Lightweight MDD for Embedded Systems

Markus Voelter
www.voelter.de
voelter@acm.org
itemis

Andreas Graf
www.itemis.de
graf@itemis.de
itemis

About itemis

1

What is a language?

INFORMAL

Set of well-defined terms

INFORMAL

Stakeholders agree on meaning

FORMAL

Metamodel

FORMAL

Metamodel
Grammar

FORMAL

Metamodel
Grammar
Notation

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.

Architecture DSLs

Architecture DSL

As you understand and develop your Architecture…

Develop a language to express it!

Language resembles architectural concepts

We express the application(s) with the language.

DEMO I

An architectural DSL for embedded systems

3

Benefits

Clear Understanding from building the language

Unambigious Vocabulary

Concepts independent from Technology

Programming Model can be defined based on Conceptual Arcitecture

Architecture „executable“
(i.e. more than rules and docs)

Why Textual?

… or: why not graphical?

Languages and Editors are easier to build

Languages and Editors are easier to build

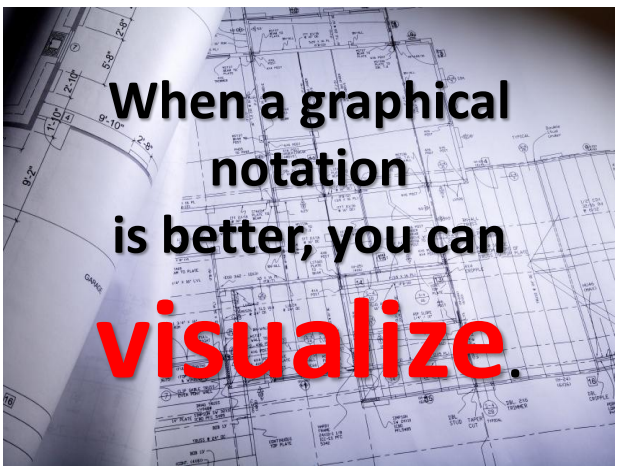
Evolve Language and simple editor as you understand and discuss the architecture, in real time!

Integrates easily with current infrastructure: CVS/SVN diff/merge

adapting existing models as the DSL evolves

Model evolution is trivial, you can always use *grep*.

Many Developers prefer textual notations



5

Tooling

Several tools available. Example: oAW Xtext

eclipse modeling PROJECT

Specify Grammar

```
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

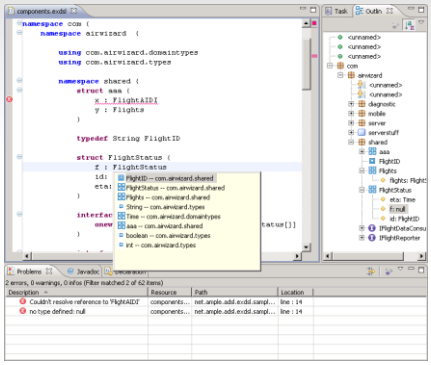
Generated Metamodel

```
exdsl
  File
    topLevelNamespace : Namespace
    featureModel : FeatureModelImport
    imports : Import
  Namespace
    components : Component
    interfaces : Interface
    featureclause : FeatureClause
    name : EString
    subNamespaces : Namespace
    usings : Using
    compositions : Composition
    datatypes : DataType
  Using
  Component
    tags : TagsClause
    featureclause : FeatureClause
    pointcut : Pointcut
    name : EString
    ports : Port
  Port
    name : EString
  MessagePort -> Port
    featureclause : FeatureClause
    interface : Interface
  ProvidedPort -> MessagePort
  RequiredPort -> MessagePort
  DataPort -> Port
```

Specify Constraints

```
import exdsl;

extension net::sample::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()+"
   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);
```

Generated Editor



DEMO II

The language-aware editor for our DSL

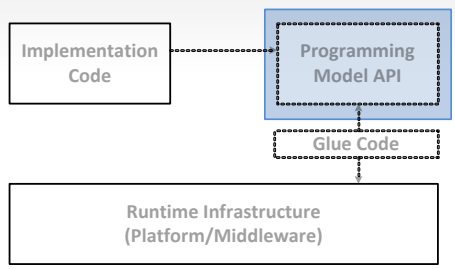
6

Generating Code

Since we already have a formal model….

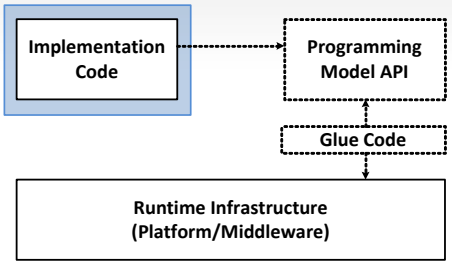
Generate API

Maps Architectural Concepts to Implementation language (non-trivial!)



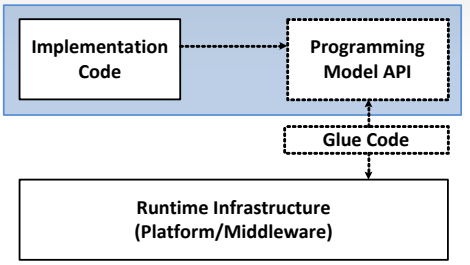
Implementation

Implementation only depends on the generated programming model API



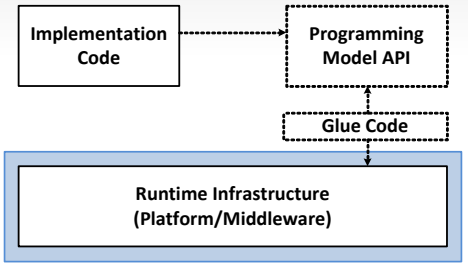
Programming Model

Generated API + Usage Idioms
Completely Technology-Independent



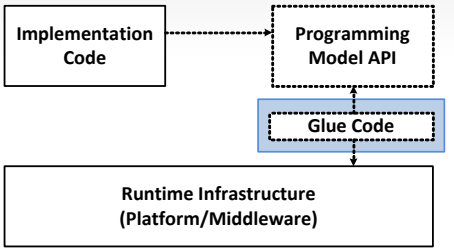
Runtime Infrastructure

Select based on fit wrt. to architectural concepts and non-functional requirements



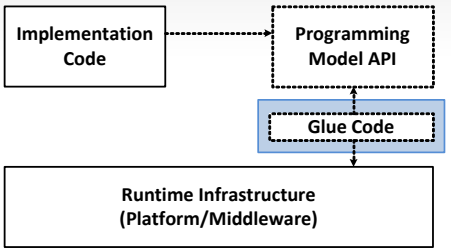
Glue Code

Aka Technology Mapping Code
Maps API to selected platform



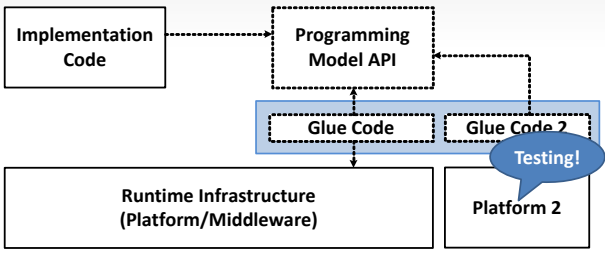
Glue Code

Contains Configuration Files for Platform
Might require „mix in models“



Several Platforms

Different Platforms, not Languages
Support for Scaling (non-functional req)

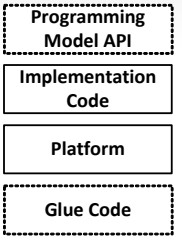


Benefits:

More Efficient Impl.
Technology Independent
Consistence/Quality
Architecture-Conformance

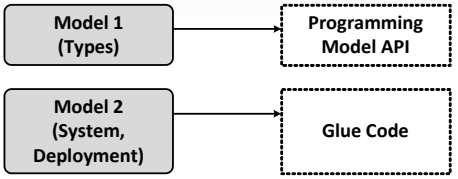
Code Gen Sequence

1) Generate API
2) Write Impl Code
3) Select Platform
4) Generate Glue Code



Separate Models

for stuff relevant for the API vs. system/deployment stuff



DEMO III

Generating C for the target device

7

Expressing Variability

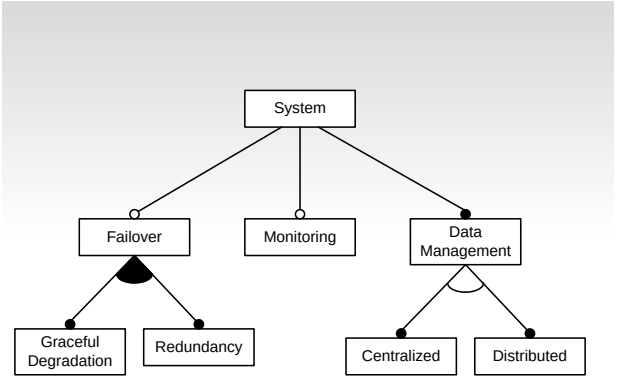
Different **Variants** of the System for different customers.

How do I **express** this in the models?

Negative Variability:
Conditionally taking something away

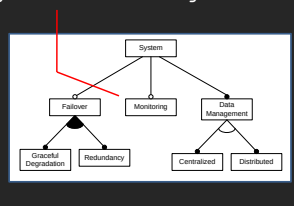
Negative Variability:
Conditionally taking something away

Feature Models

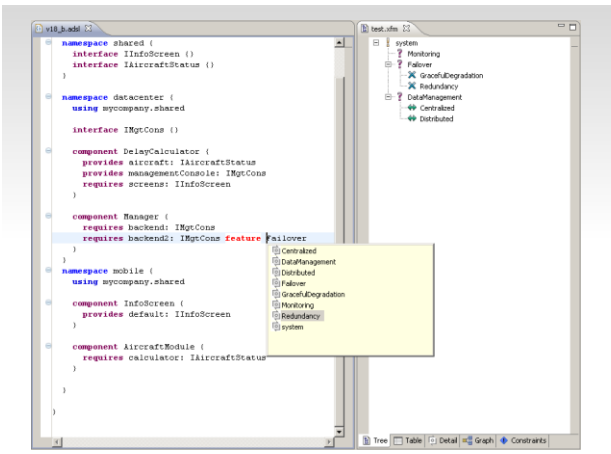



```
component DelayCalculator {
  provides default: IDelayCalculator
  requires screens[0..n]: IInfoScreen
  provides mon: IMonitoring feature monitoring
}
```

```
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

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 {
  …
}
component AircraftModule {
  …
}
component InfoScreen {
  …
}
```

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

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

Weaver is **generic**:

works with all (container) model elements

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

AO + Features

```
namespace monitoring feature monitoring {

  component MonitoringConsole …
  instance monitor: …
  dynamic connect monitor.devices …

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

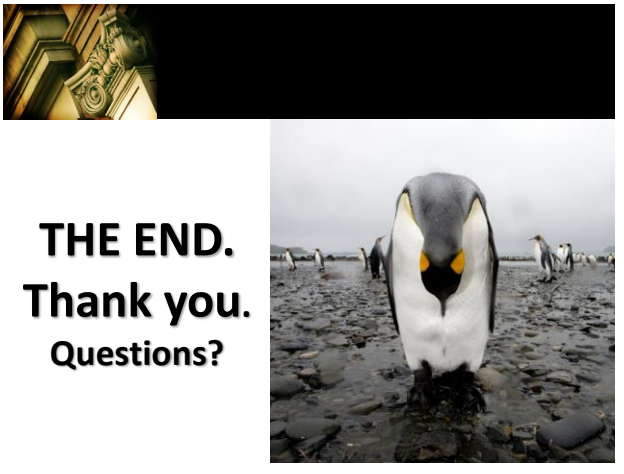
DEMO III

Adding Variability and connectivity to a feature model to the previous DSL


Based on actual
practical experience


Currently in use with
four of my customers


Benchmarked by
suitability for use in
today's projects


THE END.
Thank you.
Questions?