DSLs for Product Lines:
Approaches, Tools, Experiences

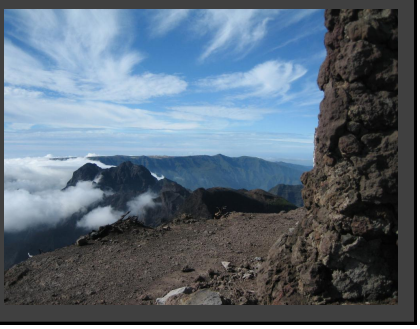
SPLC 2011

Markus Voelter
Independent/itemis
voelter@acm.org

DSLs for Product Lines:
Approaches, Tools, Experiences
SPLC 2011

1. DSLs
2. Example DSLs
3. From FMs to DSLs
4. PLE Concepts & DSLs
5. Customization and Configuration
6. PLE for Languages
7. DSL Tools

1 DSLs



programming
started
close to the hardware

abstractions
~computing

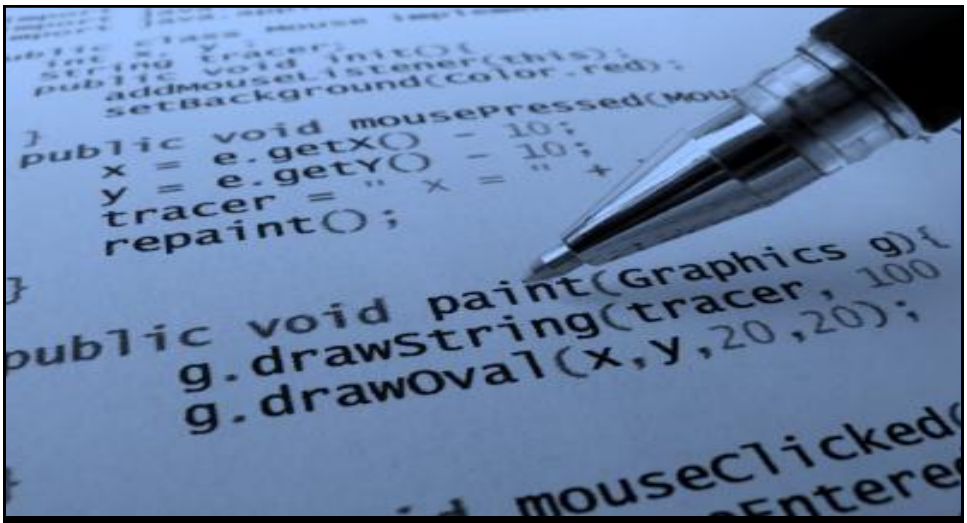
chips

abstractions
~computing

bits

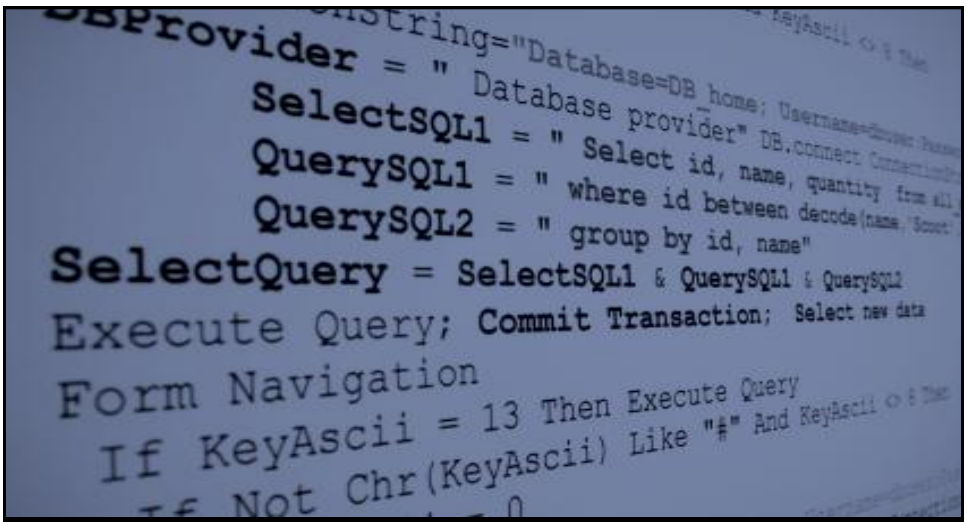
abstractions
~computing

c



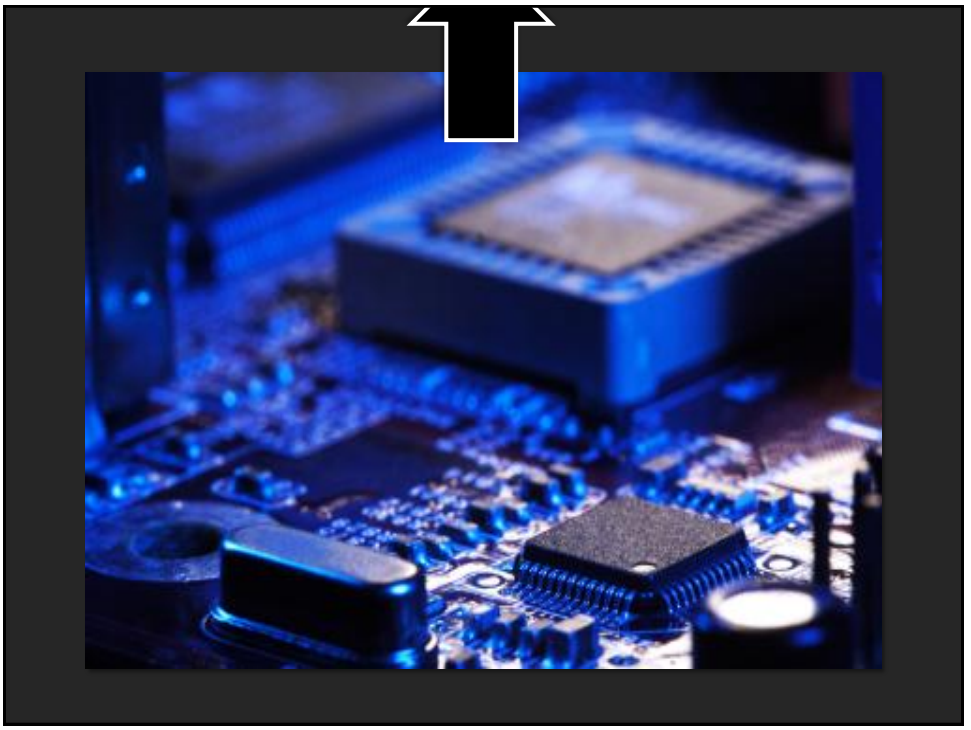
abstractions
~computing?

Java



abstractions
~computing?

SQL



general purpose

domain specific

tailor made

effective++

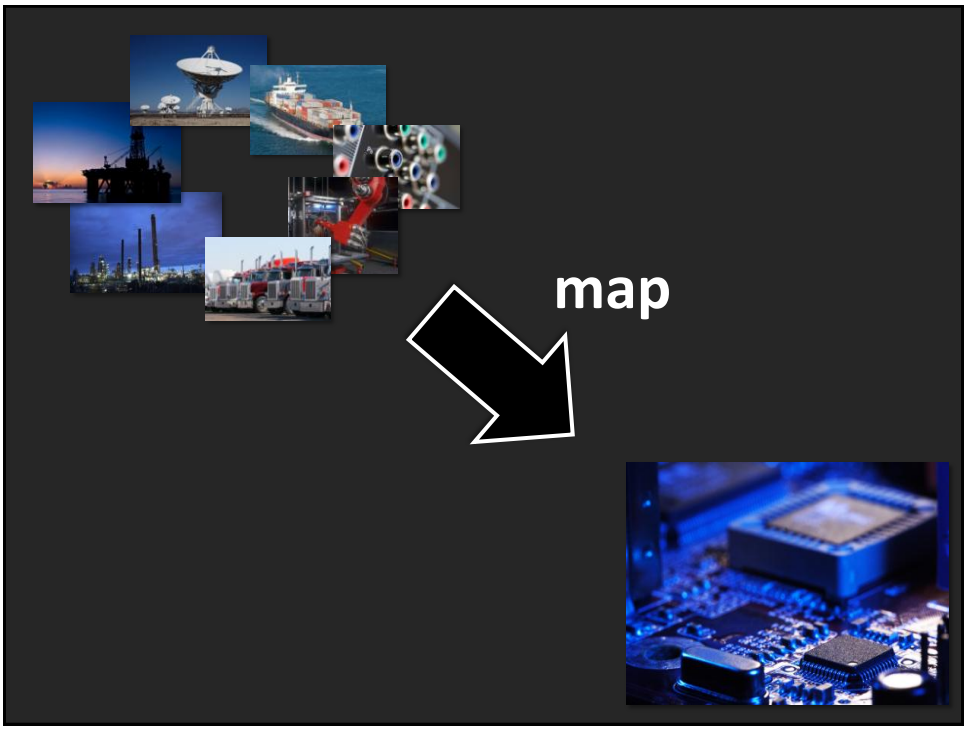
specialized, limited

used by experts

together with other specialized tools

DSL

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.

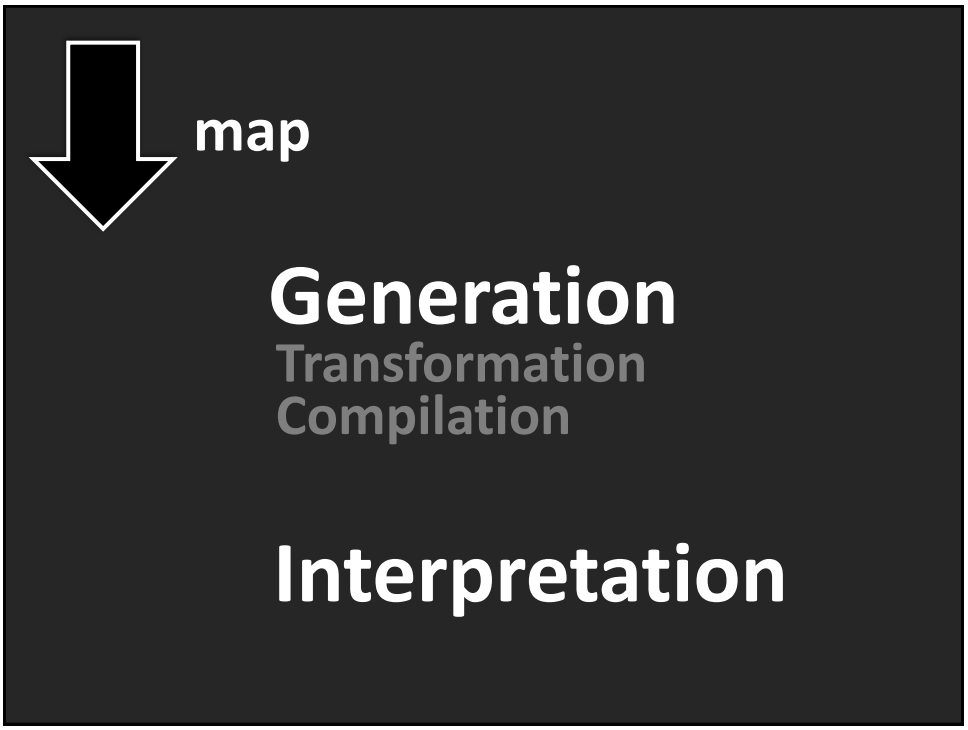


DSL Program
(aka Model)

map

automated!

GPL Program


map
Generation
Transformation
Compilation
Interpretation


LanguageOrientedProgramming
DomainSpecificModeling
GenerativeProgramming
ModelDrivenDevelopment
ModelIntegratedComputing
ModelBasedDevelopment
ModelDrivenEngineering
ModelBasedEngineering
ModelDrivenArchitecture
ModelDrivenSoftwareDevelopment

Activities

Analysing Domains

Defining Languages

Adapting/Selecting

Building Editors

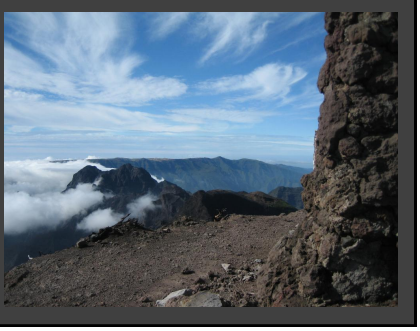
Transforming Models

Building Generators

Building Frameworks

… and using all of that to build apps

❷ Example DSLs



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 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];
    } ;
}
```

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];
    } ;
}
```

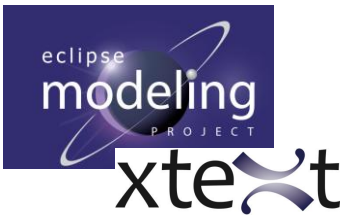
tests

```
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



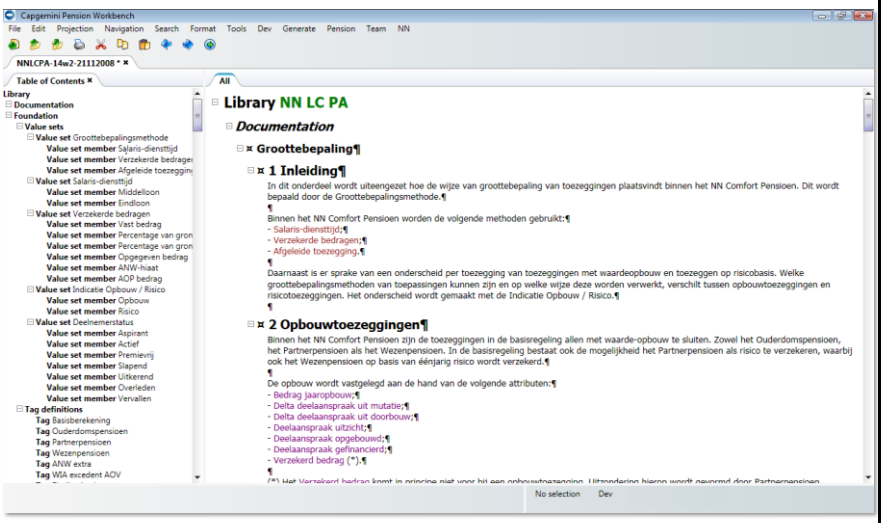
Eclipse Modeling
Eclipse Xtext

Example 2:

Pension Fund Specification

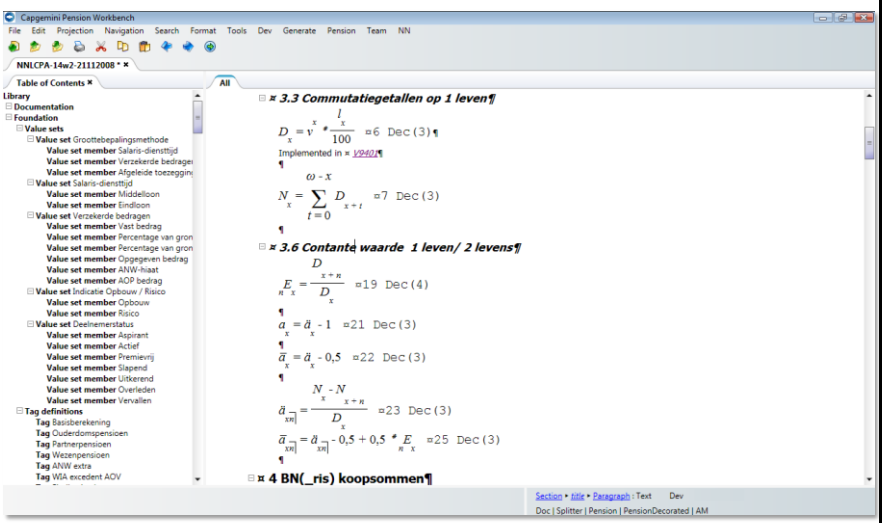
SOLUTION

Textual Documentation



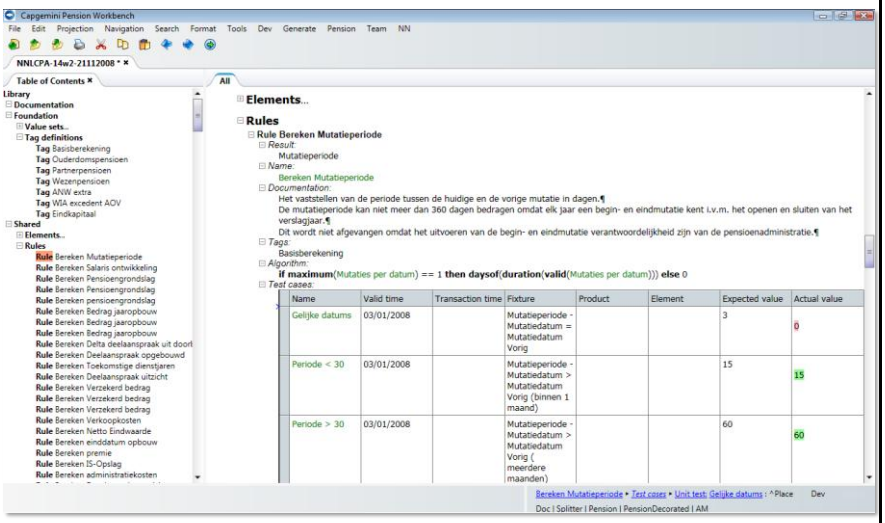
SOLUTION

Insurance Mathematics



SOLUTION

Calculation Rules and Tests



TOOLS

INTENTIONAL SOFTWARE

Intentional Software's Domain Workbench

Example 3:
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

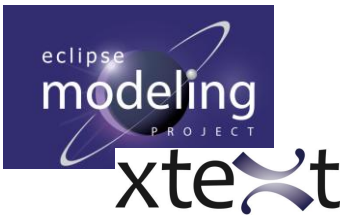
Component Specification

```
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 4:

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
        }
}
```

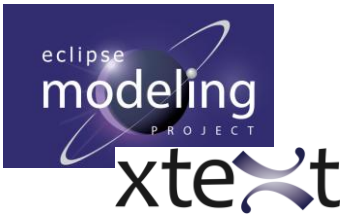
SOLUTION

Plus:

In-IDE Simulator

Unit Test Support

TOOLS



Eclipse Modeling
Eclipse Xtext

More Examples:

Miscellaneous

DOMAIN

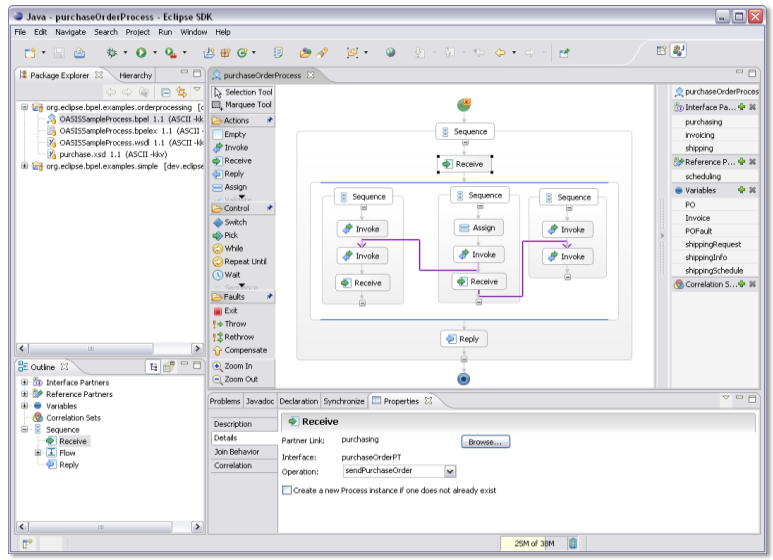
Hearing Aids

DOMAIN

Refrigerators

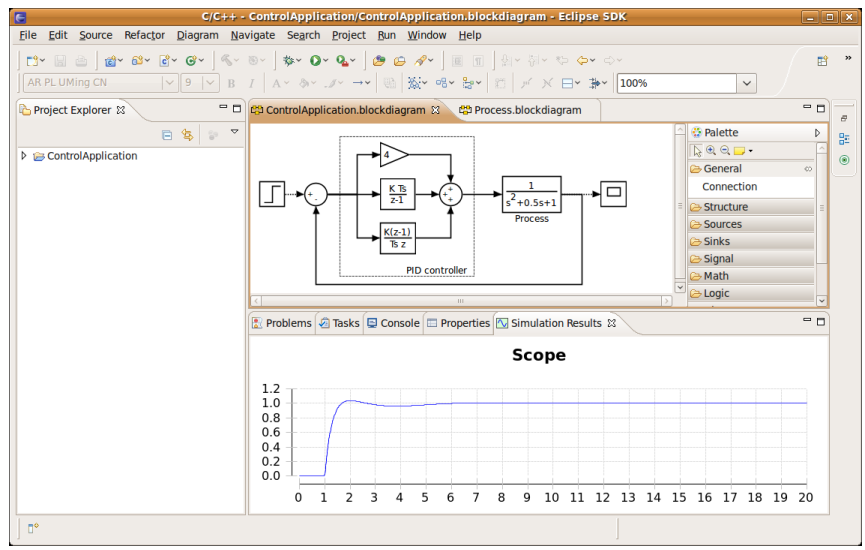
SOLUTION

BPEL Designer



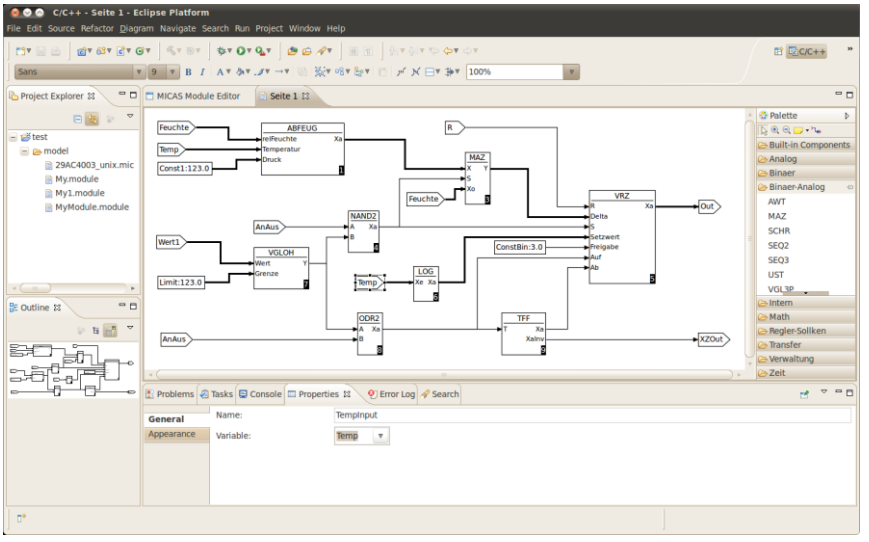
SOLUTION

Block Diagrams



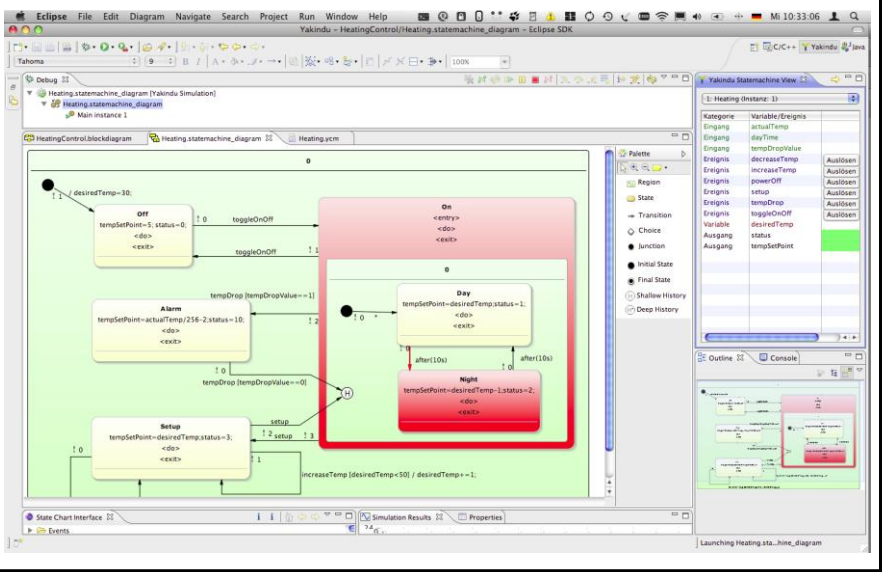
SOLUTION

PLC Programming



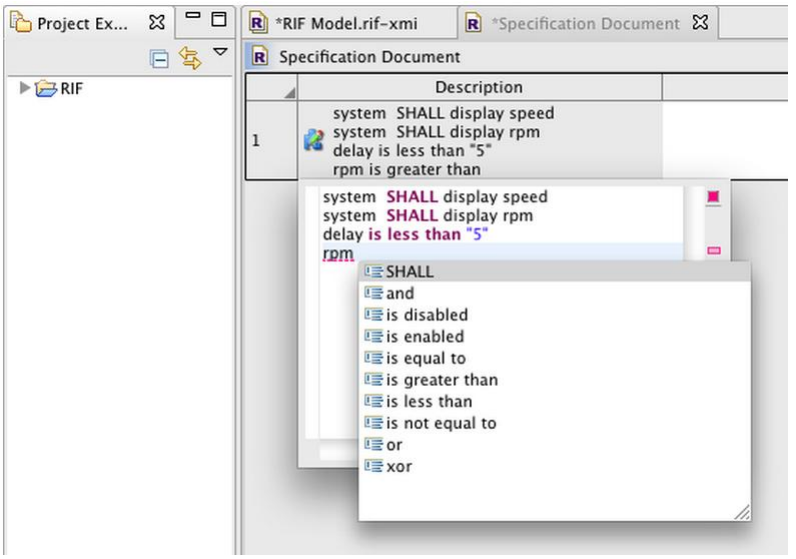
SOLUTION

State Charts



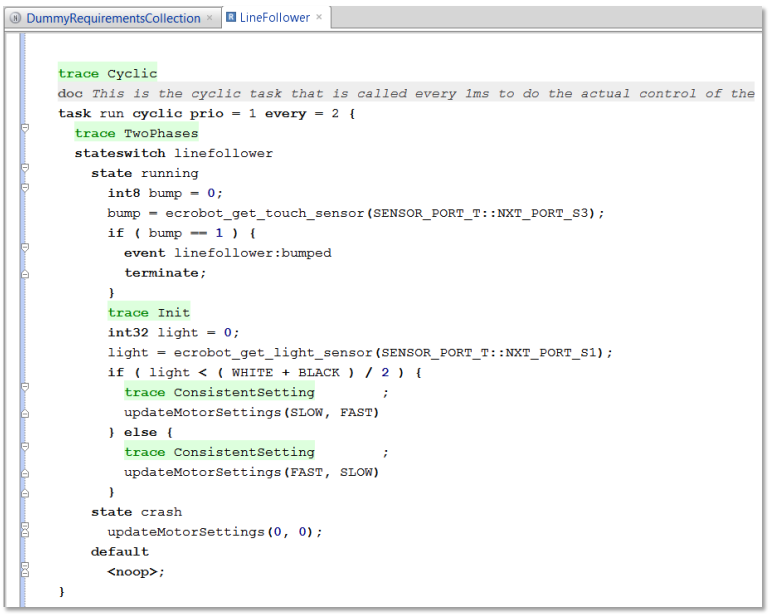
SOLUTION

Requirements (Eclipse)

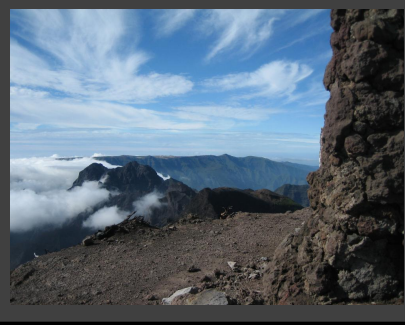


SOLUTION

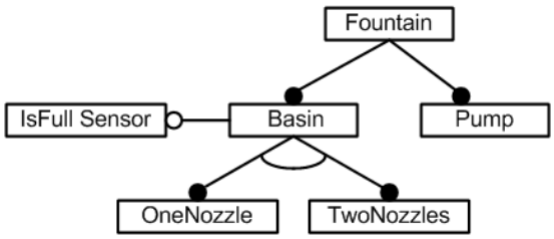
Requirements (MPS)



❸ From FMs to DSLs



Structure




```
Fountain        -> Basin PUMP(rpm:int)
Basin           -> ISFULLSENSOR? (ONENOZZLE | TWONOZZLES)
```

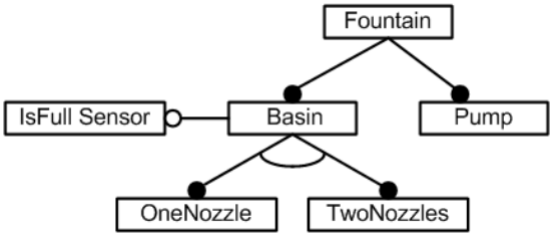
Behaviour?

- Selection vs. Creation?
- Programming in C?
- Selection – DSL – Programming in C

+ Attributes

+ Attributes + Recursion

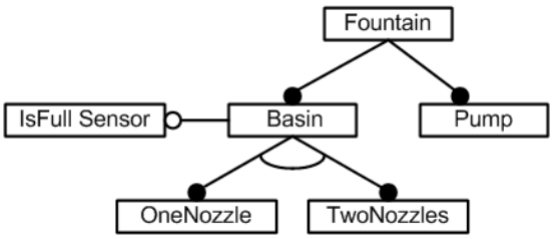
Structure + Behavior




```
Fountain        -> Basin PUMP(rpm:int) Behavior
Basin           -> ISFULLSENSOR? NOZZLE*
Behavior        -> Rule*
Rule            -> CONDITION CONSEQUENCE
```

+ Attributes + Rec + References

Structure + Behavior

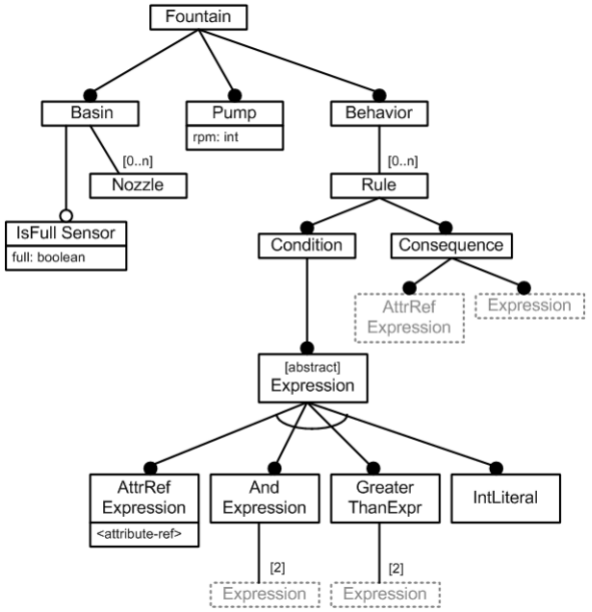



```
Fountain        -> Basin id:PUMP(rpm:int)? Behavior
Basin           -> id:ISFULLSENSOR(full:boolean)? id:NOZZLE *
Behavior        -> Rule *
Rule            -> Condition Consequence
Condition       -> Expression
Expression      -> ATTRREFEXPRESSION | AndExpression
                   GreaterThan | INTLITERAL
AndExpression   -> Expression Expression
GreaterThan     -> Expression Expression
Consequence     -> ATTRREFEXPRESSION Expression
```

+ Attributes + Rec + References

Structure + Behavior

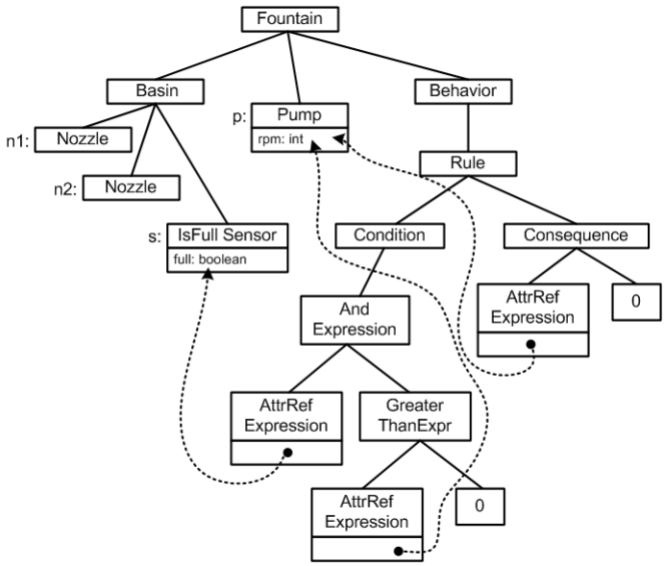
(in extended FM Notation)



+ Attributes + Rec + References

Structure + Behavior Example

(in extended FM Notation)



+ Concrete Syntax

Structure + Behavior as Grammar

```
Fountain        -> "fountain" Basin Pump Behavior
Basin           -> "basin" IsFullSensor Nozzle*
Behavior        -> Rule*
Rule            -> "if" Condition "then" Consequence
Condition       -> Expression
Expression      -> AttrRefExpression | AndExpression |
                   GreaterThanExpression | IntLiteral;
AndExpression   -> Expression "&&" Expression
GreaterThan     -> Expression ">" Expression
AttrRefExpression -> <attribute-ref-by-name>
IntLiteral      -> (0..9)*
Consequence     -> AttrRefExpression "=" Expression
IsFullSensor    -> "sensor" ID (full:boolean)?
Nozzle          -> "nozzle" ID
Pump            -> "pump" ID (rpm:int)?
```

+ Concrete Syntax

Structure + Behavior Example

```
fountain
  basin sensor s
    nozzle n1
    nozzle n2
  pump p
  if s.full && p.rpm > 0
       then p.rpm = 0
```

+ Concrete Syntax

Domain Specific Language

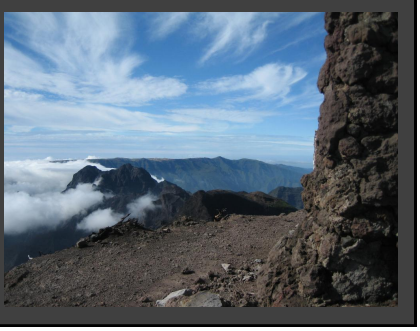
```
Fountain            -> "fountain" Basin Pump Behavior
Basin               -> "basin" IsFullSensor Nozzle*
Behavior            -> Rule*
Rule                -> "if" Condition "then" Consequence
Condition           -> Expression
Expression          -> AttrRefExpression | AndExpression |
                       GreaterThanExpression | IntLiteral;
AndExpression       -> Expression "&&" Expression
GreaterThan         -> Expression ">" Expression
AttrRefExpression   -> <attribute-ref-by-name>
IntLiteral          -> (0..9)*
Consequence         -> AttrRefExpression "=" Expression
IsFullSensor        -> "sensor" ID (full:boolean)?
Nozzle              -> "nozzle" ID
Pump                -> "pump" ID (rpm:int)?
```

Definition ↑

Program →

```
fountain
  basin sensor s
        nozzle n1
        nozzle n2
  pump p
  if s.full && p.rpm > 0
       then p.rpm = 0
```

4 PLE Concepts & DSLs



domain engineering

… domain analysis
… classifying variability
… defining DSLs

application eng.

… using the DSL to express systems

platform

… target environment
… abstractions of solution space
… can influence DSL concepts (Arch DSLs)

config knowledge

… transformations
… generators
… interpreters

core asset

… languages
… generators
… editors

variation point

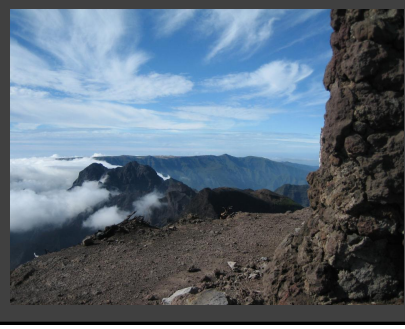
… bind how
… kind of varibility
… bind when

4 PLE Concepts & DSLs

Bind How

Kinds of Variability

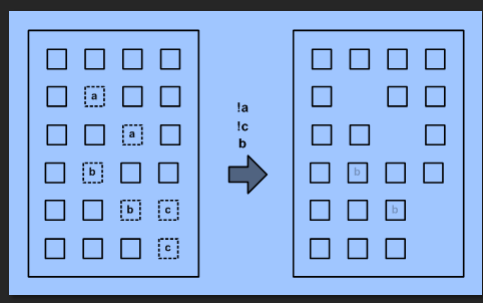
Bind When



Variability Mechanisms

Removal

… optionally take away from



Variability Mechanisms
Removal

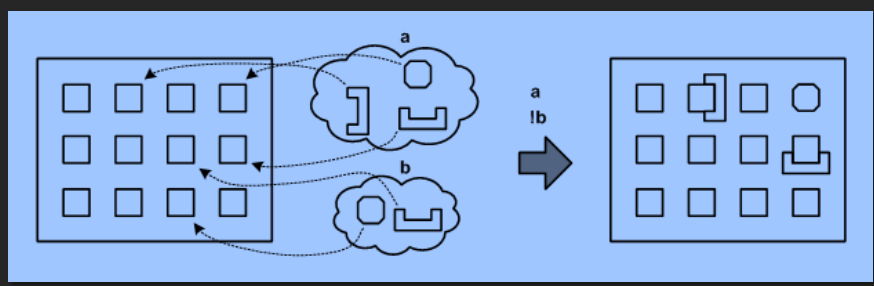
… optionally take away from overall whole

Challenge:

overall whole can get big and unwieldy

Variability Mechanisms
Injection

… optionally add to minimal core



Variability Mechanisms
Injection

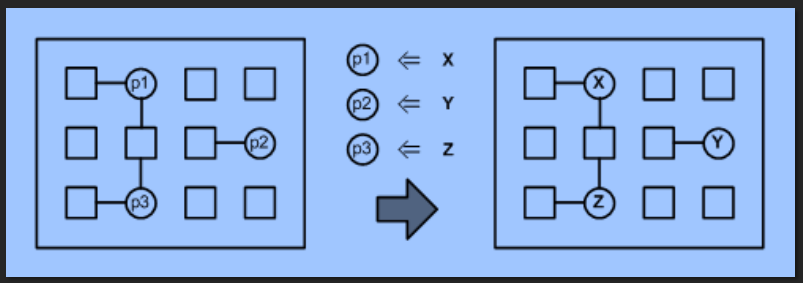
… optionally add to minimal core

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:

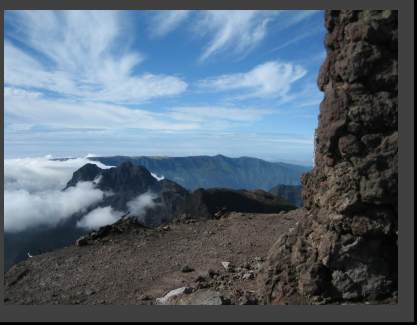
types for parameters can be non trivial (DSLs)

4 PLE Concepts & DSLs

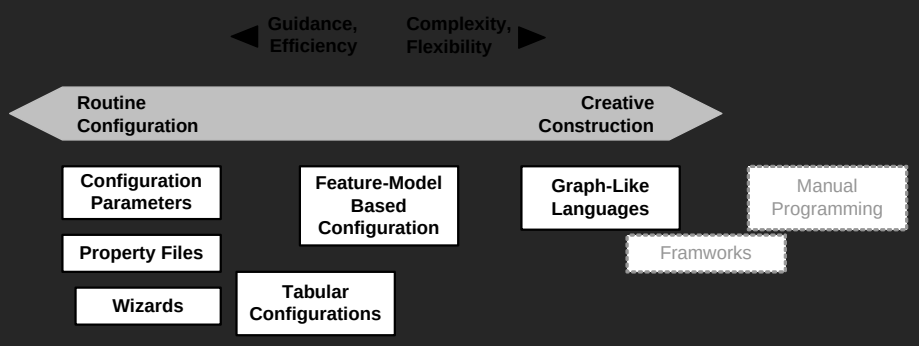
Bind How

Kinds of Variability

Bind When

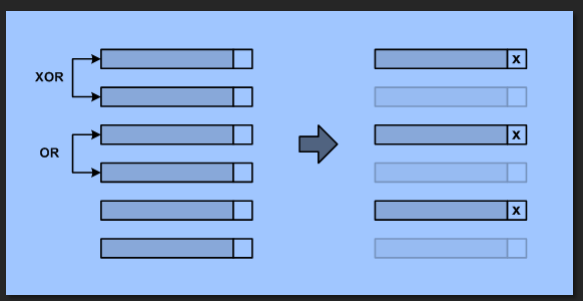


Configuration vs. Customization
Variability



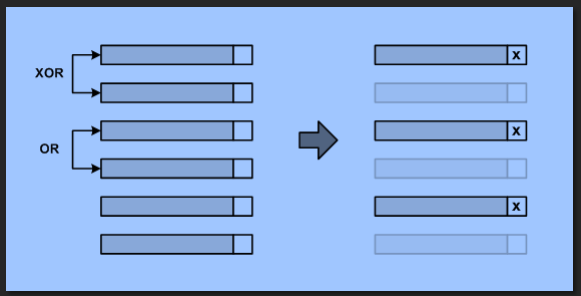
Configuration

… selecting options
… setting param values

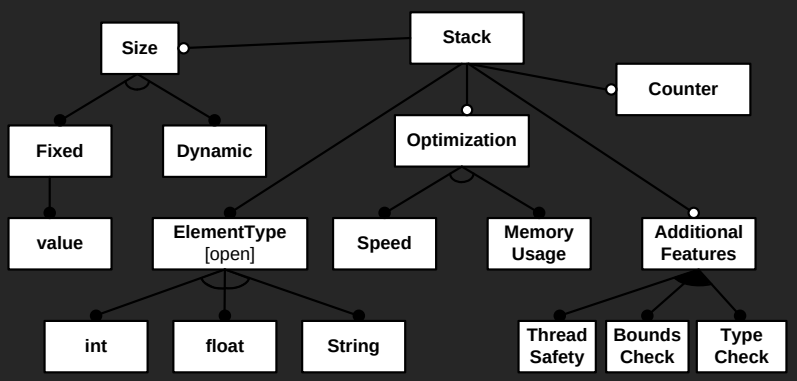


Configuration

… selecting options
… setting param values

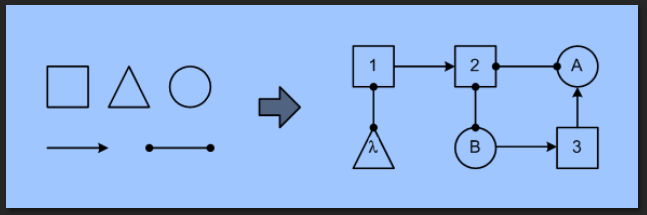


Configuration
Feature Models



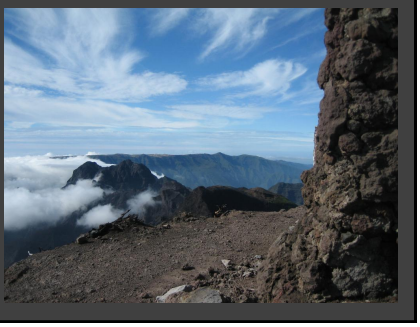
Customization

… DSLs
… instantiation
… connections



4 PLE Concepts & DSLs

Bind How
Kinds of Variability
Bind When



Binding Time

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

Interpretation vs. Generation

Generation

resulting code
can be easily
inspected

Interpretation vs. Generation

Generation

resulting code
can be easily
debugged

Interpretation vs. Generation

Generation

resulting code
can be
optimized
and more
efficient

Interpretation vs. Generation

Generation

Templates can be

derived

from existing code

Interpretation vs. Generation

Generation

work around

limitations

of target language

Interpretation vs. Generation

Generation

no changes

to target environment

(leaves no trace)

Interpretation vs. Generation

Generation

reuse

runtime infrastructure

(garbage collection, monitoring…)

Interpretation vs. Generation

Interpretation

faster

turnaround

no regeneration
test
build
deploy

Interpretation vs. Generation

Interpretation

for platform indepenence
an interpreter might be

less porting effort

Interpretation vs. Generation

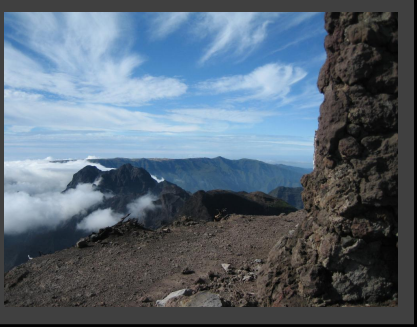
Interpretation

possibly direct feedback from

in-IDE interpreter

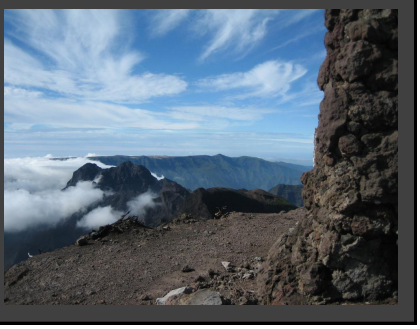
5

Customization and Configuration



5 Customization and Configuration

In-Language
Varying models
DSLs as Parameters
Trafo Variability



In language

The language provides **explicit** support!

(we can learn that from GPLs)

In language

Conditionals

conditional execution

Specialization

overriding, overwriting

Leaving Holes

for variant to fill in

Inject Stuff

several places at a time?

In language

Conditionals

#ifdef

Specialization

inheritance

Leaving Holes

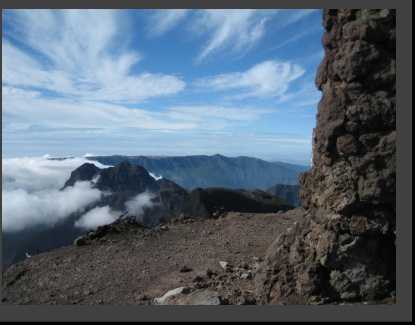
template methods

Inject Stuff

AOP

5 Customization and Configuration

In-Language
Varying models
DSLs as Parameters
Trafo Variability



Two Levels

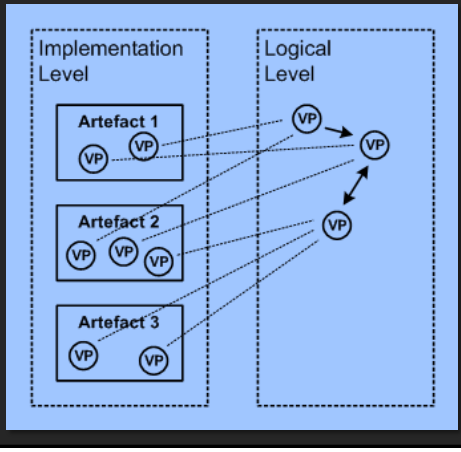
~ problem space
vs. software space

Problem Space:
Configuration

Solution Space:
Customization

Two Levels

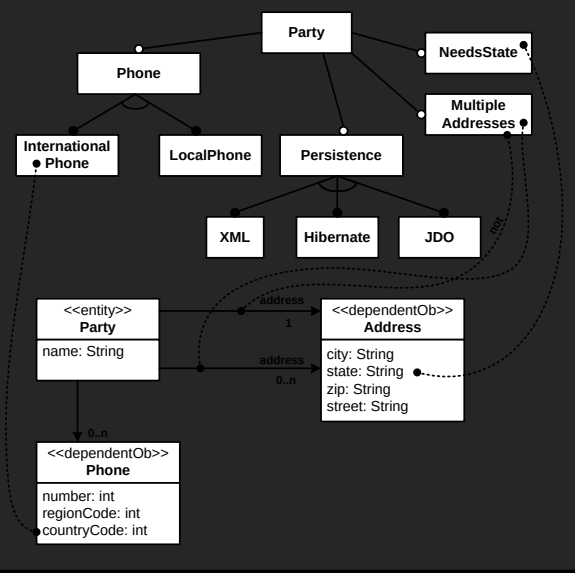
~ problem space vs. software space



Two Levels Removal

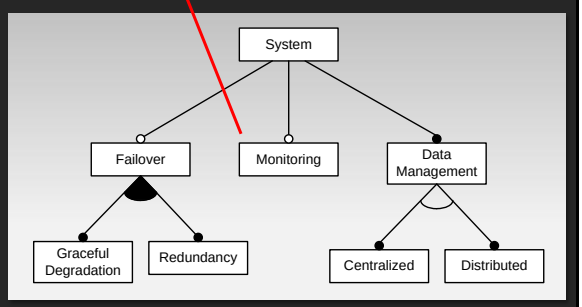
```
#if defined (ACE_HAS_TLI)
  static ssize_t t_snd_n (
         ACE_HANDLE handle,
         const void *buf,
         size_t len,
         int flags,
         ACE_Time_Value *timeout = 0,
         size_t *bytes_transferred = 0);
#endif /* ACE_HAS_TLI */
```

Two Levels Removal



Two Levels Removal

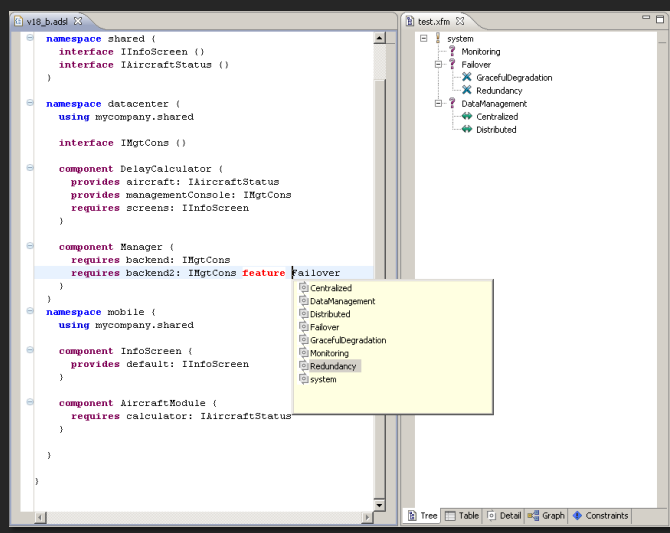
```
component DelayCalculator {
  provides default: IDelayCalculator
  requires screens[0..n]: IInfoScreen
  provides mon: IMonitoring feature monitoring
}
```



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
  }
}
```

Two Levels

~ varying models, not text

… fewer var points
… semantically meaningful
… more manageable

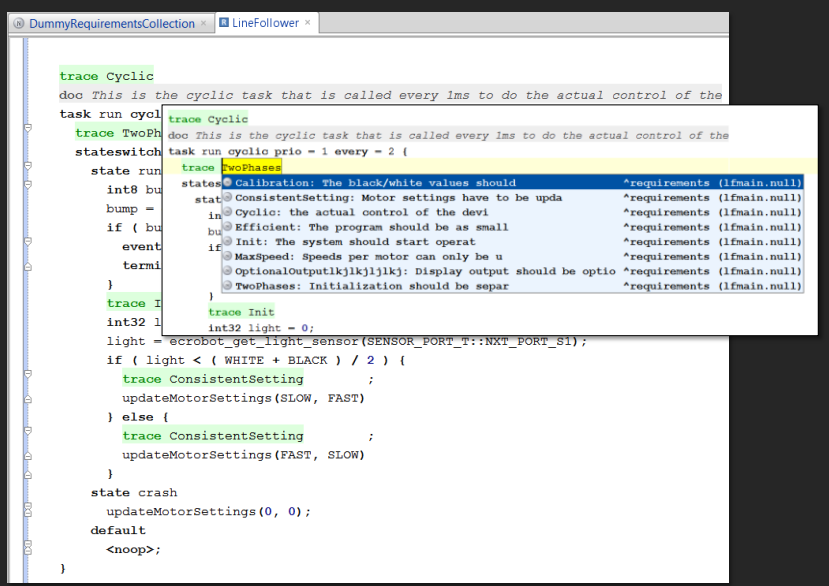
Two Levels

~ varying models, not text

… simplifies traceability

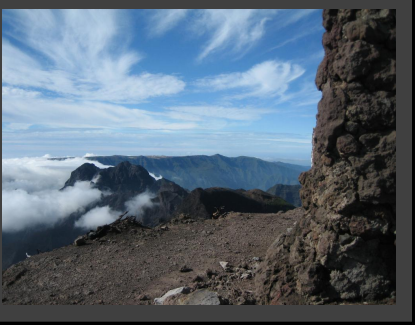
you can „point“ easily to any element -> generic traceability approach

… simplifies traceability



5 Customization and Configuration

In-Language
Varying models
DSLs as Parameters
Trafo Variability



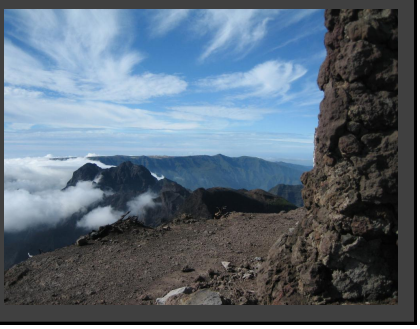
Feature Model w/ Paramters

… parameters with simple types

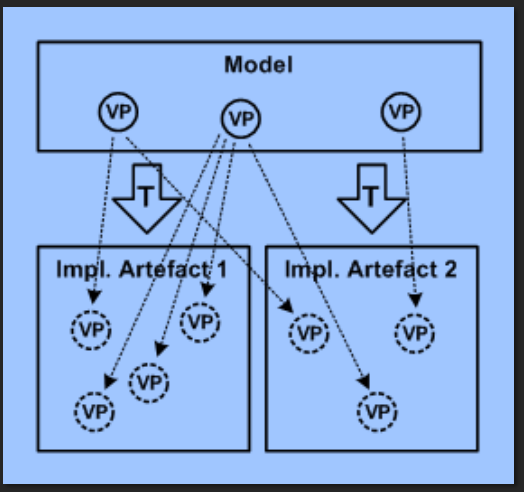
… complex types: DSL/metamodel

5 Customization and Configuration

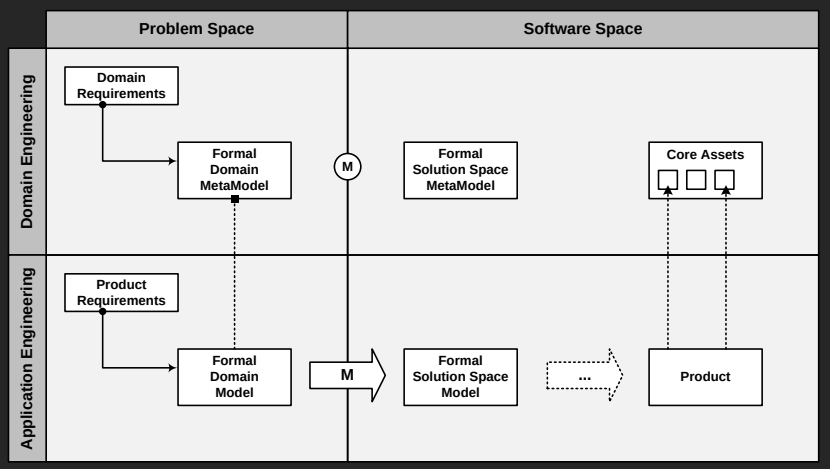
In-Language
Varying models
DSLs as Parameters
Trafo Variability



Model-Based Implementation



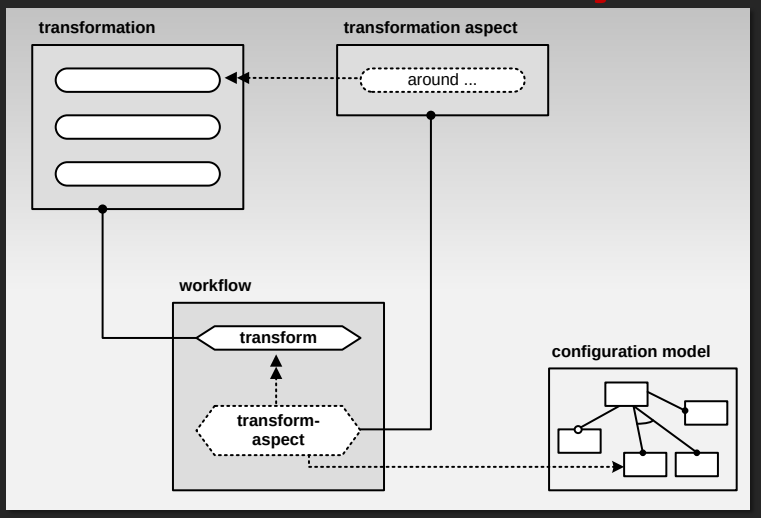
Model-Based Implementation



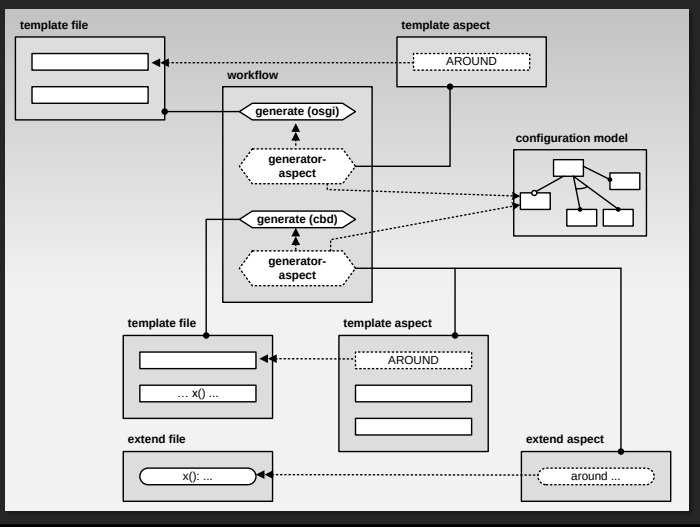
Transformation Variability

```
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
);
```

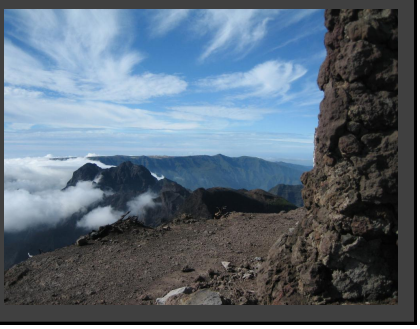
Transformation Variability



Generator Variability



6 PLE for Languages



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

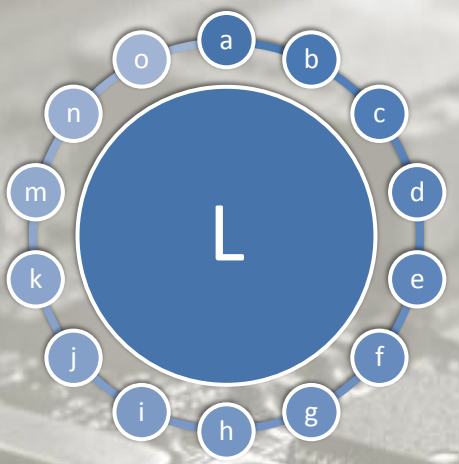
We don't want to
model,
we want to
program!

And always:
precise and tool processable

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

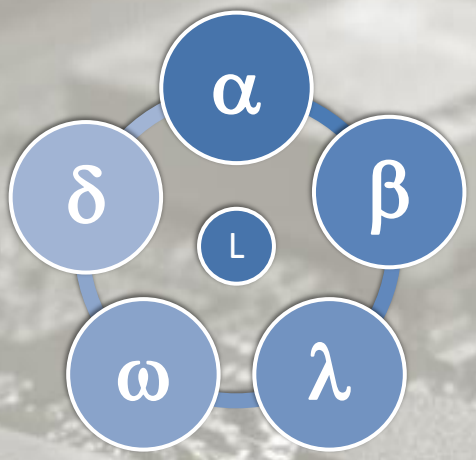
PLE for DSLs
suitable to related domains

Big Language?



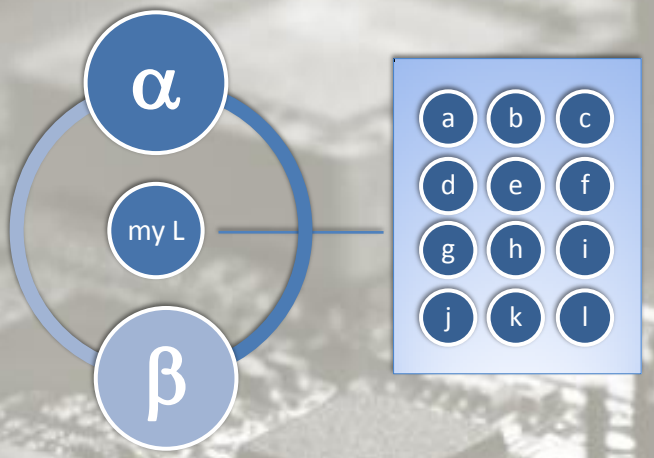
with many first class concepts!

Small Language?



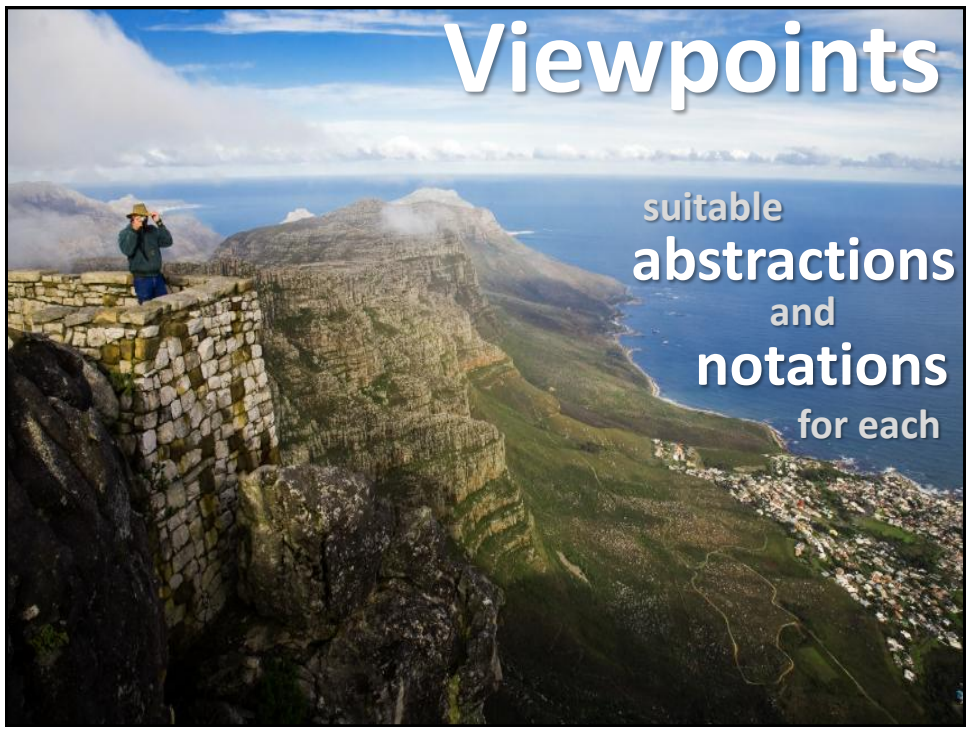
with a few, orthogonal
and poweful concepts

Modular Language

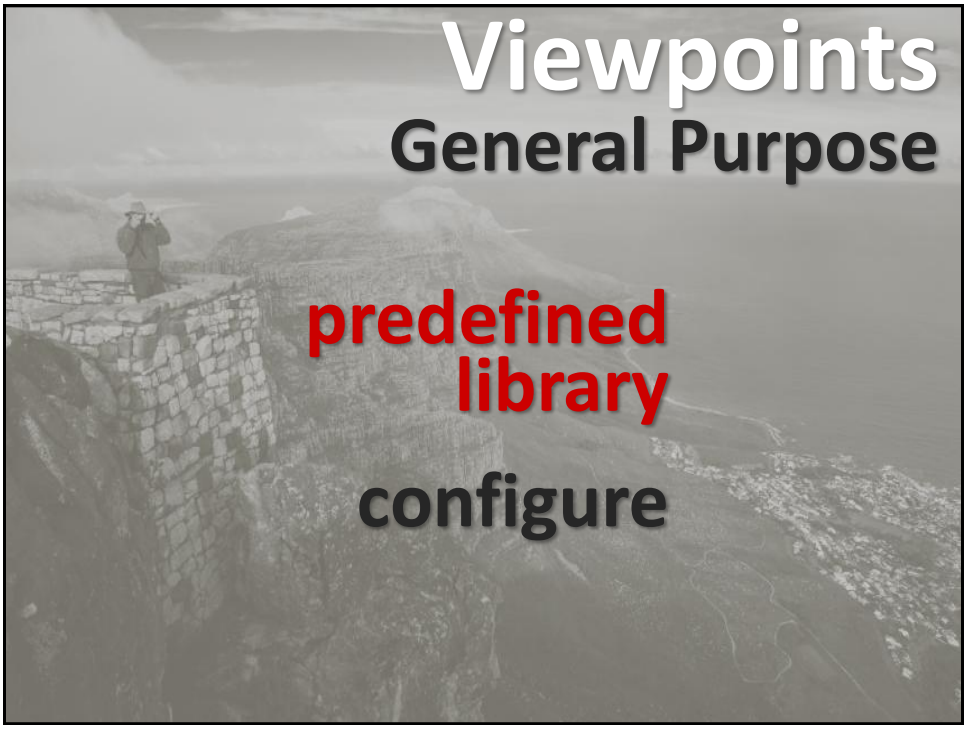


with many optional,
composable concepts


Viewpoints


Viewpoints
suitable
abstractions
and
notations
for each


Viewpoints
Integrated
via
symbolic
references
and
seamless
transitions


Viewpoints
General Purpose
predefined
library
configure

Viewpoints
Domain Specific

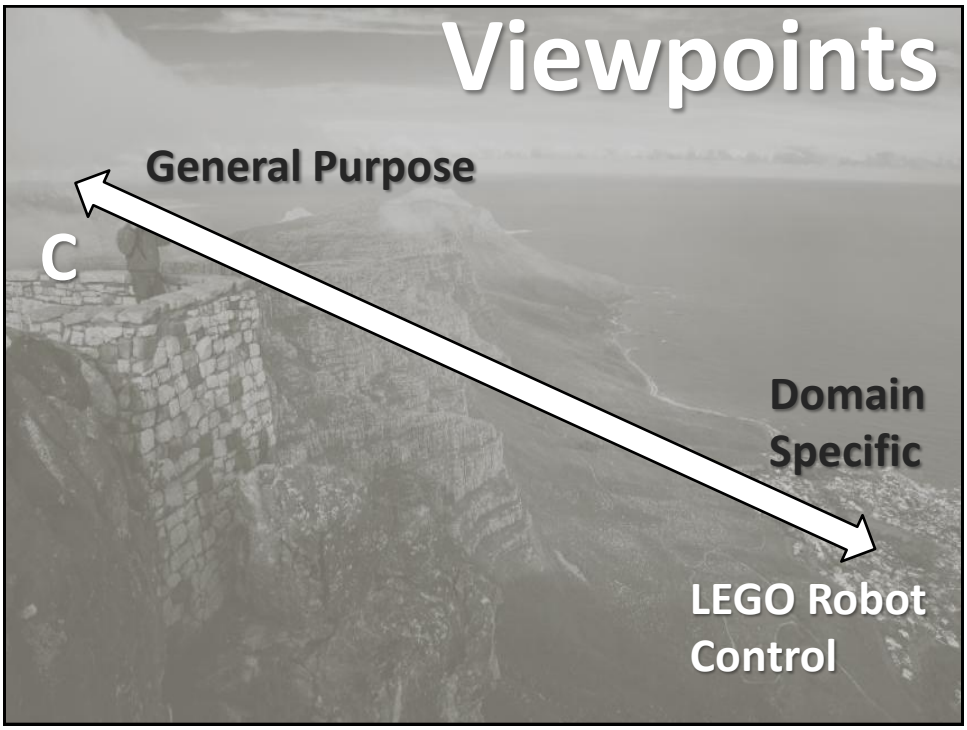
custom
purpose-built

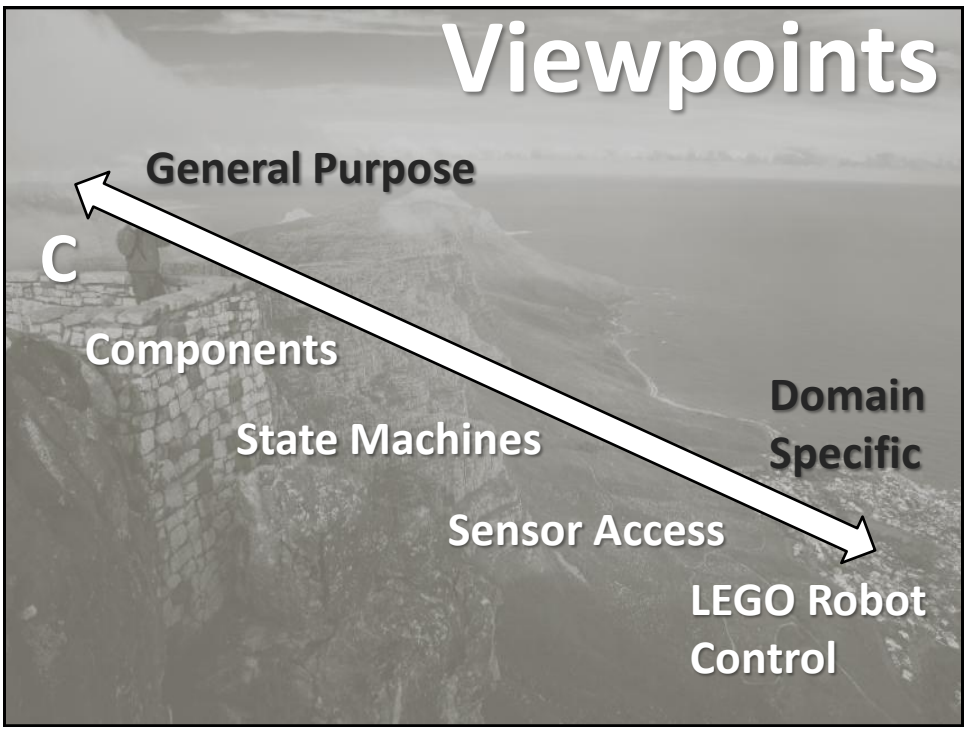
create/include

Viewpoints
Domain Specific

Custom
Notations

real
business
expert integration


Viewpoints
General Purpose
C
Domain
Specific
LEGO Robot
Control


Viewpoints
General Purpose
C
Components
State Machines
Sensor Access
Domain
Specific
LEGO Robot
Control

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!

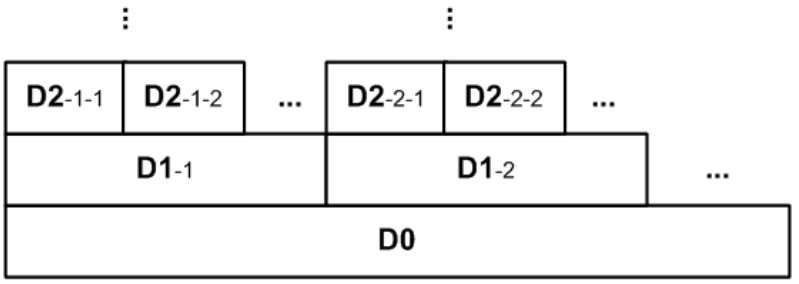
```
statemachine linefollower {
  event initialized;
  event bumped;
  event blocked;
  event unblocked;
  initial state initializing {
    initialized [true] -> running
  }
  state paused {
    entry int16 i = 1;
    unblocked [true] -> running
  }
  state running {
    blocked [true] -> paused
   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

Linguistic Abstraction

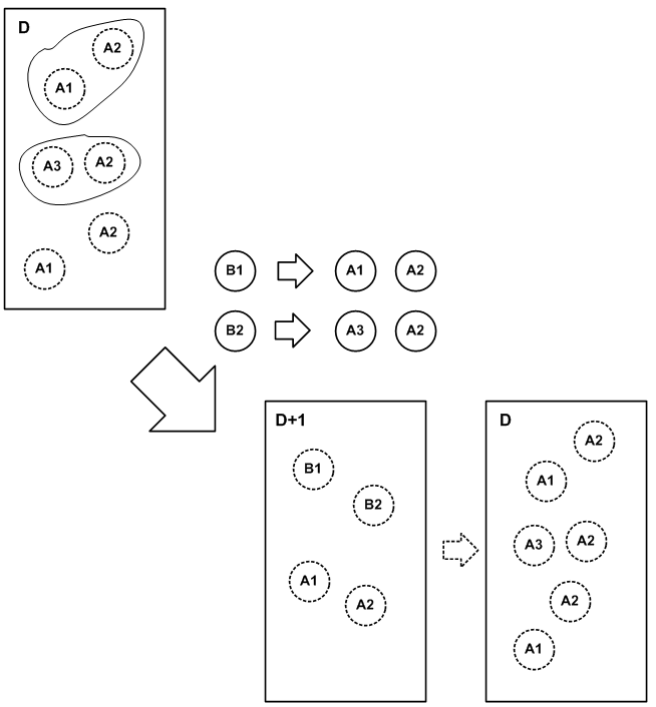
Much better!

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

Domains are Hierarchical



Creating Linguistic Abstractions

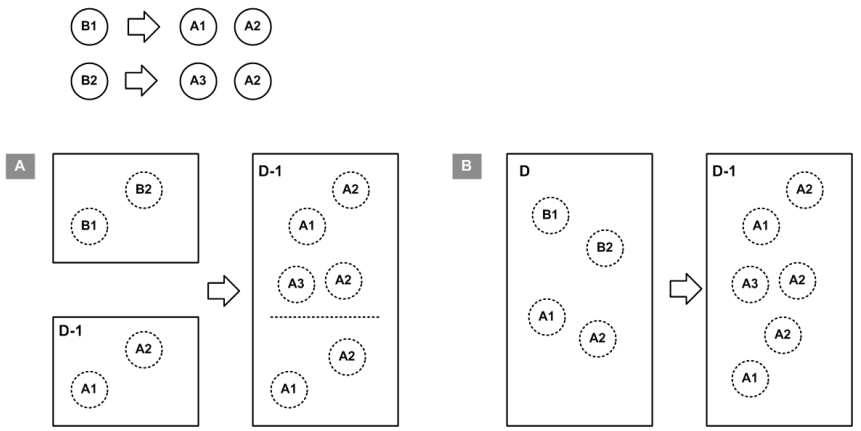


Domain Specific Languages

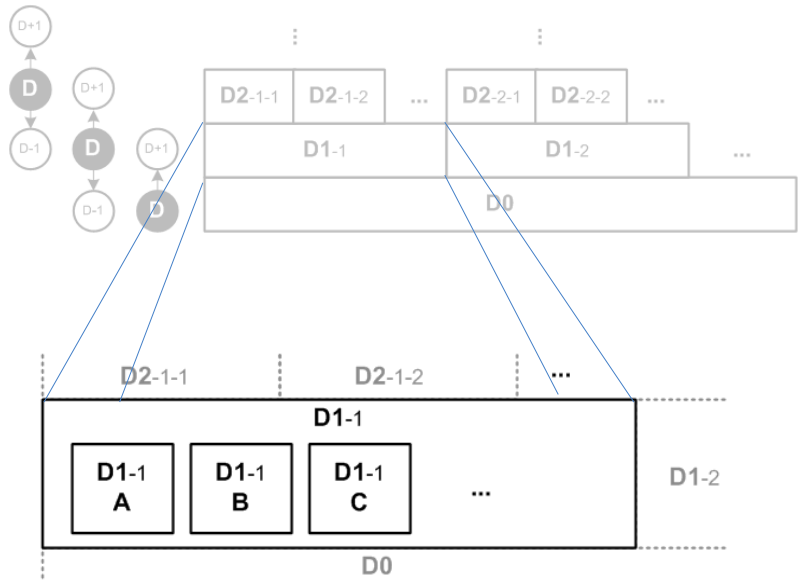
A DSL is a **language** at D that provides **linguistic abstractions** for **common patterns and idioms** of a language at D-1 when used within the domain D.

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.

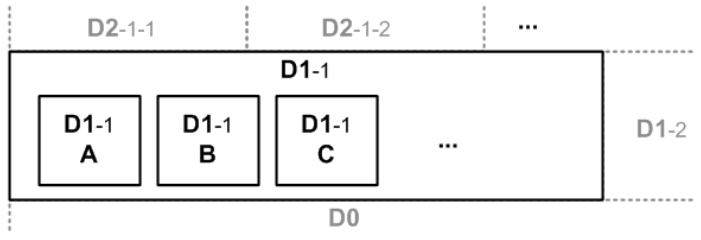
Extension vs. Cascading



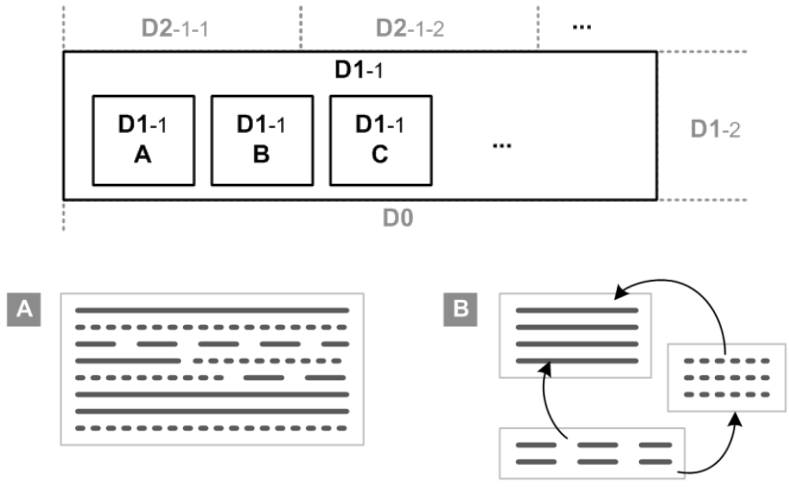
Several Concerns in a Domain



Viewpoints

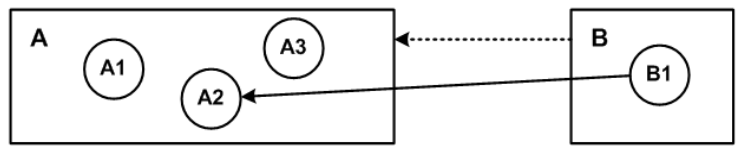


Viewpoints



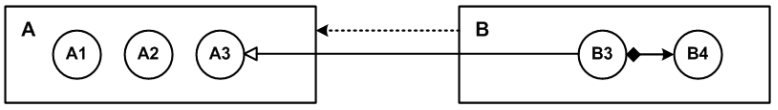
A: mixed
B: separate Viewpoints

Langugae Combination



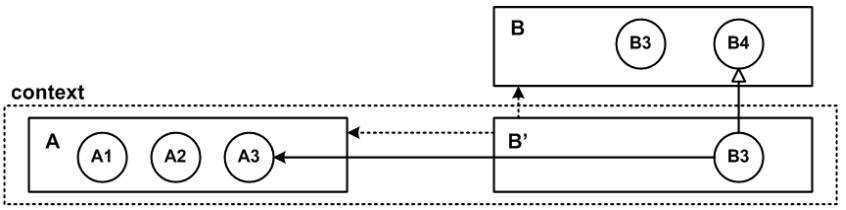
When DSL (for several concerns) are developed from scratch, as a group, then dependencies between the concerns can be materialized as dependencies between the languages and the language concepts

Langugae Extension



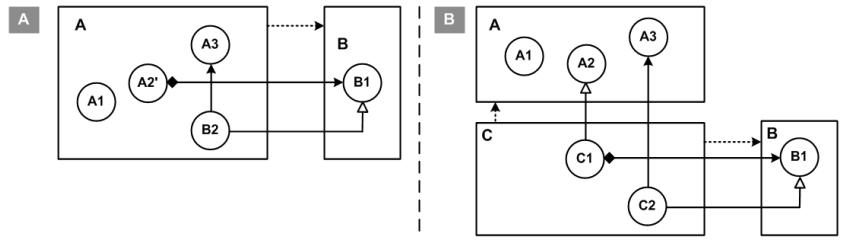
A language B extends another language A if B contains additional language concepts. This means that for programs written in B, all concepts from A are available, plus those defined in B.

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.

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 contr
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

```
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

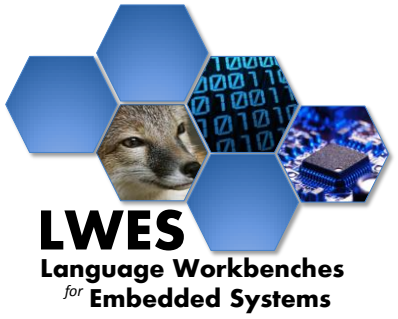
```
import blocks Environment
              Aircraft
              Stuff on the Wings
public class TestClass extends <none> implements <none> {
  <<static fields>>

  <<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>>
}
```

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

mbeddr.com

Home | Overview | Publications | MPS | Showcase | Code | Team

Search

Feeds: Posts Comments

LWES
Language Workbenches for Embedded Systems
http://mbeddr.com

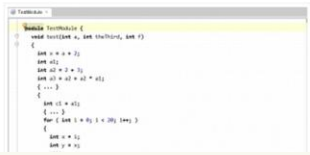
Bundesministerium für Bildung und Forschung

gefördert durch das Bundesministerium für Bildung und Forschung, Förderkennzeichen 01|S11014

First C Code working

July 17, 2011 by mpscmed

As you may know, our project relies on the idea of extending the C programming language with domain specific extensions. For that to work, we first have to make C available in MPS. While we had done this to some extent in our proof of concept, we are now implementing C much more thoroughly. As you can see in the screenshot below, some essential things are already working.



Archives

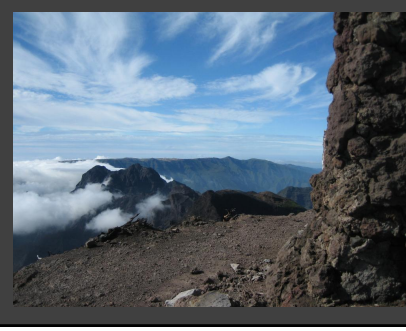
- July 2011 (3)
- June 2011 (2)
- January 2011 (2)
- July 2010 (1)
- June 2010 (2)

Categories

- code (3)
- demos'n'stuff (4)
- dev progress (1)
- news (4)
- Uncategorized (2)

Pages

7 DSL 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

xtext

MPS Meta Programming System

INTENTIONAL® SOFTWARE

xtext

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

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

MPS Meta Programming System

Open Source (Apache 2.0)
Projectional Editor
Very good at lang. Compososition
Version 2.0 is current:

- 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


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