Model-Driven Development in the context of embedded automation systems
The Challenge
Software increasingly complex
Software increasingly complex

Faster innovation cycles
Software increasingly complex

Faster innovation cycles

Product Lines - Variability
Software increasingly complex

Faster innovation cycles

Product Lines - Variability

No formal spec – correctness?
Solution Approach:

Model-Driven Software Development

Domain-Specific Languages
Code Generation
A little History
programming
started
close to the hardware

abstractions
≈ computing
abstractions
≈ computing
bits
abstractions
≈ computing
abstractions
～computing?
SQL abstractions computing?
general purpose
domain specific
tailor made
effective++
specialized, limited
used by experts
together with other specialized tools
Domain Specific Languages
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?
DSL Program
(aka Model)

automated!

map

GPL Program
map

Generation
Transformation
Compilation

Interpretation
Analysing Domains
Defining Languages
Adapting/Selecting
Building Editors
Transforming Models
Building Generators
Building Frameworks
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.
Tools
Tooling!
Tooling!
Editor
Tooling!
Editor, Debugger
Tooling!
Editor, Debugger, Testing
Tooling!
Editor, Debugger, Testing, Groupware
Tooling!
Editor, Debugger, Testing, Groupware, Scalable
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
eclipse modeling
PROJECT
EMF

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

Constraints with OCL and dialects
Building Textual Editors

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

using namespace=[Namespace|qualID];

component (pointcut=Pointcut)? "component" name=ID (tags=TagsClause)? (featureclause=FeatureClause)* 
"}";

port MessagePort | DataPort;

MessagePort: 
  ProvidedPort | RequiredPort;

ProvidedPort: 
  "provides" name=ID ":" interface=[Interface] (featureclause=FeatureClause)* 

RequiredPort:
```
TMF / Xtext

Building Textual Editors
M2T

Model-to-Text Transformations

JET: Java Emitter Templates

Xpand: oAW’s template engine
System Modeling
Leitstandsoftware
Leitstandssoftware
A central coordinator
Leitstandssoftware
A central coordinator
Many „plug-in“ cards
Leitstandssoftware
A central coordinator
Many „plug-in“ cards
With different configs
Leitstandssoftware
A central coordinator
Many „plug-in“ cards
With different configs
With different hardware
Leitstandssoftware
A central coordinator
Many „plug-in“ cards
With different configs
With different hardware
Running components
Leitstandsoftware
A central coordinator
Many „plug-in“ cards
With different configs
With different hardware
Running components
Communicating via bus
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

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

struct DID {
    int16  ch1_16;
    int32  hs_counter;
}  

halImpl DigitalInHALImpl for DigitalInHAL;
halImpl WinBiHALImpl for DigitalInHAL;
```
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];
  };
}
```plaintext
system DemoInput target atmel extendedModuleType 0x30 rack 1 {

dip: DigitalIn slot 1 with DigitalInHALImpl channels 16
using datatypes {
    SinglePointIndicationWithoutTime;
    SinglePointIndicationWithTime;
    BitStringType10BitWithoutTime;
    BitStringType10BitWithTime;
}

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

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;
    };
};
Process
Initial Familiarization Workshop
Initial Familiarization Workshop
Reference Implementation
Initial Familiarization Workshop
Reference Implementation
„Meta Stuff“ in parallel
Initial Familiarization Workshop Reference Implementation „Meta Stuff“ in parallel DSL developed incrementally
Initial Familiarization Workshop
Reference Implementation
„Meta Stuff“ in parallel
DSL developed incrementally
DSL forces decisions -> spec++
Initial Familiarization Workshop
Reference Implementation
„Meta Stuff“ in parallel
DSL developed incrementally
DSL forces decisions -> spec++
Fallback: continue manually
Initial Familiarization Workshop
Reference Implementation
„Meta Stuff“ in parallel
DSL developed incrementally
DSL forces decisions \( \rightarrow \) spec++
Fallback: continue manually
Testing: first class citizen
Expected Benefits
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
Lessons Learned
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
- Examples
- Grammar
- Concepts
- Editor Customization
- Constraints
THE END.