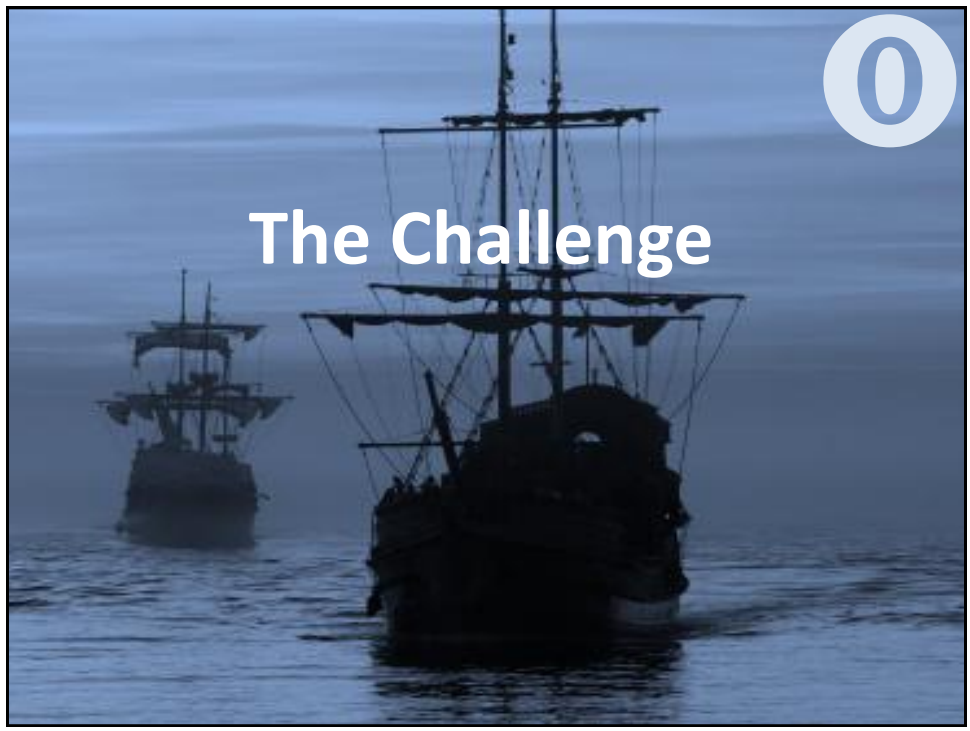
Model-Driven Development
in the context of embedded systems with Eclipse Tools
Peter Hellmann
Matthias Zacharias
matthias.zacharias@bmk-solutions.de
peter.hellmann@bmk-solutions.de
BMK electronic solutions
Markus Voelter
Andreas Graf
Philipp Bollbach
voelter@itemis.de
graf@itemis.de
bollbach@itemis.de
itemis


0
The Challenge

Software increasingly **complex**

Faster innovation cycles

Product Lines - **Variability**

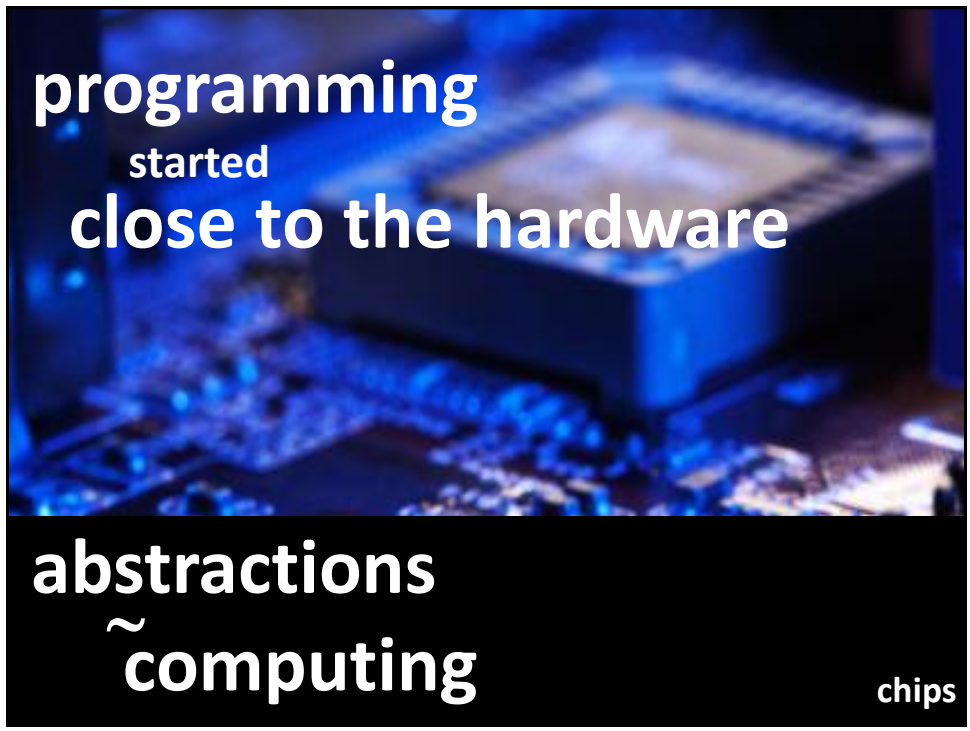
No formal spec – **correctness**?

Solution Approach:

Model-Driven Software Devlopment

Domain-Specific Languages
Code Generation

1
A little History

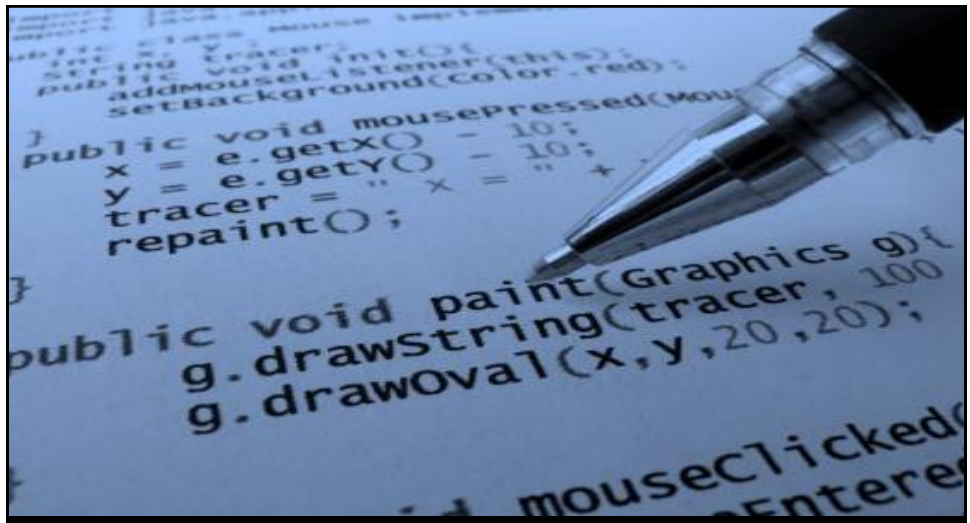

programming
started
close to the hardware
abstractions
~computing
chips

abstractions
~computing

bits

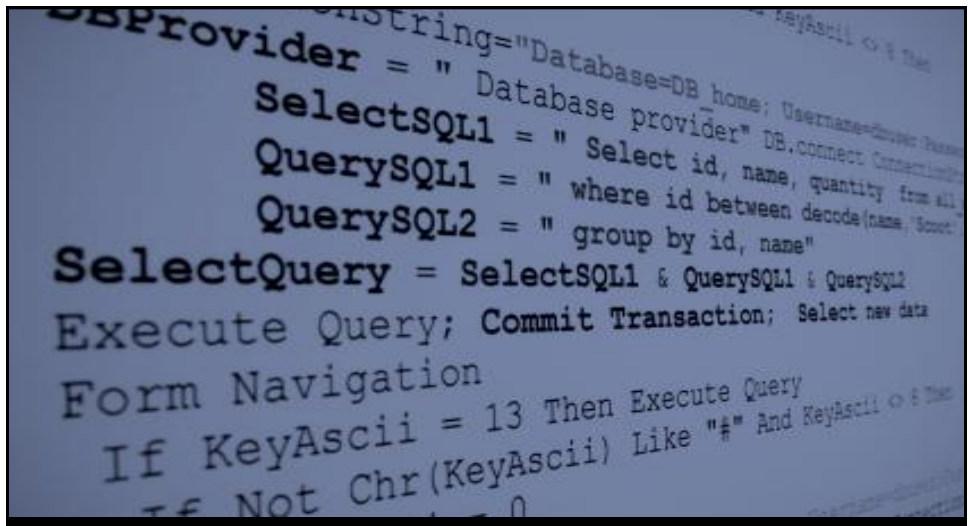
abstractions
~computing

c



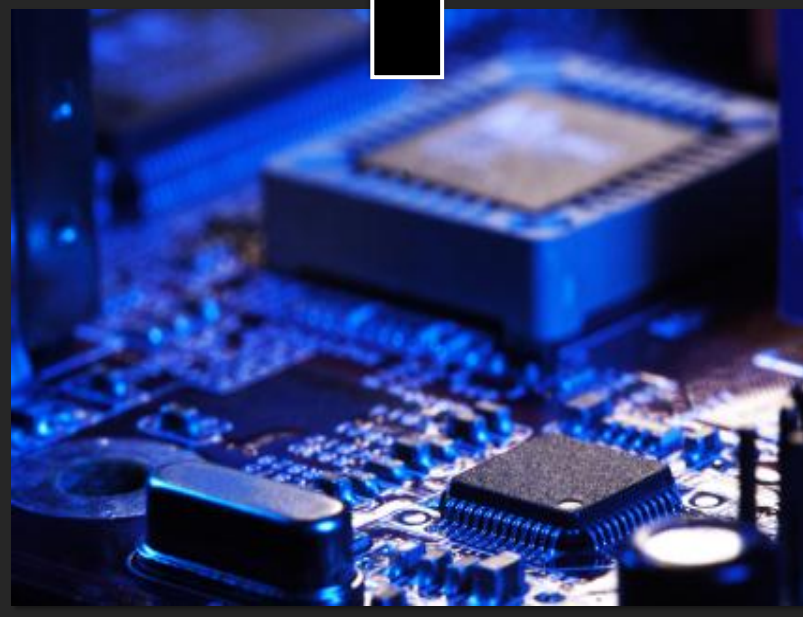
abstractions
~computing?

Java

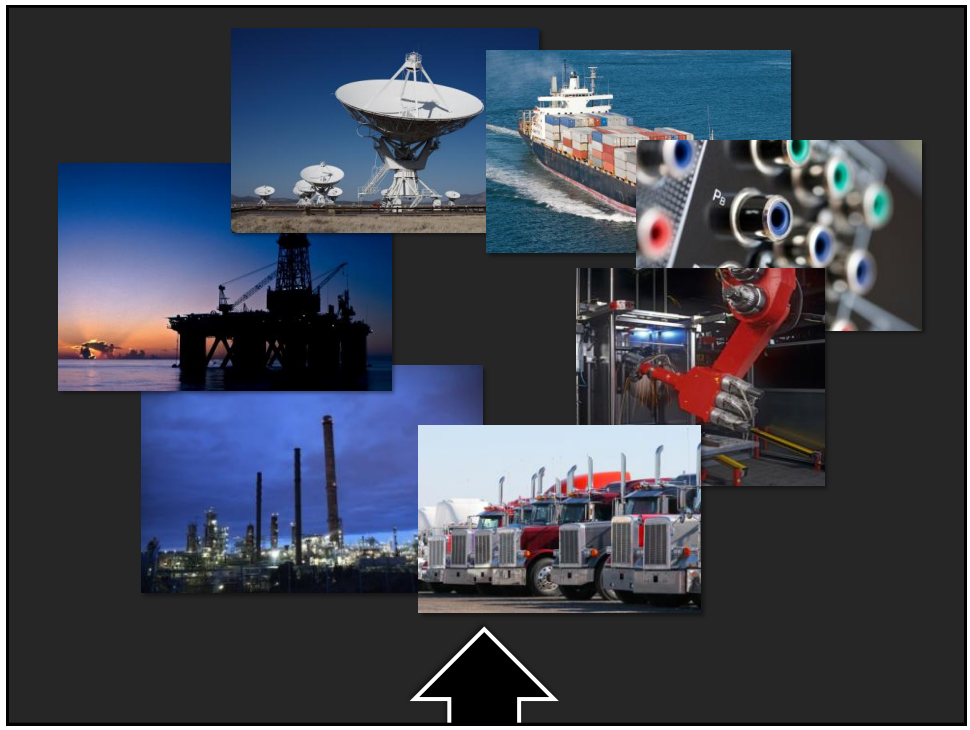


abstractions
~computing?

SQL



?



general purpose

domain specific

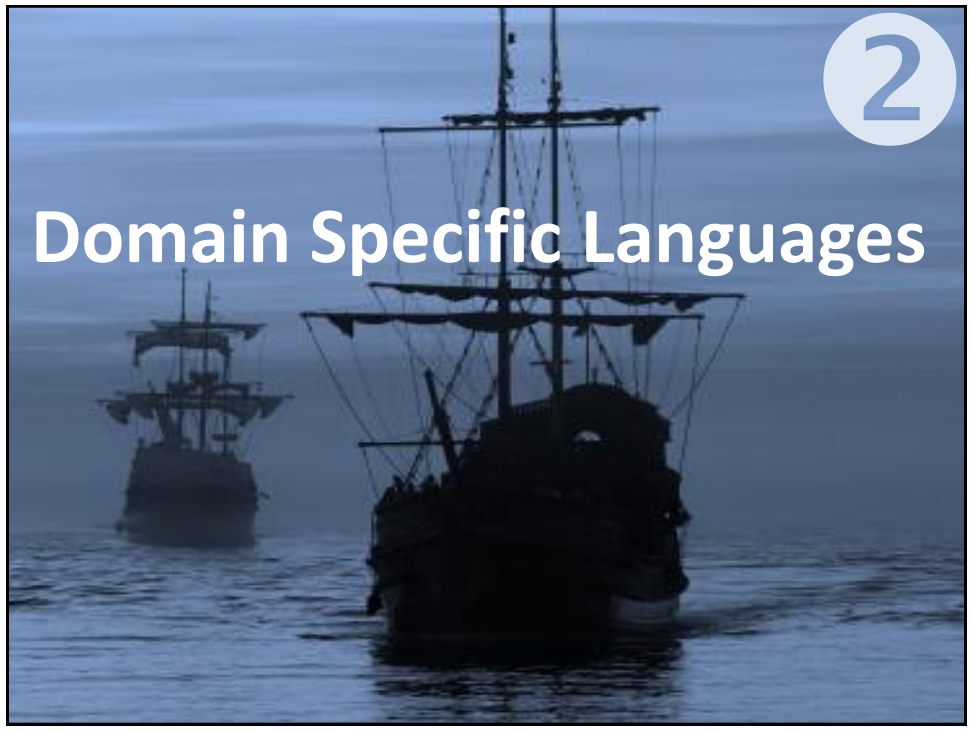
tailor made

effective++

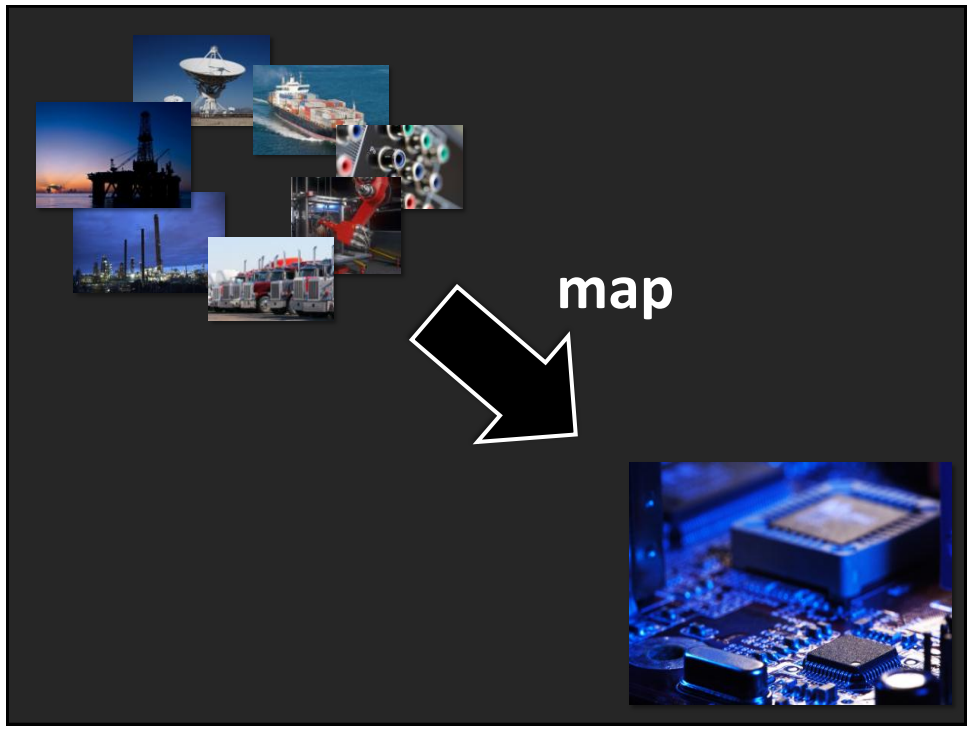
specialized, limited

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.

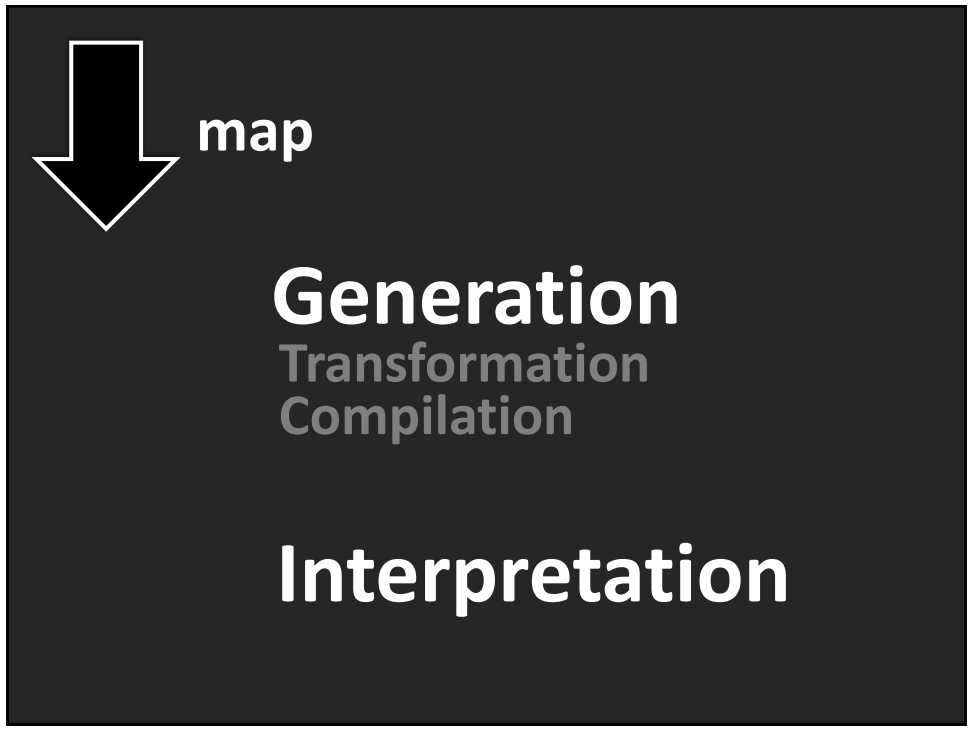


DSL Program
(aka Model)

map

automated!

GPL Program


map
Generation
Transformation
Compilation
Interpretation

LanguageOrientedProgramming
DomainSpecificModeling
GenerativeProgramming
ModelDrivenDevelopment
ModelIntegratedComputing
ModelDrivenEngineering
ModelDrivenArchitecture
ModelBasedDevelopment
ModelBasedEngineering
ModelDrivenSoftwareDevelopment

Activities

Analysing Domains

Defining Languages

Adapting/Selecting

Building Editors

Transforming Models

Building Generators

Building Frameworks

… and using all of that to build apps

Architecture

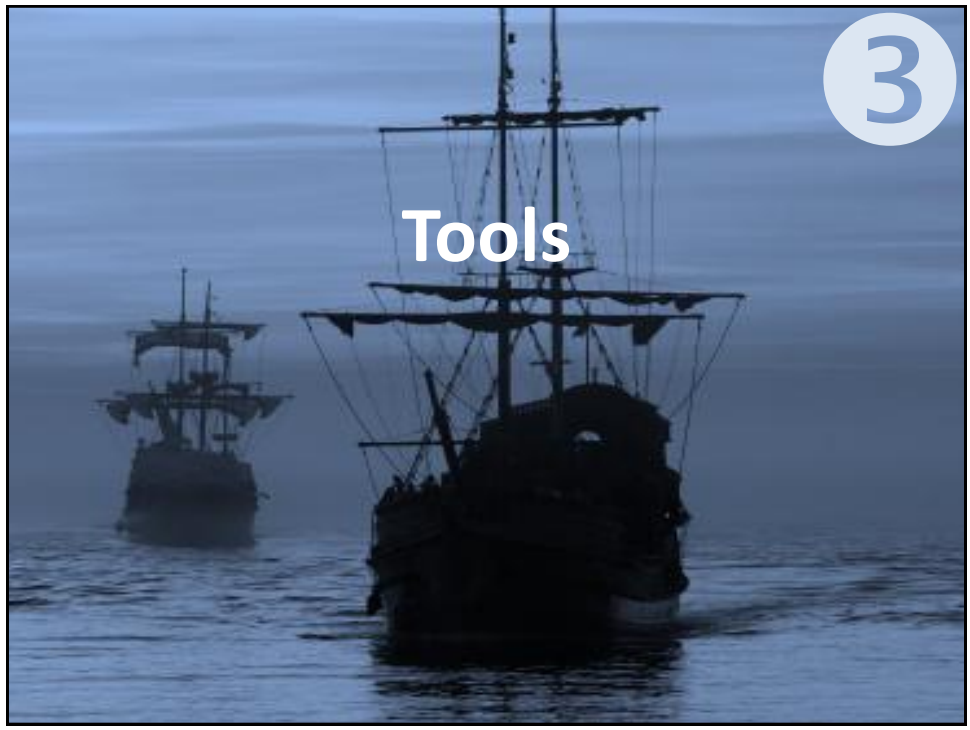
DSL

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.


3
Tools

Tooling!
Editor, Debugger, Testing, Groupware,
Scalable, „All in Eclipse“

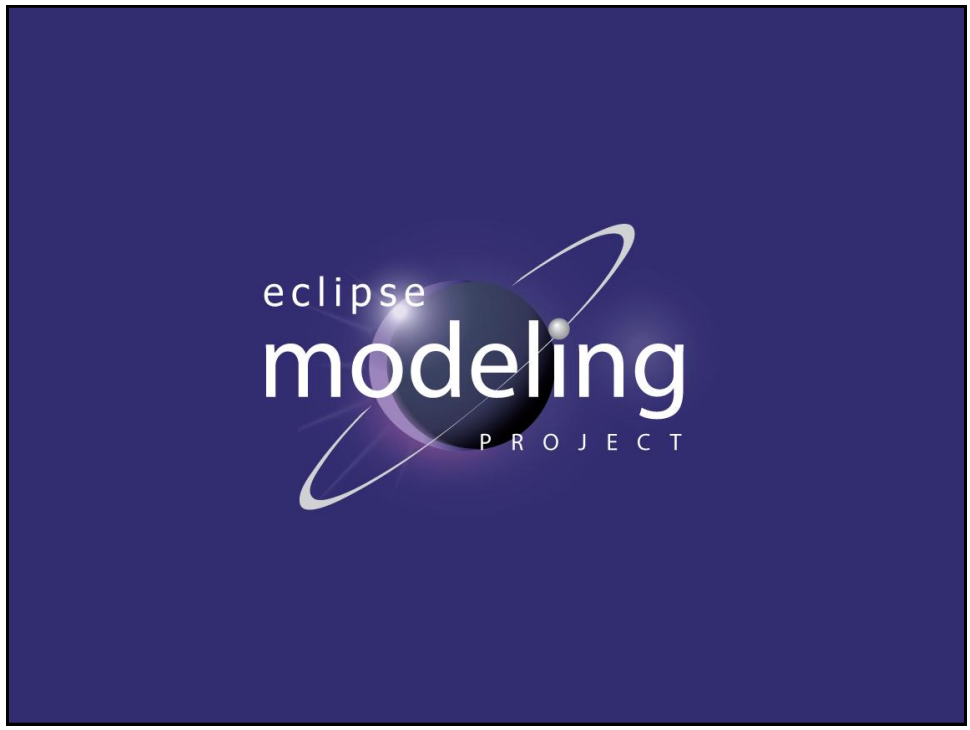
Tools Tooling

Language Definition Tools
abstract syntax, concrete syntax, constraints

Editor Frameworks

Transformation Languages

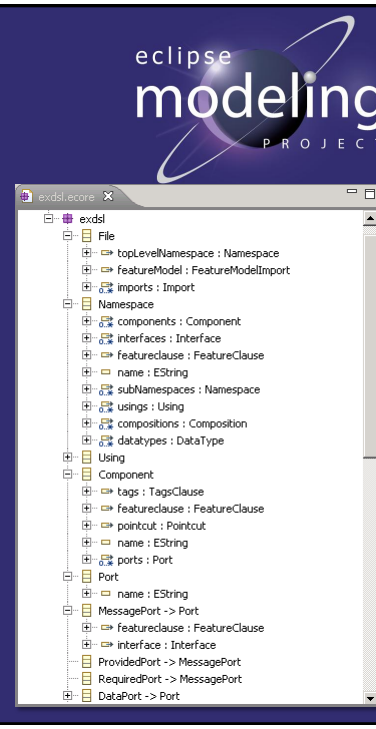
Code Generation Tools



EMF

Metamodel As Tree

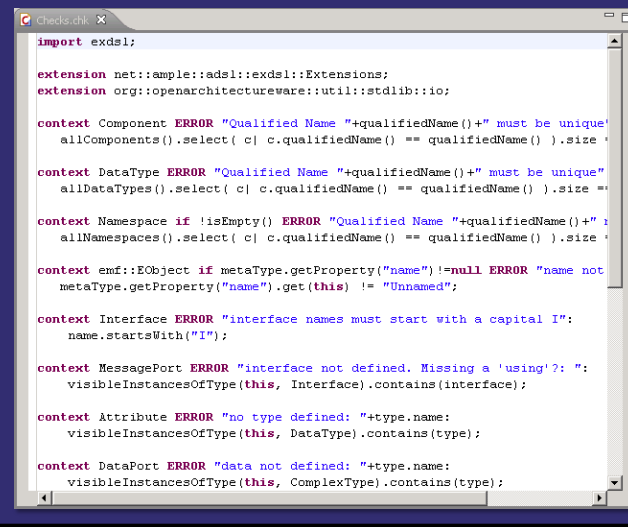
can also be edited
as UML-like diagram



EMF

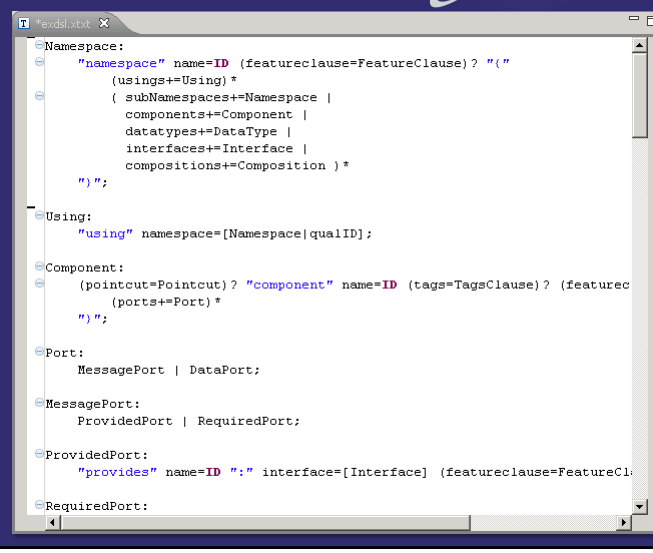
Constraints

with OCL
and dialects



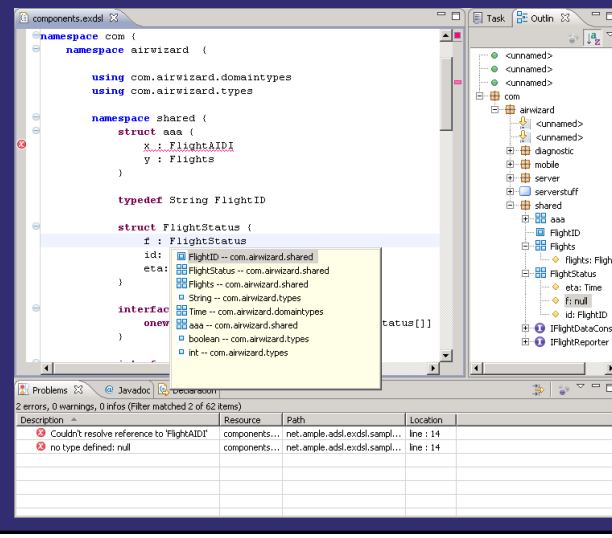
TMF / Xtext

Building Textual Editors



TMF / Xtext

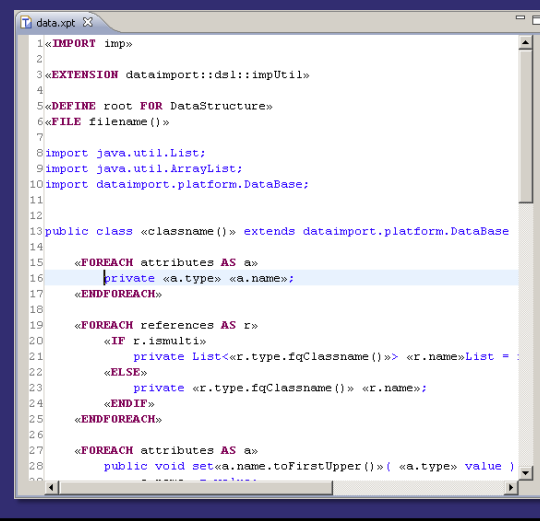
Building Textual Editors

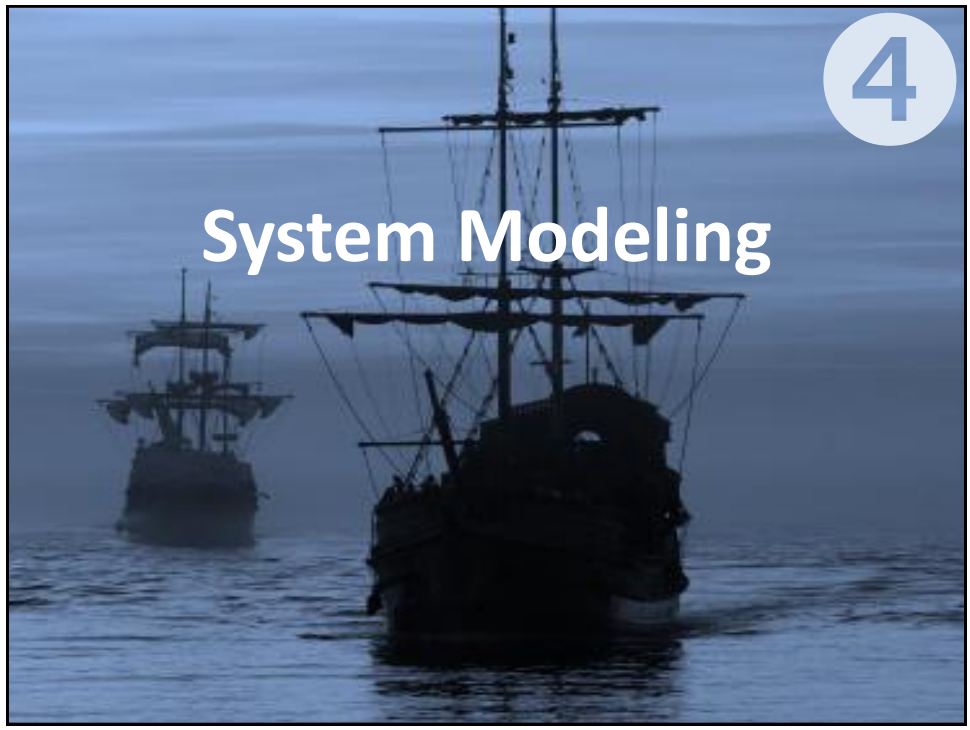


M2T

Model-to-Text Transformations

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





Leitstandssoftware

A central coordinator

Many „plug-in“ **cards**

With different **configs**

With different **hardware**

Running **components**

Communicating via **bus**

component definition

```
processing DigitalIn "BI" moduletype 0x08 hal = DigitalInHAL  {

    datatypes {
        SinglePointIndicationWithoutTime;
        SinglePointIndicationWithTime;
        DoublePointIndicationWithoutTime;
        DoublePointIndicationWithTime;
        BitStringTypeI8BitWithoutTime;
        BitStringTypeI8BitWithTime;
    }

    parametertypes {
        DataType default {
            subattr db0 # intendedDataType == pdt SinglePointIndicationWithTime;
        };
        DebounceFilterTime default {
            attr filterTimeInMs == 0x02;
            subattr db1 # SP == 0x00;
            subattr db1 # IN == 0x00;
        } ;
        MaximumOscillatingFrequency;
    }

    function READDATA () : ProcessData;
    function WRITEDATA(input : ProcessData);

    struct ProcessData {
        int8 channel;
        int8 fixData[4];
    }

    struct Memory {
        int8 state;
        ProcessData data;
    }

    instance memory Memory ;
}
```

Hardware specifics

```
hal DigitalInHAL memory DID {
    command HighSpeedCounterInit;
    command HardwareInit;

    struct DID {
        int16 ch1_16;
        int32 hs_counter;
    }
}

halImpl DigitalInHALImpl for DigitalInHAL;
halImpl WinBiHALImpl for DigitalInHAL;
```

bus protocol spec

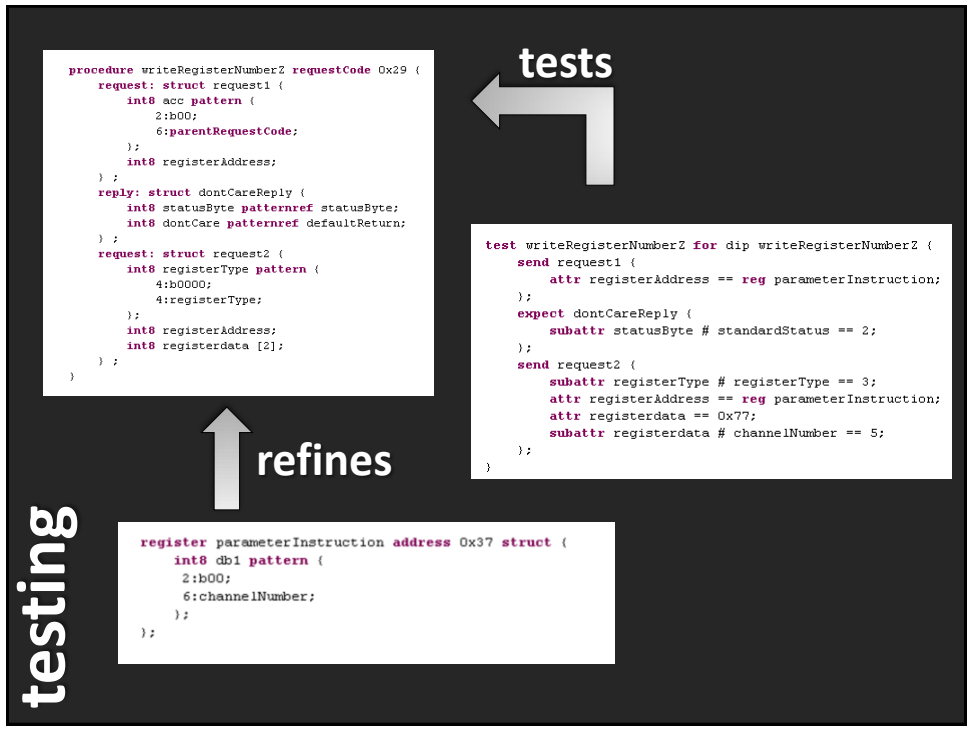
```
procedure writeRegisterNumberZ requestCode 0x29 {
    request: struct request1 {
        int8 acc pattern {
            2:b00;
            6:parentRequestCode;
        };
        int8 registerAddress;
    } ;
    reply: struct dontCareReply {
        int8 statusByte patternref statusByte;
        int8 dontCare patternref defaultReturn;
    } ;
    request: struct request2 {
        int8 registerType pattern {
            4:b0000;
            4:registerType;
        };
        int8 registerAddress;
        int8 registerdata [2];
    } ;
}
```

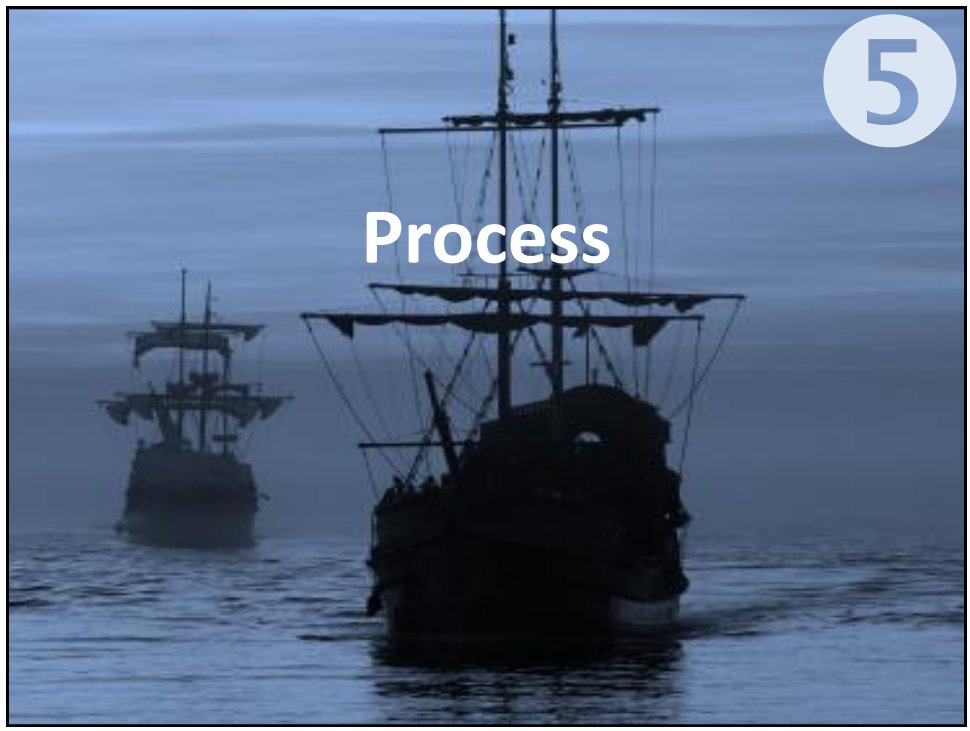
System/card definition

```
system DemoInput target atmel extendedModuleType 0x30 rack 1 {

    dip: DigitalIn slot 1 with DigitalInHALImpl channels 16
        using datatypes {
            SinglePointIndicationWithoutTime;
            SinglePointIndicationWithTime;
            BitStringTypeI8BitWithoutTime;
            BitStringTypeI8BitWithTime;
        };

    dispatcher {
        processingcomponent DigitalIn {
            on normalizeModule;
            on readStandardStatus;
            on setModuleInService;
            on readDataOfInputChannelX;
            on repeatLastReply;
            on resetModule;
            on writeRegisterNumberZ;
        }
    }
}
```


testing
tests
refines
procedure writeRegisterNumberZ requestCode 0x29 {
request: struct request1 {
int8 acc pattern {
2:b00;
6:parentRequestCode;
};
int8 registerAddress;
} ;
reply: struct dontCareReply {
int8 statusByte patternref statusByte;
int8 dontCare patternref defaultReturn;
} ;
request: struct request2 {
int8 registerType pattern {
4:b0000;
4:registerType;
};
int8 registerAddress;
int8 registerdata [2];
} ;
}
test writeRegisterNumberZ for dip writeRegisterNumberZ {
send request1 {
attr registerAddress == reg parameterInstruction;
};
expect dontCareReply {
subattr statusByte # standardStatus == 2;
};
send request2 {
subattr registerType # registerType == 3;
attr registerAddress == reg parameterInstruction;
attr registerdata == 0x77;
subattr registerdata # channelNumber == 5;
};
}
register parameterInstruction address 0x37 struct {
int8 db1 pattern {
2:b00;
6:channelNumber;
};
};

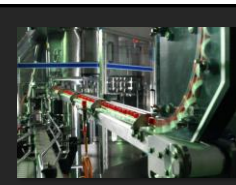

5
Process

Initial Familiarization Workshop
Reference Implementation
„Meta Stuff“ in parallel
DSL developed incrementally
DSL forces decisions -> spec++
Fallback: continue manually
Testing: first class citizen

Capture Domain Knowledge

formalized into languages and models

… Formal Definition of Bus Protocol and Data Structures
… Components, Interfaces

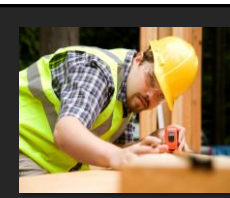


Capture Implementation Strategy

in the generators

… How to implement Dispatcher
… Define the API for accessing data and interfaces
… Data Structures (down to bit level)

Automation
faster, deterministic



Increased Quality
well defined structures allthrough the system

Meaningful Validation

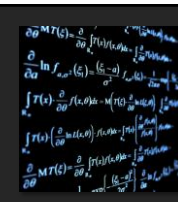
more semantics in the model

… Are all protocol messages handled?
… Runtime: Diagnostics

Testing Support

more semantics in the model

… models to “base” test cases on
… generate test cases from separate test models using the protocol definition



Suitable Notations

textual, graphical, tabular

- … models as documentation
- … generated diagrams/visualizations
- … traceability to requirements docs

Technology Independence

generate „technology glue code“

- … simple migration to new platforms (through regeneration)
- … hand-written code against generated, platform-independent APIs
- … reuse components

Abstraction w/o Runtime Overhead

generator „optimizes away“

… meaningful architectural concepts
… generator generates efficient code
… PLE variability is statically resolved

Everything is a model

including for example (some) hardware

… bus protocol
… hardware structures
… channels

MDSD
does not result in
loss of
efficiency
for embedded systems

Open Source MDSD tools are **mature**

MDSD increases **quality of requirements**.

MDSD is

cost-effective.

Strong benefits in

automated testing.

Enforces

Modularization.

External Coach
can help to
introduce
the new approach

Iterate!

Generator
Editor Customization
Examples
Grammar
Concepts
Constraints

THE END.