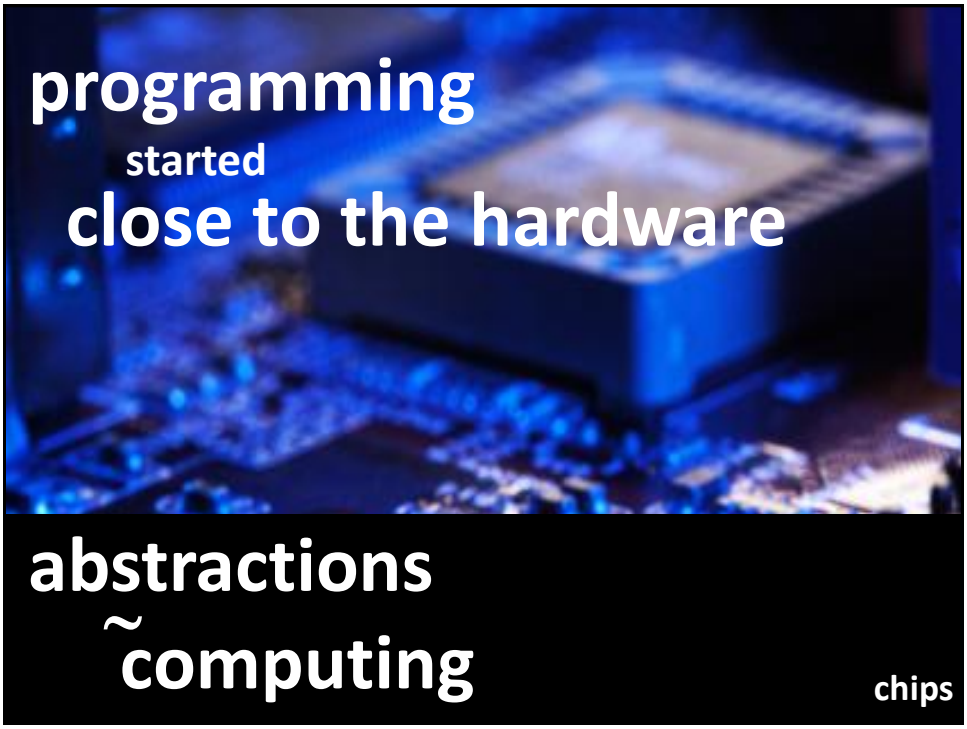
Model-Driven Development and DSLs

in the context of embedded sytems

Markus Voelter
Independent/itemis
voelter@acm.org

1. A little History
2. Domain Specific Languages
3. Examples
4. Tools
5. Benefits
6. Architecture DSLs
7. A more theoretical View
8. Programming and Modeling
9. The LWES Project
10. Bonus: Best Practices


1
A little History

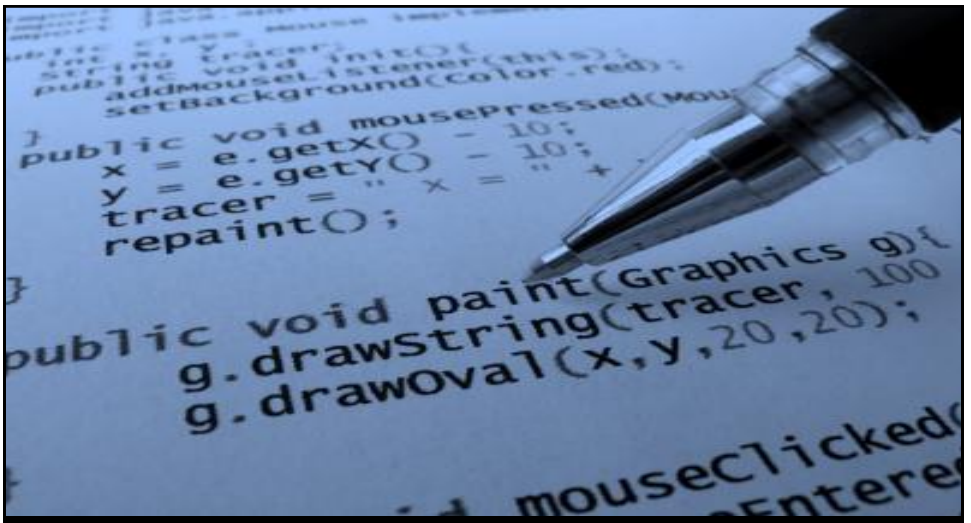

programming
started
close to the hardware
abstractions
~computing
chips

abstractions
~computing

bits

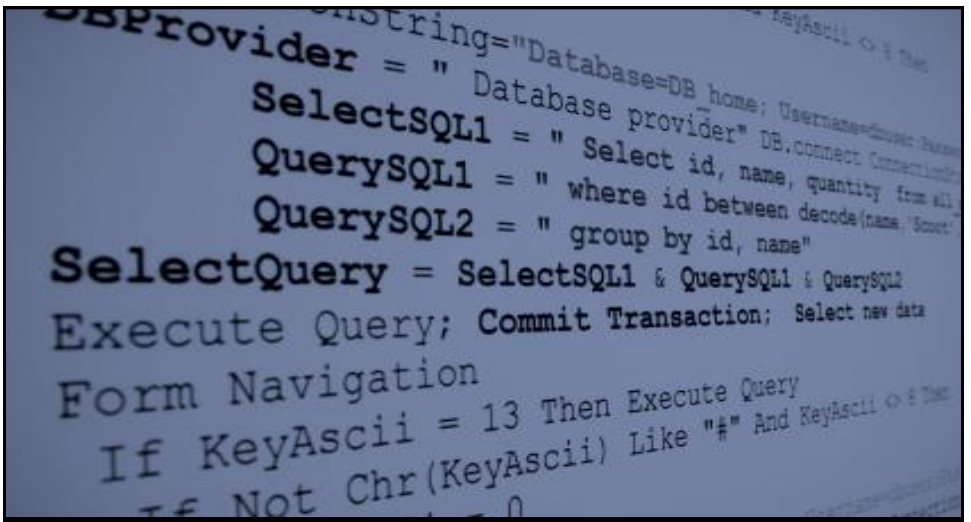
abstractions
~computing

c



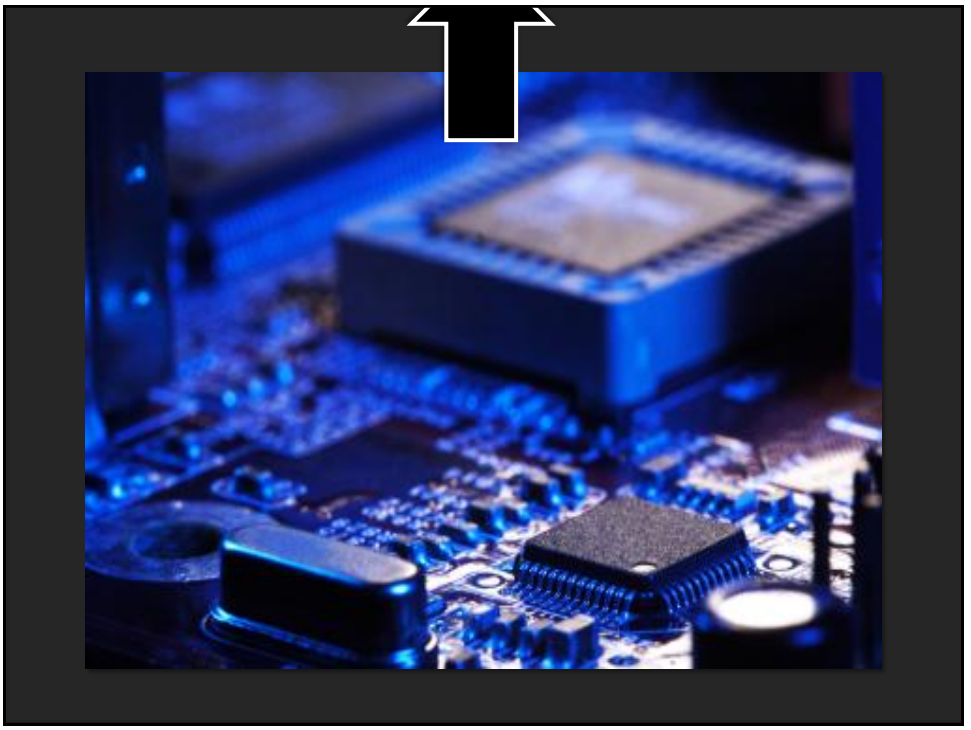
abstractions
~computing?

Java



abstractions
~computing?

SQL



general purpose

domain specific

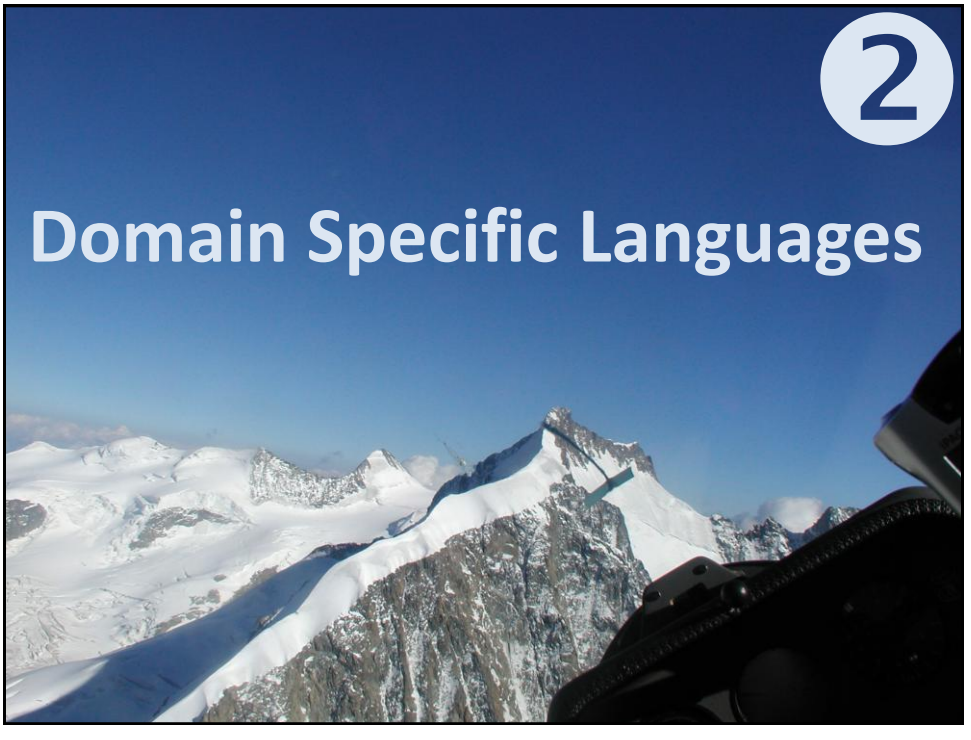
tailor made

effective++

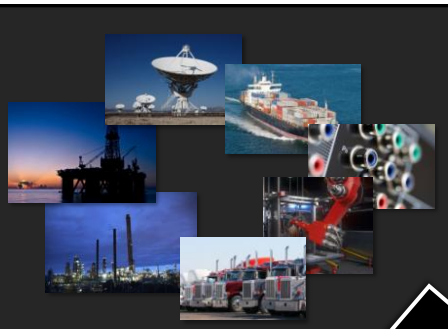
specialized, limited

used by experts

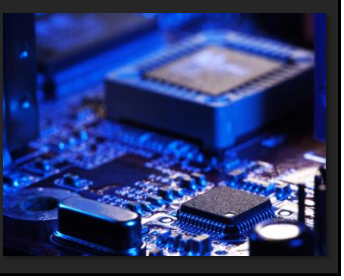
together with other specialized tools



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.



map



DSL Program
(aka Model)

map

automated!

GPL Program

map

Generation
Transformation
Compilation

Interpretation

Activities

Analysing Domains

Defining Languages

Adapting/Selecting

Building Editors

Transforming Models

Building Generators

Building Frameworks

… and using all of that to build apps

internal

vs.

external

compiled

vs.

interpreted

customization

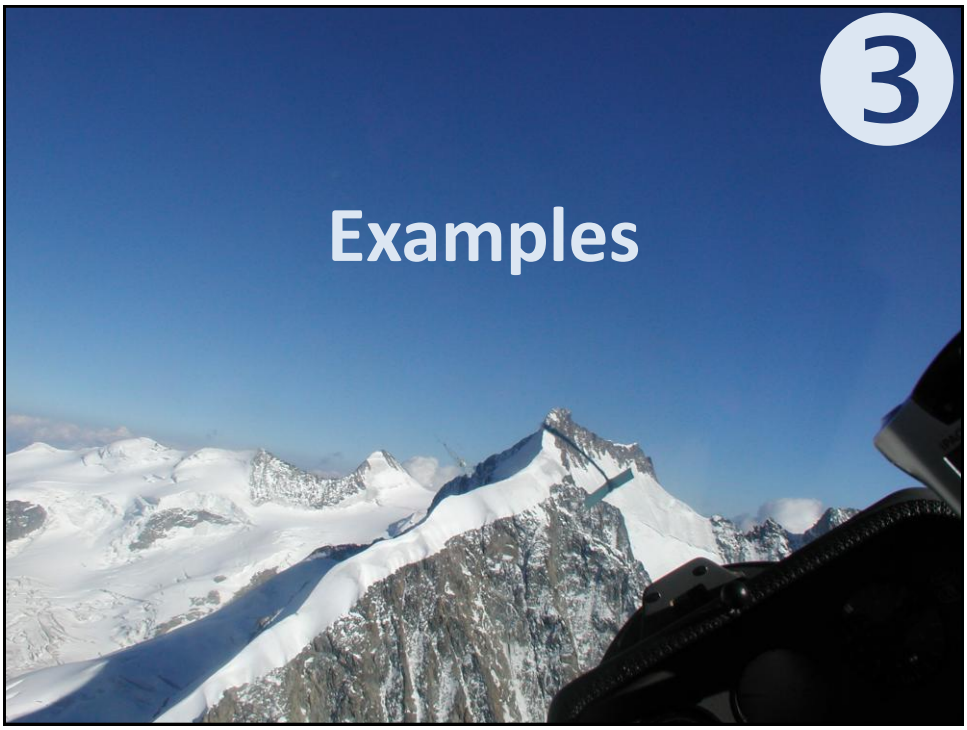
vs.

configuration

graphical

vs.

textual



Example 1:

Embedded Protocol Handler

SOLUTION

Component Specification

```
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 ;
}
```

SOLUTION

Message Format Definition

```
procedure writeRegisterNumber2 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];
    } ;
}
```

SOLUTION

Testing

```
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];
    } ;
}
```

tests

```
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;
    };
}
```

refines

```
register parameterInstruction address 0x37 struct {
    int8 db1 pattern {
        2:b00;
        6:channelNumber;
    };
};
```

TOOLS

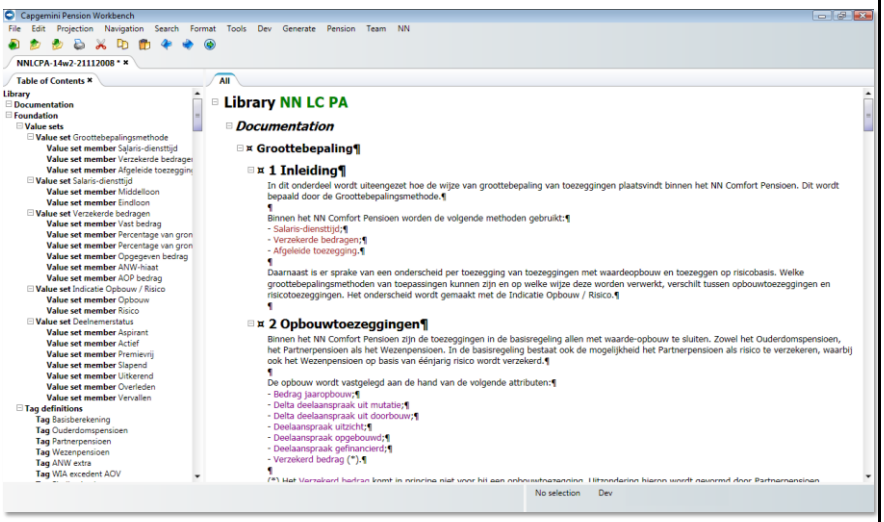
Eclipse Modeling
Eclipse Xtext

Example 2:

Pension Fund Specification

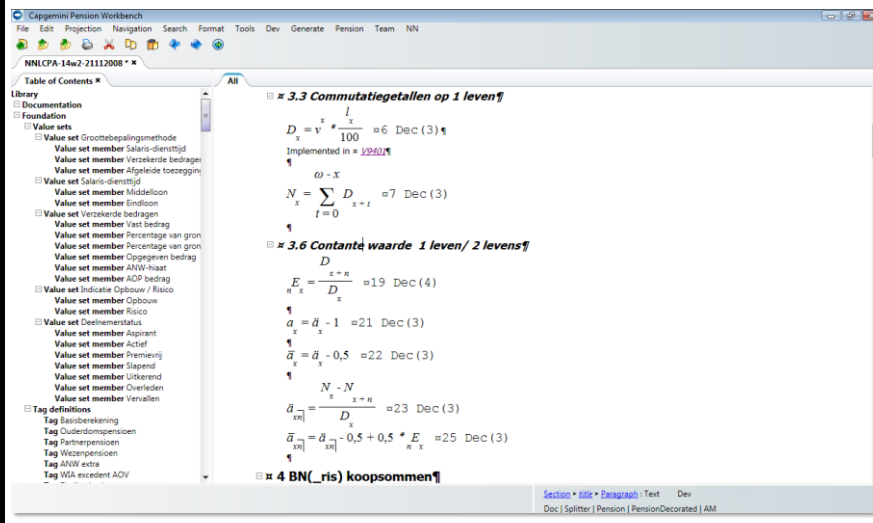
SOLUTION

Textual Documentation



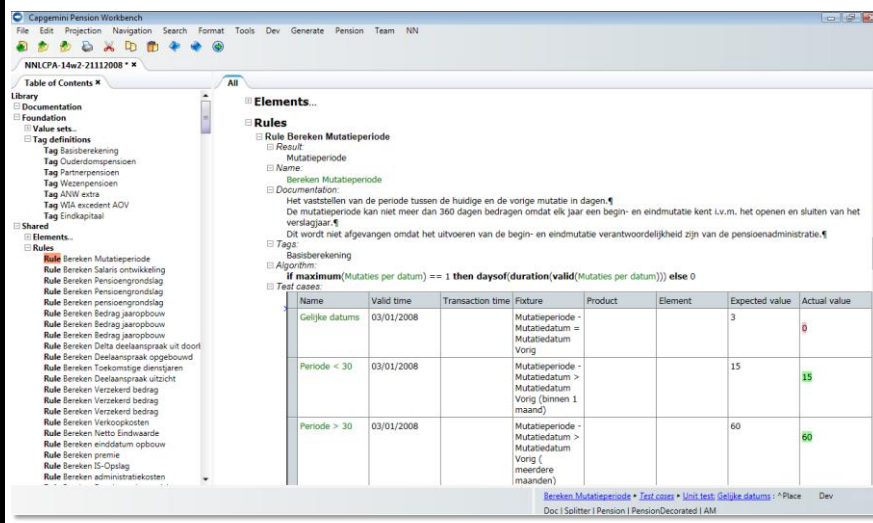
SOLUTION

Insurance Mathematics



SOLUTION

Calculation Rules and Tests



TOOLS

INTENTIONAL SOFTWARE

Intentional Software's Domain Workbench

Example 3:

Radar Systems Engineering

SOLUTION

Component Definition

test.cdsl | test.mm

```
import "classpath:/test.mm"

quantity voltage is double
quantity temperature is double

source component Sensor (
    produces m: (measurement: voltage, sensortemp: temperature)
    behavior {
        m.measurement <= sensorM[]
        m.sensortemp <= sensorT[]
    }
)

processing component TempCalibration (
    consumes input: Sensor::m
    produces calibrated: ( measurement: voltage )
    behavior {
        calibrated.measureme
    }
)

processing component Process
    consumes input: ( measur
    produces earthtemp: ( te
    behavior {
        earthtemp.temp <= pr
    }
)
```

```
sink component Output (
    consumes t : Processor::earthtemp
)

system satellite (
  s: Sensor
  tc: TempCalibration
  p: Processor
  o: Output

  s.m -> tc.input
  tc.calibrated -> p.input
  p.earthtemp -> o.t

  export o.t.temp as temperature
  export s.m.measurement as originalMeasurement

)
```

SOLUTION

Component Behavior Specification

ComponentDSL.xtext | test.m

```
BeginPackage["MappingSatellite`"]

Begin["`Private`"]


sensorM[t_Int] := 23*t

sensorT[t_Int] := 300

calibrate[ m_Double, temp_Double ] := m - temp/10

process[ m_Double ] := m*3


End[ ]

EndPackage[ ]

(* Mathematica Raw Program *)
```

SOLUTION

Resulting System Behaviour

satellite.m

```
BeginPackage["satellite`"]
Begin["`Private`"]


temperature[t_Int] := process[ calibrate[ sensorM[ t ], sensorT[ t ] ] ]

originalMeasurement[t_Int] := sensorM[ t ]


End[ ]
EndPackage[ ]
```

SOLUTION

Analysis

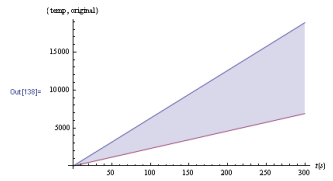
Untitled-17

Wolfram Mathematica | PRODUCT TRIAL

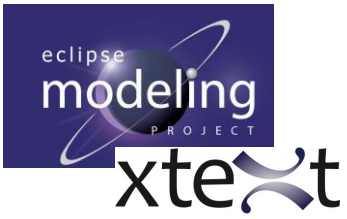
Learning Center | Help | Contact Us | Buy Mathematica

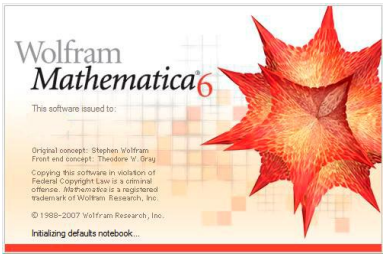
```
In[123]:= Import["L:\\mathematica-test-35\\mmtest\\src-gen\\satellite.m"]

In[138]:= Plot[{temperature[x], originalMeasurement[x]}, {x, 0, 300},
           Filling → {1 → {2}}, AxesLabel → {t[s], {temp, original}}]
```

Out[138]=



TOOLS





Eclipse Modeling
Eclipse Xtext
Wolfram Mathematica
Mathematica Workbench

Example 4:

Alarm System Menus

SOLUTION

Menu Structure

```
import "classpath:/units.md"
import "classpath:/software.swc"

namespace s1

    uses units

    condition Locked
    condition BlinkingLight

    menu Normal label "Standardmenue"
        item unlockNow sys(TurnOffAlarm) if Locked
            button label "Unlock"
        submenu Manual label "Manual Settings"
            item alarmLevel sys(AlarmLevel)
                valuerange SoundLevel restrict 10..80
            item useLight sys(TurnOffAlarm) if BlinkingLight
                bool
        end
        submenu autoLocking label "Automatic Locking"
            item startTime sys(TurnOnAlarm)
                valuerange Time
            item endTime sys(TurnOffAlarm)
                valuerange Time
            template areaSettings [size=15, area=1, sw=sys(TurnOnAlarm)] area1Settings
            template areaSettings [size=10, area=2, sw=sys(TurnOnAlarm)] area2Settings
        end
    end

    template [size: int, area: int, sw: swref] areaSettings
        item onOrOff sys(TurnOffAlarm) labelexpr "Autolock "+area+" on/off"
            bool true = label(size) "On" false = label "Off"
        item test sys(AlarmLevel) label "Test"
            bool
        item alarmLevel sys(AlarmLevel)
            valuerange SoundLevel restrict size..80
    end

    menu Expert extends Normal
        item master sys(UnlockNow) afterItem unlockNow
            bool
    end


end
```

SOLUTION

Software Components

```
message TurnOffAlarm
message TurnOnAlarm
message AlarmLevel
message UnlockNow

component AlarmManager {
    receives TurnOffAlarm
    receives TurnOnAlarm
    receives AlarmLevel
}

component MasterSwitch {
    receives UnlockNow
}
```

TOOLS

Eclipse Modeling
Eclipse Xtext

Example 5:

Requirements Tracability

SOLUTION

Imported Requirements

DummyRequirementsCollection

```
DUMMY REQUIREMENTS (to be replaced by interface to real RE tool)
show trace true
```

Init	The system should start operating only after it has been initialized property refines TwoPhases
Efficient	The program should be as small regarding memory footprint as possible refines Init refines MaxSpeed
Cyclic	the actual control of the device should be based on a cyclic task
Calibration	The black/white values should be easily calibrated
MaxSpeed	Speeds per motor can only be up to 80
OptionalOutputlkjlkjljlkj	Display output should be optional
TwoPhases	Initialization should be separate from operation
ConsistentSetting	Motor settings have to be updated consistently

SOLUTION

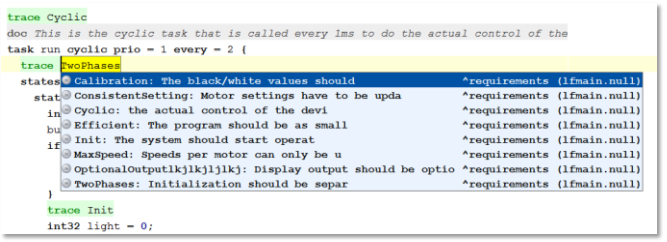
Program Code with Annotations (green)

DummyRequirementsCollection | LineFollower

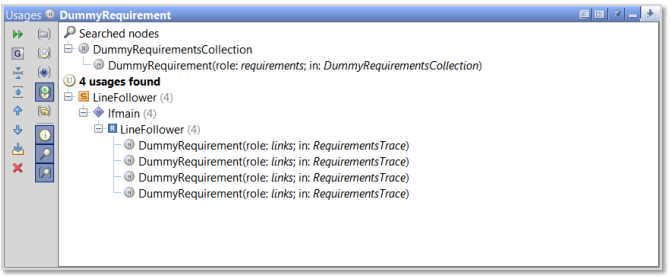
```
trace Cyclic
doc This is the cyclic task that is called every 1ms to do the actual control of the
task run cyclic prio = 1 every = 2 {
  trace TwoPhases
  stateswitch linefollower
    state running
      int8 bump = 0;
      bump = ecrobot_get_touch_sensor(SENSOR_PORT_T::NXT_PORT_S3);
      if ( bump == 1 ) {
        event linefollower:bumped
        terminate;
      }
      trace Init
      int32 light = 0;
      light = ecrobot_get_light_sensor(SENSOR_PORT_T::NXT_PORT_S1);
      if ( light < ( WHITE + BLACK ) / 2 ) {
        trace ConsistentSetting        ;
        updateMotorSettings(SLOW, FAST)
      } else {
        trace ConsistentSetting        ;
        updateMotorSettings(FAST, SLOW)
      }
    state crash
      updateMotorSettings(0, 0);
    default
      <noop>;
}
```

SOLUTION

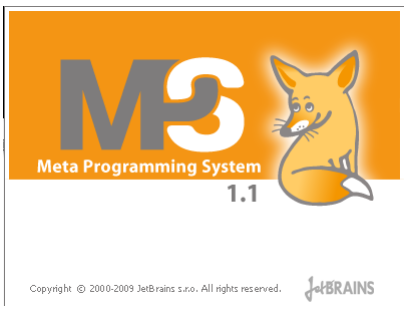
Selecting from the Requirements



Find Usages of Requirements



TOOLS



JetBrains MPS

Example 5:
OSGi-based System

SOLUTION

Component Specification

```
subsystem the.world.scenarios {

    public:

        immutable type ProblemReport {
            problem: string
            severity: int
            emergency: bool
        }

        interface Radio {
            report( a: ProblemReport ): void
        }

        interface Press {
            broadcast( a: string ): string
        }

    private:

        component Houston {
            provides uplink: Radio
            requires press: Press
        }

        component PrintingPress {
            provides source: Press
        }

}

subsystem the.moon.scenarios {

    uses the.world.scenarios

    private:

        component Armstrong {
            task sayHello scheduled onceUponStartup
            requires home: Radio [0..1]
        }

}
```

SOLUTION

Component Specification

```
subsystem the.world.scenarios {

    public:

        immutable type ProblemReport {
            problem: string
            severity: int
            emergency: bool
        }

        interface Radio {
            report( a: ProblemReport ): void
        }

        interface Press {
            broadcast( a: string ): string
        }

    private:

        component Houston {
            provides uplink: Radio
            requires press: Press
        }

        component PrintingPress {
            provides source: Press
        }

}

subsystem the.moon.scenarios {

    uses the.world.scenarios

    private:

        component Armstrong {
            task sayHello scheduled onceUponStartup
            requires home: Radio [0..1]
        }

}
```

```
import "classpath:/twoworlds.compds1"

scenario Simple : the.moon.scenarios, the.world.scenarios
    actor IRP ==> { Armstrong |

    var $m = ProblemReport {
                severity = 12
                emergency = true
                problem = "there's cheese"
            }

    home.report(msg#$m) -> { Houston |
        var $res = press.broadcast( < #msg.problem > ) -> {
            return "thanks"
        }
        log.info "they replied "
        log.info $res
        assert equals returnValueIsThanks : $res == "thanks"
        var $res2 = press.broadcast( < #msg.problem > ) -> {
            return "again"
        }
        log.info "and then they replied "
        log.info $res2
        assert equals returnValueIsAgain : $res2 == "again"
    }

}
```

647x551

SOLUTION

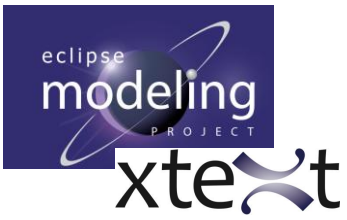
Component Specification

```
subsystem the.world.scenarios {

    public:

        immutable type ProblemReport {
            problem: string
            severity: int
            emergency: bool
        }

        interface Radio {
            report( a: ProblemReport ): void
        }

        interface Press {
            broadcast( a: string ): string
        }

    private:

        component Houston {
            provides uplink: Radio
            requires press: Press
        }

        component PrintingPress {
            provides source: Press
        }

}

subsystem the.moon.scenarios {

    uses the.world.scenarios

    private:

        component Armstrong {
            task sayHello scheduled onceUponStartup
            requires home: Radio [0..1]
        }

}
```

```
import "classpath:/twoworlds.compds1"

scenario Simple : the.moon.scenarios, the.world.scenarios
    actor IRP ==> { Armstrong |

    var $m = ProblemReport {
                severity = 12
                emergency = true
                problem = "there's cheese"
            }

    home.report(msg#$m) -> { Houston |
        var $res = press.broadcast( < #msg.problem > ) -> {
            return "thanks"
        }
        log.info "they replied "
        log.info $res
        assert equals returnValueIsThanks : $res == "thanks"
        var $res2 = press.broadcast( < #msg.problem > ) -> {
            return "again"
        }
        log.info "and then they replied "
        log.info $res2
        assert equals returnValueIsAgain : $res2 == "again"
    }

}

import "classpath:/twoworlds.compds1"
import "classpath:/twoworlds.scenario"

system SunSystem scenario Simple {
    node moon {
        subsystem the.moon.scenarios
    }
    node earth {
        subsystem the.world.scenarios
    }
}
```

647x551

TOOLS



Eclipse Modeling
Eclipse Xtext

Example 6:

Math, Science and Java

SOLUTION

```
env block Aircraft

  v       : double [m/s] / current aircraft speed

  A       : double [m m] / cross area of the wing

  c_a     : double [1  ] / Auftriebsbeiwert

  c_w     : double [1  ] / Widerstandsbeiwert

  n_wings : double [1  ] / Number of Wings
```

SOLUTION

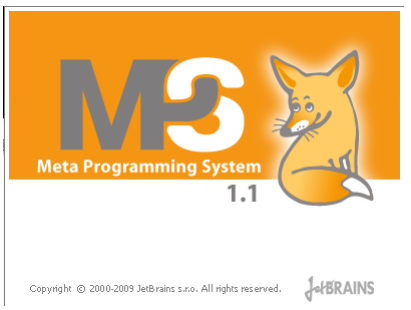
```
function block Fundamental Stuff
  uses Aircraft, Environment

  The dynamic pressure p_dyn is calculated from the current air density rho and the square of the flight speed v

  exported p_dyn : double [Pa] = 1/2 * rho * v^2        [kg/(m s s)]

    v=0 rho=1.225  -> 0
    v=10 rho=1.225 -> 61.25
    v=20 rho=1.225 -> 245
```

```
function block Stuff on the Wings
  uses Environment, Aircraft, Fundamental Stuff

  Aus dem Staudruck p_dyn lässt sich dann der aktuelle Auftrieb F_A berechnen; die Form wird
   c_a beschrieben  und die Fläche durch  A

  exported F_A : double [N] = p_dyn * A * c_a        [m kg/(s s)]

    c_a=0.3 p_dyn=61.25 A=2  -> 183.75
    p_dyn=61.25 A=10 c_a=0.6 -> 367.5

  Auch der Widerstand F_W berechnet sich entsprechend mit Hilfe des Beiwertes c_w :

  exported F_W : double [N] = p_dyn * A * c_w        [kg m/(s s)]

  Angenommen wir haben mehrere Flügel n_wings am Flugzeug, dann berechnet sich der Autrieb
```

SOLUTION

```
import blocks Environment
              Aircraft
              Stuff on the Wings
public class TestClass extends <none> implements <none> {
  <<static fields>>

  <<static initializer>>
  <<fields>>
  <<properties>>
  <<initializer>>
  public TestClass() {
    <no statements>
  }

  public void m() {
    values air = (| Environment.rho = 1.225 |);
    values planeStatic = (| Aircraft.A = 10, Aircraft.c_a = 0.5, Aircraft.c_w = 0.3, Aircraft.v = 100 |);
    double auftrieb = Stuff on the Wings.F_A (air, planeStatic);
    System.err.println(auftrieb);
  }

  <<static methods>>

  <<nested classifiers>>
}
```

TOOLS



JetBrains MPS

Example 7:

Fountains

CONTEXT

SOLUTION

Hardware Structure

```
feature BasicOnePump
    pump compartment cc1
        static compressor c1

feature AtLeastOneZone extends BasicOnePump
    water compartment comp1
        pumped by c1
        compartment levelsensor ct_f1
        light l_f1

feature[f] SuperPowerCompartment
    water compartment adds to f
        superPowerMode

feature WithAlarm
    level alarm a1

fountain StdFountain extends AtLeastOneZone
```

SOLUTION

Behaviour

```
pumping program P1 for AtLeastOneZone + WithAlarm +
                       SuperPowerCompartment[f=comp1]  {
    parameter defaultWaterLevel : int
    parameter superWaterLevel: int
    event superPowerTimeout

    init {
        set comp1->targetHeight = defaultWaterLevel
    }

    start:
        on (comp1->needsPower == true) && !(comp1->isPumping) {
            do comp1->pumpOn
        }
        on comp1->enough {
            do comp1->pumpOff
        }
        on comp1.superPumping->turnedOn {
            set comp1->targetHeight = superWaterLevel
            raise event superPowerTimeout after 20
        }
        on comp1.superPumping->turnedOff or superPowerTimeout {
            set comp1->targetHeight = defaultWaterLevel
        }
}
```

SOLUTION

Plus:

In-IDE Simulator
Unit Test Support

TOOLS

Eclipse Modeling
Eclipse Xtext

More Examples8:
Miscellaneous

Hearing Aids

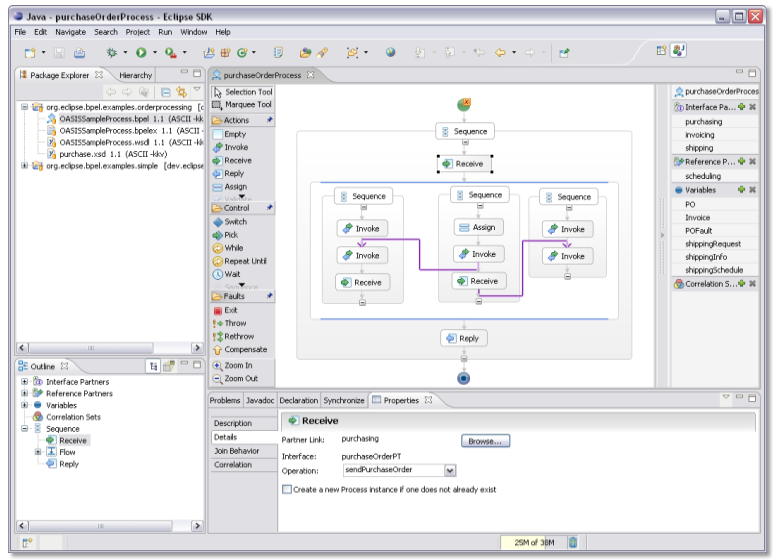
DOMAIN

DOMAIN

Refrigerators

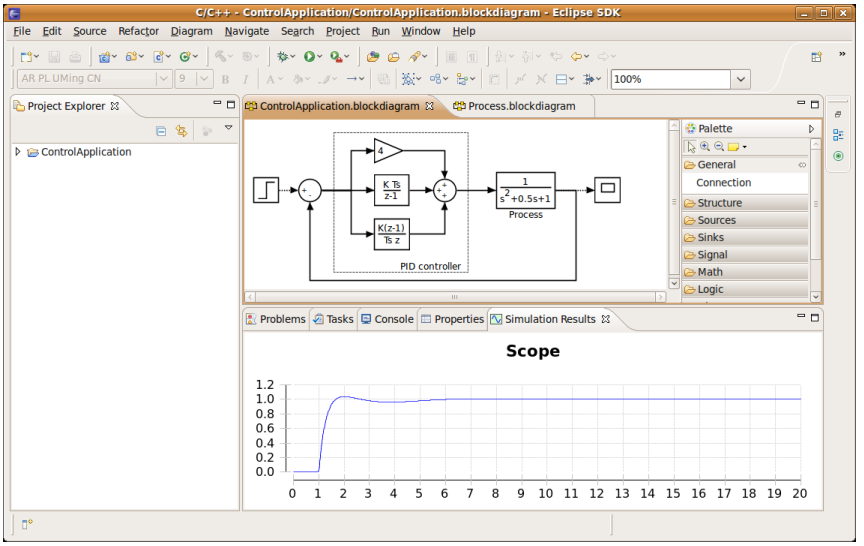
SOLUTION

BPEL Designer



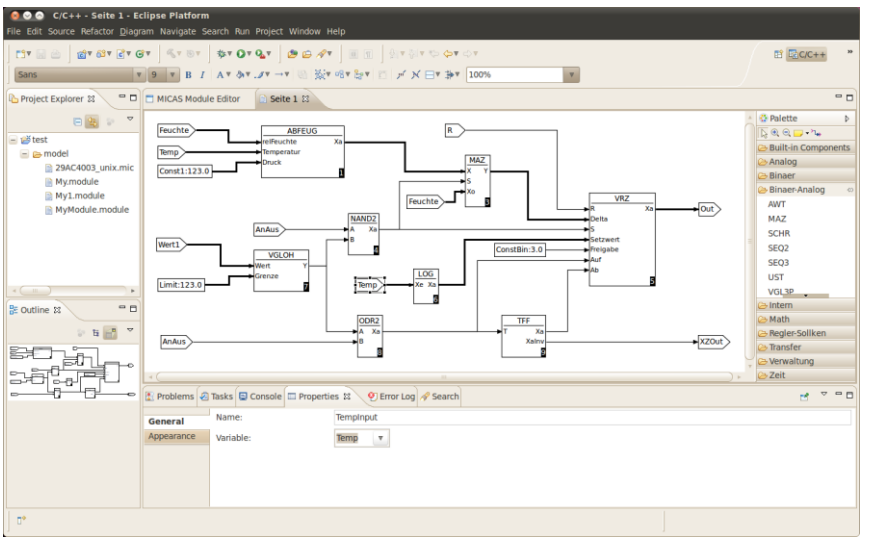
SOLUTION

Block Diagrams



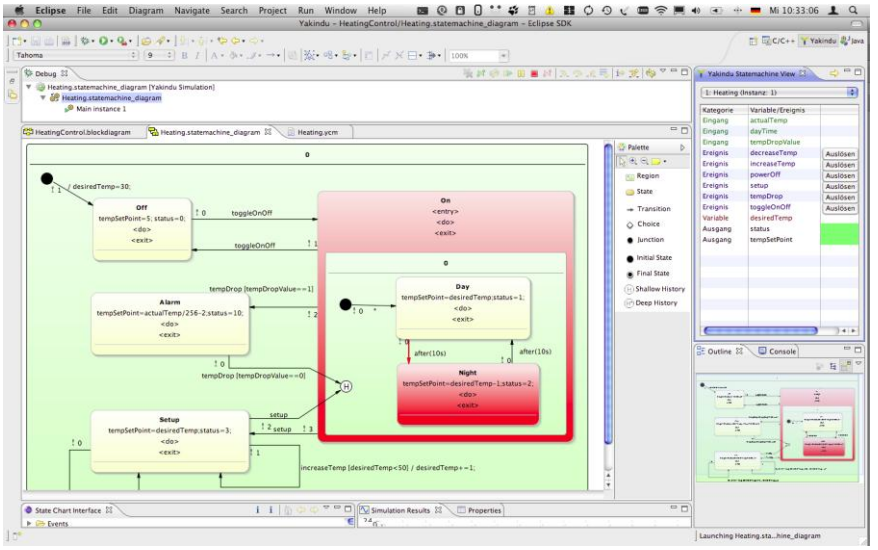
SOLUTION

PLC Programming



SOLUTION

State Charts

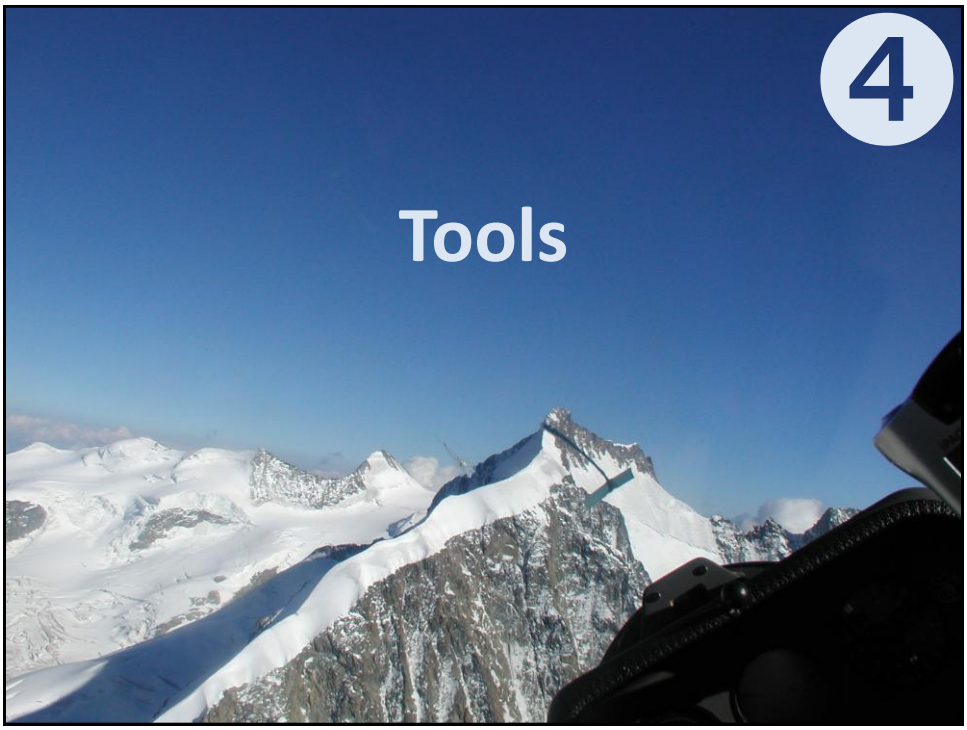


TOOLS

eclipse modeling PROJECT

xtext

YAKINDU


4
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

Open Source (EPL)
Eclipse-based, Eclipse Project
Very flexible, very popular!
Current Version 2.0:

- improved performance
- Xbase: expressions for reuse
- Xtend2: „Better Java“, with support for Xpand-like templates

Open Source (Apache 2.0)
Projectional Editor
Very good at lang. Composition
Current Version 2.0:
- Improved performance
- Unified generate/compile/build
- Debug MPS in MPS
- Tables in the editor
- (Diagrams planned for 2.1)

Commercial Tool.
Projectional Editor
Very flexible notations
Version 1.8 is current

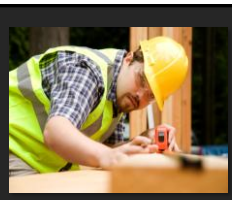
Way More:

Spoofax
Rascal
oomega
The Whole Platform

see also
http://languageworkbenches.net

Automation

faster, deterministic



Increased Quality

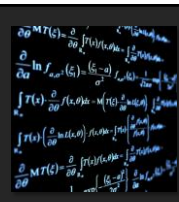
well defined structures allthrough the system

Meaningful Validation

more semantics in the model

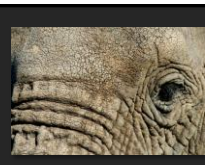
Capture Domain Knowledge

formalized into languages and models



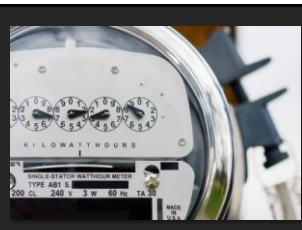
Suitable Notations

textual, graphical, tabular



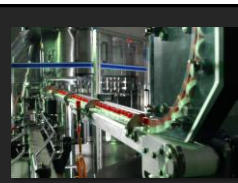
Technology Independence

generate „technology glue code“



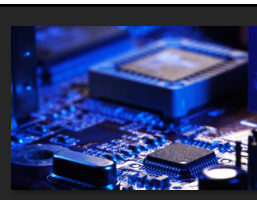
Abstraction w/o Runtime Overhead

generator „optimizes away“



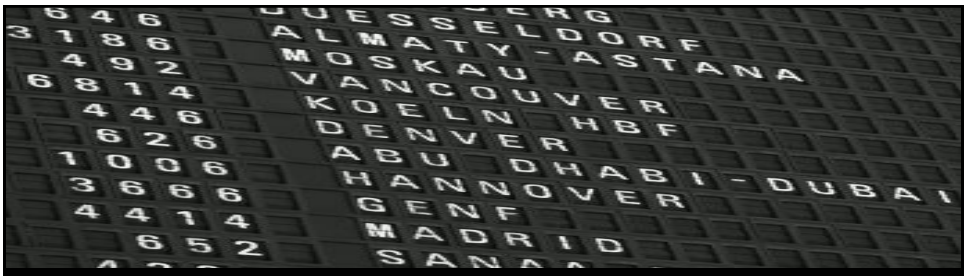
Capture Implementation Strategy

in the generators

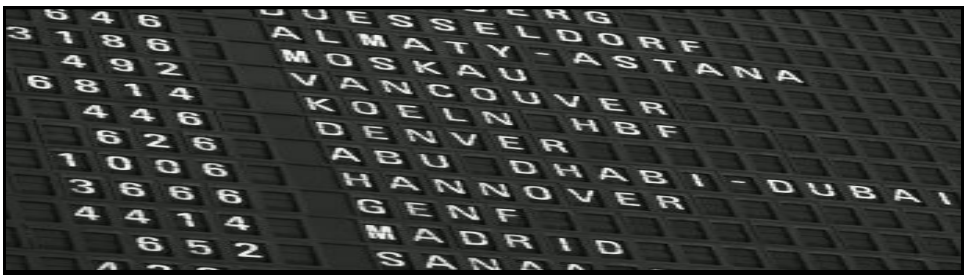


Everything is a model

including for example hardware (some) hardware



Airport Management System



Monitors
Website
Aircraft-Module

Data Center

```
component DelayCalculator {}

component InfoScreen {}

component AircraftModule {}
```

```
component DelayCalculator {
  provides IDelayCalculator
  requires IInfoScreen
}
component InfoScreen {
  provides IInfoScreen
}
component AircraftModule {
  provides IAircraftModule
  requires IDelayCalculator
}

interface IDelayCalculator {}
interface IInfoScreen {}
interface IAircraftModule {}
```

```
component InfoScreen {
  provides IInfoScreen
}

instance screen1: InfoScreen
instance screen2: InfoScreen
…
```

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

component InfoScreen {
  provides default: IInfoScreen
}

component AircraftModule {
  provides default: IAircraftModule
  requires calculator[1]: IDelayCalculator
}
```

```
component DelayCalculator {
  provides aircraft: IAircraftStatus
  provides managementConsole:
                IManagementConsole
  requires screens[0..n]: IInfoScreen
}

component Manager {
  requires backend[1]: IManagementConsole
}

component InfoScreen {
  provides default: IInfoScreen
}

component AircraftModule {
  requires calculator[1]: IAircraftStatus
}
```

```
component DelayCalculator {
  requires screens[0..n]: IInfoScreen
  …
}

component InfoScreen {
  provides default: IInfoScreen
}

instance dc: DelayCalculator
instance screen1: InfoScreen
instance screen2: InfoScreen

connect dc.screens
    to (screen1.default, screen2.default)
```

```
namespace com.mycompany.production {
  instance dc: DelayCalculator

  // InfoScreen instances are created and
  // started in other configurations
  dynamic connect dc.screens every 60 query {
    type = IInfoScreen
    status = active
  }
}
```

```
interface IAircraftStatus {

  oneway message reportPosition
                 (aircraft: ID, pos: Position )

  request-reply message reportProblem {
    request (aircraft: ID, problem: Problem,
             comment: String)
    reply (repairProcedure: ID)
  }

}
```

```
struct FlightInfo {
  from: Airport
  to: Airport
  scheduled: Time
  expected: Time
  …
}

replicated singleton flights {
  flights: FlightInfo[]
}

component DelayCalculator {
  publishes flights
}

component InfoScreen {
  consumes flights
}
```

```
interface IAircraftStatus {
  oneway message registerAircraft(aircraft: ID! )
  oneway message unregisterAircraft(aircraft: ID! )
  oneway message reportPosition(aircraft: ID!,
                 pos: Position! )
  request-reply message reportProblem {
    request (aircraft: ID!, problem: Problem!,
             comment: String!)
    reply (repairProcedure: !ID)
  }
  protocol initial = new {
    state new {
      registerAircraft => registered
    }
    state registered {
      unregisterAircraft => new
      reportPosition
      reportProblem
    }
  }
}
```

As you understand and develop your Architecture…

Develop a language to express it!

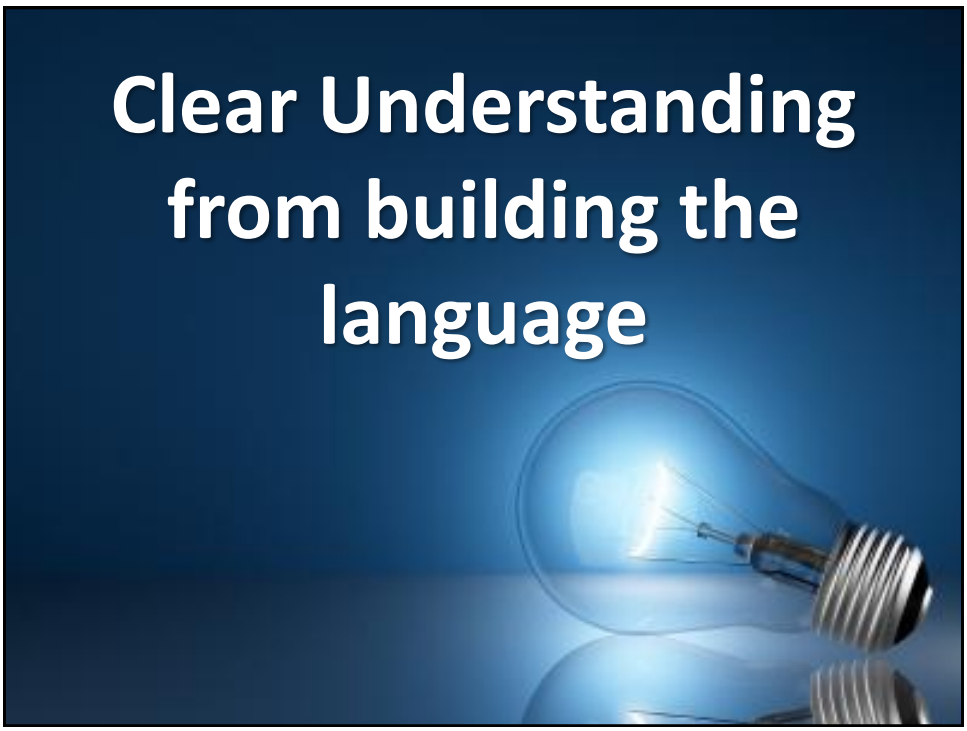
Language resembles architectural concepts

We express the application(s) with the language.

Architecture

DSL

Clear Understanding from building the language



Unambigious Vocabulary

Concepts independent from Technology

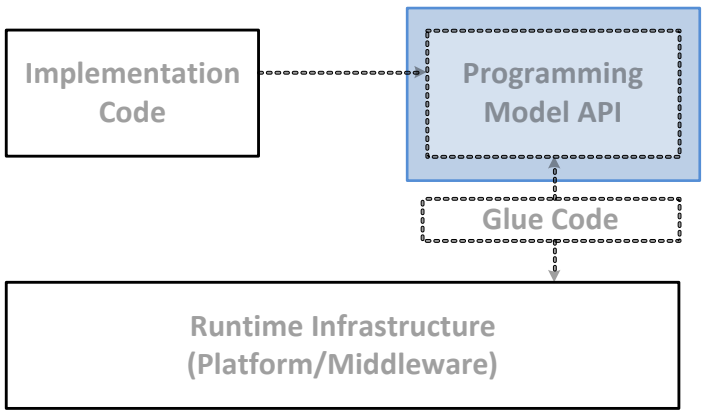
Programming Model can be defined based on Conceptual Arcitecture

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

Since we already have a formal model….

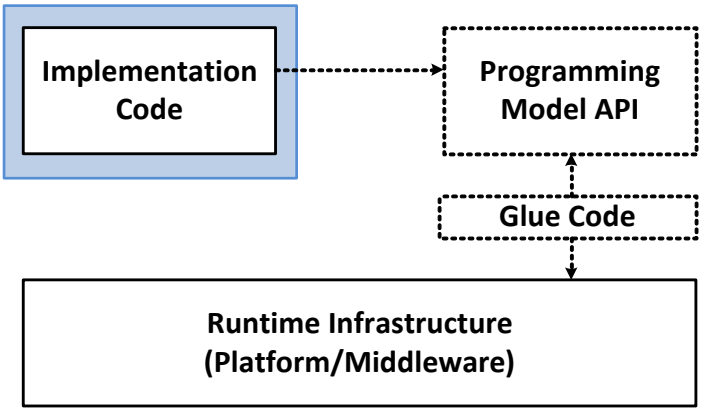
Generate API

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



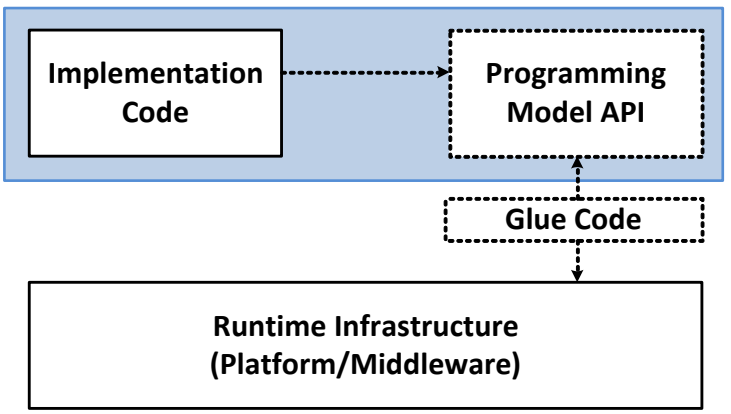
Implementation

Implementation only depends on the generated programming model API



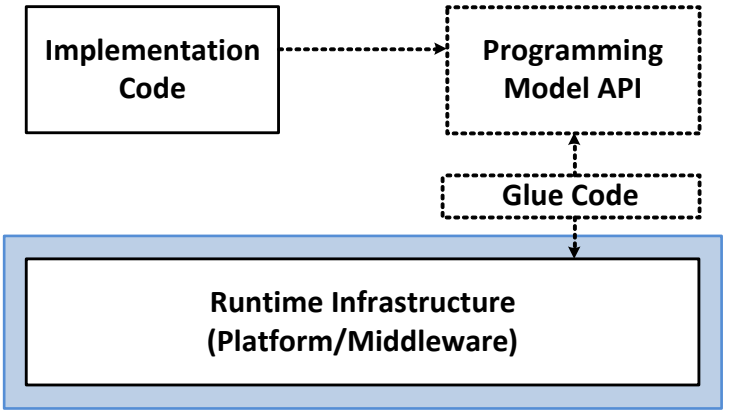
Programming Model

Generated API + Usage Idioms
Completely Technology-Independent



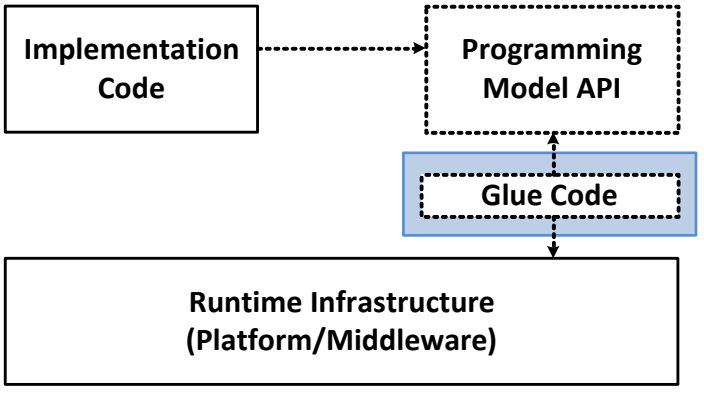
Runtime Infrastructure

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



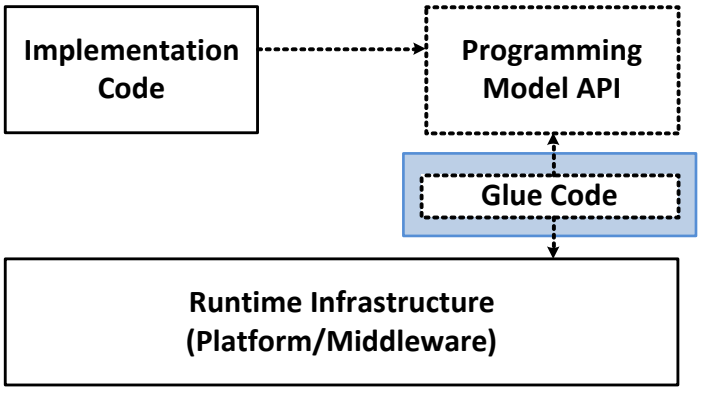
Glue Code

Aka Technology Mapping Code
Maps API to selected platform



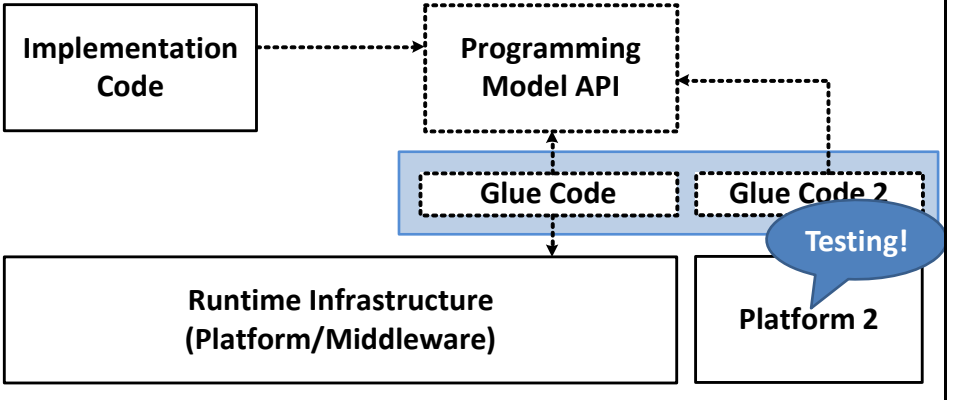
Glue Code

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



Several Platforms

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



Formal Architecture Description **is not new:**

ADLs, UML

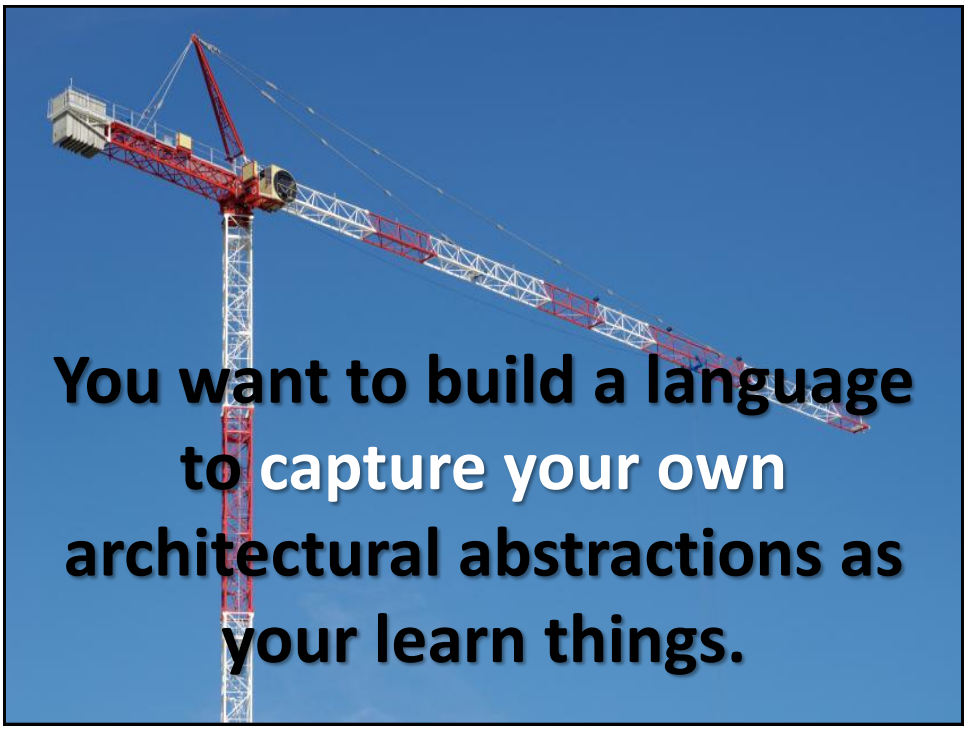
But all of those use

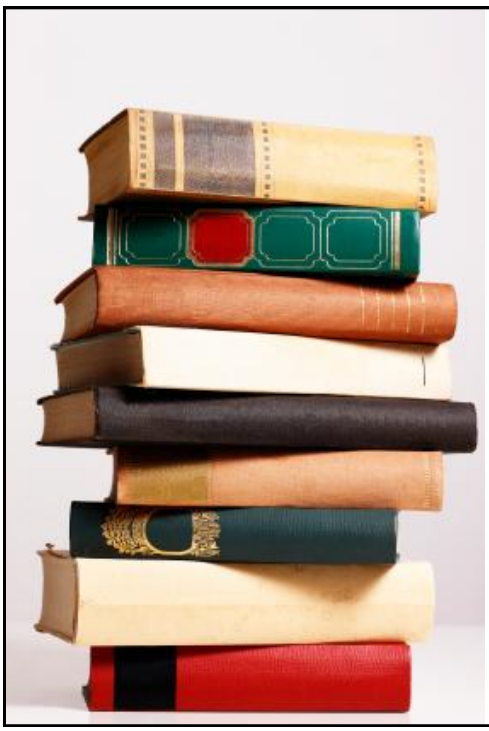
existing, generic

languages!

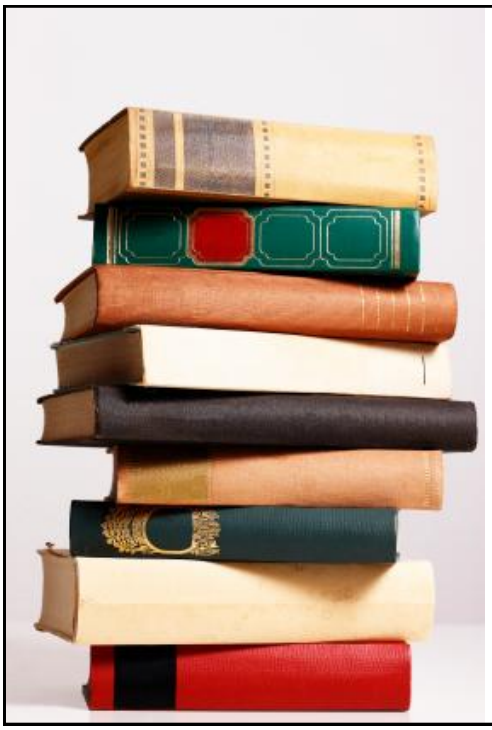
This misses the point!

Trying to express **your specific** architecture with **predefined** abstractions is not useful!

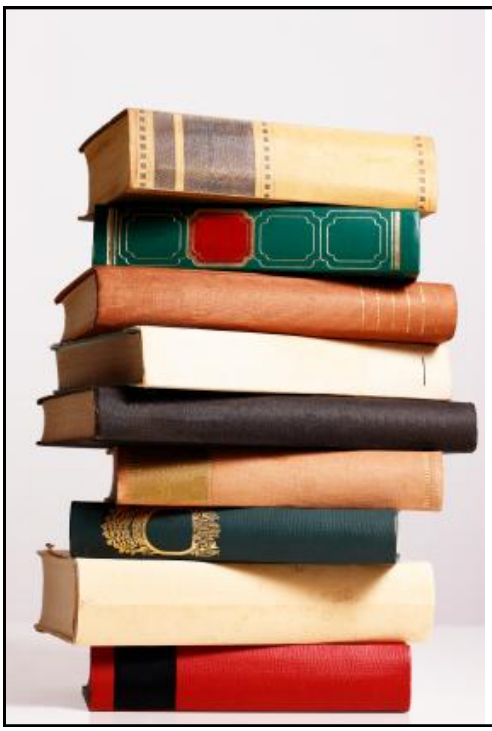




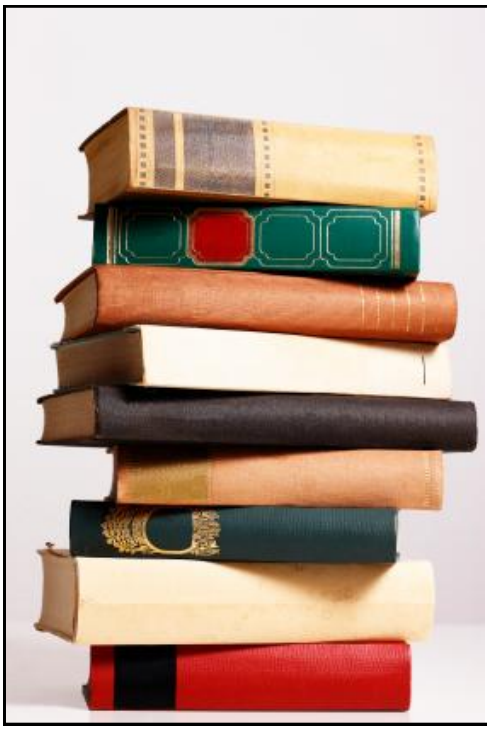
Where are standards useful?



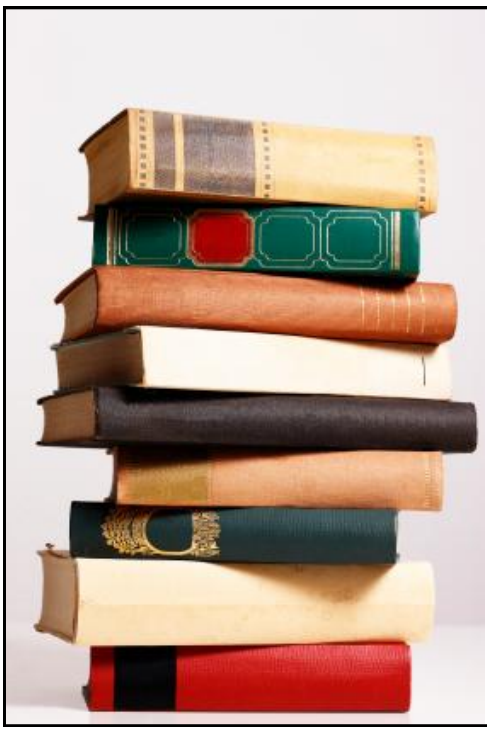
People have to learn architectural concepts anyway.



Is UML with a profile still a standard language?



On which meta level do I want to standardize? M2 (UML), M3 (MOF)?



Isn't a DSL based on MOF as „standard" as a profile based on UML?

UML Profiles instead?

You'll think more about UML-ities than your own concepts

UML Profiles instead?

Tool integration issues (repository, diff/merge, versioning)

UML Profiles instead?

Tools are often complex, heavyweight, bloated. Acceptance limited.

UML Generally Useless?

No. UML can be used for documentation (sequence diagrams, eg)

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.

Linguistic Abstraction

What's the Problem here?

```
// A
int[] arr = ...
for (int i=0; i<arr.size(); i++) {
    sum += arr[i];
}
```

```
// B
List<int> l = ...
for (int i=0; i<arr.size(); i++) {
    l.add( arr[i] );
}
```

Linguistic Abstraction

What's the Problem here?

```
// A
int[] arr = ...
for (int i=0; i<arr.size(); i++) {
    sum += arr[i];
}
```

```
// B
List<int> l = ...
for (int i=0; i<arr.size(); i++) {
    l.add( arr[i] );
}
```

Much better with new linguistic abstraction

```
// A'
for (int i in arr) {
    sum += i;
}
```

```
// B'
seqfor (int i in arr) {
    l.add( arr[i] );
}
```

No sophisticated analysis required to understand the semantics of a contruct and treat it "correctly".

Linguistic Abstraction

What's this? And what's the Problem?

```
var linefollower_states_enum linefollower_currentstate = linefollower_states_enum::STATE_INITIALIZING;

enum linefollower_events_enum { EVENT_INITIALIZED, EVENT_BUMPED, EVENT_BLOCKED, EVENT_UNBLOCKED }

enum linefollower_states_enum { STATE_INITIALIZING, STATE_PAUSED, STATE_RUNNING, STATE_CRASH }

void linefollower_execute( linefollower_events_enum event ) {
  if ( linefollower_currentstate == linefollower_states_enum::STATE_INITIALIZING ) {
    if ( event == linefollower_events_enum::EVENT_INITIALIZED ) {
      if ( true ) {
        linefollower_currentstate = linefollower_states_enum::STATE_RUNNING;
        return;
      }
    }
  }
  if ( linefollower_currentstate == linefollower_states_enum::STATE_PAUSED ) {
    if ( event == linefollower_events_enum::EVENT_UNBLOCKED ) {
      if ( true ) {
        linefollower_currentstate = linefollower_states_enum::STATE_RUNNING;
        return;
      }
    }
  }
  if ( linefollower_currentstate == linefollower_states_enum::STATE_RUNNING ) {
    if ( event == linefollower_events_enum::EVENT_BLOCKED ) {
      if ( true ) {
        linefollower_currentstate = linefollower_states_enum::STATE_PAUSED;
        int16 i = 1;
        return;
      }
    }
    if ( event == linefollower_events_enum::EVENT_BUMPED ) {
      if ( true ) {
        linefollower_currentstate = linefollower_states_enum::STATE_CRASH;
        return;
      }
    }
  }
}
```

Linguistic Abstraction

Much better!

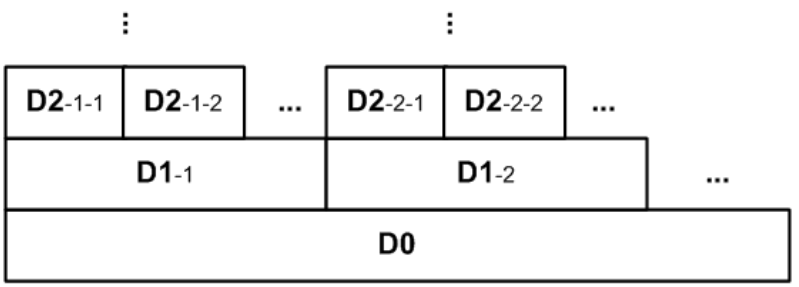
```
statemachine linefollower {
  event initialized;
  event bumped;
  event blocked;
  event unblocked;
  initial state initializing {
    initialized [true] -> running
  }
  state paused {
    entry int16 i = 1;
    unblocked [true] -> running
  }
  state running {
    blocked [true] -> paused
    bumped [true] -> crash
  }
  state crash {
    <<transitions>>
  }
}
```

Linguistic Abstraction

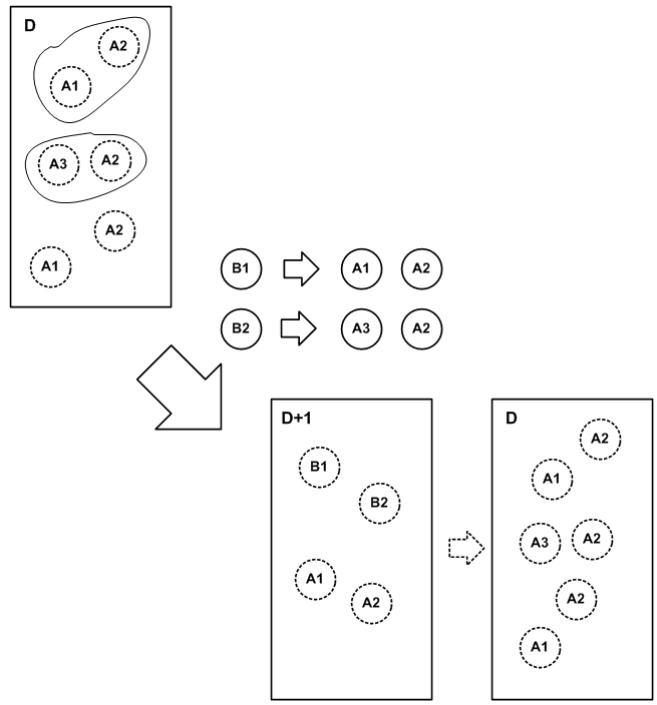
Much better!

linefollower	initializing	paused	running	crash
initialized	true running			
bumped			true crash	
blocked			true paused	
unblocked		true running		

Domains are Hierarchical



Creating Linguistic Abstractions

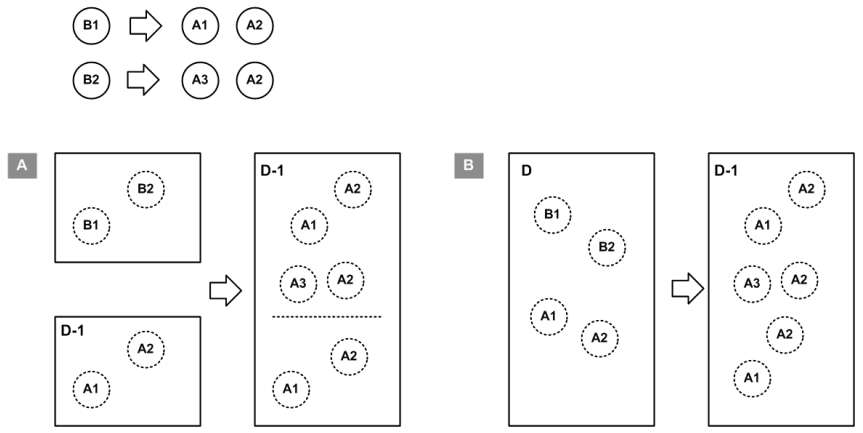


Domain Specific Languages

A DSL is a **language** at D that provides **linguistic abstractions** for **common patterns and idioms** of a language at D-1 when used within the domain D.

A good DSL does **not** require the use of patterns and idioms to express **semantically interesting** concepts in D. Processing tools do not have to do "semantic recovery" on D programs.

Extension vs. Cascading



Modeling

… (Mostly) Graphical Notations

… Abstract Syntax Storage

… Projecting Editors

… Different editable views for model

Programming

… (Mostly) Textual Notations

… Concrete Syntax Storage

… (Fancy) ASCII Editors

… Read-Only Visualizations

Why

the difference?

Programming
the way we do
Modeling?

Modeling
the way we do
Programming?

Modeling == Programming
Programming == Modeling

We don't want to
model,
we want to
program!

… at different levels of abstaction
… from different viewpoints
… integrated!

We don't want to
model,
we want to
program!

… with different degrees of domain-specificity
… with suitable notations
… with suitable expressiveness

We don't want to
model,
we want to
program!

And always:
precise and tool processable

Programming
Languages
are not
MODULAR
enough.

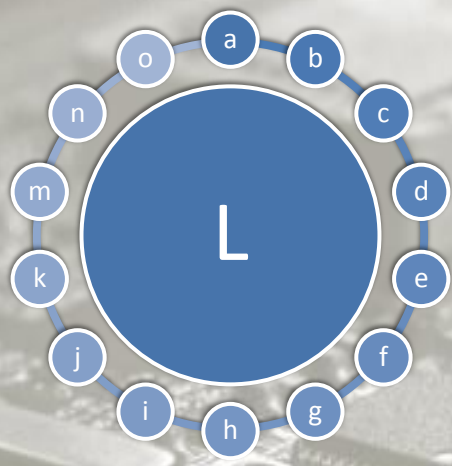
Programming Languages are not **COMPOSABLE** enough.

Programming Languages are not **CONFIGURABLE** enough.

Programming Languages are not **ADAPTABLE** enough.

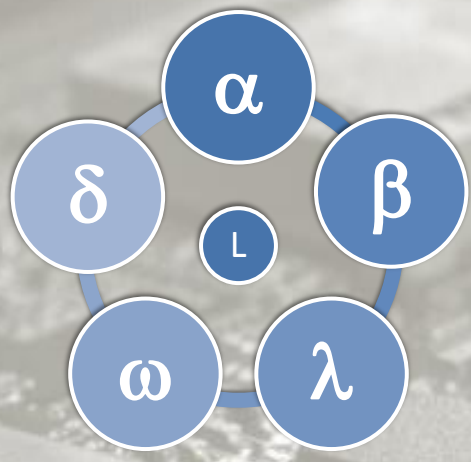
Programming Language Syntax is not **FLEXIBLE** enough.

Big Language?



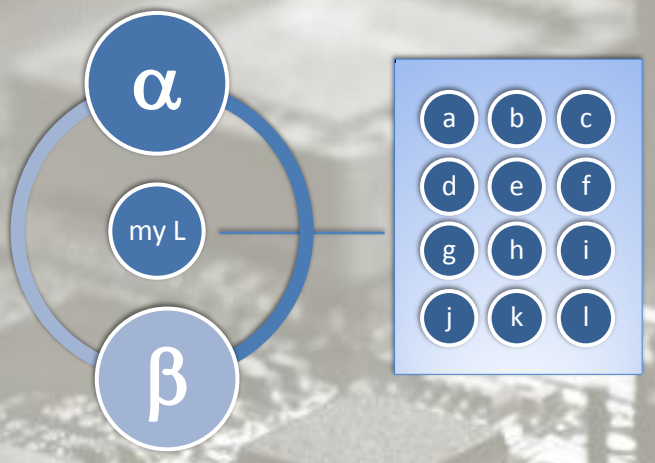
with many first class concepts!

Small Language?



with a few, orthogonal
and poweful concepts

Modular Language



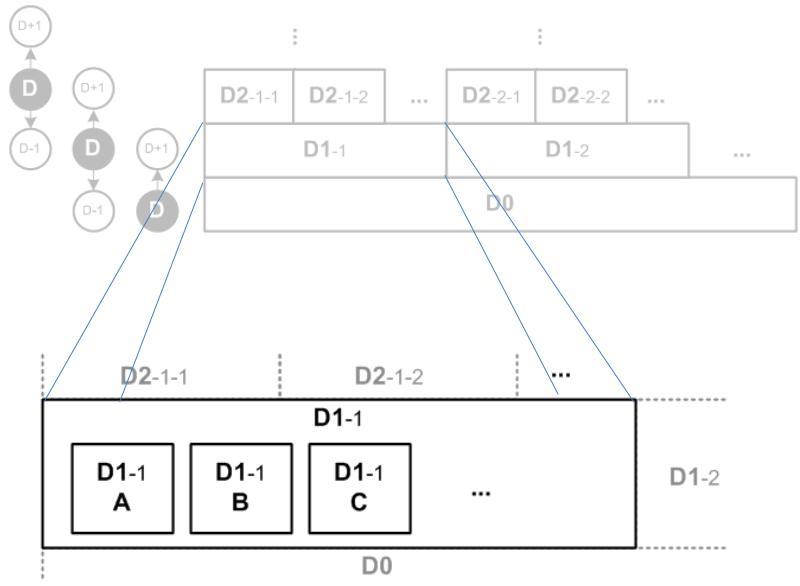
with many optional, composable concepts

Modular Language

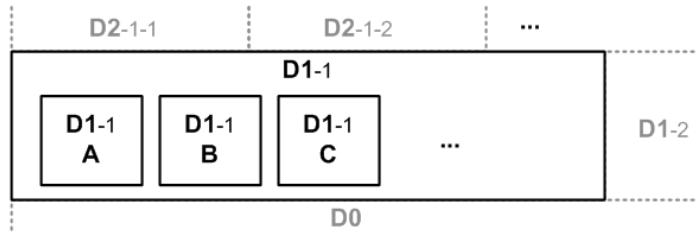
Like frameworks and libraries,

but with syntax and IDE support

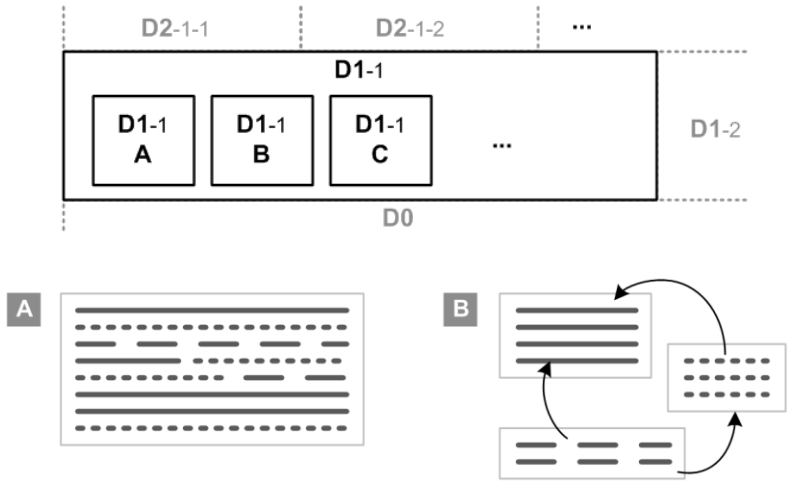
Several Concerns in a Domain



Viewpoints

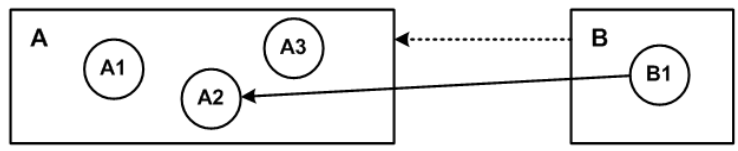


Viewpoints



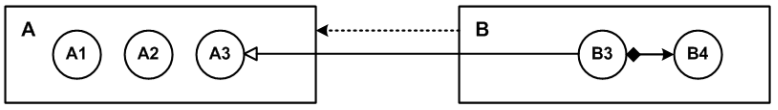
A: mixed
B: separate Viewpoints

Langugae Combination



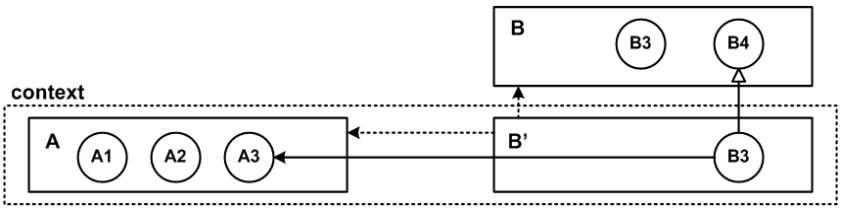
When DSL (for several concerns) are developed from scratch, as a group, then dependencies between the concerns can be materialized as dependencies between the languages and the language concepts

Langugae Extension



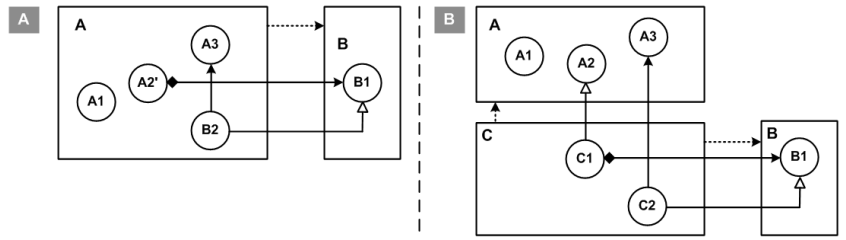
A language B extends another language A if B contains additional language concepts. This means that for programs written in B, all concepts from A are available, plus those defined in B.

Langugae Reuse



A language has been developed to be used in contexts not known at the time of development. No dependencies allowed! The reusable language has to be extended so it can reference concepts from languages in that context.

Langugae Embedding



Composition is a special case of reuse, where the reused language is syntactically embedded into languages from the context.

LWES
Language Workbenches
for Embedded Systems

Bundesministerium für Bildung und Forschung

http://mbeddr.com

LWES
Language Workbenches
for Embedded Systems

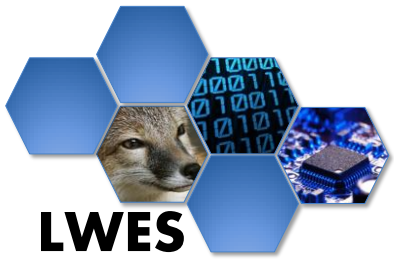
Bundesministerium für Bildung und Forschung

http://mbeddr.com

Incremental Extension of C with DSLs for Embedded Systems, integrated with Formal Methods and support for PLE and Requirements Tracing

Bundesministerium für Bildung und Forschung

http://mbeddr.com



Bundesministerium für Bildung und Forschung

http://mbeddr.com

itemis fortiss SICK LEAR CORPORATION

EXAMPLE CASE

SOLUTION

```
doc This module represents the code for the line follower lego robot. It has a coupl
module main imports OsekKernel, EcAPI, BitLevelUtilies {

  constant int WHITE = 500;

  constant int BLACK = 700;

  constant int SLOW = 20;

  constant int FAST = 40;

  doc Statemachine to manage the t
  statemachine linefollower {
    event initialized;
    initial state initializing {
      initialized [true] -> runnin
    }
    state running {

    }
  }

  initialize {
    ecrobot_set_light_sensor_activ
    event linefollower:initialized
  }
```

```
doc This is the cyclic task that is called every 1ms to do the actual contro
task run cyclic prio = 2 every = 2 {
  stateswitch linefollower
    state running
      int32 light = 0;
      light = ecrobot_get_light_sensor(SENSOR_PORT_T::NXT_PORT_S1);
      if ( light < ( WHITE + BLACK ) / 2 ) {
        updateMotorSettings(SLOW, FAST);
      } else {
        updateMotorSettings(FAST, SLOW);
      }
    default
      <noop>;
}

doc This procedure actually configures the motors based on the speed values
void updateMotorSettings( int left, int right ) {
  nxt_motor_set_speed(MOTOR_PORT_T::NXT_PORT_C, left, 1);
  nxt_motor_set_speed(MOTOR_PORT_T::NXT_PORT_B, right, 1);
}
```

SOLUTION

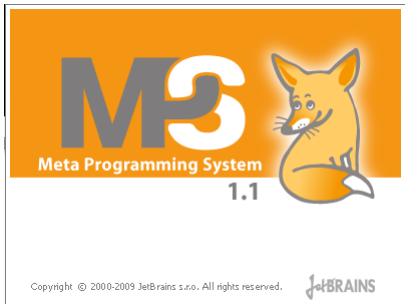
```
exported interface MotorControl {
  void stop(   );
  void setLeftSpeed( int8 speed );
  void setRightSpeed( int8 speed );
}

exported component Motors {
  provides motorControl : MotorControl;
}
```

```
exported component implementation MotorsNXT : Motors {

  procedure void motorControl.stop( ) {
    nxt_motor_set_speed(MOTOR_PORT_T::NXT_PORT_B, 0, 1);
    nxt_motor_set_speed(MOTOR_PORT_T::NXT_PORT_C, 0, 1);
  }

  procedure void motorControl.setLeftSpeed( int8 speed ) {
    nxt_motor_set_speed(MOTOR_PORT_T::NXT_PORT_C, speed, 1);
  }

  procedure void motorControl.setRightSpeed( int8 speed ) {
    nxt_motor_set_speed(MOTOR_PORT_T::NXT_PORT_B, speed, 1);
  }
}
```

SOLUTION

```
module impl imports <<imports>> {

  int speed( int val ) {
    return 2 * val;
  }

  robot script stopAndGo
    block main on bump block retreat on bump <no bumpReaction>
                                stop
                                accelerate to 0 - 30  within 2000
                                drive on for 2000
                                decelerate to 0  within 1000
                                stop
      accelerate to speed(25) within 3000
      drive on for 2000
      turn left for 2000
      block driveMore on bump <no bumpReaction>
        accelerate to 80  within 2000
        turn right for 3000
      decelerate to 0  within 3000
      stop

}
```

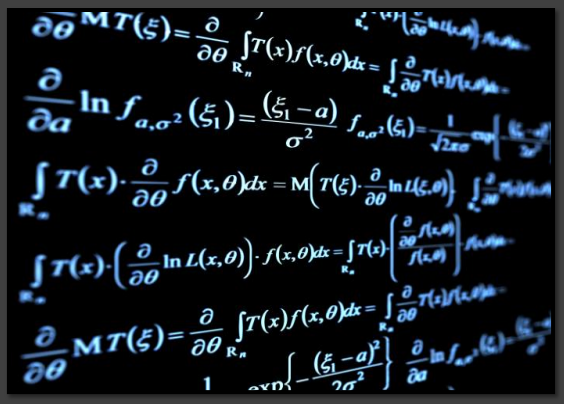
TOOLS



JetBrains MPS

Limit Expressiveness

Notation, Notation, Notation

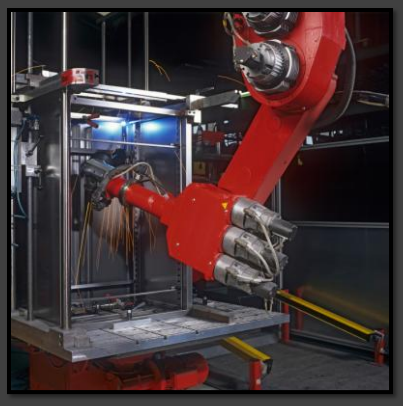


Graphical vs. Textual

Viewpoints

Teamwork Support

Interpretation
vs.
Generation



Rich Domain Specific Platform

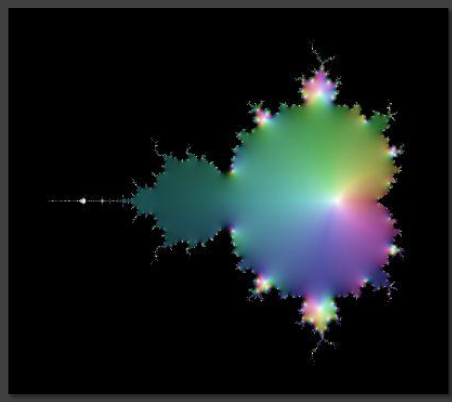
Checks First and Separate

Cascading

Annotation Models

Don‘t forget Testing

Iterate!



Co-Evolve Language and Concepts

Domain Users Programming?

Compatible Organization


THE END.
.coordinates
web www.voelter.de
email voelter@acm.org
twitter markusvoelter
xing http://www.xing.com/profile/Markus_Voelter
linkedin http://www.linkedin.com/pub/0/377/a31