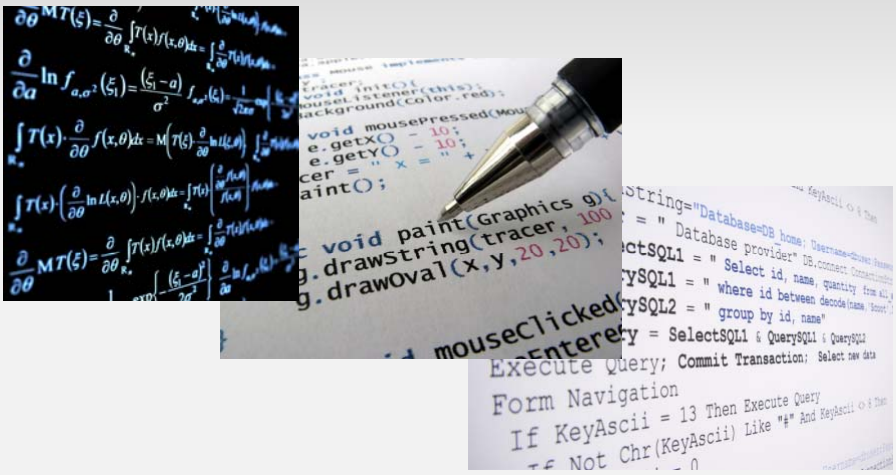
Customization

… „real languages“
… instantiation
… connections

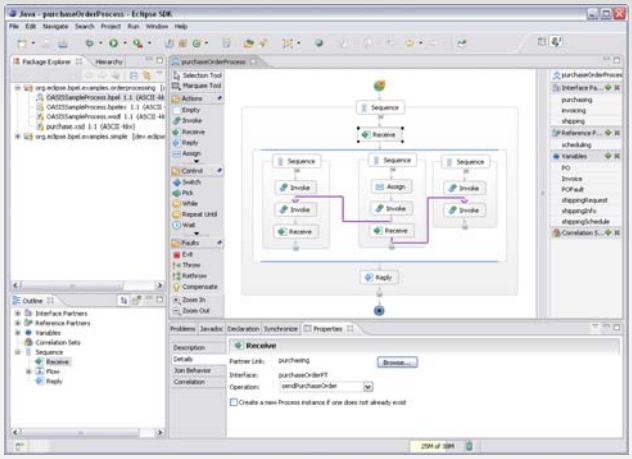


Customization

Languages



Customization
Languages

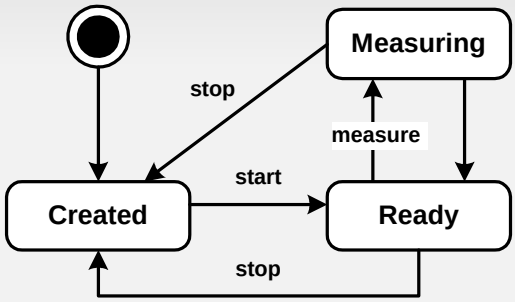


Customization
Languages

```
data {

    data Patient persistent {
        name: String
        age : int range 0..120
        firstName: String[20] regex "[a-z]*"
        birthDate: String
        insuranceName: String
        insuranceNumber: String
        -> adr: Adress
        -> tel: Telephone*
        -> stays: Stay*
    }
    data Adress persistent {}

    data Telephone {
        countryCode: String
        cityCode: String
        localNumber: String
    }
    data Stay {
        beginDate: String
        -> stations: Station*
        endDate: String
    }
    data Station {
        date: String
        stationName: String
    }
}
```

```
records {

    record P001 {
        p: Patient
        1 -> p.name
        2 -> p.firstName
        3 -> p.birthDate


        record A001 {
            a: Adress
            1 -> a.street
            2 -> a.zip
            3 -> a.city
            p.adr = a
        }

        record V001 {
            1 -> p.insuranceName
            2 -> p.insuranceNumber
        }
        record T001 {}
        record S001 {}
    }
}
```

Customization
Languages



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

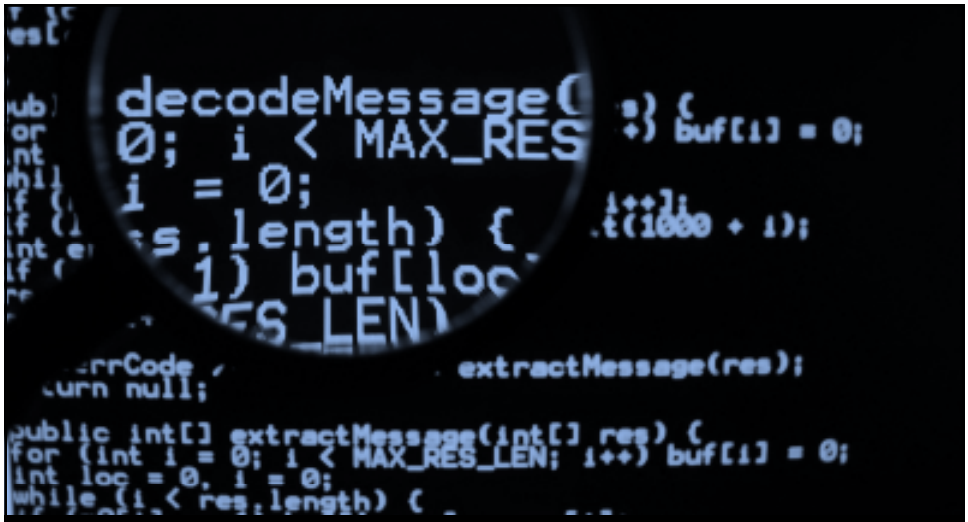
programming
started
close to the hardware

abstractions
~computing

chips

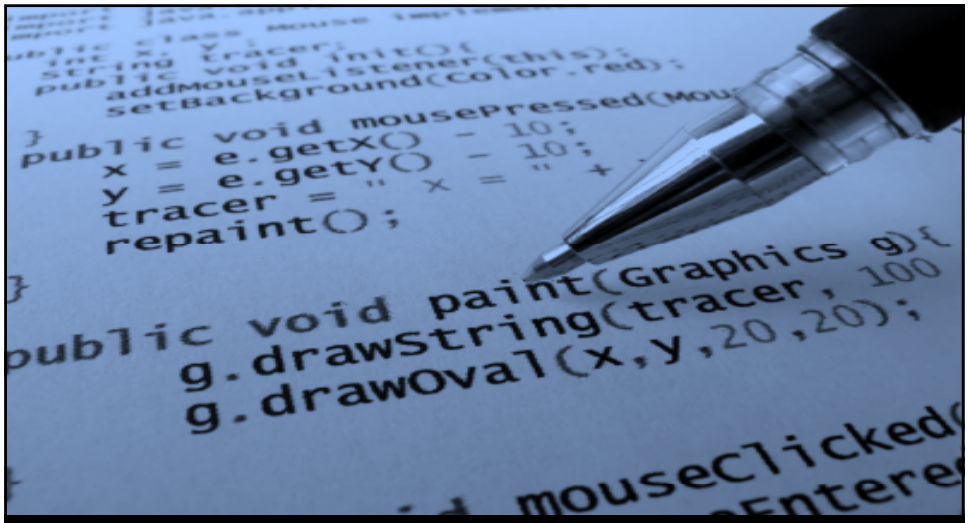
abstractions
~computing

bits



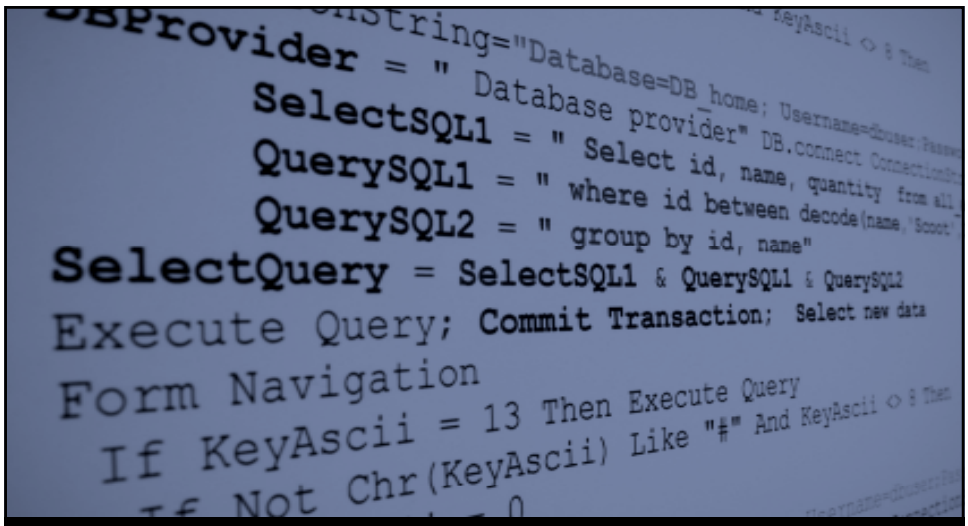
abstractions
~computing

C



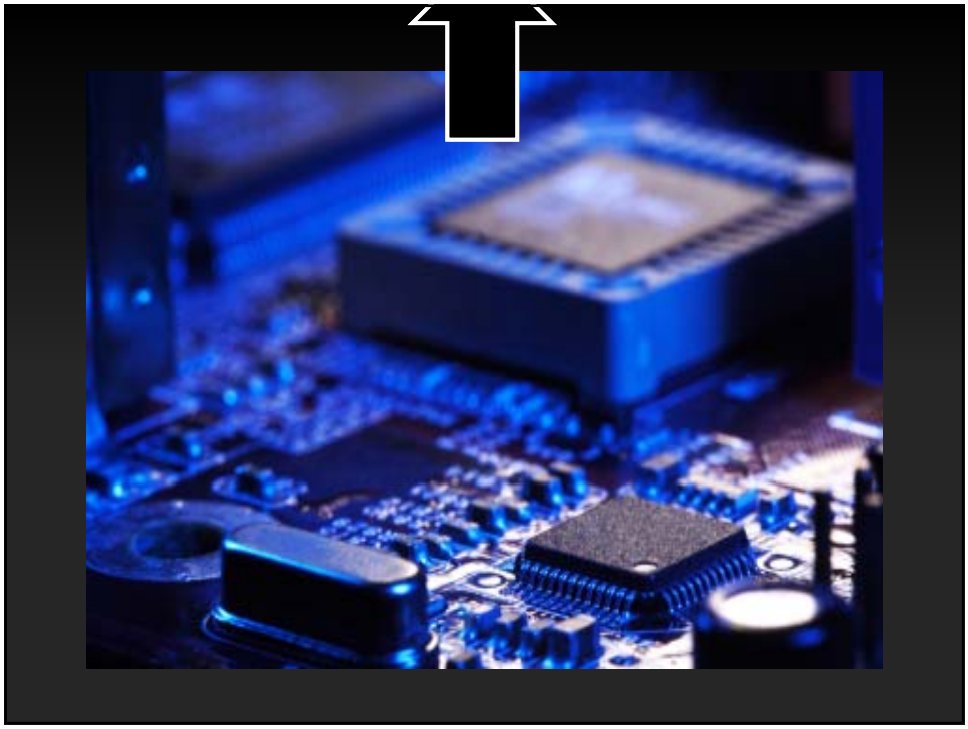
abstractions
~computing?

Java



abstractions
~computing?

SQL



?

general purpose

domain specific

tailor made

effective++

specialized, limited

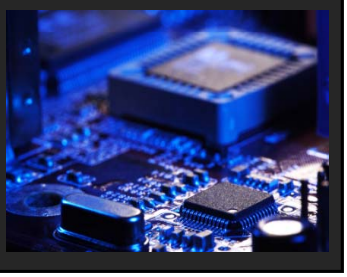
used by experts

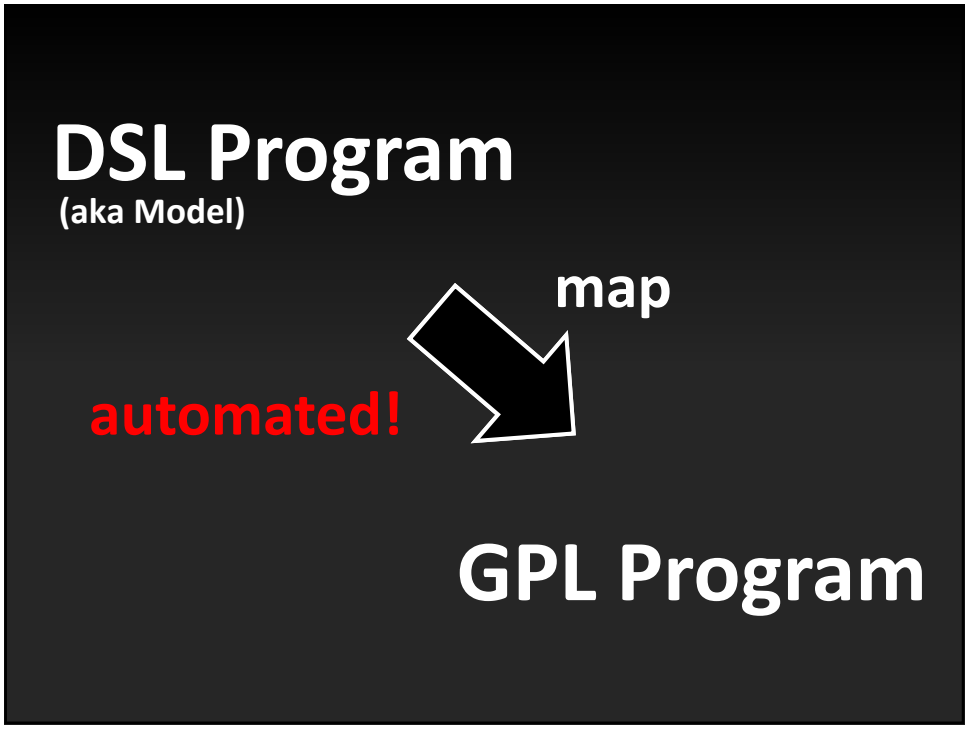
together with other specialized tools

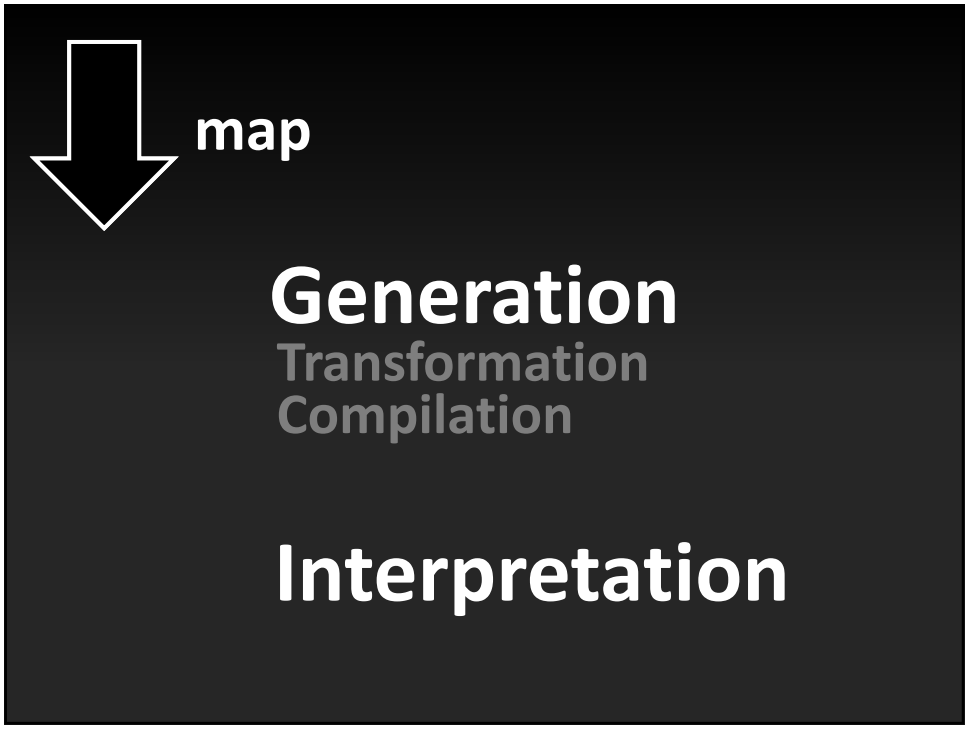
A DSL is a **focussed**, **processable language** for describing a specific **concern** when building a system in a specific **domain**. The **abstractions** and **notations** used are natural/suitable for the **stakeholders** who specify that particular concern.

execute?

map




DSL Program
(aka Model)
map
automated!
GPL Program


map
Generation
Transformation
Compilation
Interpretation

LanguageOrientedProgramming
DomainSpecificModeling
GenerativeProgramming
ModelDrivenDevelopment
ModelIntegratedComputing
ModelDrivenEngineering
ModelDrivenArchitecture
ModelBasedDevelopment
ModelBasedEngineering
ModelDrivenSoftwareDevelopment

Automation
faster, deterministic

Increased Quality

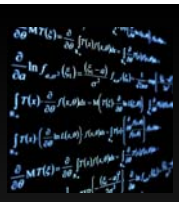
well defined structures allthrough the system

Meaningful Validation

more semantics in the model

Capture Domain Knowledge

formalized into languages and models

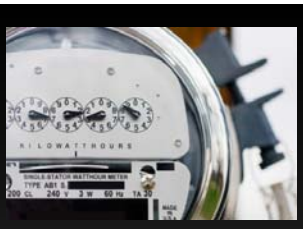


Suitable Notations

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

Variability
Configuration
Customization
MDD Intro

MDD Tooling

Model Variability
Transformation Var.
Summary & Wrapup

eclipse modeling PROJECT

EMF

Ecore meta meta model

+

Editing

Transactions

Validation

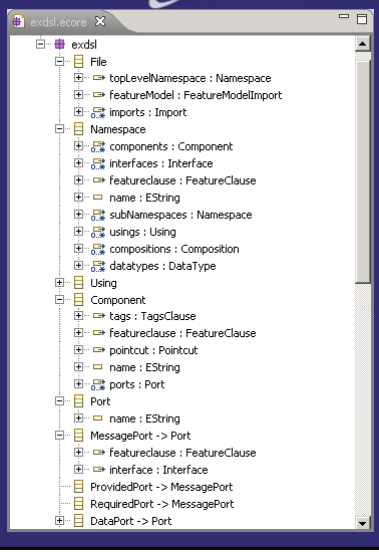
Query

Distribution/Persistence

EMF

Metamodel As Tree

can also be edited
as UML-like diagram



EMF

Constraints

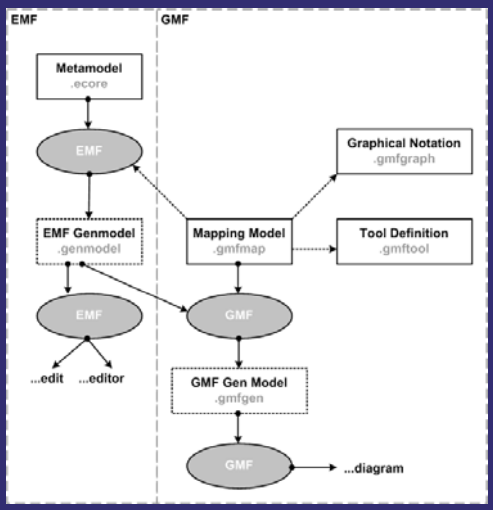
with OCL
and dialects

Checks.chk

```
import exdsl;

extension net::ample::adsl::exdsl::Extensions;
extension org::openarchitectureware::util::stdlib::io;

context Component ERROR "Qualified Name "+qualifiedName()+" must be unique"
   allComponents().select( c| c.qualifiedName() == qualifiedName() ).size =

context DataType ERROR "Qualified Name "+qualifiedName()+" must be unique"
   allDataTypes().select( c| c.qualifiedName() == qualifiedName() ).size ==

context Namespace if !isEmpty() ERROR "Qualified Name "+qualifiedName()+" n
   allNamespaces().select( c| c.qualifiedName() == qualifiedName() ).size =

context emf::EObject if metaType.getProperty("name")!=null ERROR "name not
   metaType.getProperty("name").get(this) != "Unnamed";

context Interface ERROR "interface names must start with a capital I":
    name.startsWith("I");

context MessagePort ERROR "interface not defined. Missing a 'using'?: ":
    visibleInstancesOfType(this, Interface).contains(interface);

context Attribute ERROR "no type defined: "+type.name:
    visibleInstancesOfType(this, DataType).contains(type);

context DataPort ERROR "data not defined: "+type.name:
    visibleInstancesOfType(this, ComplexType).contains(type);
```

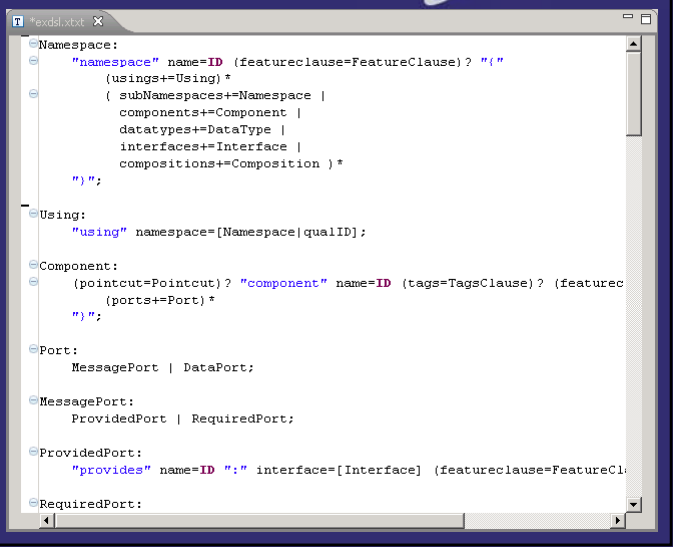
GMF

Graphical Box/Line editors based on EMF



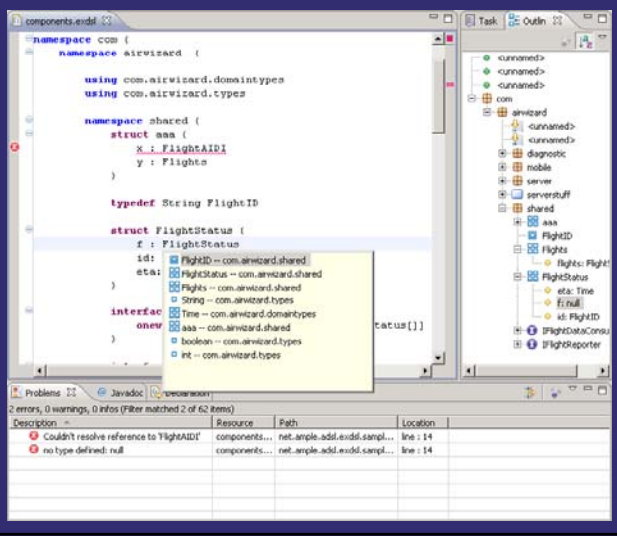
TMF / Xtext

Building Textual Editors



TMF / Xtext

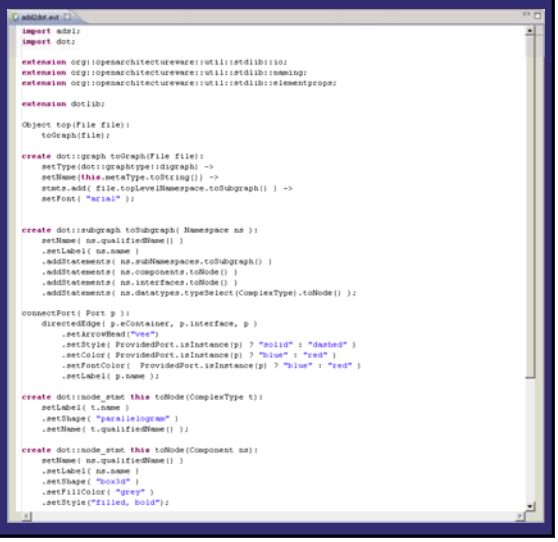
Building Textual Editors



M2M

Model-to-Model Transformations

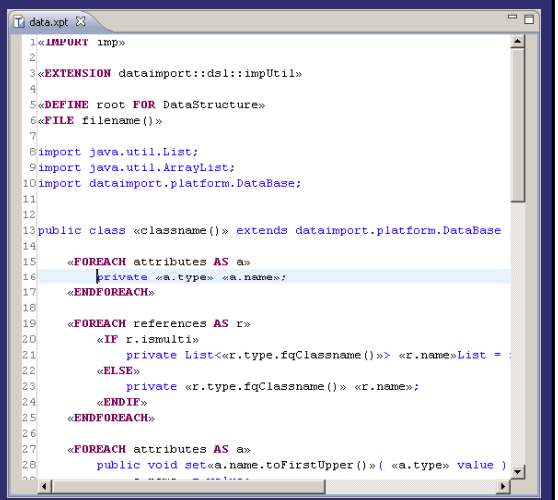
- INRIA's ATL
- QVT
- Xtend



M2T

Model-to-Text Transformations

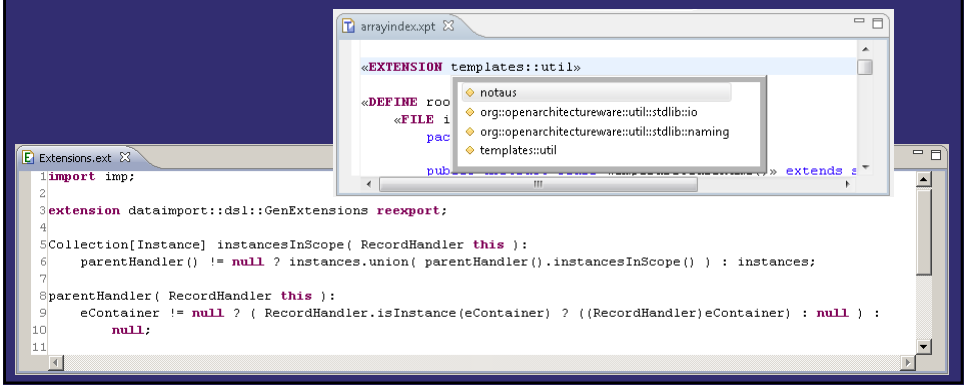
- JET: Java Emitter Templates
- Xpand: oAW's template engine



M2T

Model-to-Text Transformations

Extensions to modularize complex expressions



openArchitectureWare

www.openarchitectureware.org

One Stop Toolkit for DSLs + X

Version **5.0 is current**

Lively ecosystem of tools and extensions

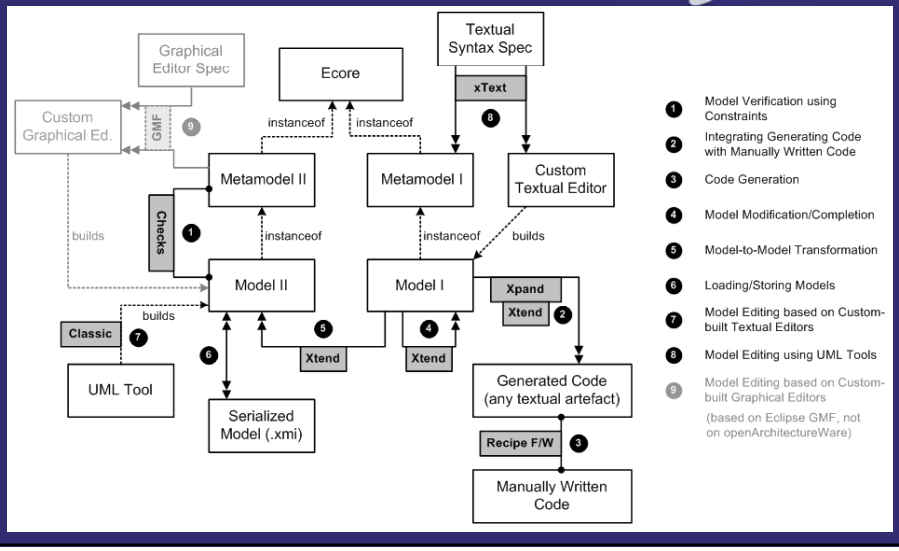
Proven track record in various domains & project contexts

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

openArchitectureWare

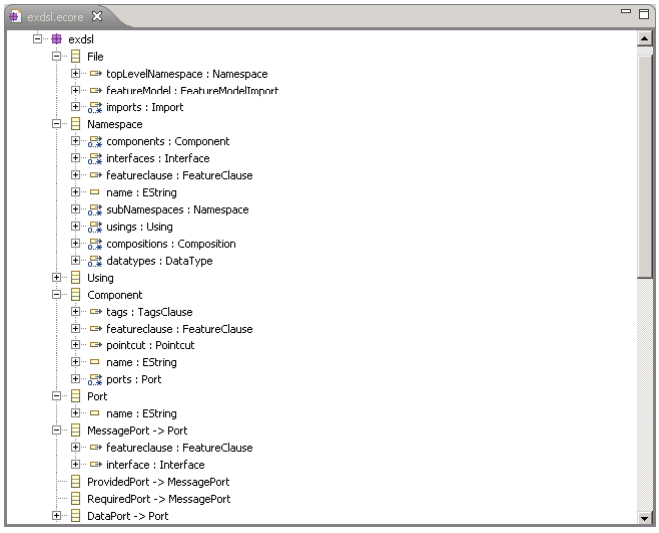


Specify Grammar

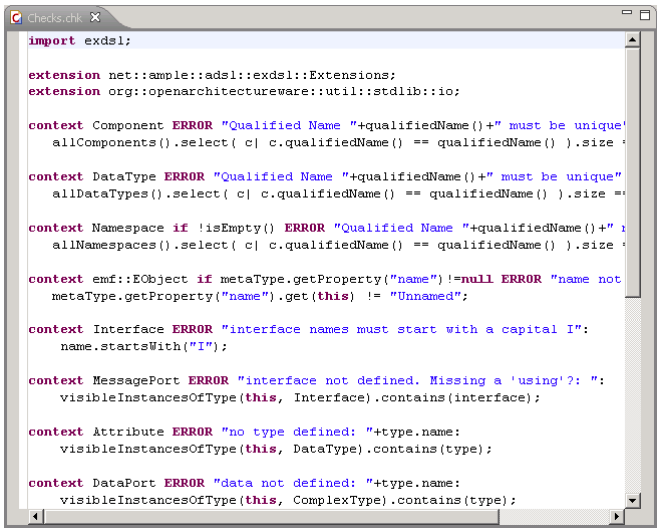
```
*exdsl.xtxt
Namespace:
    "namespace" name=ID (featureclause=FeatureClause)? "{"
        (usings+=Using)*
        ( subNamespaces+=Namespace |
          components+=Component |
          datatypes+=DataType |
          interfaces+=Interface |
          compositions+=Composition )*
    "}";

Using:
    "using" namespace=[Namespace|qualID];

Component:
    (pointcut=Pointcut)? "component" name=ID (tags=TagsClause)? (featurec
        (ports+=Port)*
    "}";

Port:
    MessagePort | DataPort;

MessagePort:
    ProvidedPort | RequiredPort;

ProvidedPort:
    "provides" name=ID ":" interface=[Interface] (featureclause=FeatureCl

RequiredPort:
```

Antlr Grammar and Parser is generated from this specification

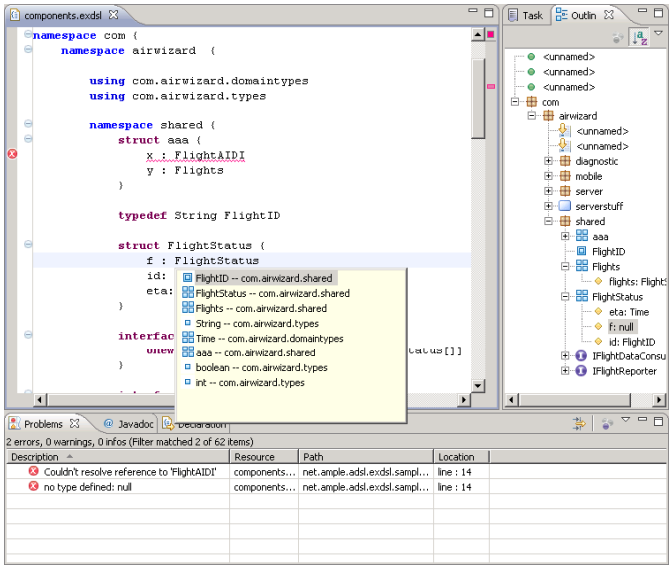
Generated Metamodel



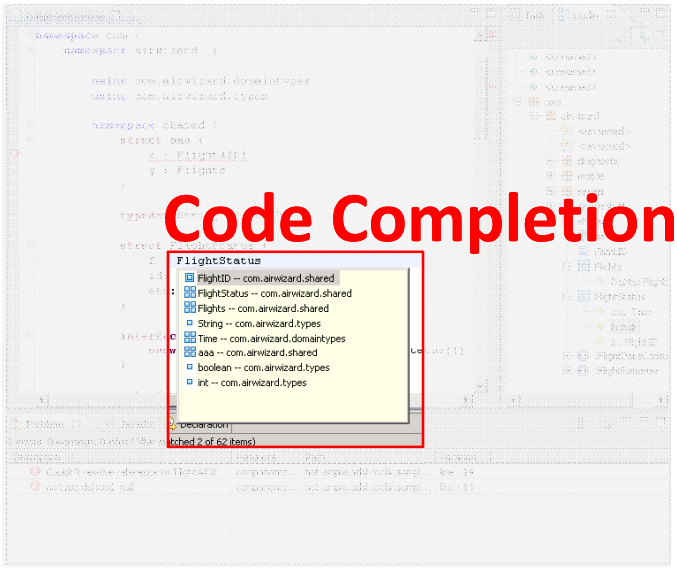
Specify Constraints



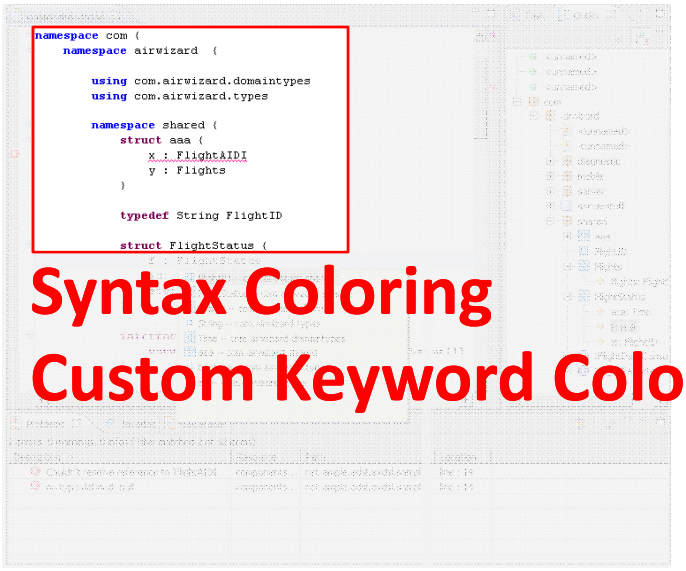
Generated Editor



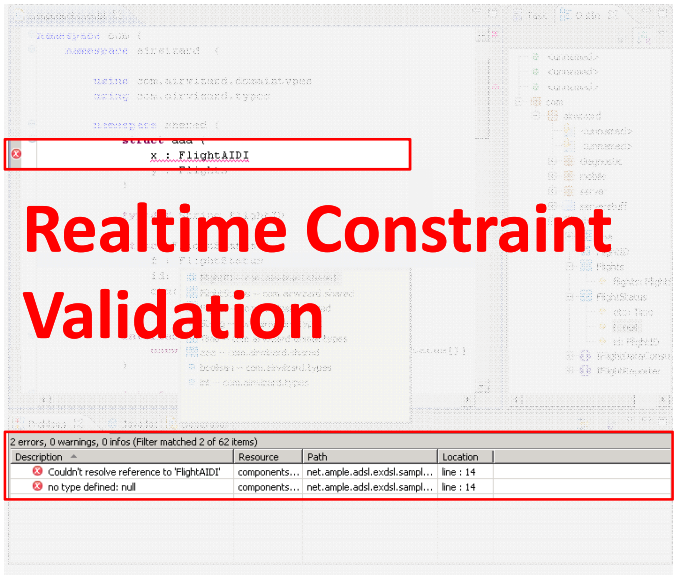
Generated Editor



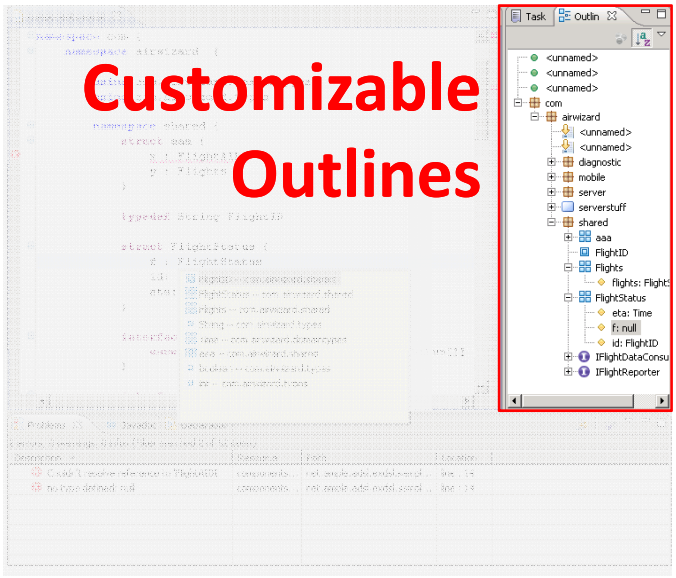
Generated Editor



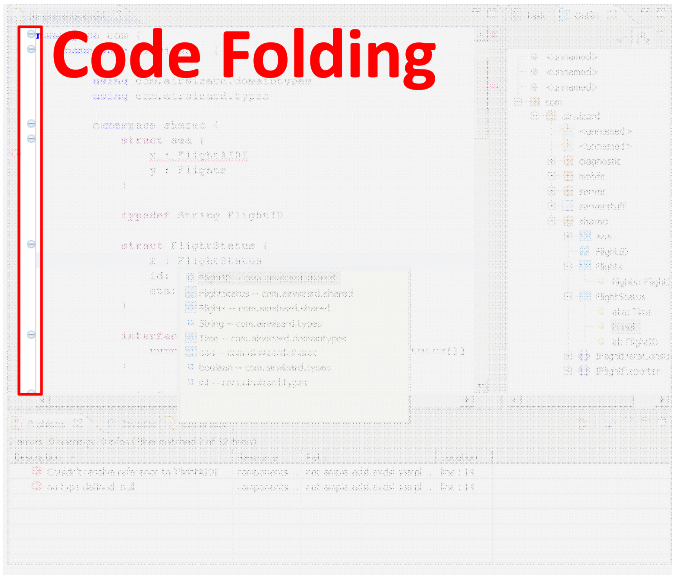
Generated Editor



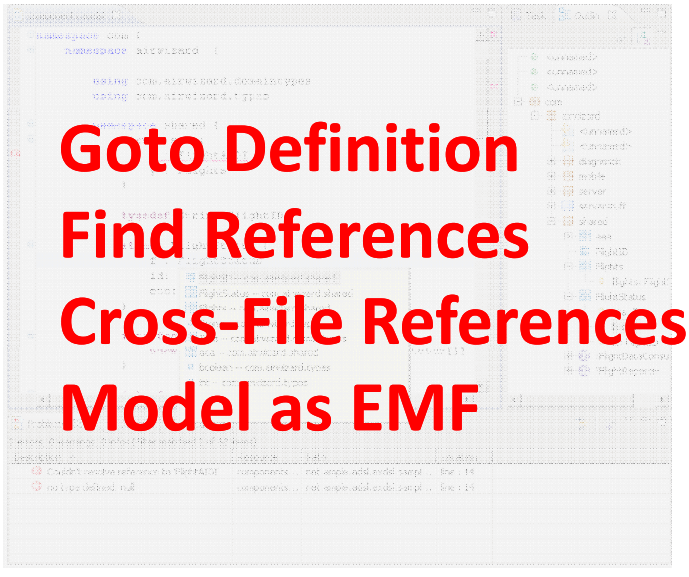
Generated Editor



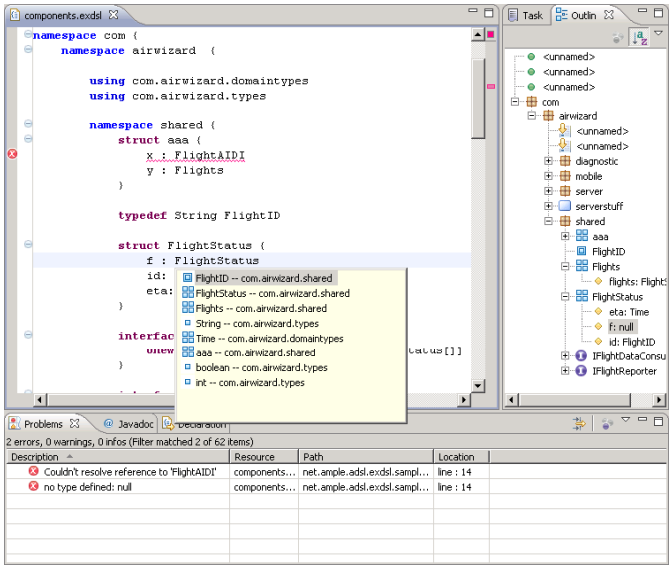
Generated Editor



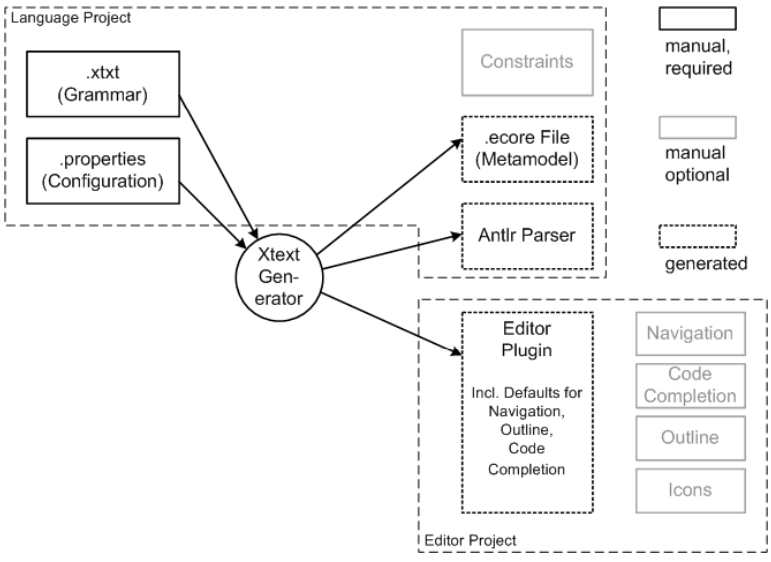
Generated Editor



Generated Editor



Xtext Overview



DEMO

Building a sample textual DSL and code generator for a simple domain using Eclipse TMF/openArchitectureWare

Another Tool…?

Another Tool…?

also do…

IntelliJ IDEA
Resharper

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)

Java++

extend base language

build DSLs that reuse parts of BaseLanguage

Language Extension Example

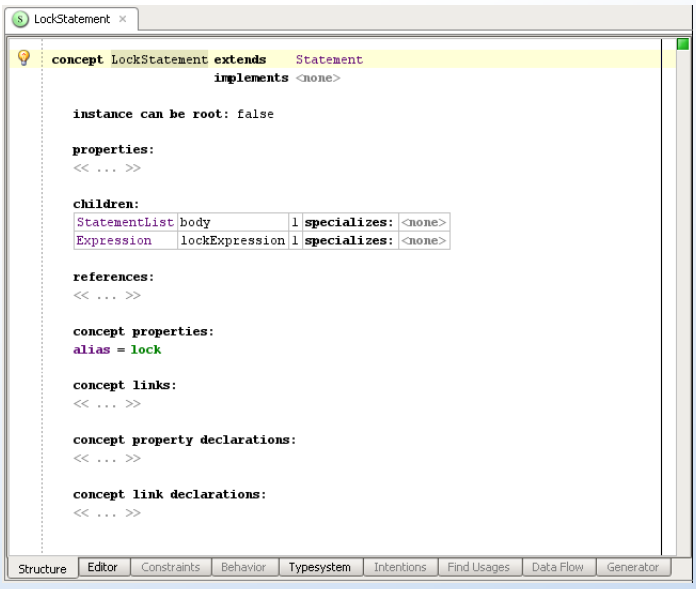
Old

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

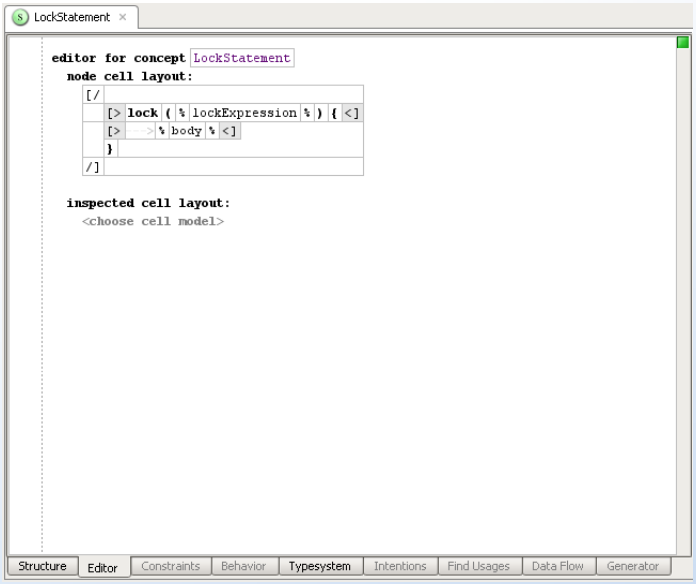
New

```
ReadWriteLock l = …
lock (l) {
  //code
}
```

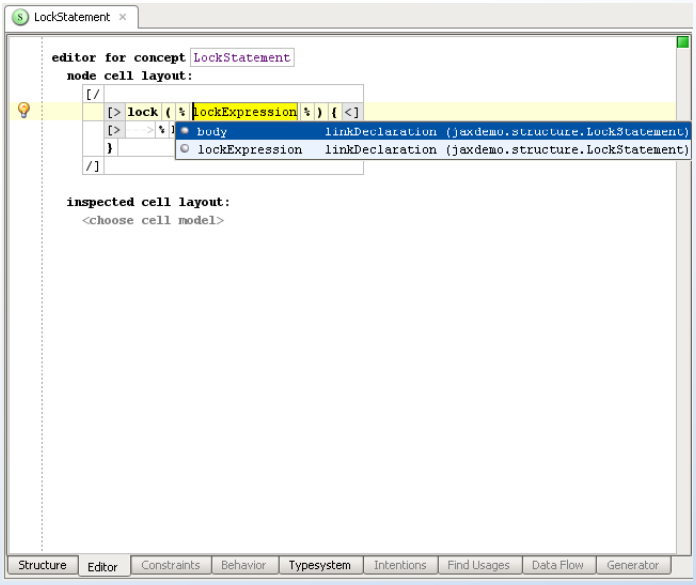
Structure ◆ Editor ◆ Typesystem ◆ Generator



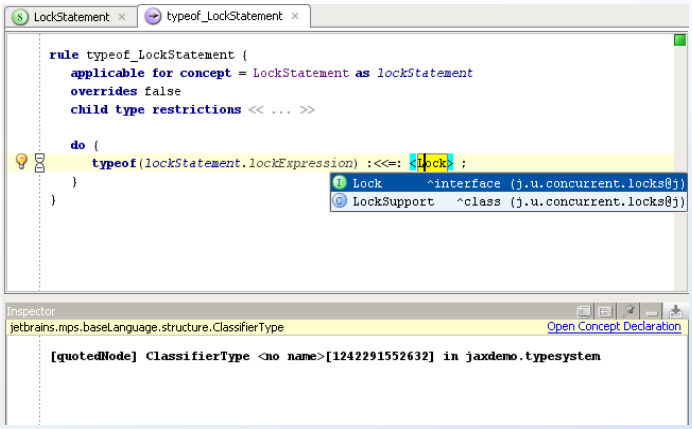
Structure ◆ Editor ◆ Typesystem ◆ Generator



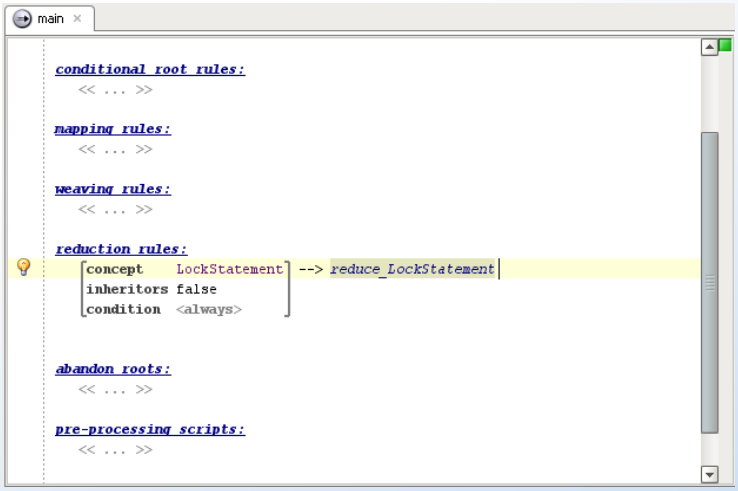
Structure ◆ Editor ◆ Typesystem ◆ Generator



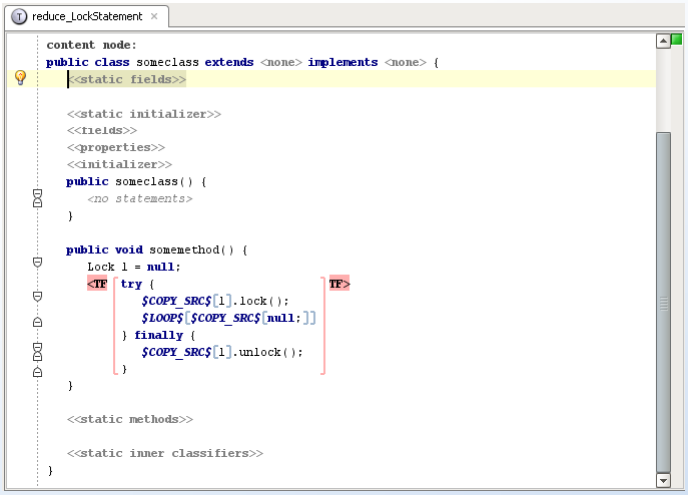
Structure ◆ Editor ◆ Typesystem ◆ Generator



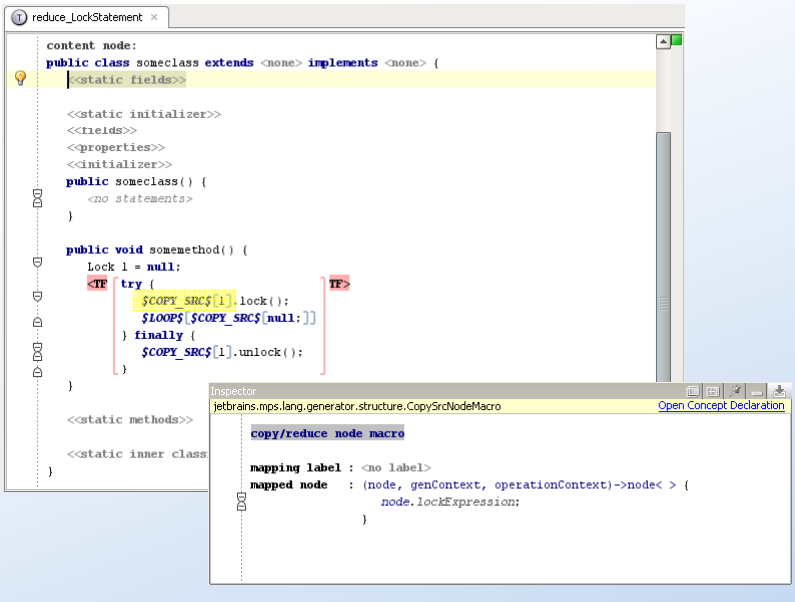
Structure ◆ Editor ◆ Typesystem ◆ Generator



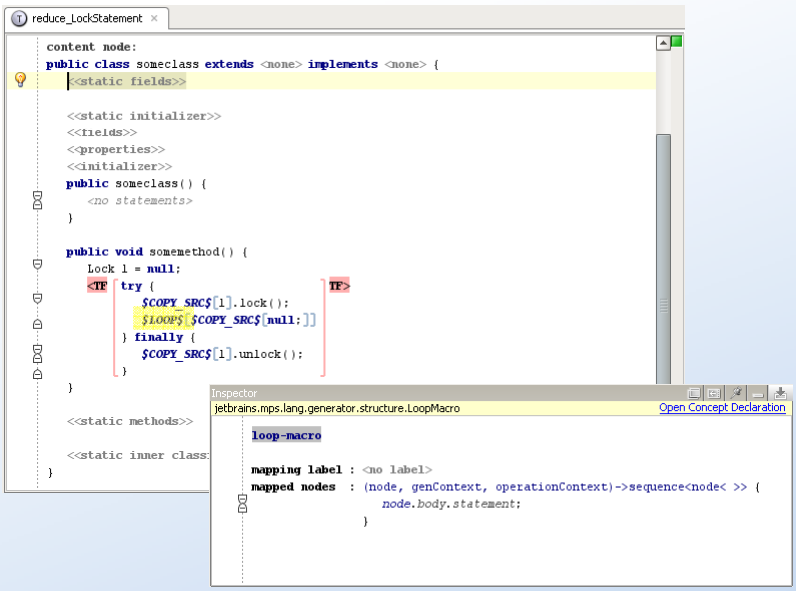
Structure ◆ Editor ◆ Typesystem ◆ Generator



Structure ◆ Editor ◆ Typesystem ◆ Generator



Structure ◆ Editor ◆ Typesystem ◆ Generator

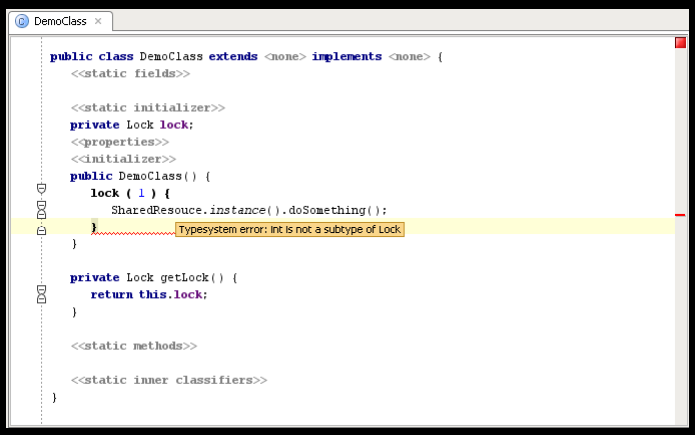


Language Extension Example

Result behaves like a native base language construct

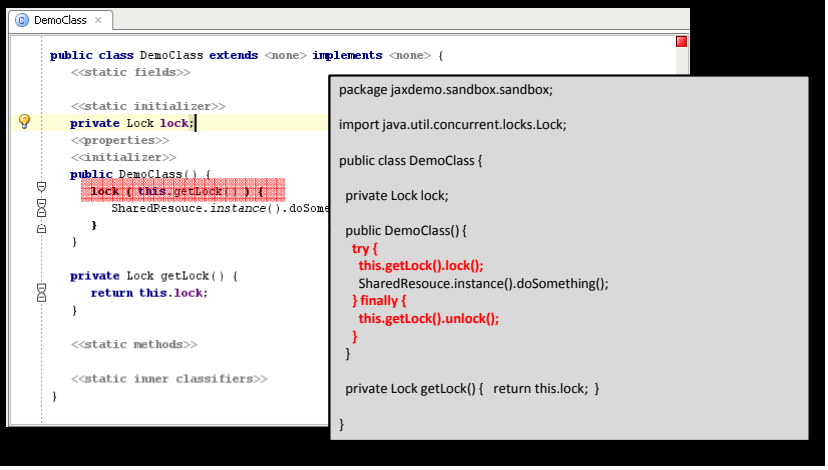
Language Extension Example

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 NewLanguageDialog

  IDEDialog
    left: 100 top: 100 width: 600 height: 200
    title = New Language

    content pane:
    Grid
      Row
        [Label
          text : "Language Namespace:"]
      Row
        [TextField ( name )
          text : ( this.languageNamespace )]
      Row
        [Label
          text : "Language Path:"]
      Row
        [PathField ( path )
          path : ( this.languagePath )]

    buttons
    [text OK                ][text Cancel
    default true            ][default false
                            ]
    [handler this.onOk()    ][handler this.onCancel()]
```

Example Languages

HTML Templates

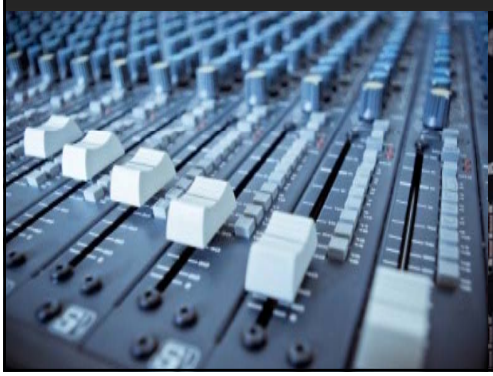
```
html template MessageHeader(Message message, boolean hideUser, boolean userLast)

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

<< variables >>

rss links: << rss links >> << rss template uris>>

<< explicit js import >>

if (!(hideUser) && !(userLast)) {
  <span class: right-margin-span>
    [html include <visible>
      def[]: UserLink(user: message.getUser(), bold: true)]
  </span>
}
<span class: right-margin-span>
  [html include <visible>
    def[]: Date(date: message.created)]
</span>
if (null != message.updated) {
  <span class: right-margin-span>
    updated:
    [html include <visible>
      def[]: Date(date: message.updated)]
  </span>
}
```

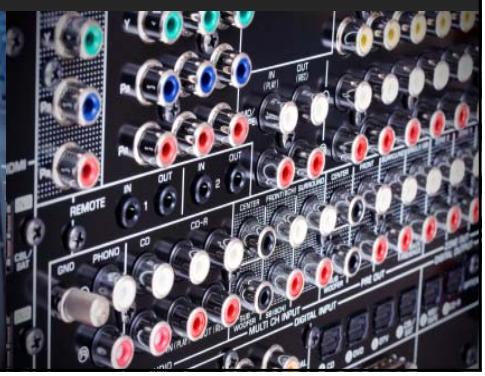
Example Languages

Persistent Classes

```
public persistent class Forum extends <none> implements <none> {

  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 opts;
  public unordered child FThread[0..n] threads opts;
  public unordered bidirectional association User[0..n] subscribers onDelete(clear);
  public unordered bidirectional association User[0..n] watchers onDelete(clear);

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


  << destructor >>
```

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





Two Levels

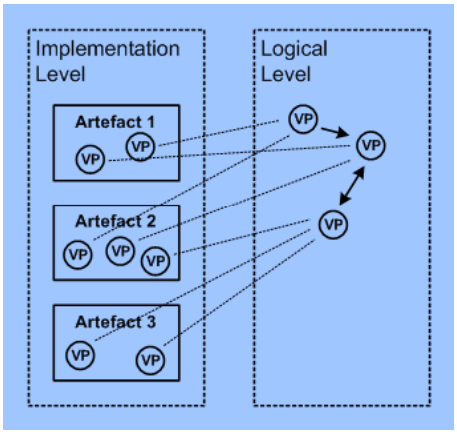
~ problem space
vs. software space

Problem Space:
Configuration

Software Space:
Customization

Two Levels

~ problem space
vs. software space



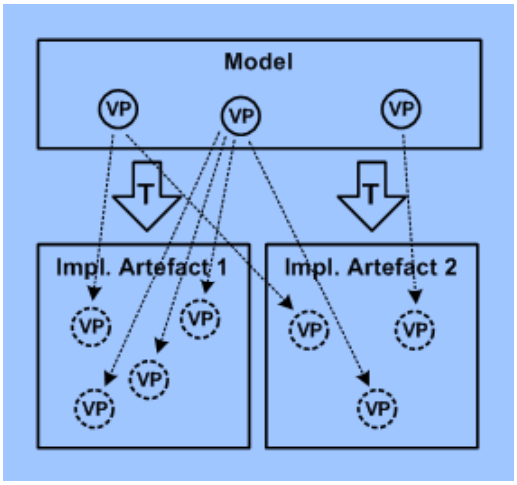
Two Levels Removal

```
#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 */
```

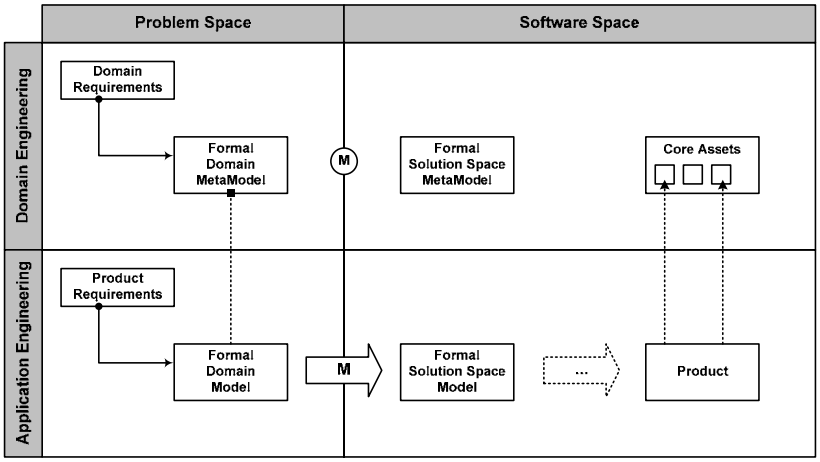
Model-Based Implementation

- … customization in problem space
- … Problem-Space DSL

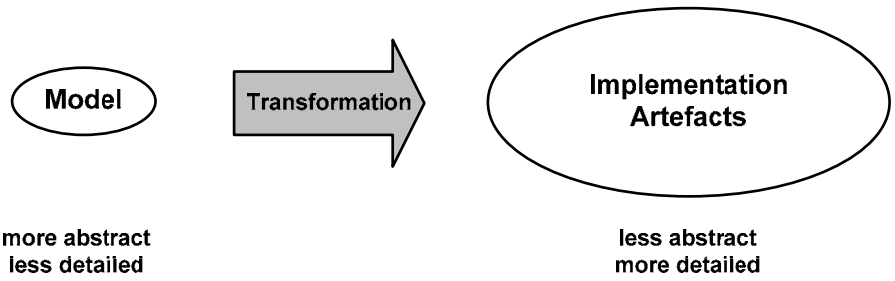
Model-Based Implementation



Model-Based Implementation



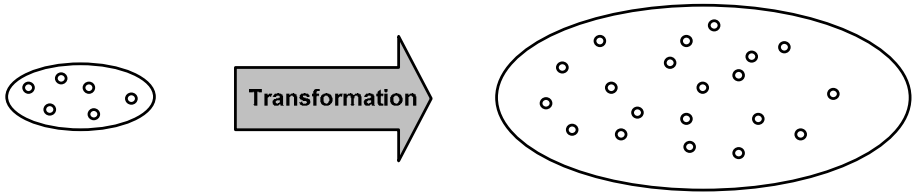
MDSD - Thumbnail



more abstract
less detailed

less abstract
more detailed

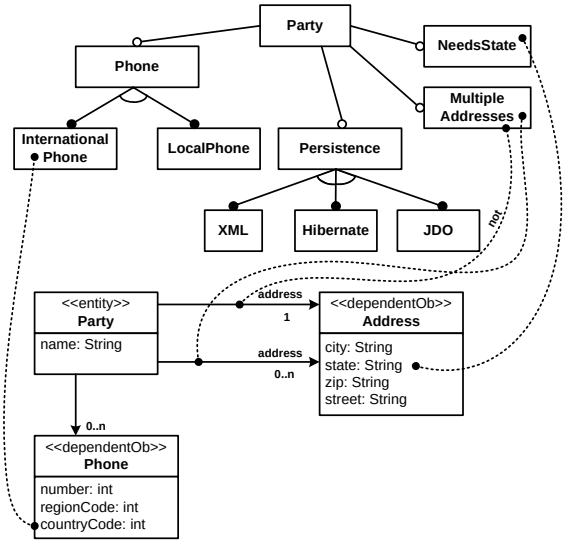
MD-PLE - Thumbnail



= Variation Point

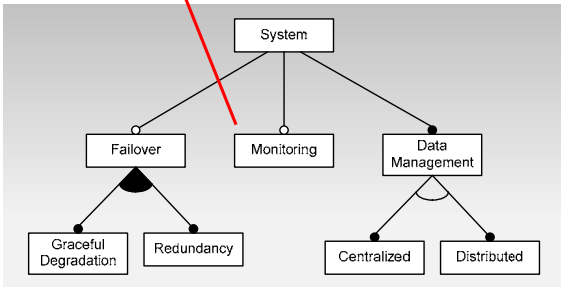
fewer!

Two Levels Removal



Two Levels Removal

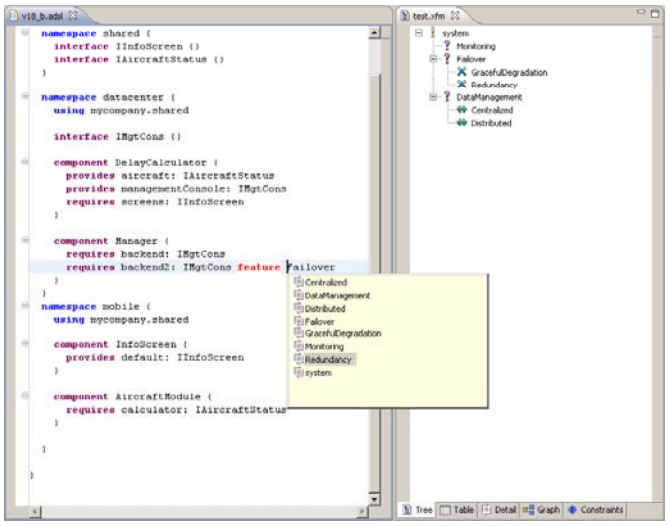
```
component DelayCalculator {
  provides default: IDelayCalculator
  requires screens[0..n]: IInfoScreen
  provides mon: IMonitoring feature monitoring
}
```



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

```
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 (*) compnent {
   provides mon: IMonitoring
}
```

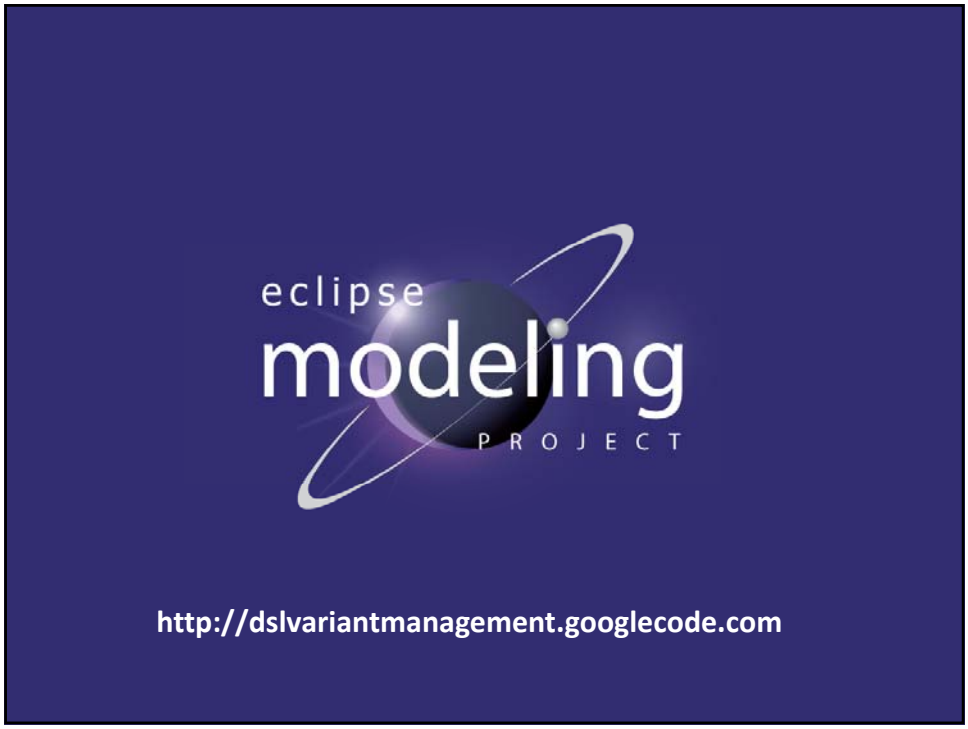
```
component DelayCalculator {
  …
  provides mon: IMonitoring
}
component AircraftModule {
  …
  provides mon: IMonitoring
}
component InfoScreen {
  …
  provides mon: IMmonitoring
}
```

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

}
```

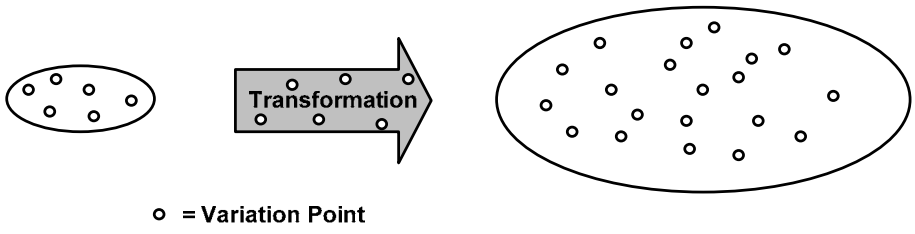


DEMO

Extending the sample DSL to include feature-based variability connected to an external feature model

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

MD-PLE – Thumbnail II

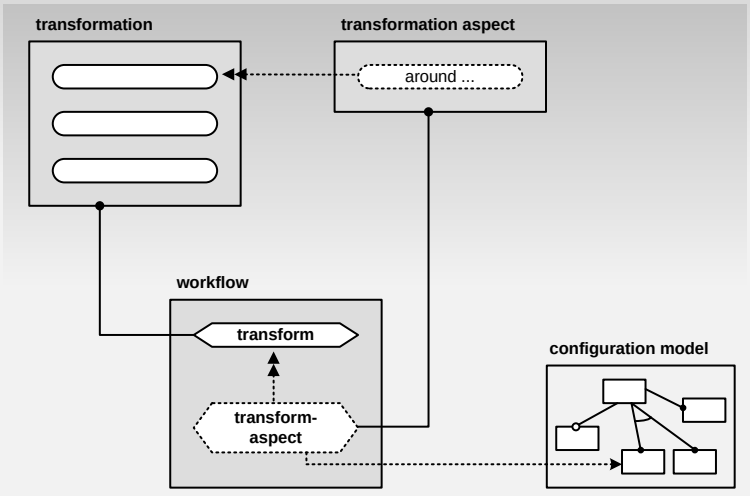


more options

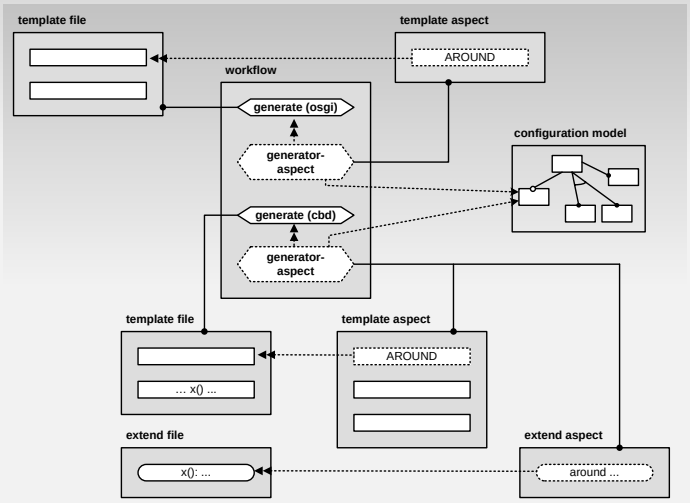
Transformation
Variability

```
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
);
```

Transformation
Variability



Generator Variability



DEMO

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.

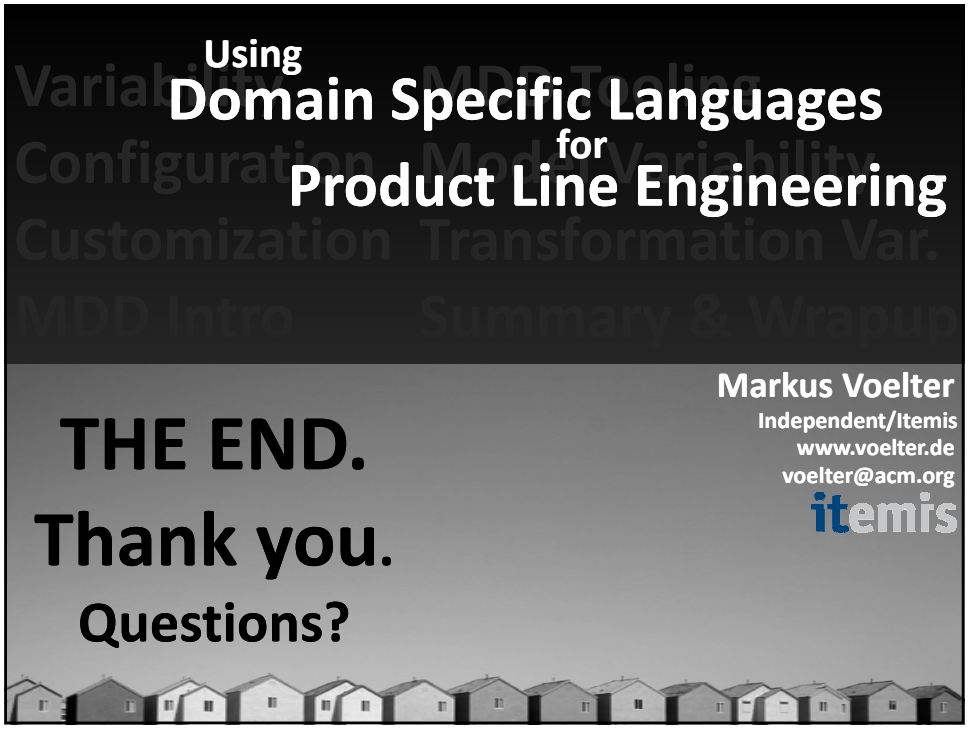
Transformation and Generation can be used to map PS to SS

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
Variability
MDD Tooling
Configuration
Model Variability
Customization
Transformation Var.
MDD Intro
Summary & Wrapup
THE END.
Thank you.
Questions?
Markus Voelter
Independent/Itemis
www.voelter.de
voelter@acm.org
itemis