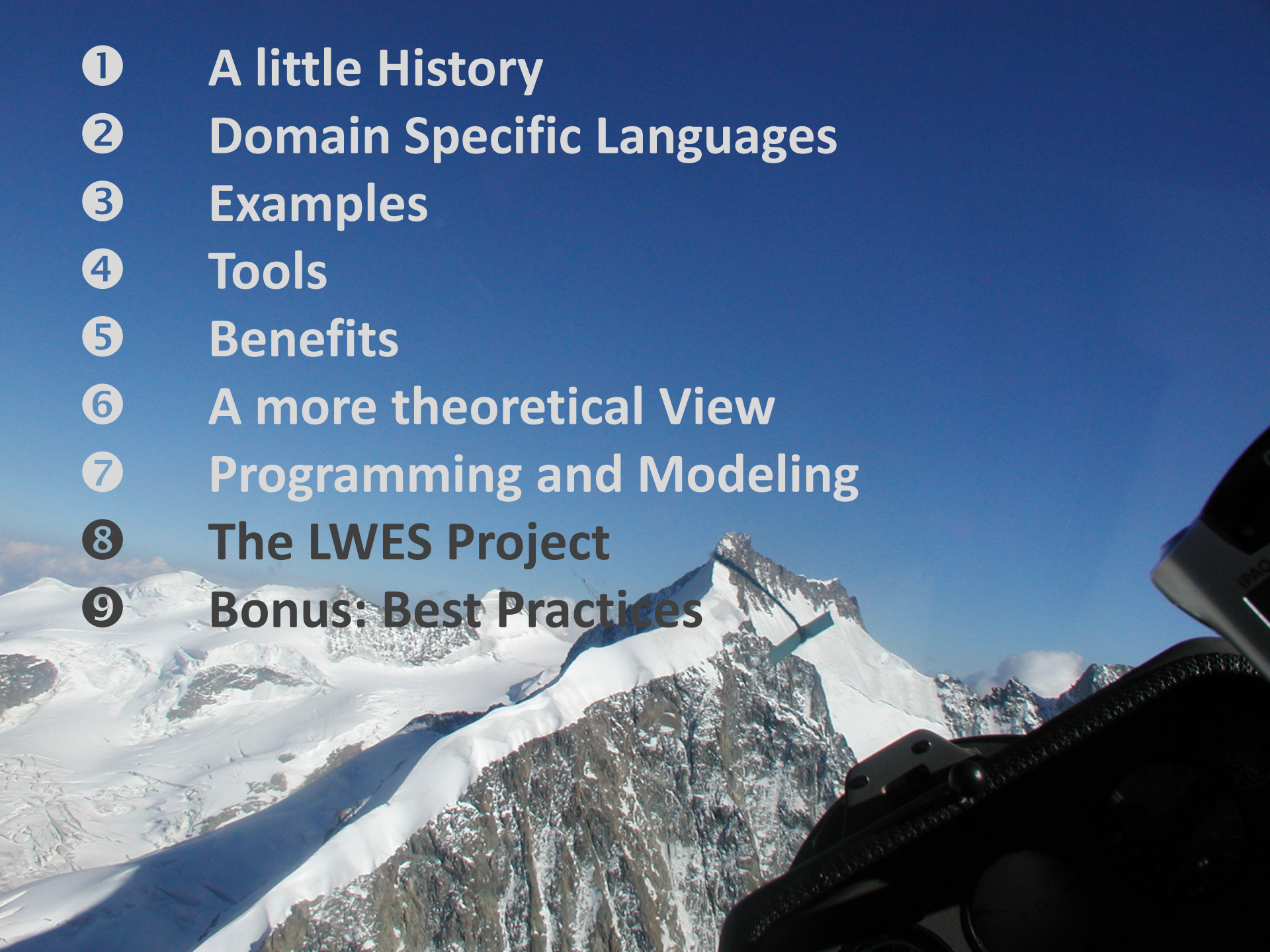
Domain Specific Languages

Markus Voelter
Independent/itemis
voelter@acm.org


❶ A little History
❷ Domain Specific Languages
❸ Examples
❹ Tools
❺ Benefits
❻ A more theoretical View
❼ Programming and Modeling
❽ The LWES Project
❾ Bonus: Best Practices

1

A little History

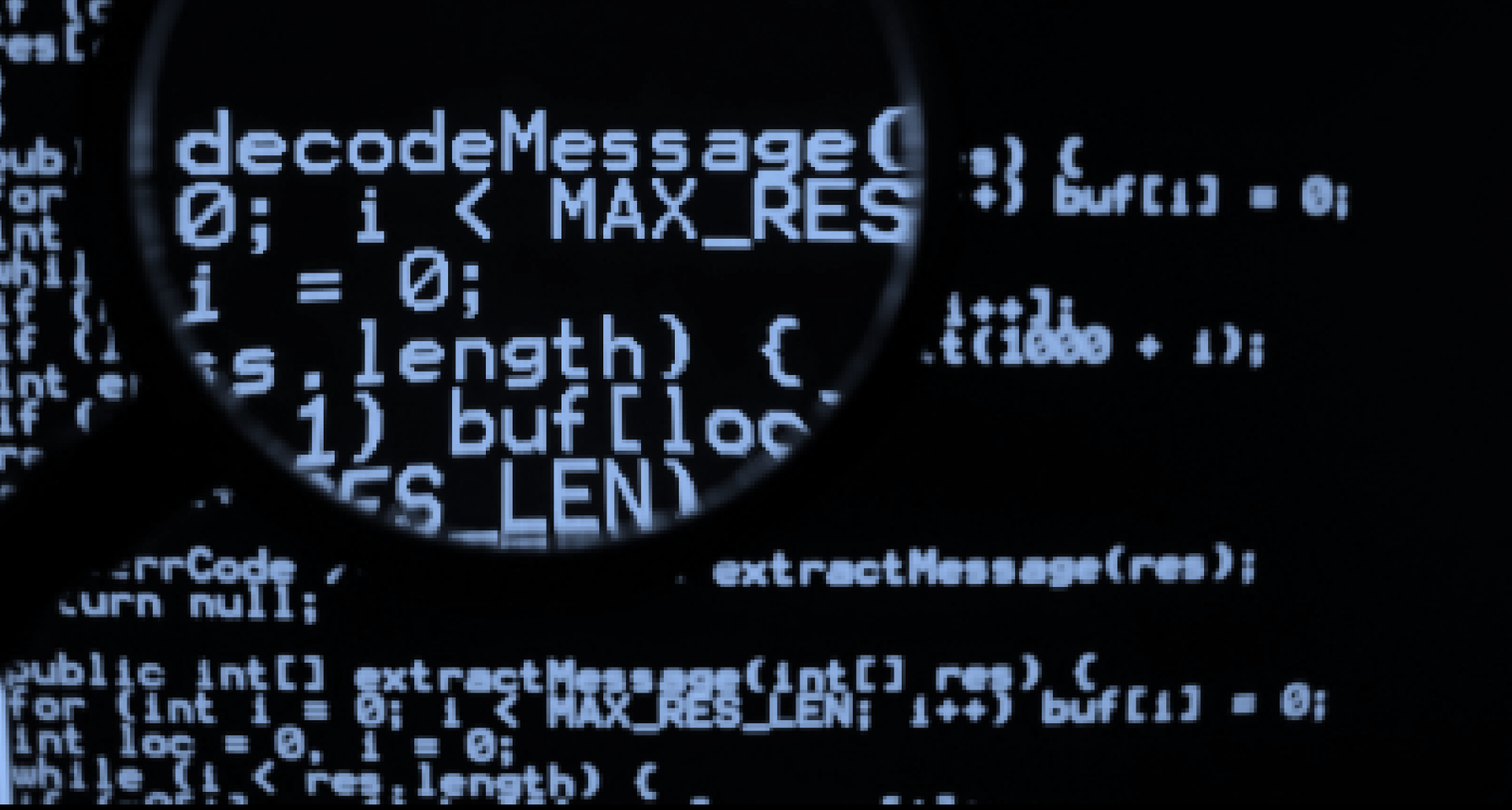
abstractions

~

computing

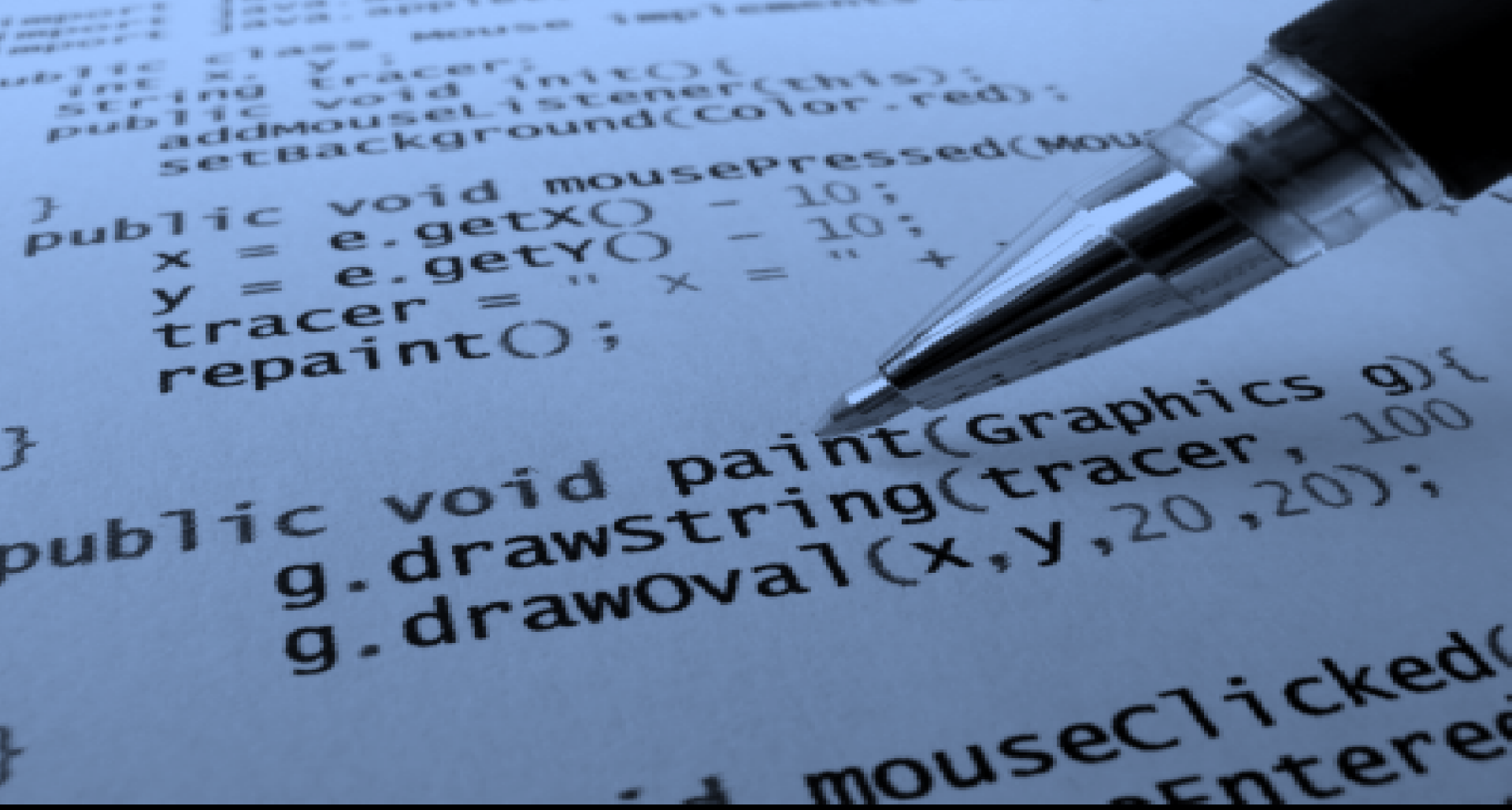
chips


abstractions
~
computing
bits



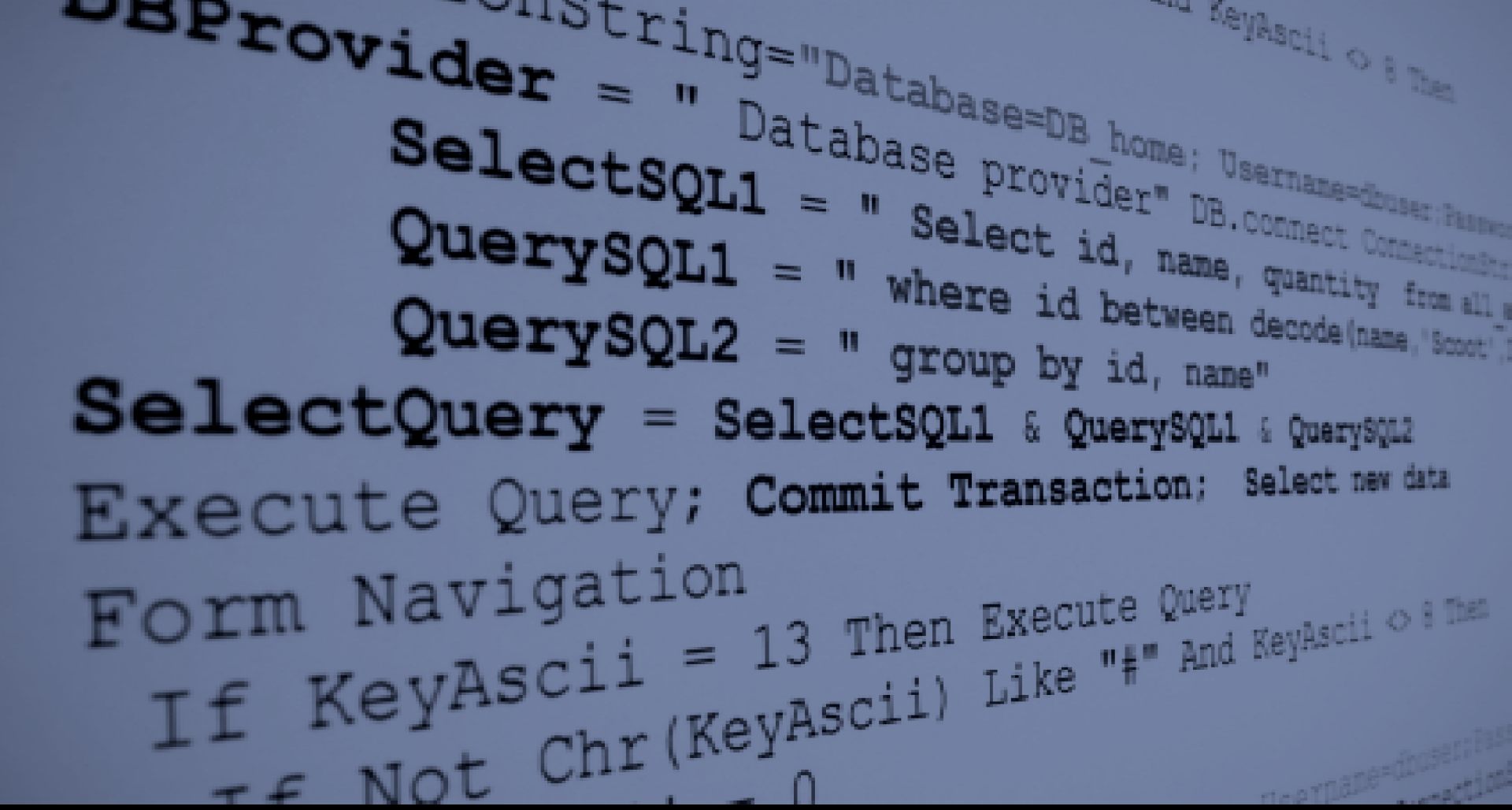
abstractions
~
computing

c



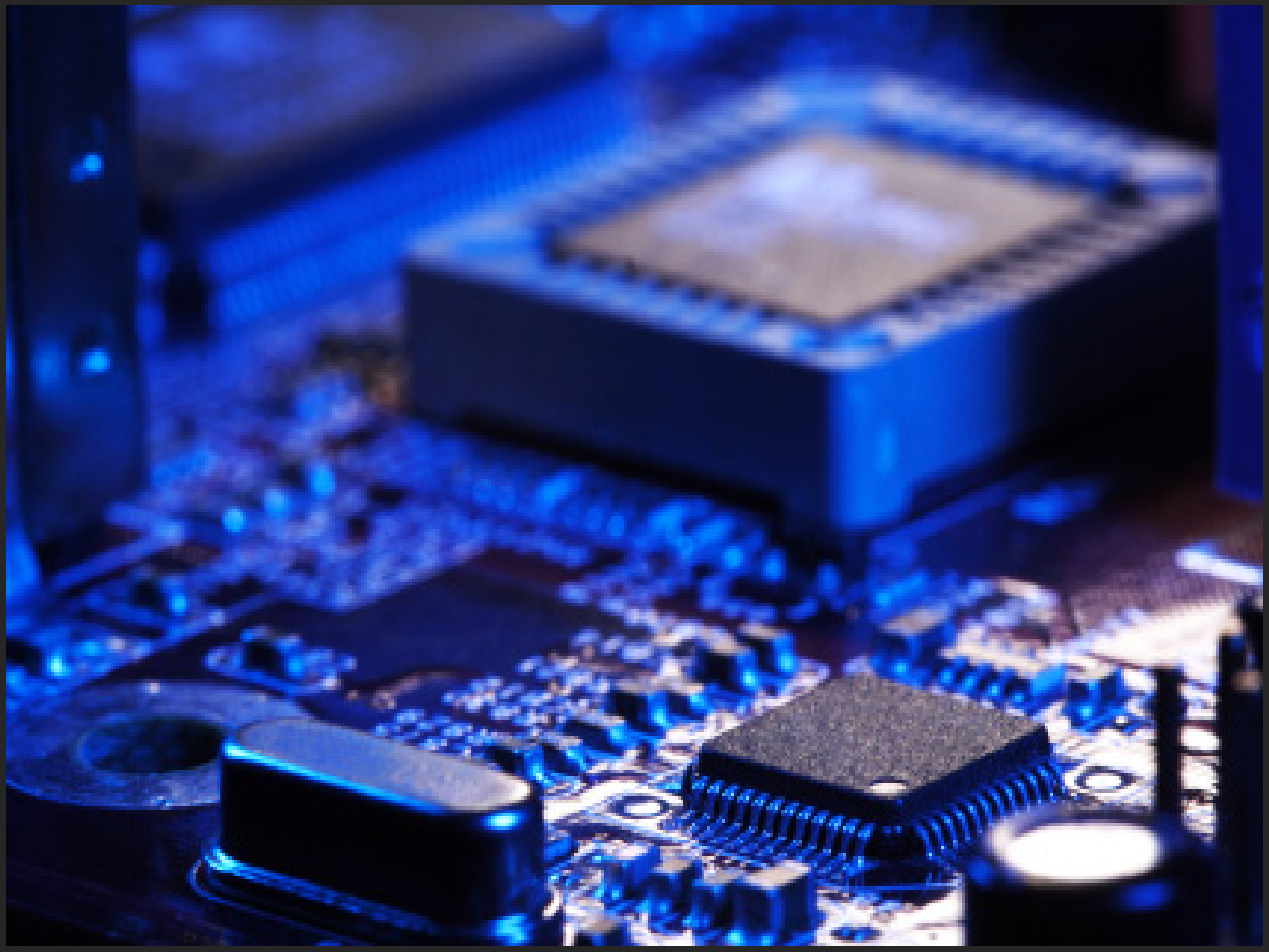
abstractions ~ computing?

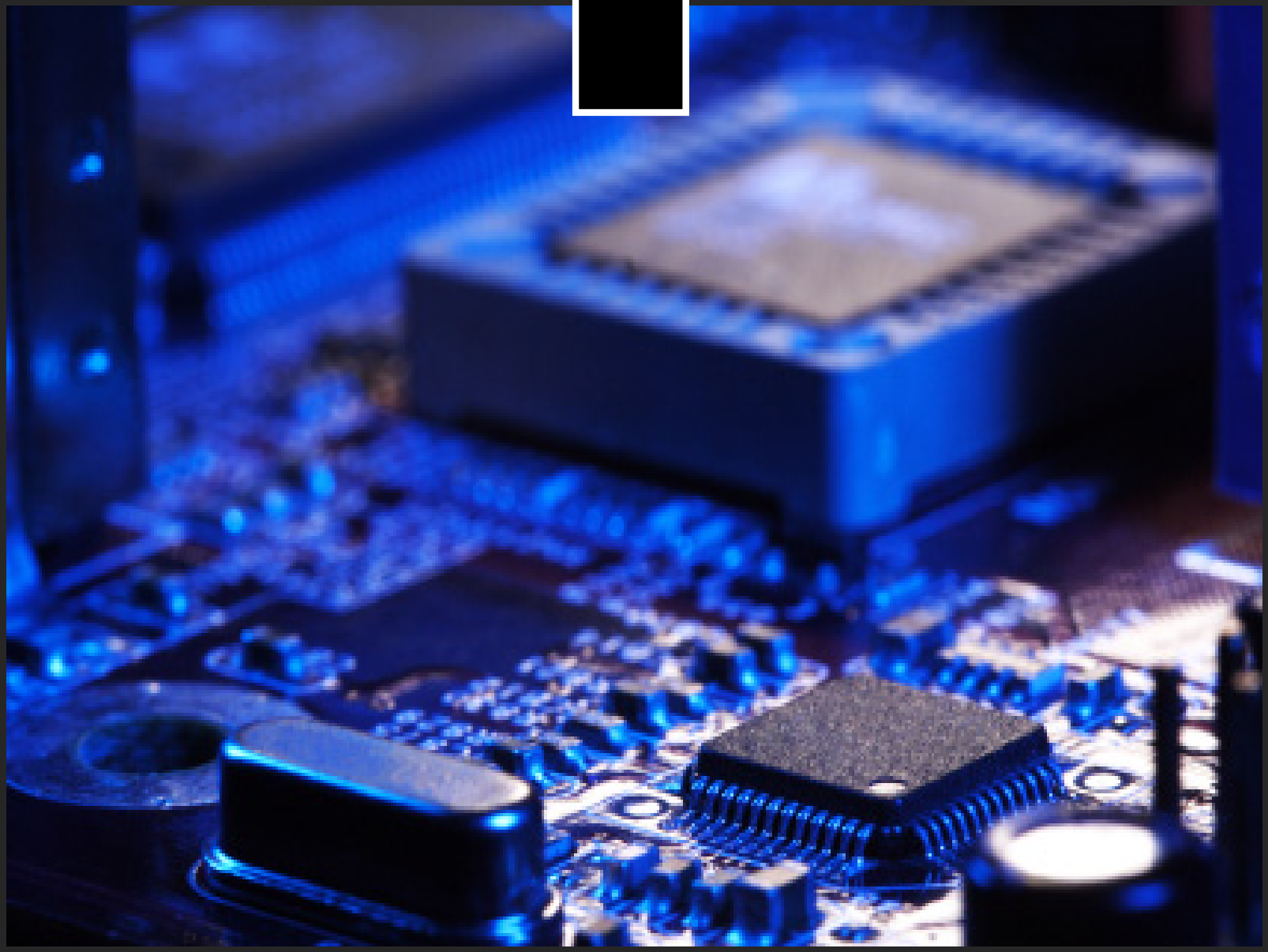
Java



abstractions ~ computing?

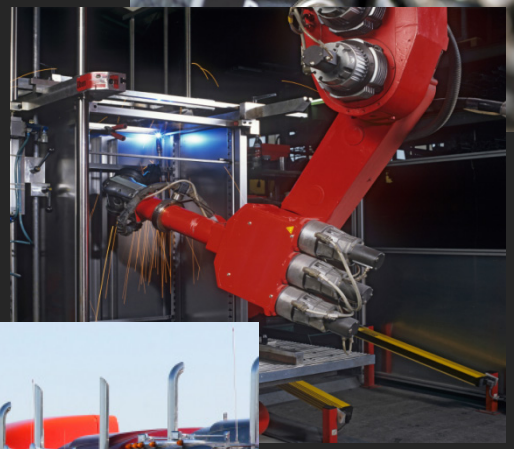
SQL





?


PB



general purpose

domain specific

tailor made

effective++

specialized, limited

used by experts

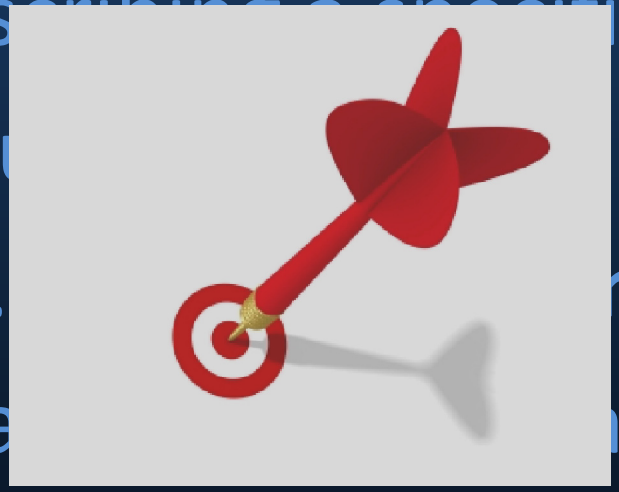
together with other specialized tools

2

Domain Specific Languages

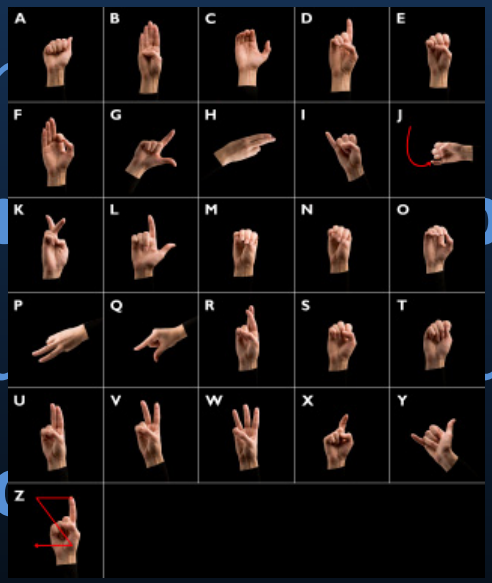
A DSL is a **focussed**, **processable language** for describing a specific **concern** when building a system in a specific **domain**. The **abstractions** and **notations** used are natural/suitable for the **stakeholders** who specify that particular concern.

A DSL is a **focussed**, **processable** **language** for des ic **concern** when bu in a specific **domain**. **ns** and **notations** used a ble for the **stakeholders** who specify that particular concern.



A DSL is a **focussed**, **processable** **langua** pecific **concer** ystem in a specifi **actions** and **notati** /suitable for the **sta** ify that particular concern.

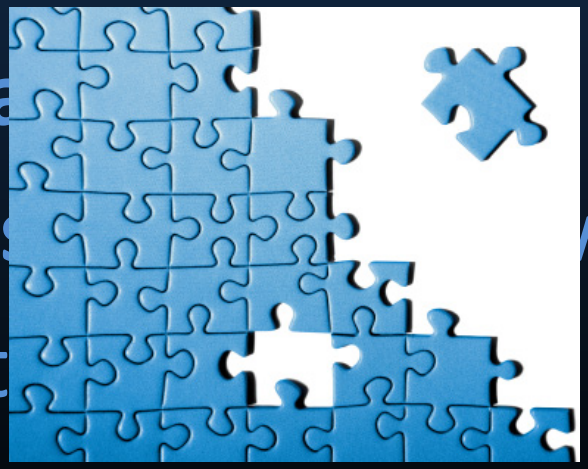
A DSL is a **focussed**, **processable** **language** for describing a specific **concern** wh a system in a specific **do** **ostractions** and **notations** u ural/suitable for the **stakeho** pecify that particular concern.



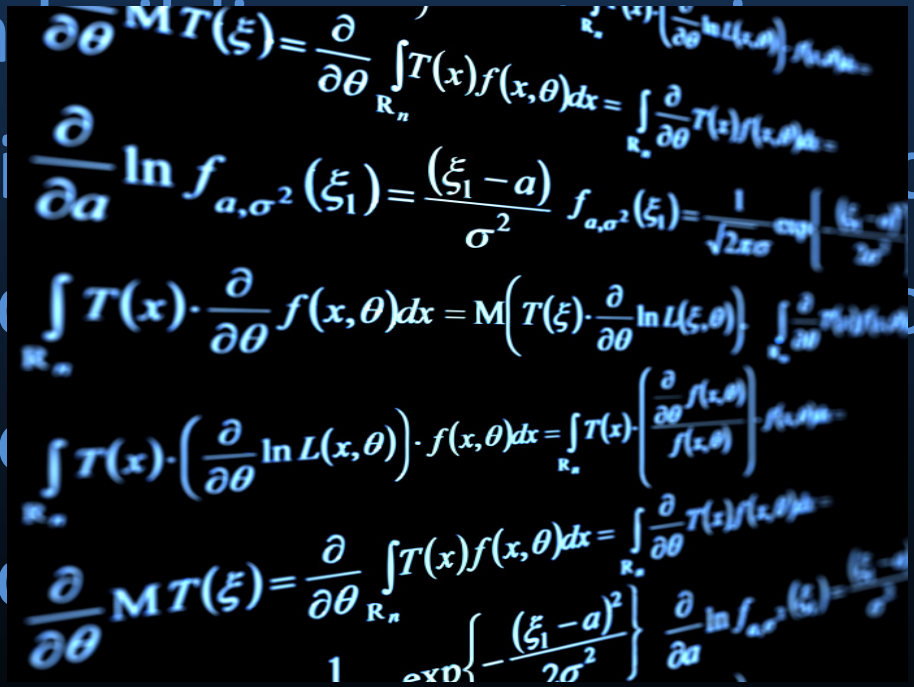
A DSL is a **focussed**, **processable language** for describing a specific **concern** when building a system in a specific **domain** and **notations** used for the **stakeholders** with particular concerns

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 a e for the **stakeholders** particular concer

A DSL is a **focussed**, **processable language** for describing a specific **concern** when building a system in a specific **domain**. The **abstractions** and **nota** natural/suitable for the s who specify that part



A DSL is a **focussed**, **processable** **language** for describing a specific **concern** when

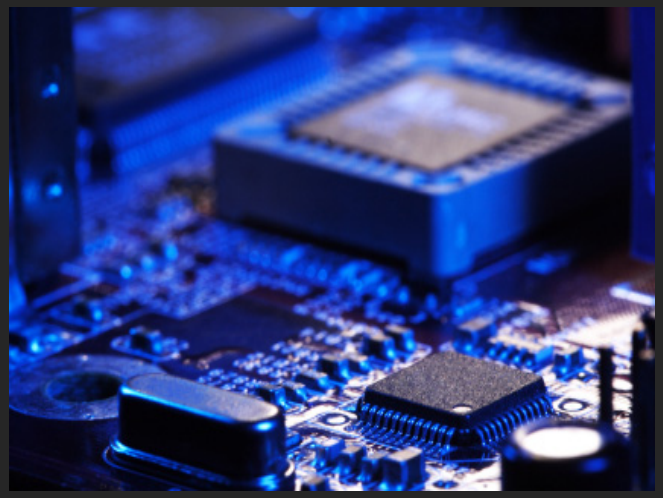


specific **domai** **notations** use the **stakehold** particular con

A DSL is a **focussed**, **processable language** for describing a specific **concern** when building a system in a specific **domain**. The a s and **notations** used are na le for the **stakeholders** who t particular concern.

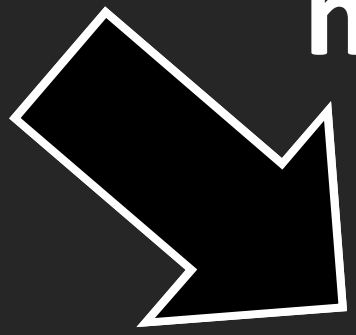
execute?

map



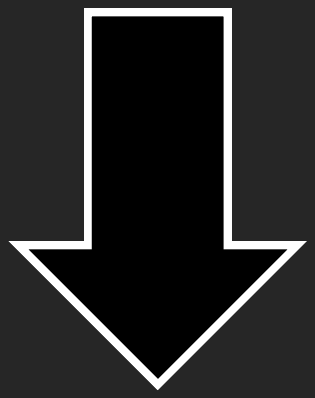
DSL Program
(aka Model)

map



automated!

GPL Program



map

Generation

Transformation

Compilation

Interpretation


LanguageOrientedProgramming
DomainSpecificModeling
GenerativeProgramming
ModelDrivenDevelopment
ModelIntegratedComputing
ModelDrivenEngineering
ModelBasedDevelopment
ModelDrivenArchitecture
ModelBasedEngineering
ModelDrivenSoftwareDevelopment

Activities

Analysing Domains

Defining Languages

Adapting/Selecting

Building Editors

Transforming Models

Building Generators

Building Frameworks

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


3
Examples

Example 1:
Embedded Protocol Handler

Component Specification

SOLUTION

```
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 ;
}
```

Message Format Definition

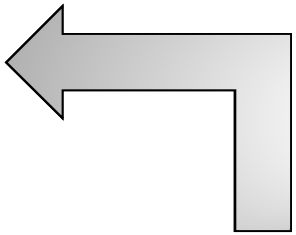
SOLUTION

```
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];
    } ;
}
```

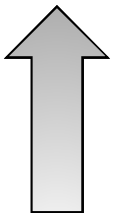
Testing

SOLUTION

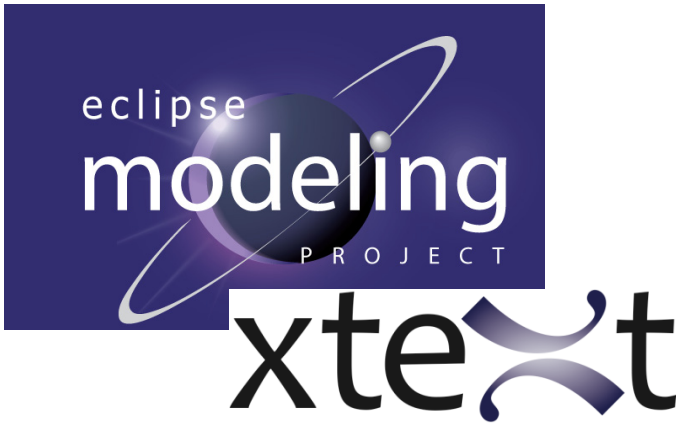
```
procedure writeRegisterNumberZ requestCode 0x29 {
    request: struct request1 {
        int8 acc pattern {
            2:b00;
            6:parentRequestCode;
        };
        int8 registerAddress;
    } ;
    reply: struct dontCareReply {
        int8 statusByte patternref statusByte;
      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




```
test writeRegisterNumberZ for dip writeRegisterNumberZ {
    send request1 {
        attr registerAddress == reg parameterInstruction;
    };
    expect dontCareReply {
        subattr statusByte # standardStatus == 2;
    };
    send request2 {
        subattr registerType # registerType == 3;
        attr registerAddress == reg parameterInstruction;
        attr registerdata == 0x77;
        subattr registerdata # channelNumber == 5;
    };
}
```




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



Eclipse Modeling
Eclipse Xtext

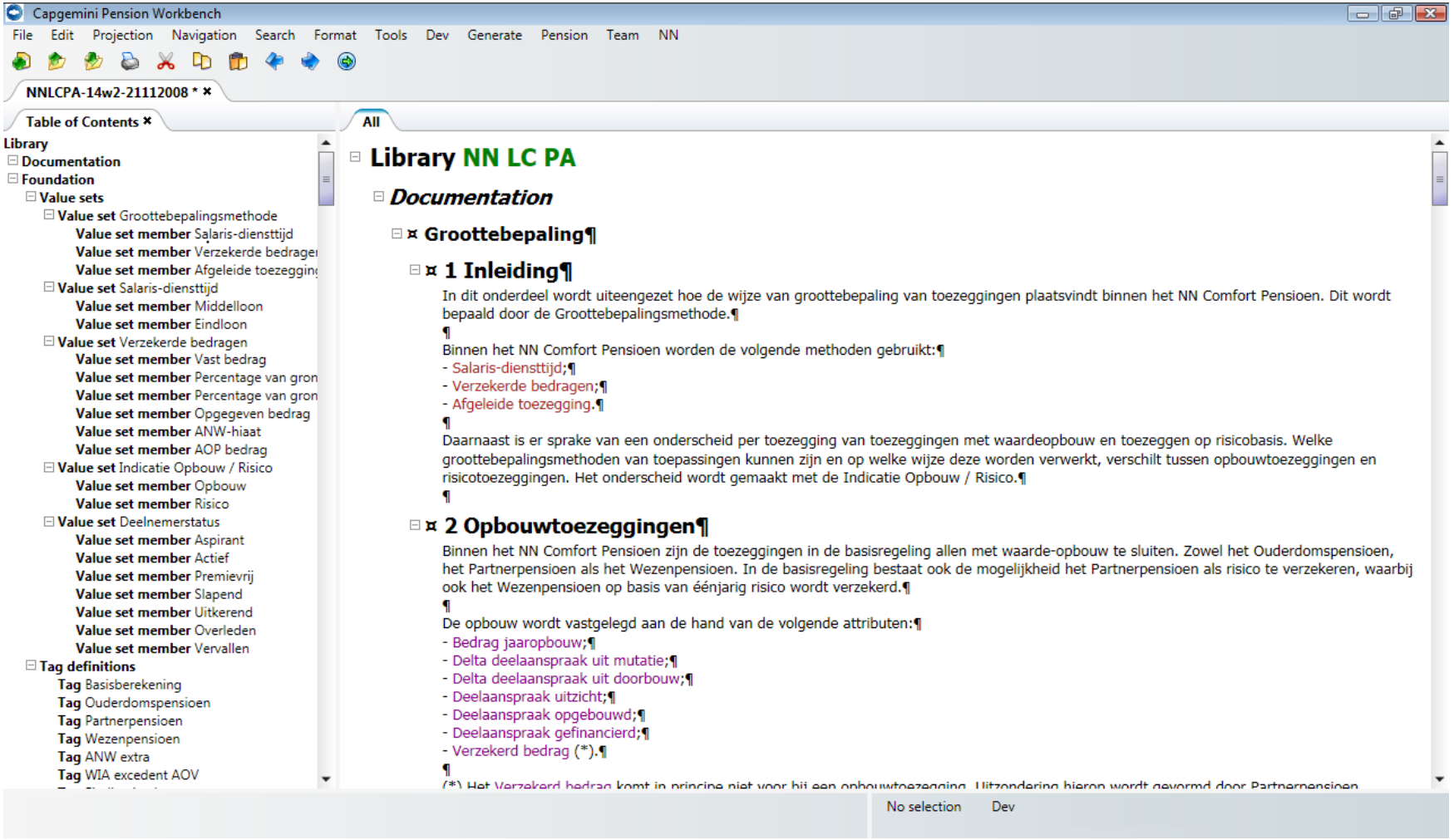
TOOLS

Example 2:

Pension Fund Specification

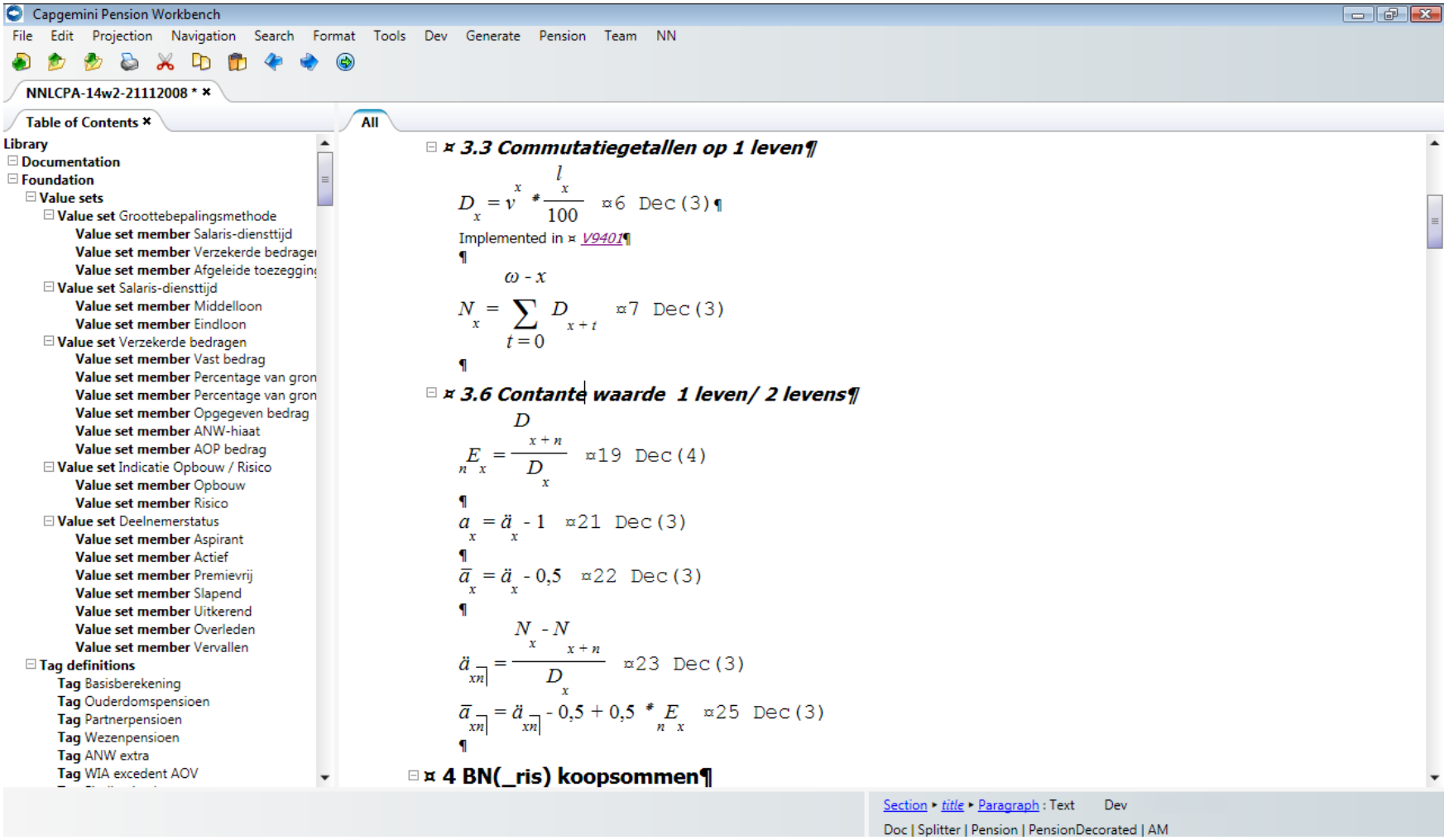
Textual Documentation

SOLUTION



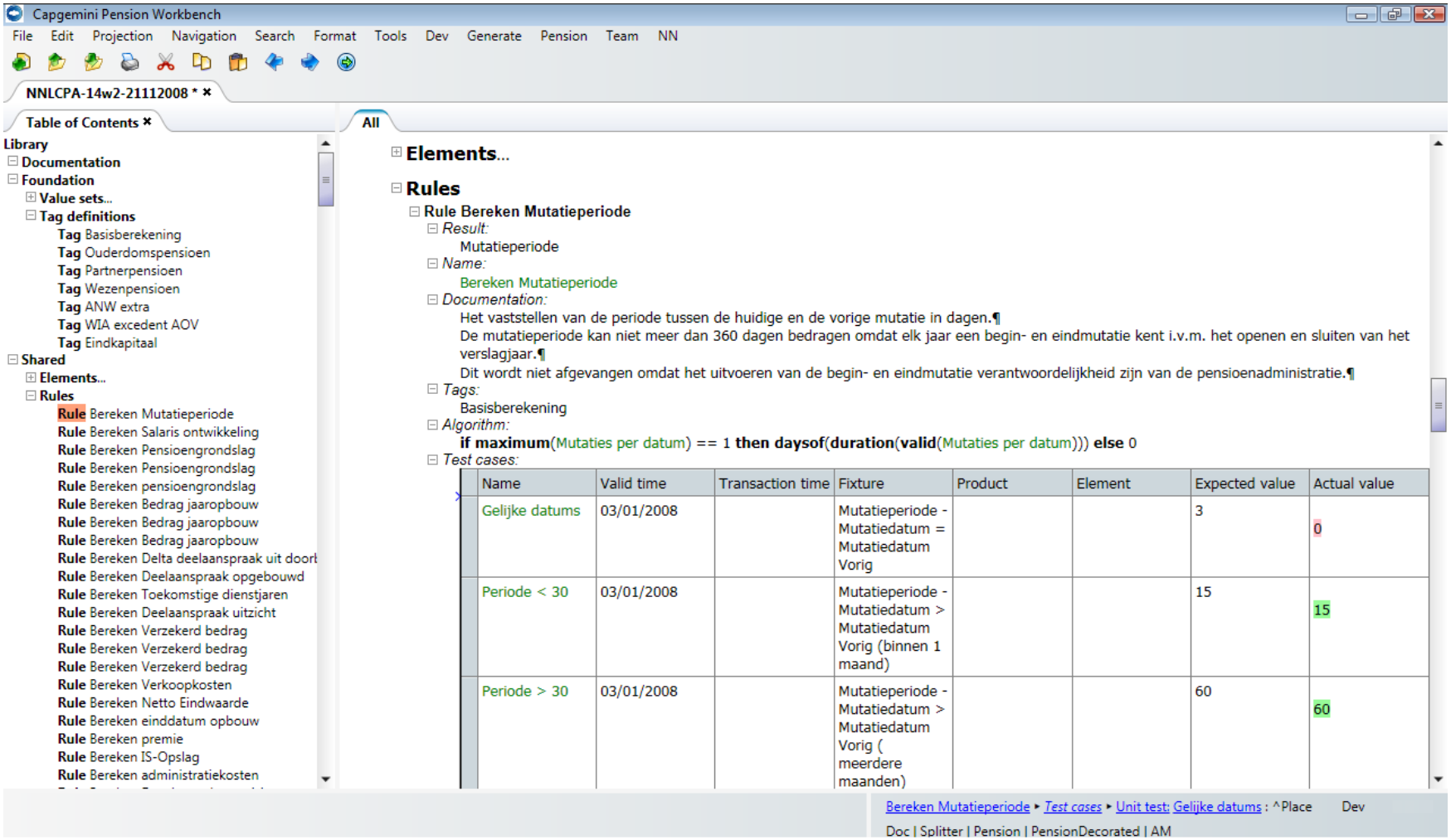
Insurance Mathematics

SOLUTION



Calculation Rules and Tests

SOLUTION



Intentional Software‘s Domain Workbench

TOOLS

Example 3:

Radar Systems Engineering

Component Definition

SOLUTION

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

}
```

Component Behavior Specification

SOLUTION

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 *)
```

Resulting System Behaviour

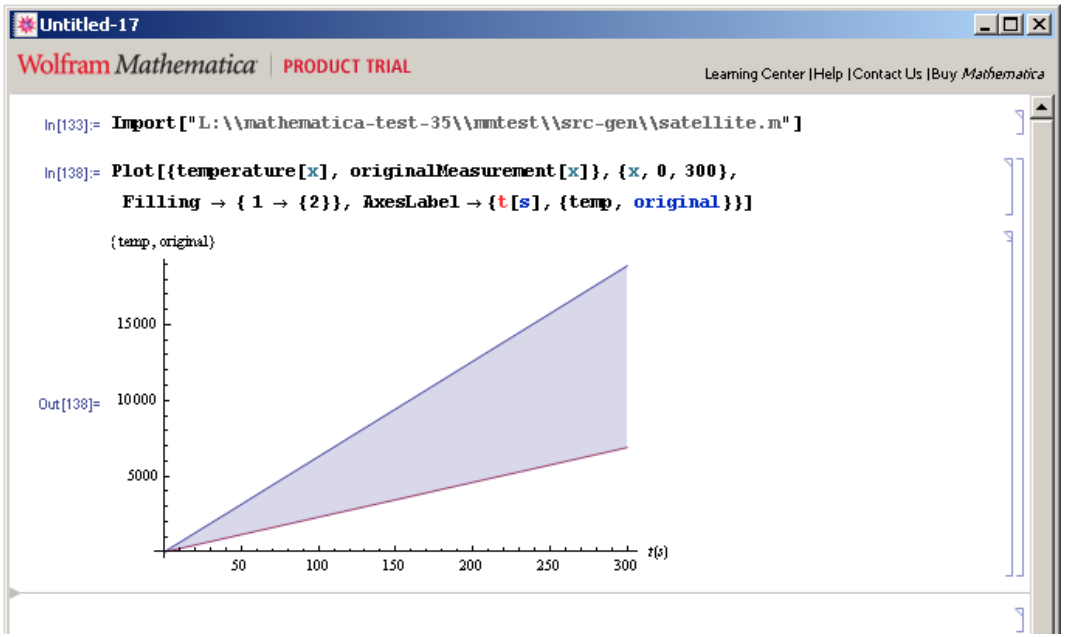
SOLUTION

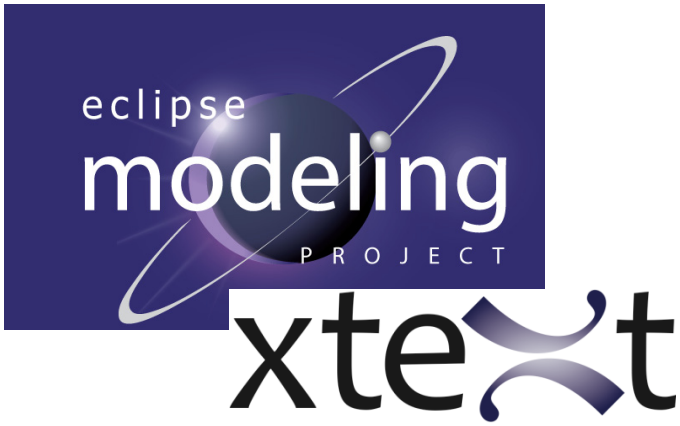
satellite.m

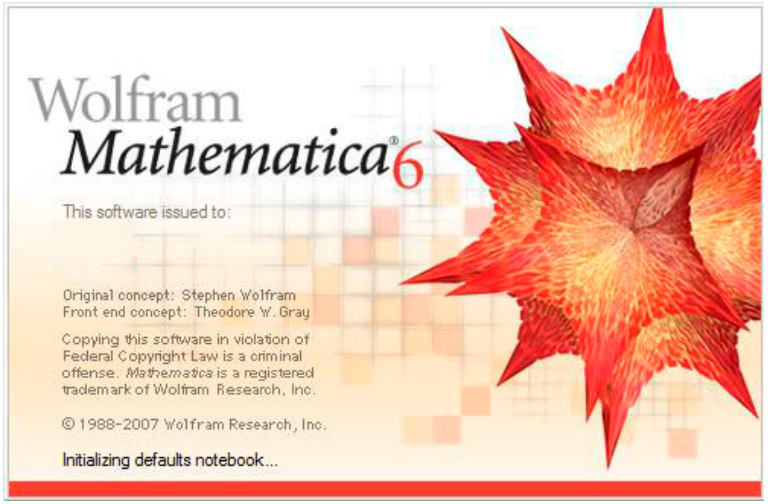
```
BeginPackage["satellite`"]
Begin["`Private`"]


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

originalMeasurement[t_Int] := sensorM[ t ]


End[ ]
EndPackage[ ]
```

Analysis

SOLUTION







Eclipse Modeling
Eclipse Xtext
Wolfram Mathematica
Mathematica Workbench

TOOLS

Example 4:

Alarm System Menus

Menu Structure

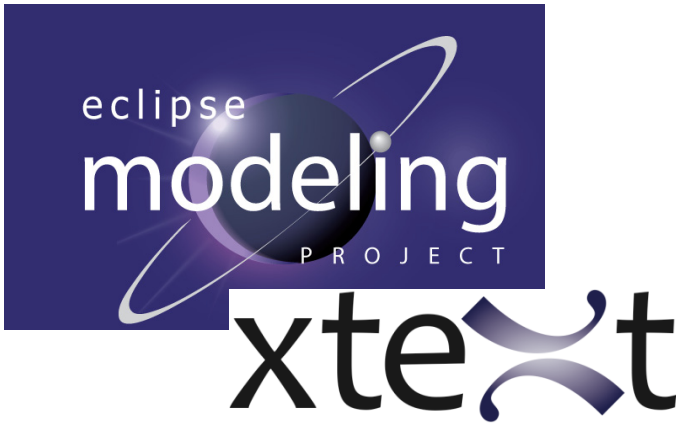
SOLUTION

```
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
```

Software Components

SOLUTION

```
message TurnOffAlarm
message TurnOnAlarm
message AlarmLevel
message UnlockNow

component AlarmManager {
    receives TurnOffAlarm
    receives TurnOnAlarm
    receives AlarmLevel
}

component MasterSwitch {
    receives UnlockNow
}
```



Eclipse Modeling
Eclipse Xtext

TOOLS

Example 5:

Requirements Tracability

Imported Requirements

SOLUTION

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

Program Code with Annotations (green)

SOLUTION

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>;
}
```

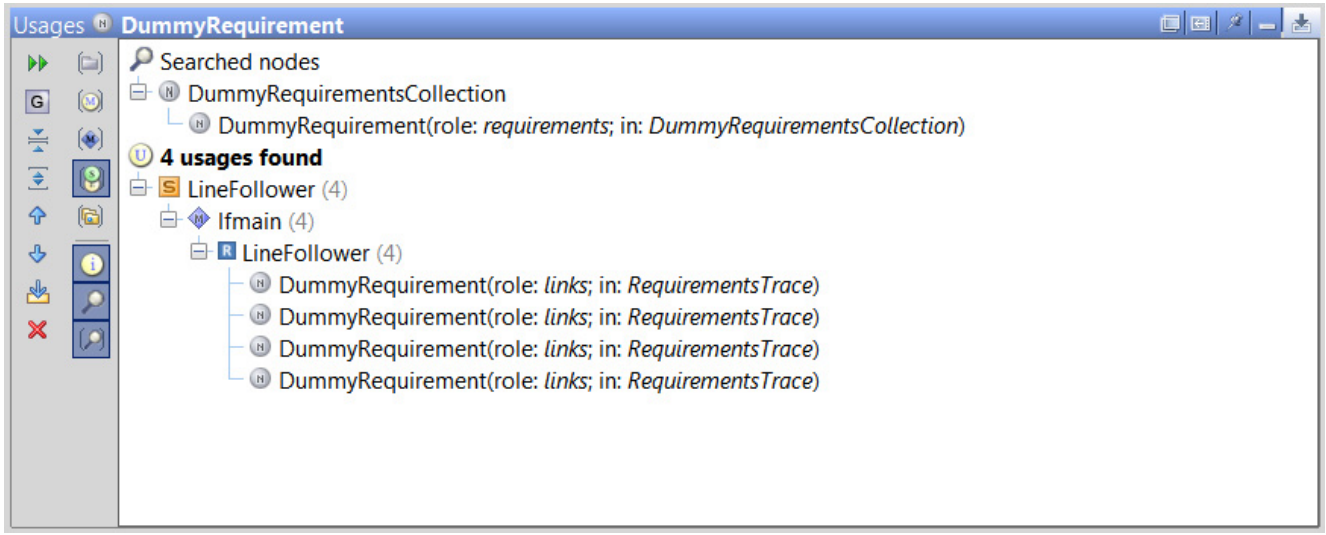
SOLUTION

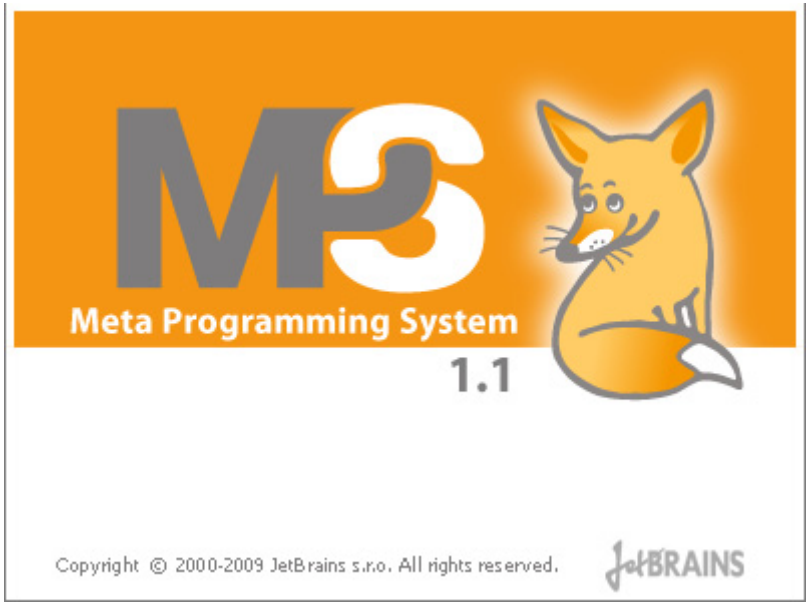
Selecting from the Requirements

```
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
  states
    stat
      in
      bu
      if
      }
      trace Init
      int32 light = 0;
```

```
Calibration: The black/white values should                ^requirements (lfmain.null)
ConsistentSetting: Motor settings have to be upda         ^requirements (lfmain.null)
Cyclic: the actual control of the devi                    ^requirements (lfmain.null)
Efficient: The program should be as small                 ^requirements (lfmain.null)
Init: The system should start operat                      ^requirements (lfmain.null)
MaxSpeed: Speeds per motor can only be u                  ^requirements (lfmain.null)
OptionalOutputlkjlkjljlkj: Display output should be optio ^requirements (lfmain.null)
TwoPhases: Initialization should be separ                 ^requirements (lfmain.null)
```

Find Usages of Requirements





JetBrains MPS

TOOLS

Example 5:
OSGi-based System

Component Specification

SOLUTION

```
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]
        }

}
```

Component Specification

SOLUTION

```
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.compdsl"

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

Component Specification

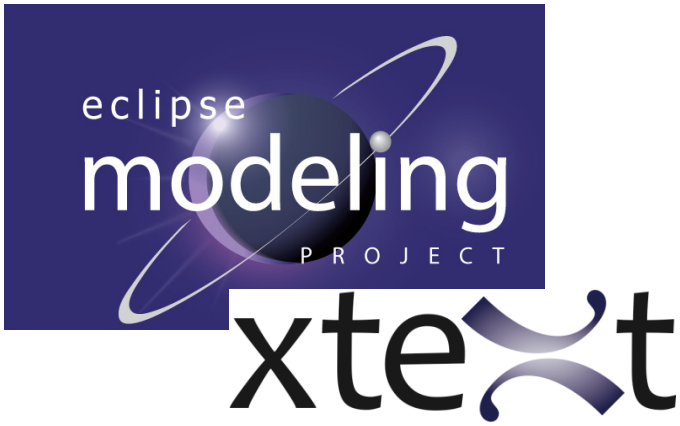
SOLUTION

```
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.compdsl"

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

```
import "classpath:/twoworlds.compdsl"
import "classpath:/twoworlds.scenario"

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



Eclipse Modeling
Eclipse Xtext

TOOLS

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
```

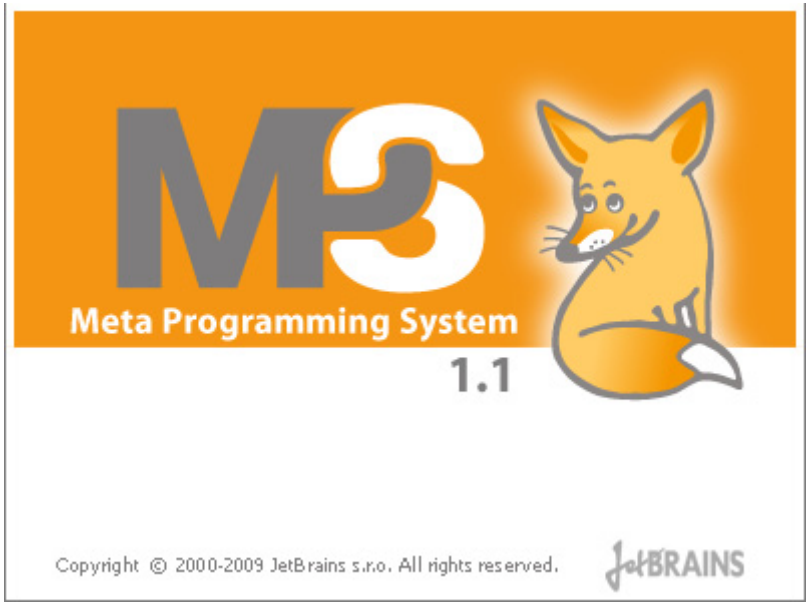
SOLUTION

```
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>>
}
```



JetBrains MPS

TOOLS

Example 7:

Fountains

CONTEXT

Hardware Structure

SOLUTION

```
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
```

Behaviour

SOLUTION

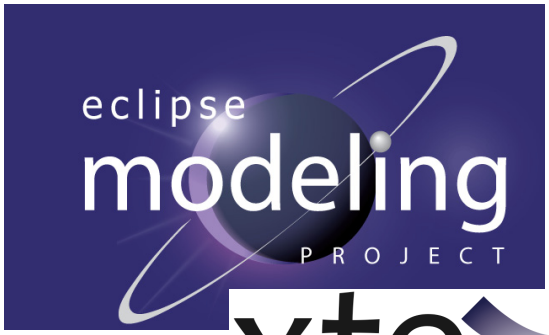
```
pumping program P1 for AtLeastOneZone + WithAlarm +
                       SuperPowerCompartment[f=comp1]  {
    parameter defaultWaterLevel : int
    parameter superWaterLevel: int
    event superPowerTimeout

    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



Eclipse Modeling
Eclipse Xtext

TOOLS

More Examples8:
Miscellaneous

Hearing Aids

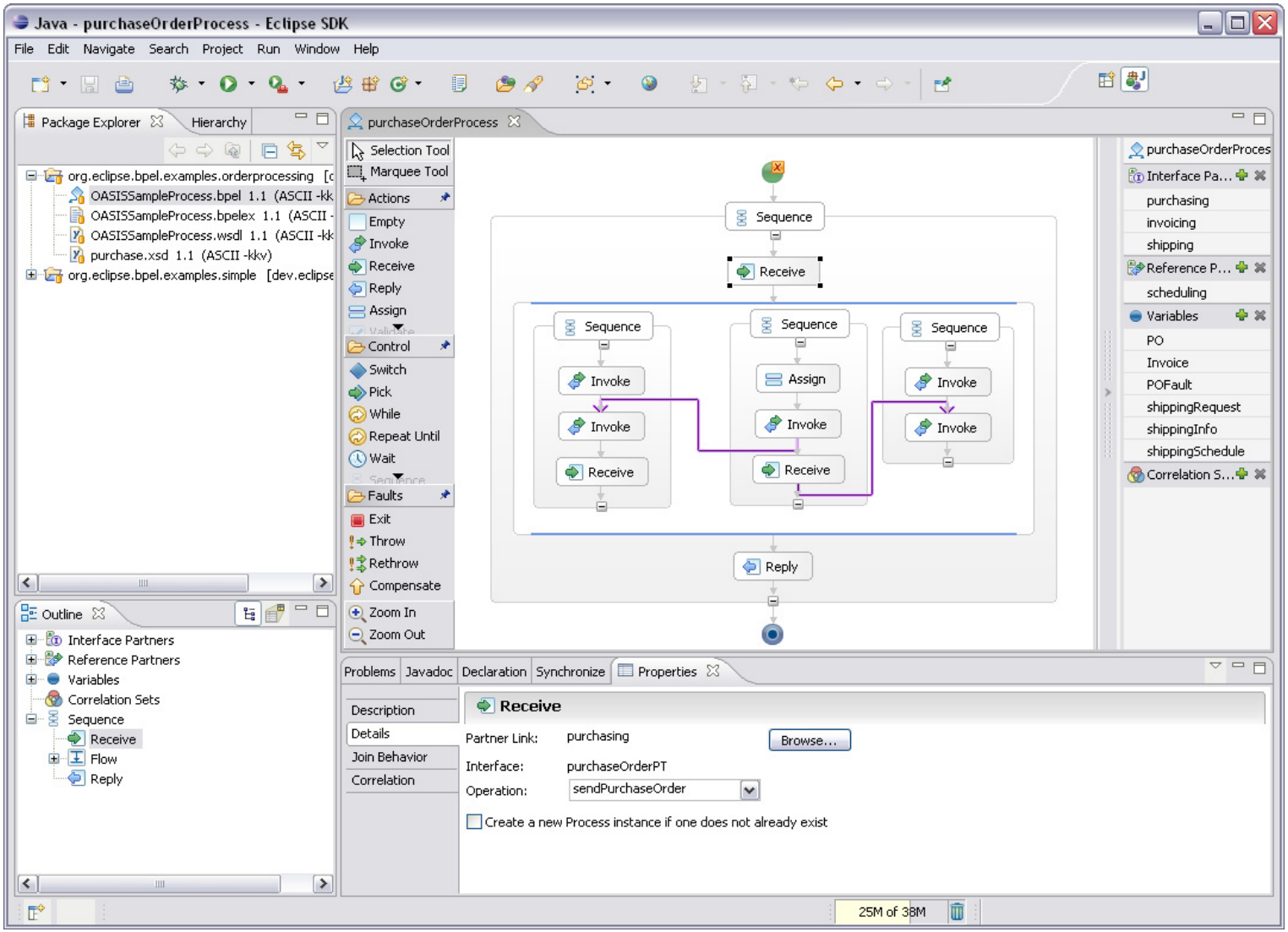
DOMAIN

Refrigerators

DOMAIN

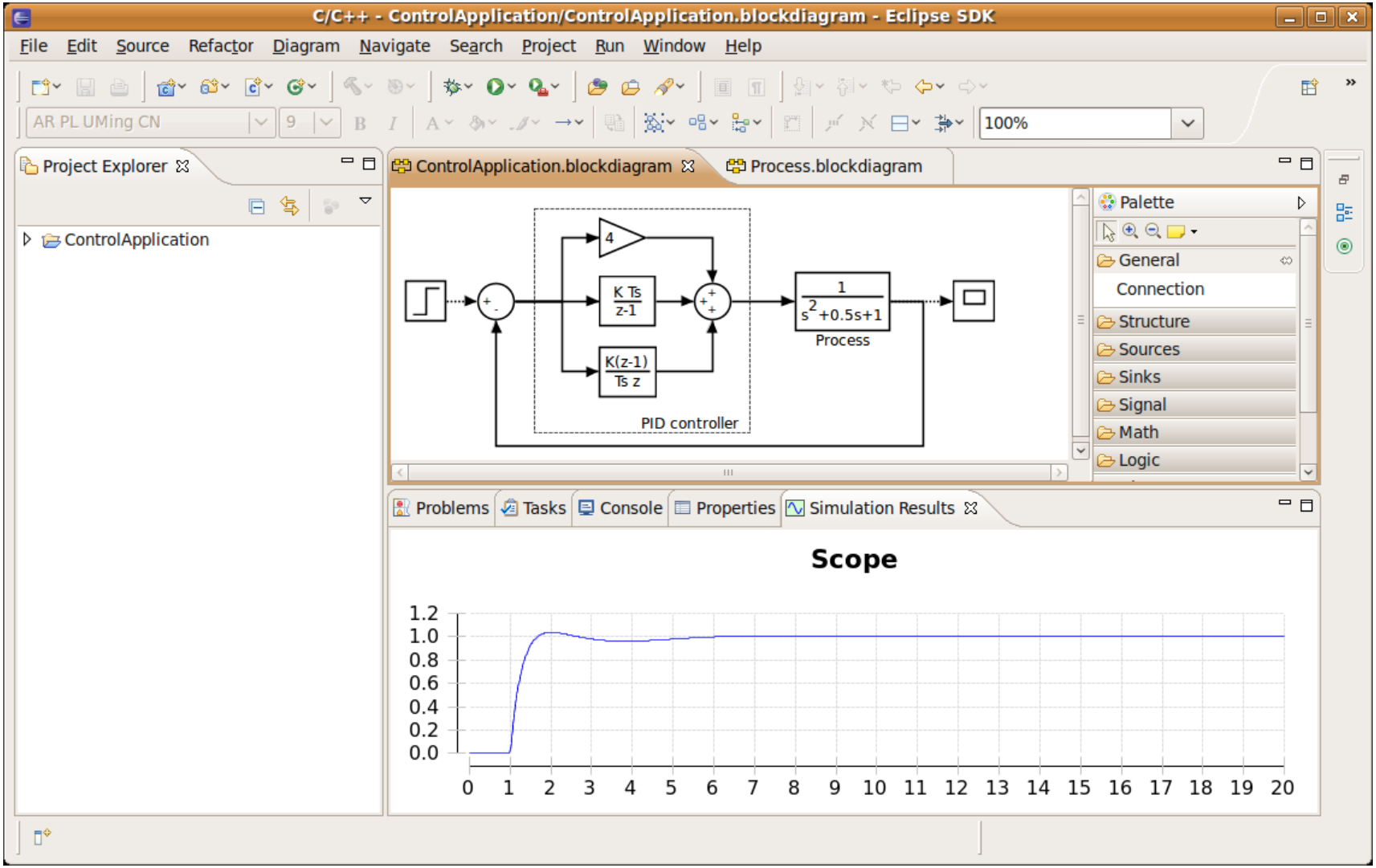
BPEL Designer

SOLUTION



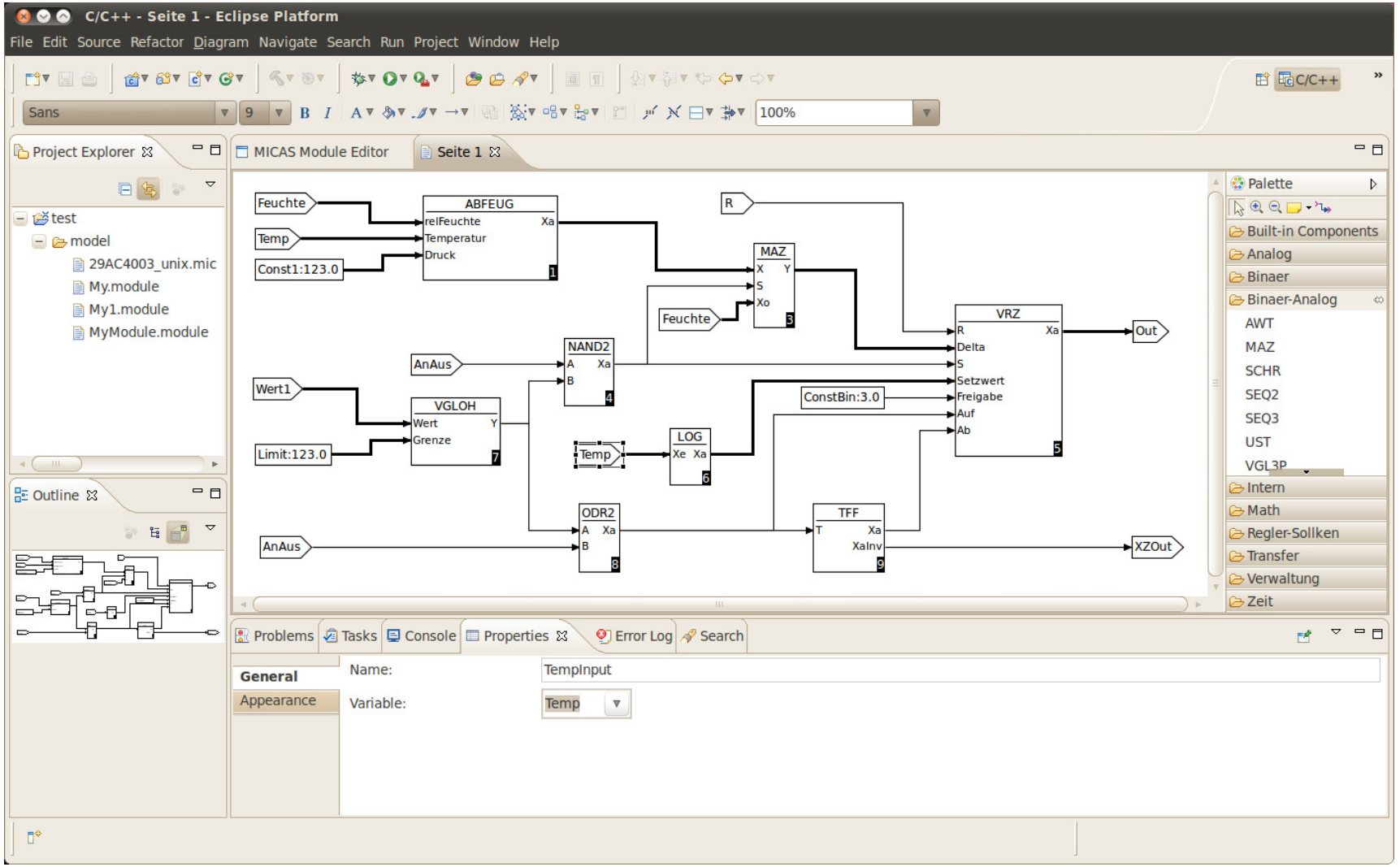
Block Diagrams

SOLUTION



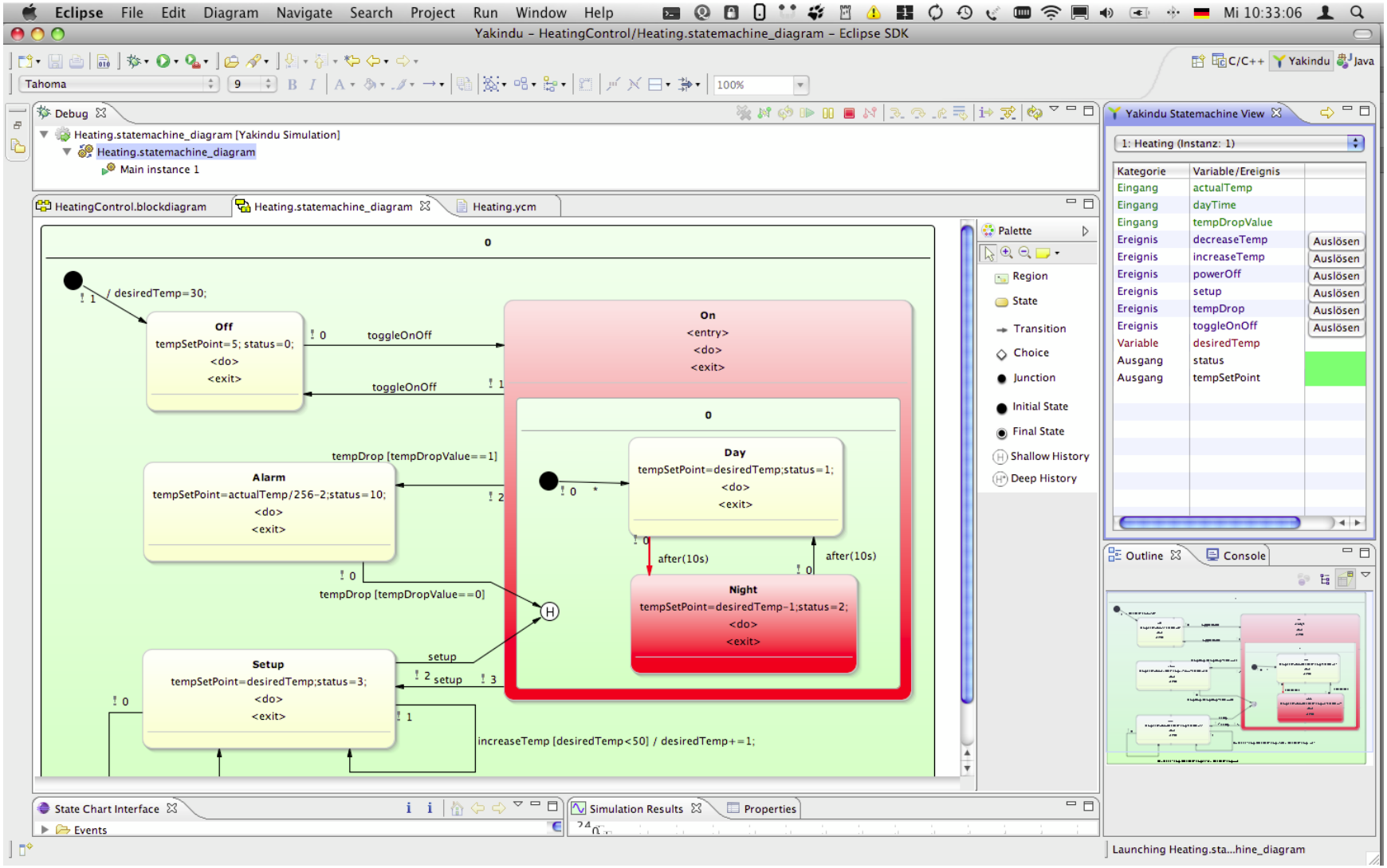
PLC Programming

SOLUTION

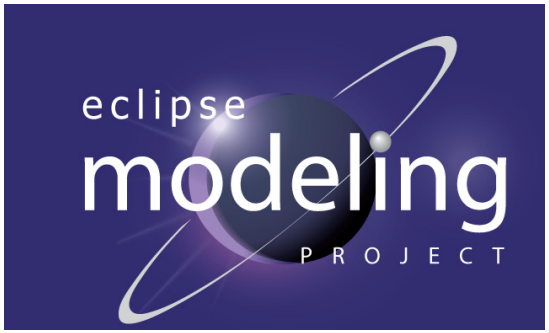


State Charts

SOLUTION



TOOLS



4

Tools

Tooling!

Tooling!

Editor

Tooling!

Editor, Debugger

Tooling!

Editor, Debugger, Testing

Tooling!

Editor, Debugger, Testing, Groupware

Tooling!

Editor, Debugger, Testing, Groupware, Scalable

Tooling!

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

Tools Tooling

Language Definition Tools

abstract syntax, concrete syntax, constraints

Editor Frameworks

Transformation Languages

Code Generation Tools


xtext


MPS
Meta Programming System


INTENTIONAL®
SOFTWARE

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

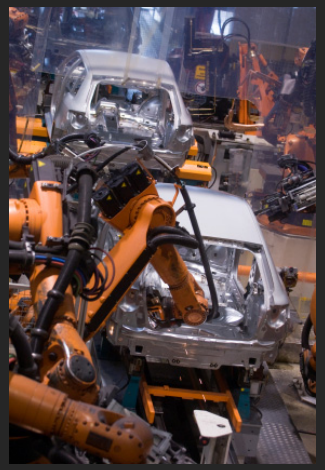
The Whole Platform

see also

http://languageworkbenches.net

5

Benefits



Automation

faster, deterministic

Increased Quality

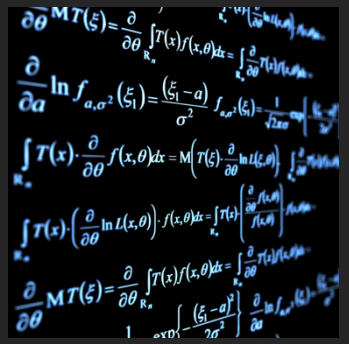
well defined structures allthrough the system

Meaningful Validation

more semantics in the model

Capture Domain Knowledge

formalized into languages and models

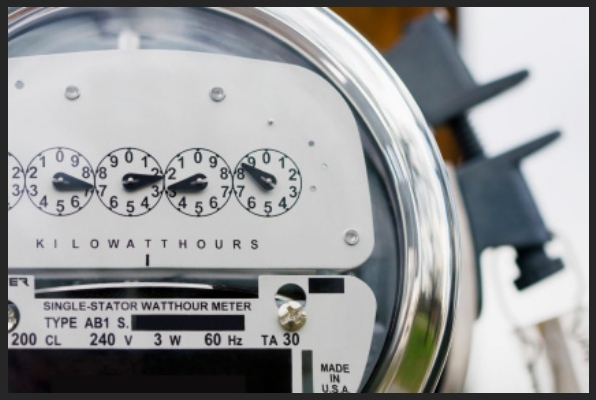


Suitable Notations

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

6

A more theoretical View

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.

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!

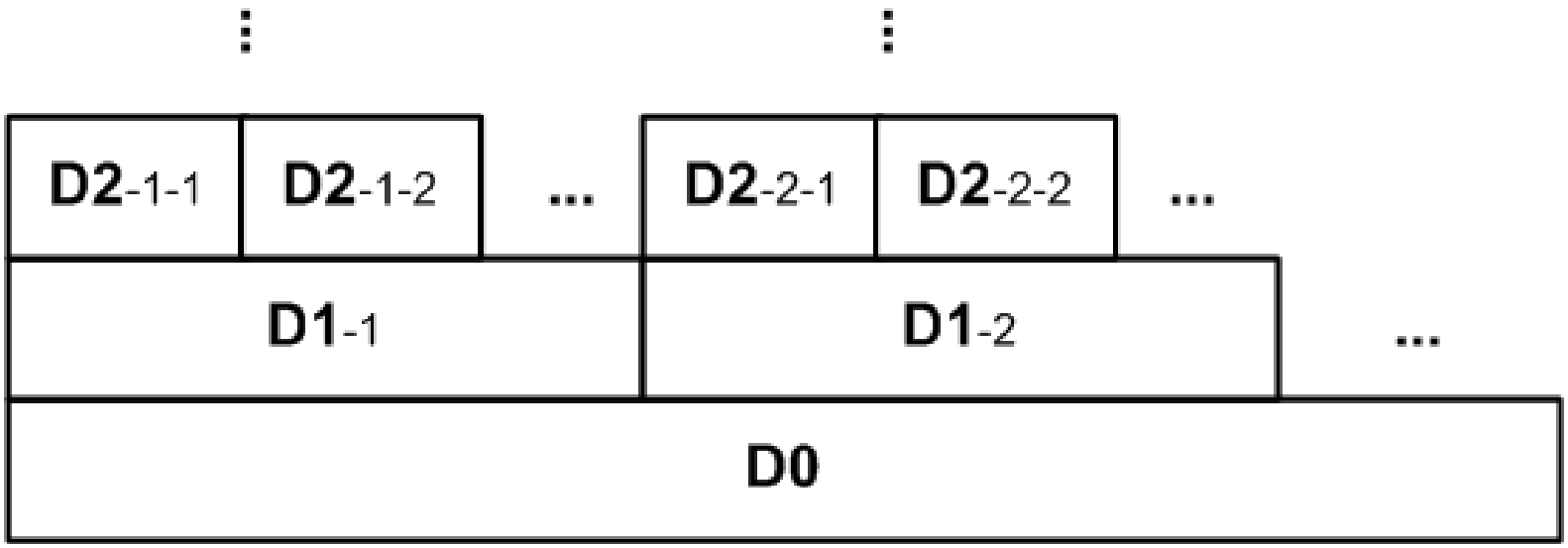
```
statemachine linefollower {
  event initialized;
  event bumped;
  event blocked;
  event unblocked;
  initial state initializing {
    initialized [true] -> running
  }
  state paused {
    entry int16 i = 1;
    unblocked [true] -> running
  }
  state running {
    blocked [true] -> paused
   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

Linguistic Abstraction

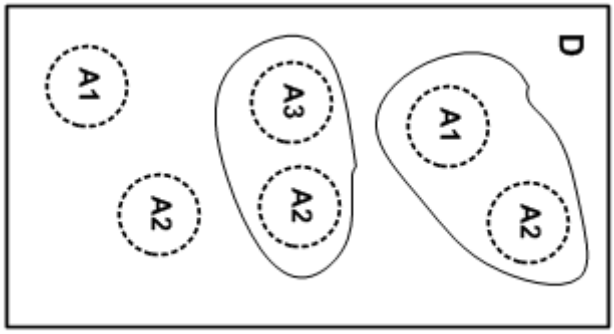
Much better!

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

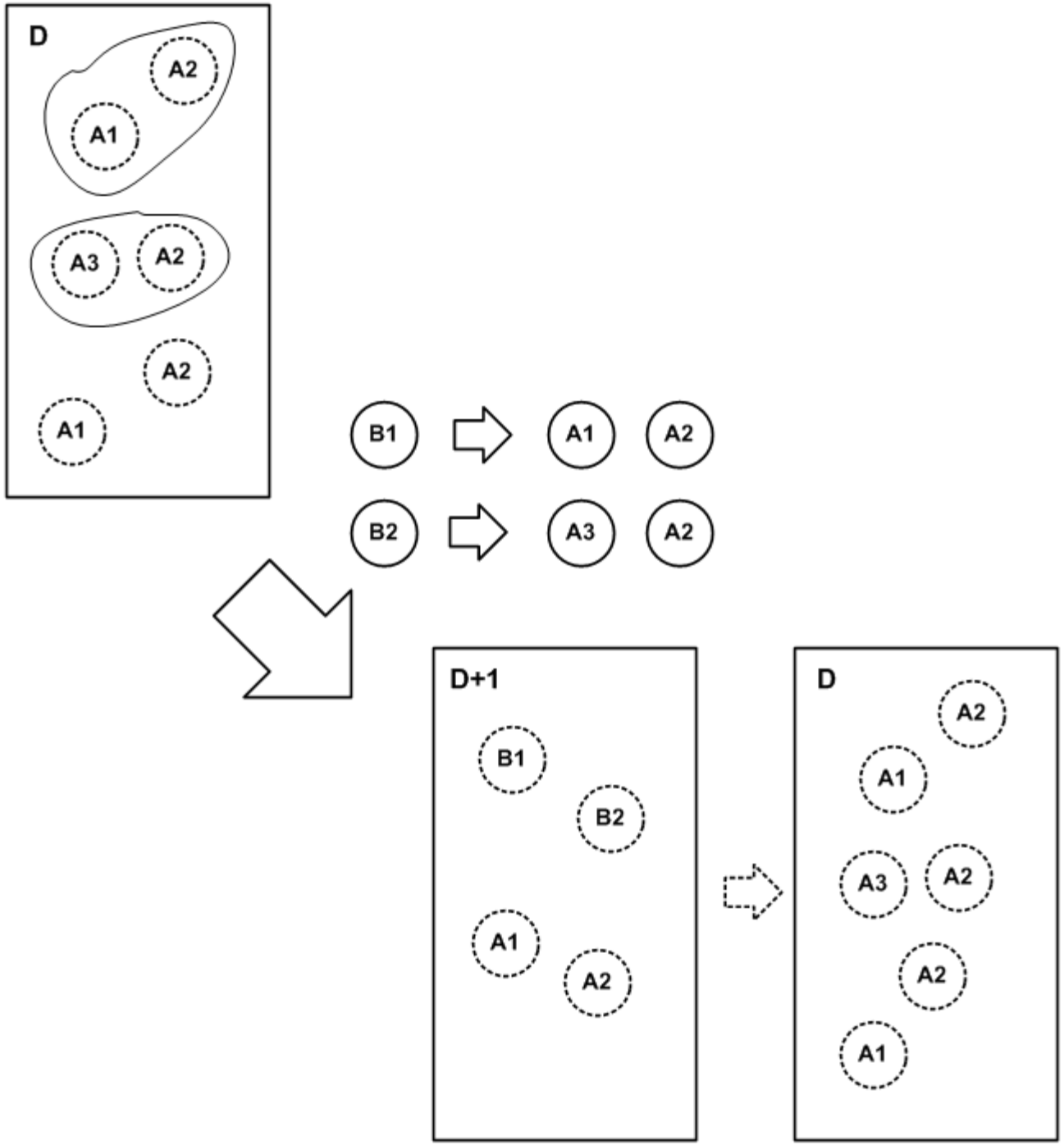
Domains are Hierarchical



Creating Linguistic Abstractions



Creating Linguistic Abstractions



Domain Specific Languages

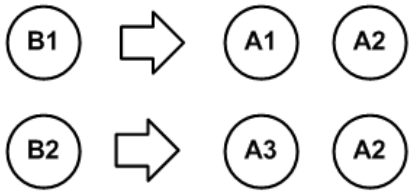
A DSL is a **language** at D that provides **linguistic abstractions** for **common patterns and idioms** of a language at D-1 when used within the domain D.

Domain Specific Languages

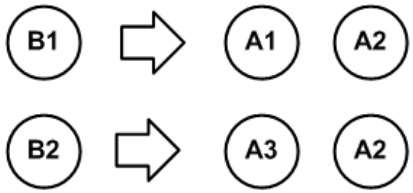
A DSL is a **language** at D that provides **linguistic abstractions** for **common patterns and idioms** of a language at D-1 when used within the domain D.

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.

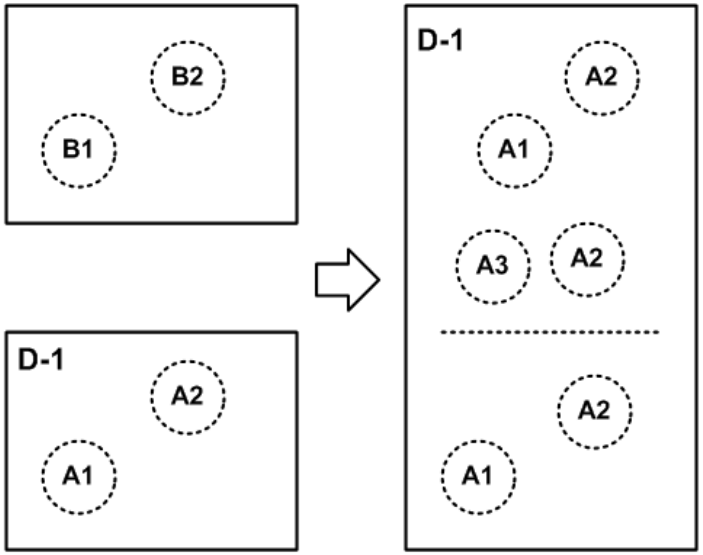
Extension vs. Cascading



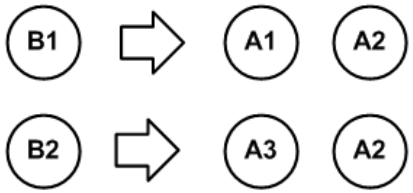
Extension vs. Cascading

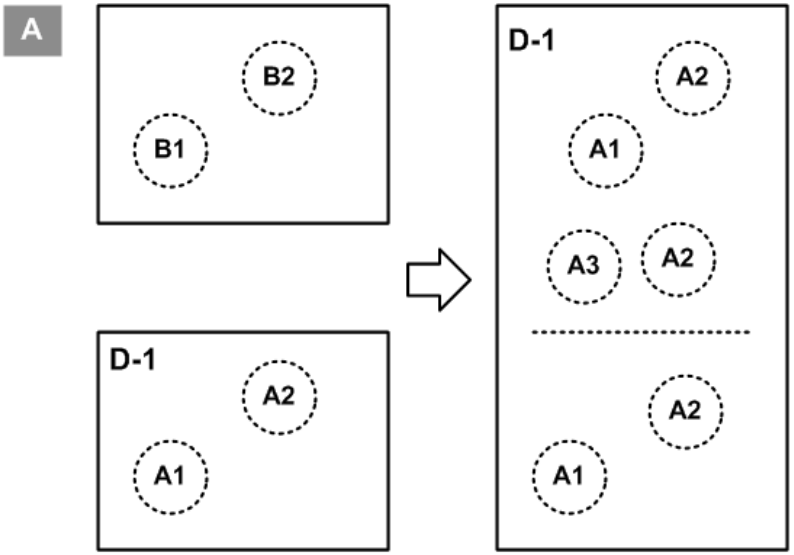


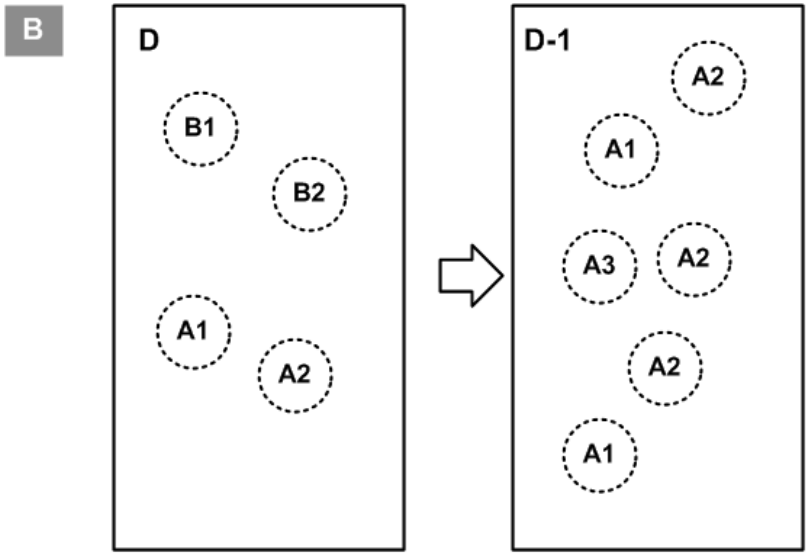
A



Extension vs. Cascading

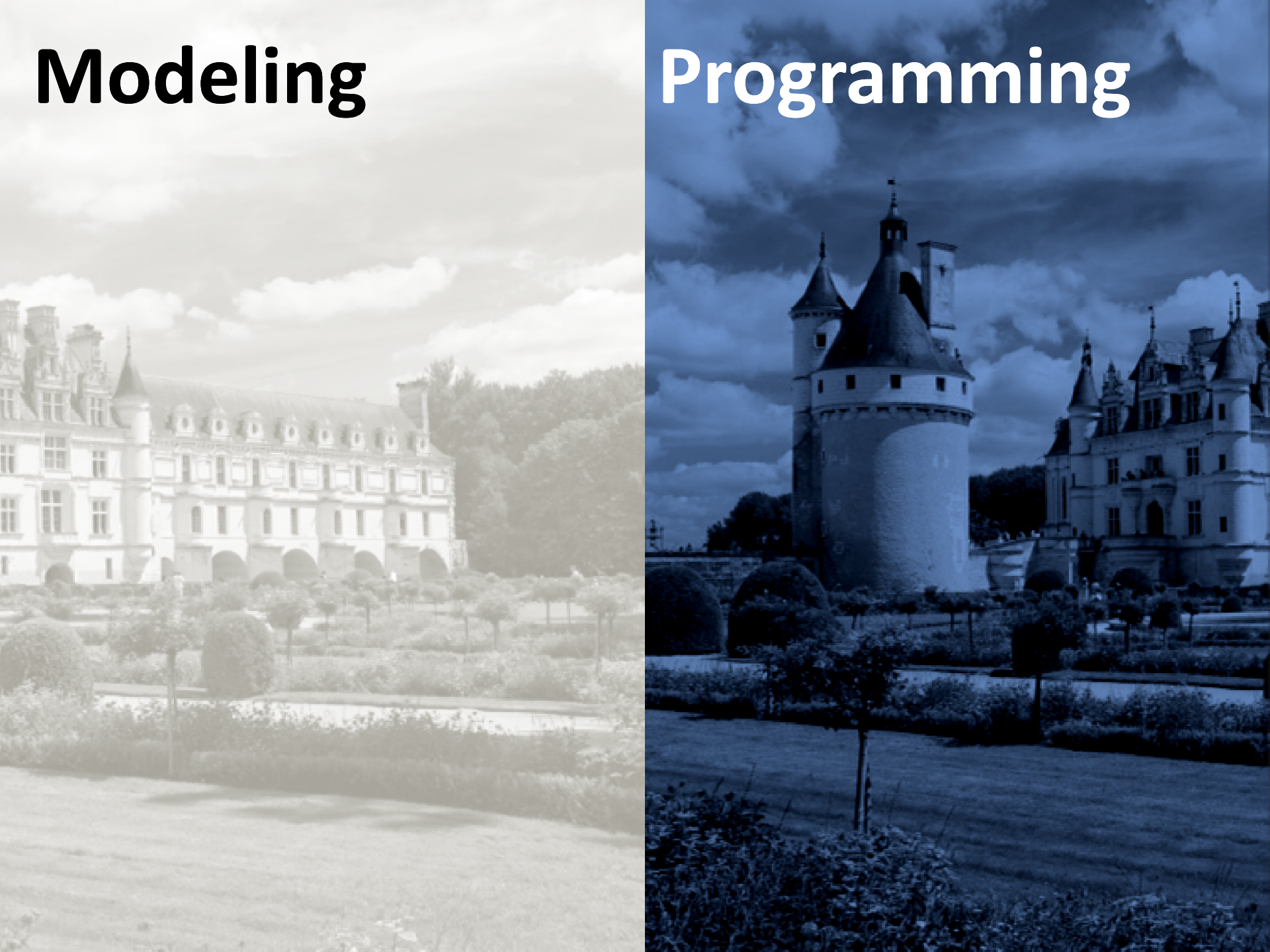


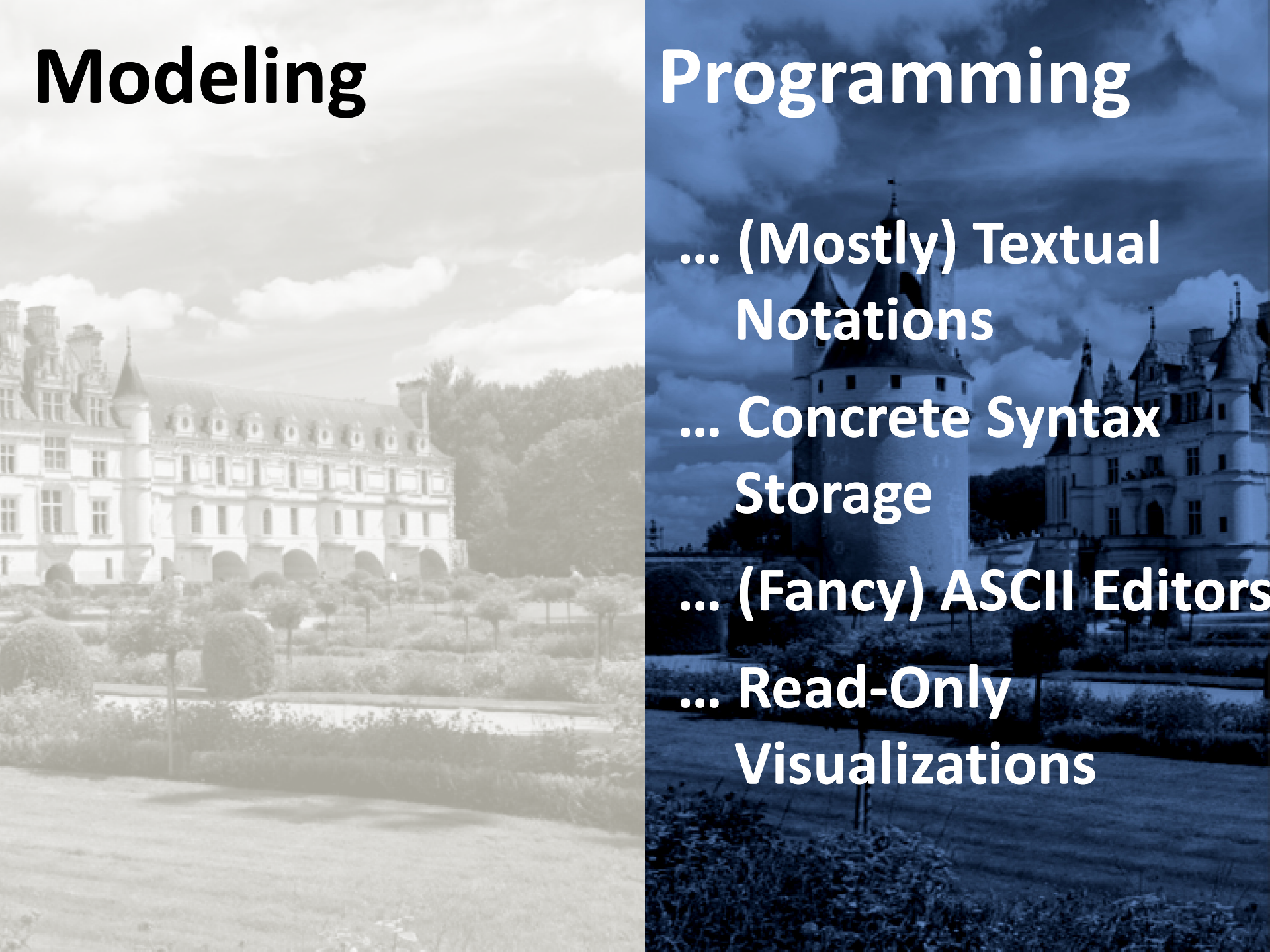




7

Programming and Modeling


Modeling
Programming



Modeling

Programming

… (Mostly) Textual Notations

… Concrete Syntax Storage

… (Fancy) ASCII Editors

… Read-Only Visualizations

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

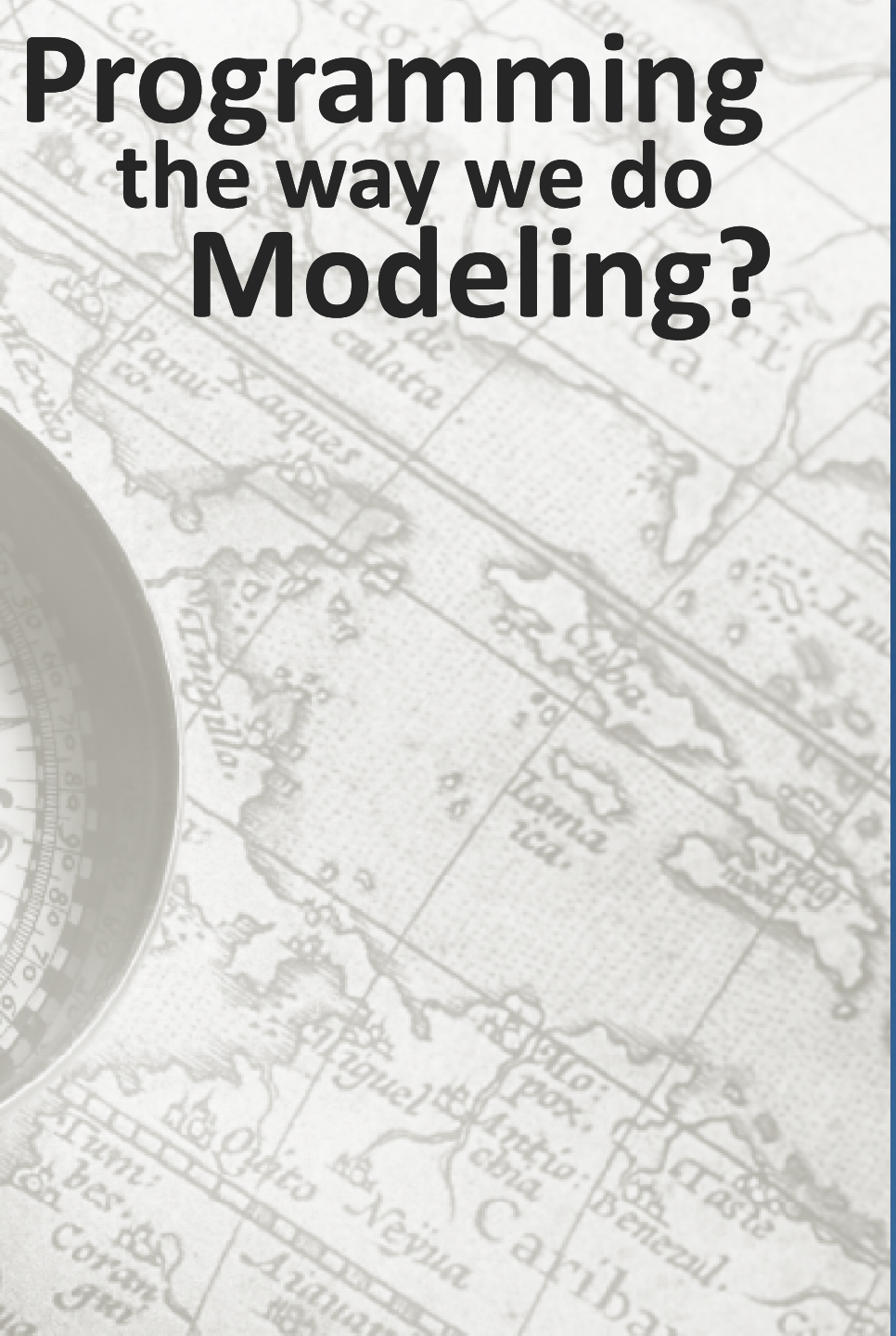
Why

the difference?

It is time for …

… a Different Perspective

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!

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!

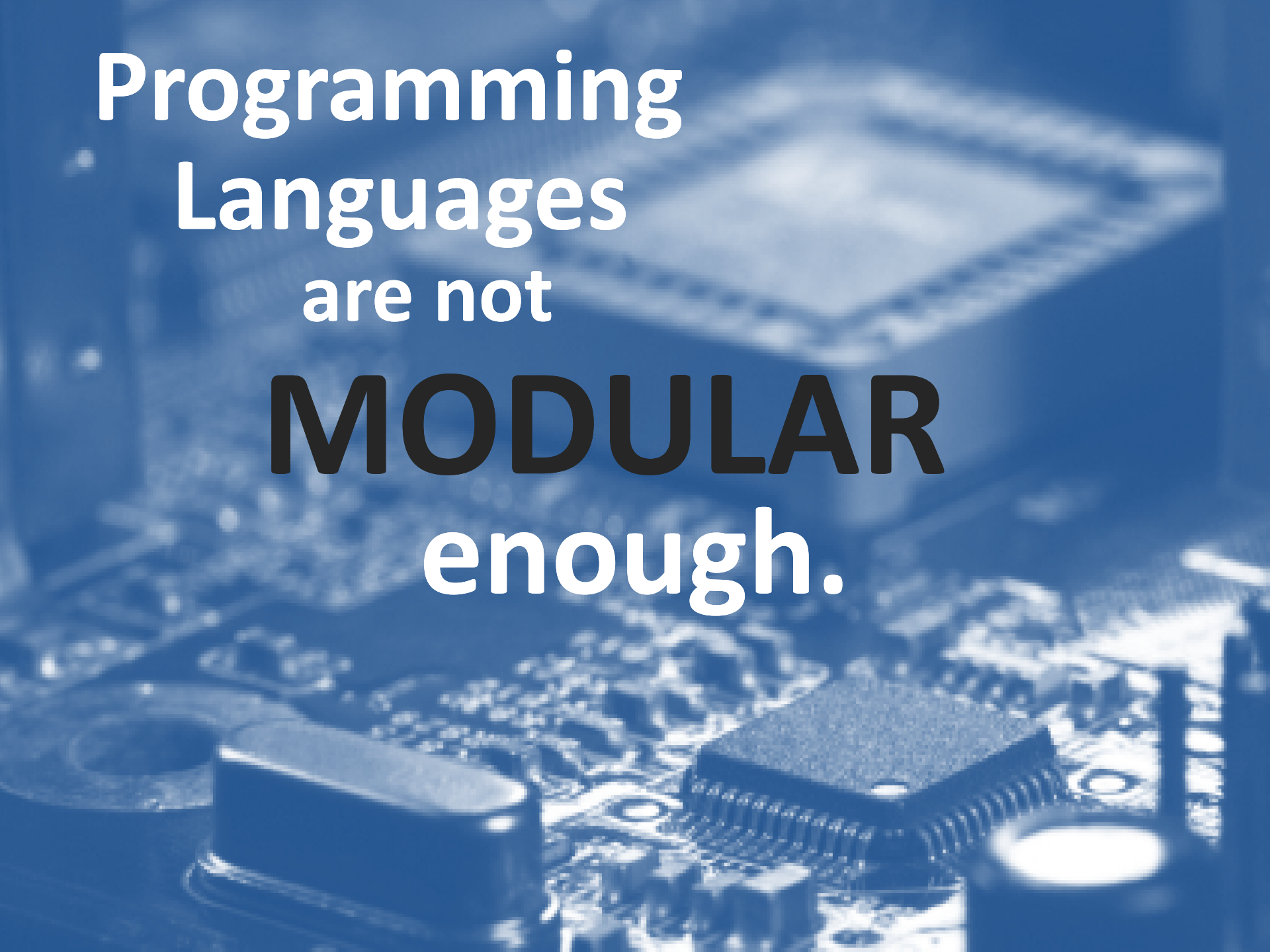
… with different degrees of domain-specificity

… with suitable notations

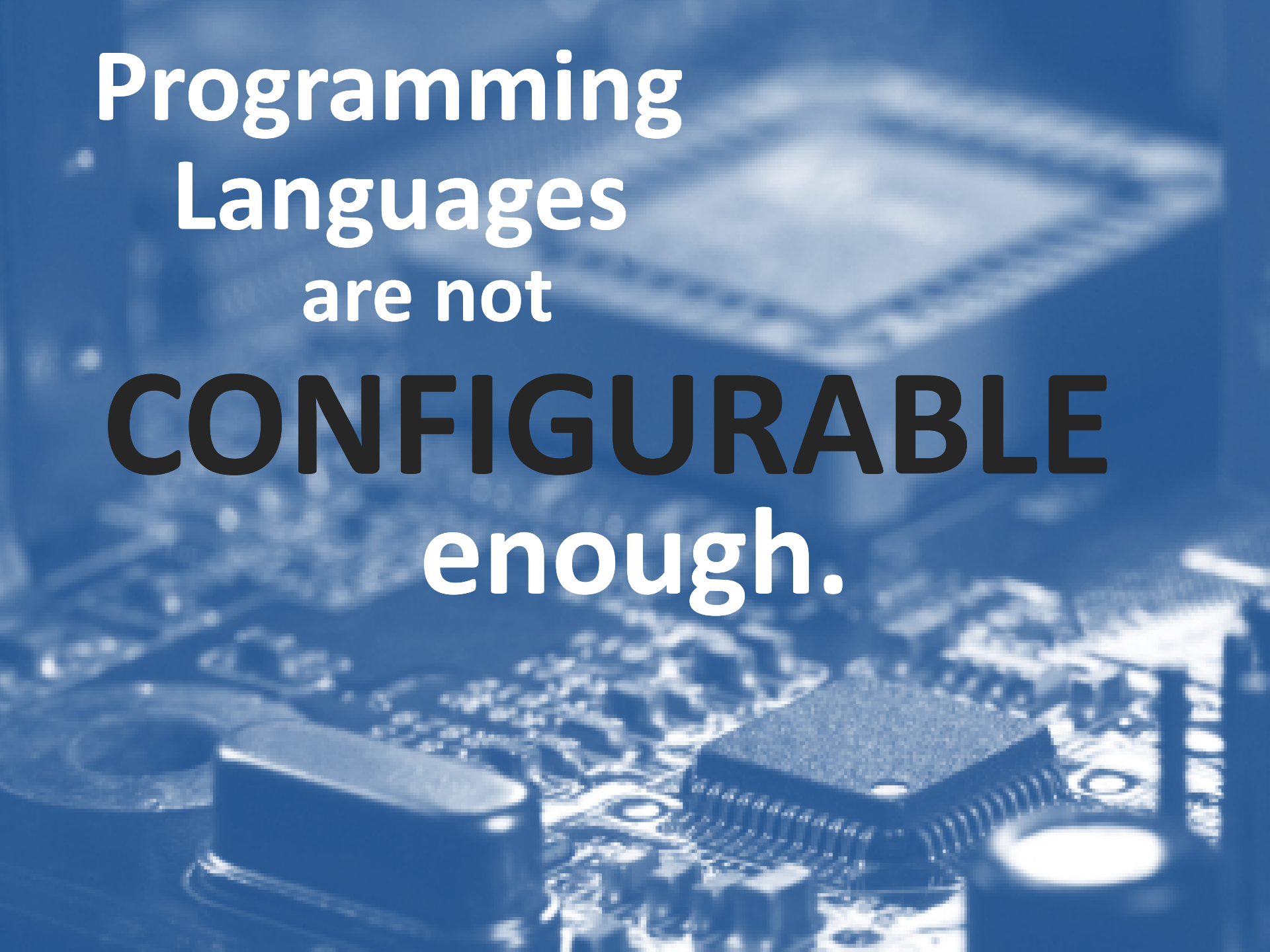
… with suitable expressiveness

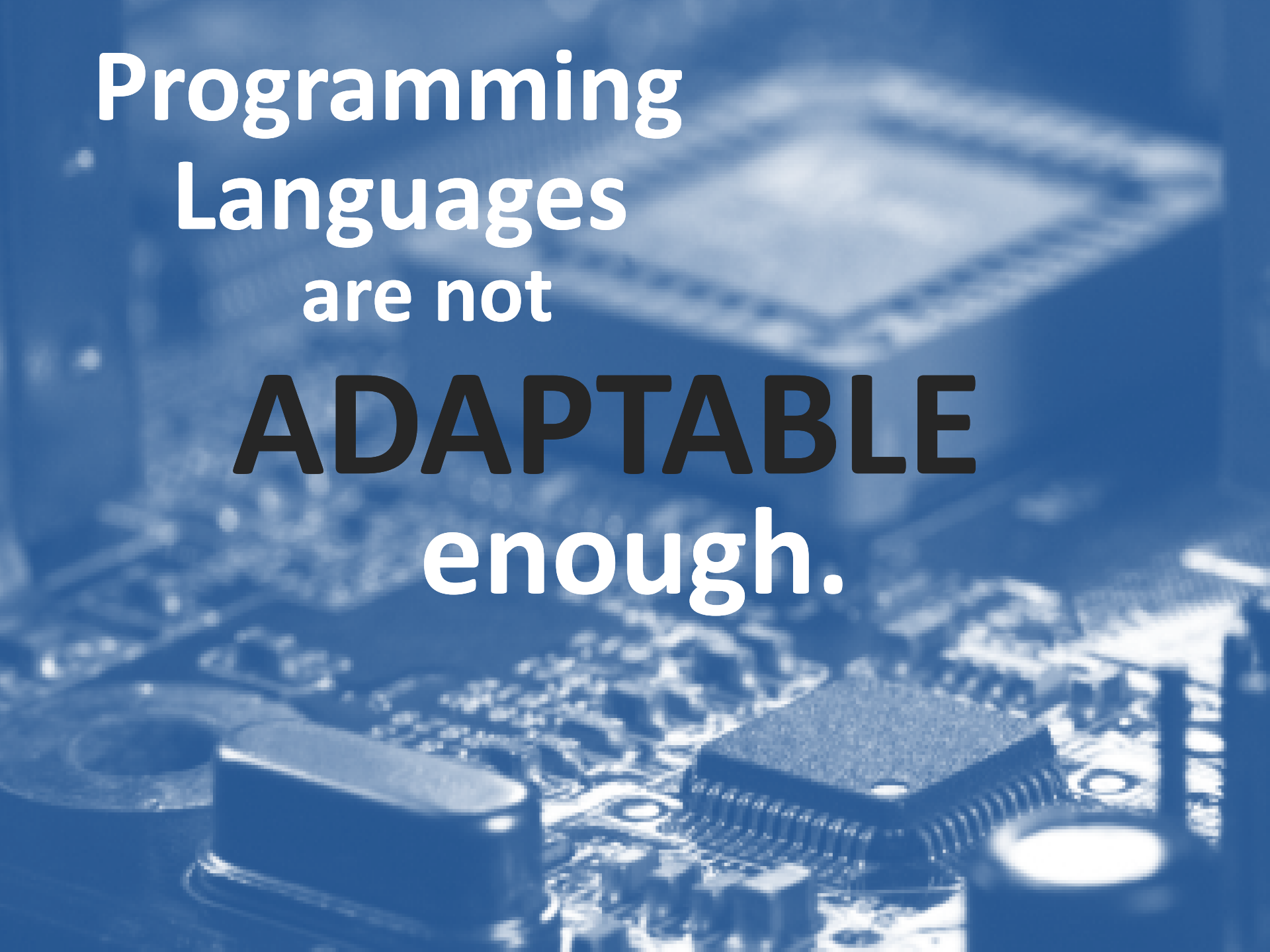
We don‘t want to
model,
we want to
program!

And always:
precise and tool processable


Programming
Languages
are not
MODULAR
enough.

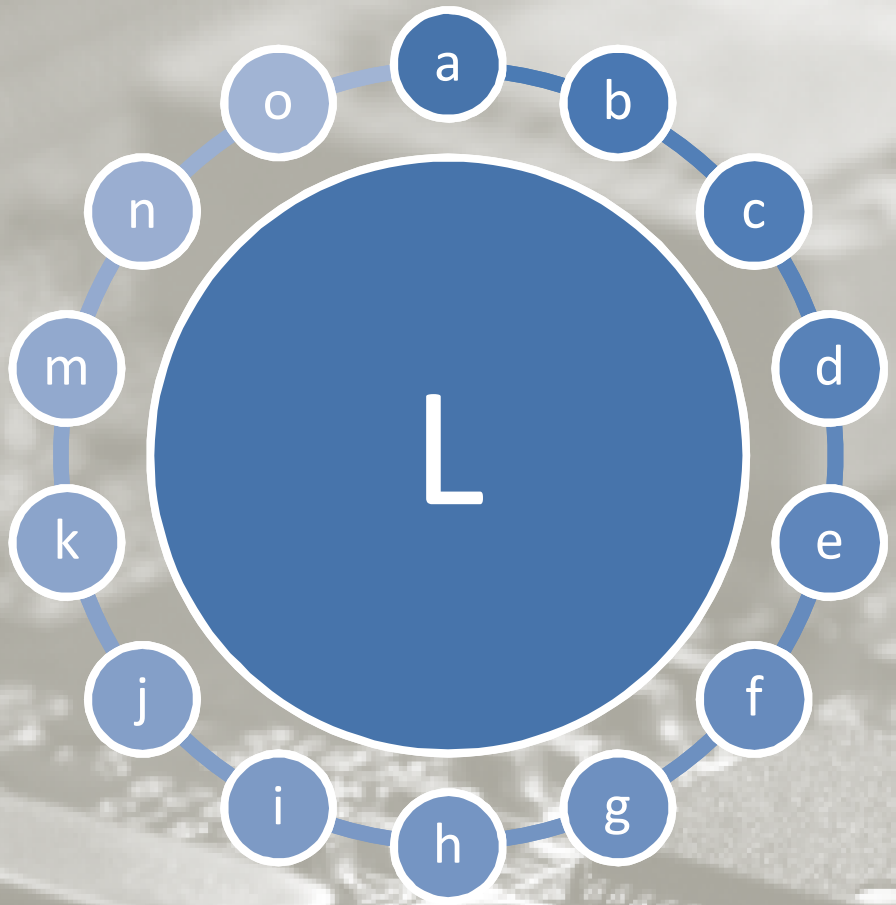

Programming
Languages
are not
COMPOSABLE
enough.


Programming
Languages
are not
CONFIGURABLE
enough.


Programming
Languages
are not
ADAPTABLE
enough.

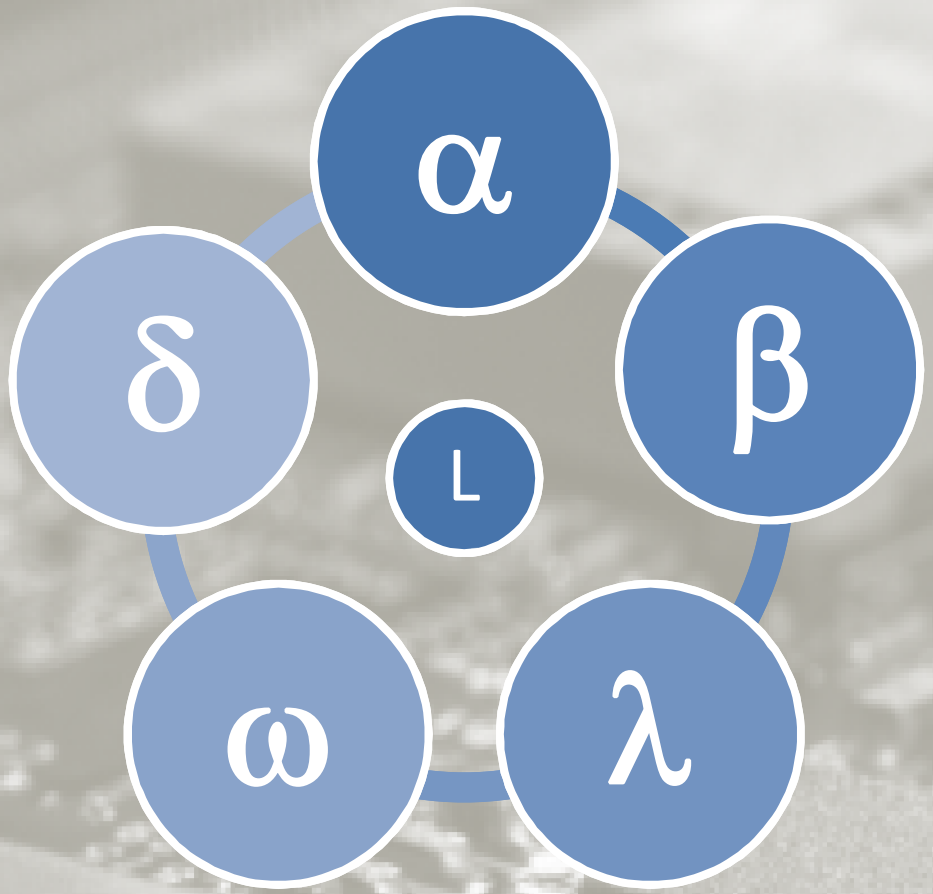

Programming
Language Syntax
is not
FLEXIBLE
enough.

Big Language?



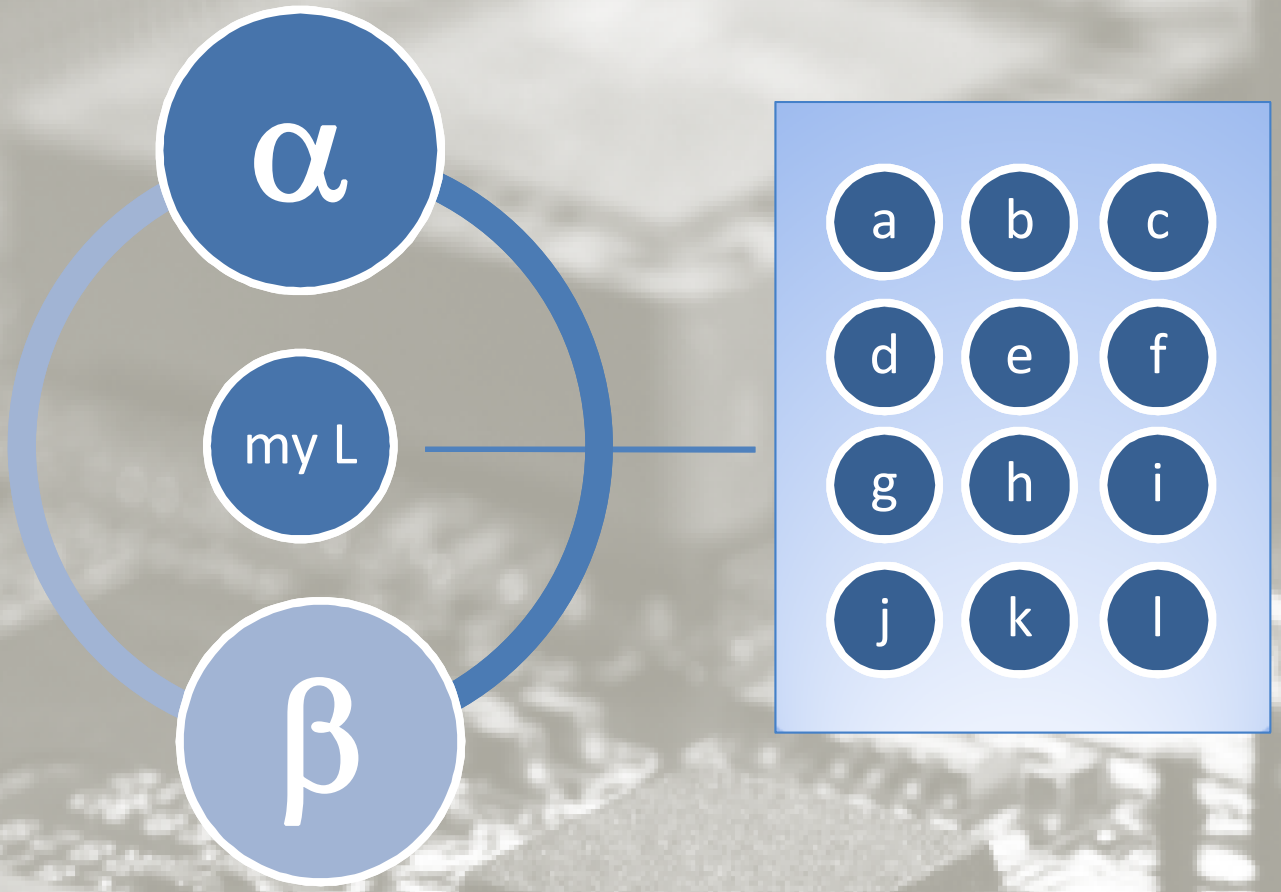
with **many** first class concepts!

Small Language?



with a few, orthogonal
and poweful concepts

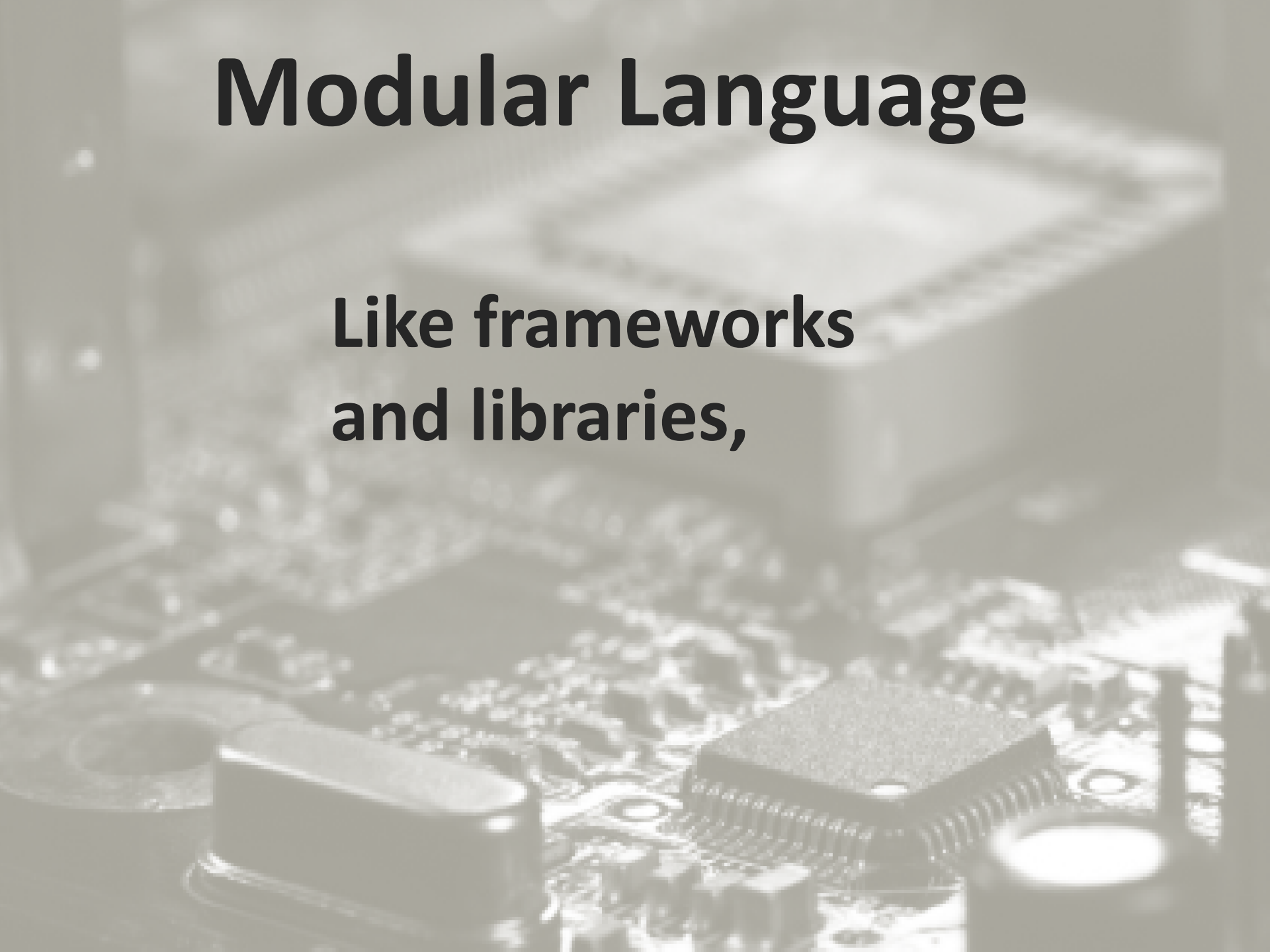
Modular Language



with many optional, composable concepts

Modular Language

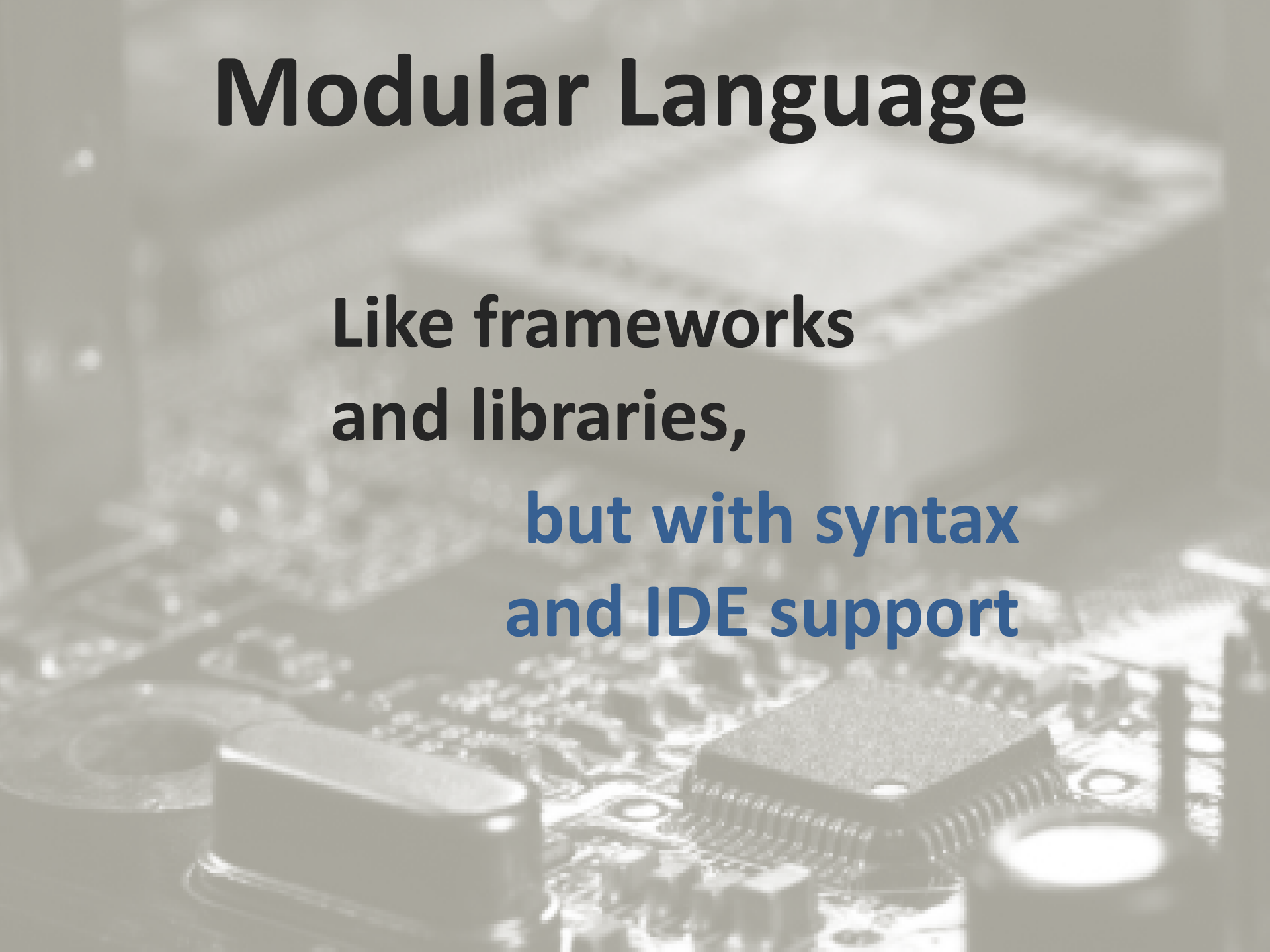
Like frameworks
and libraries,



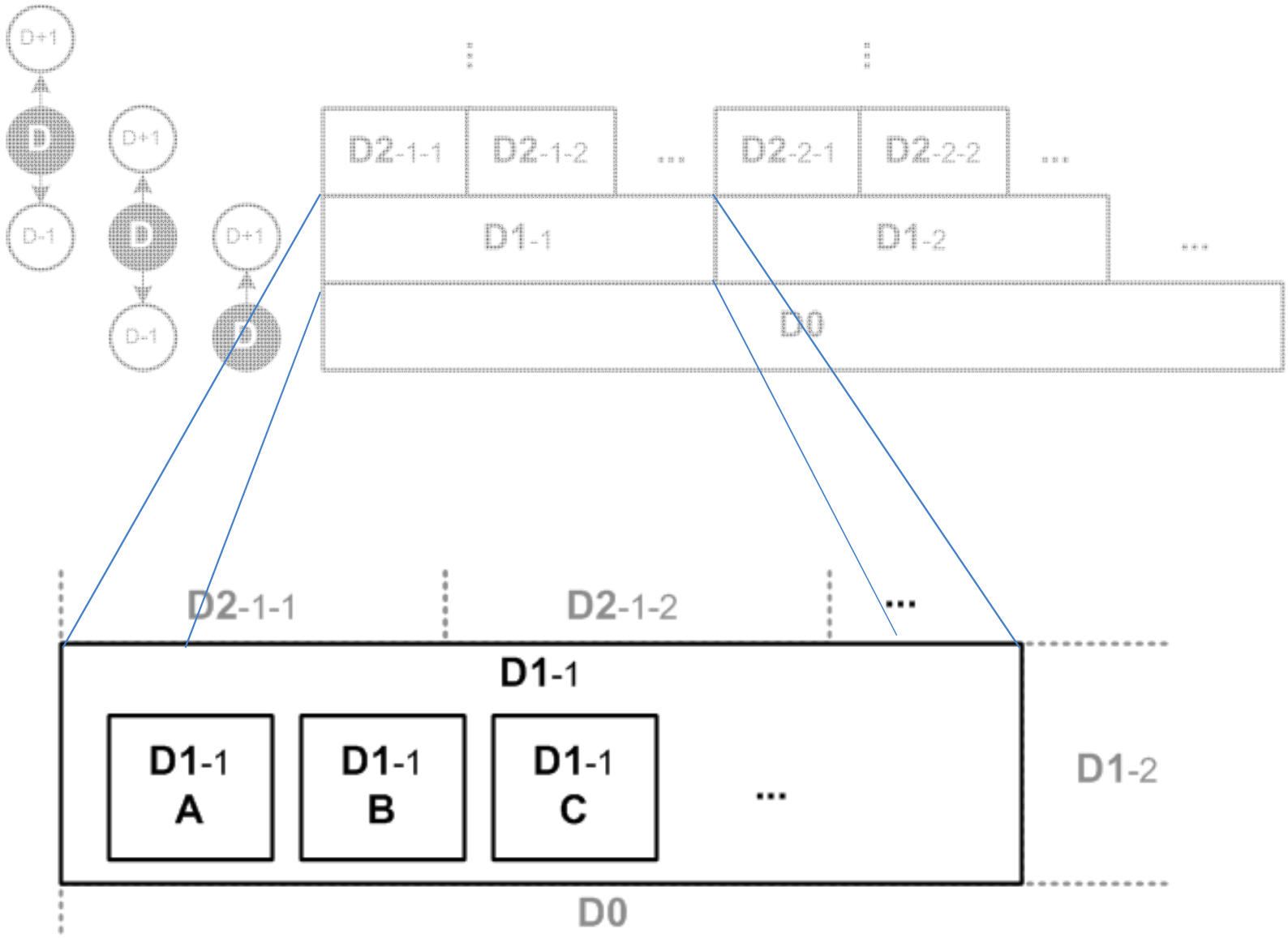
Modular Language

Like frameworks
and libraries,

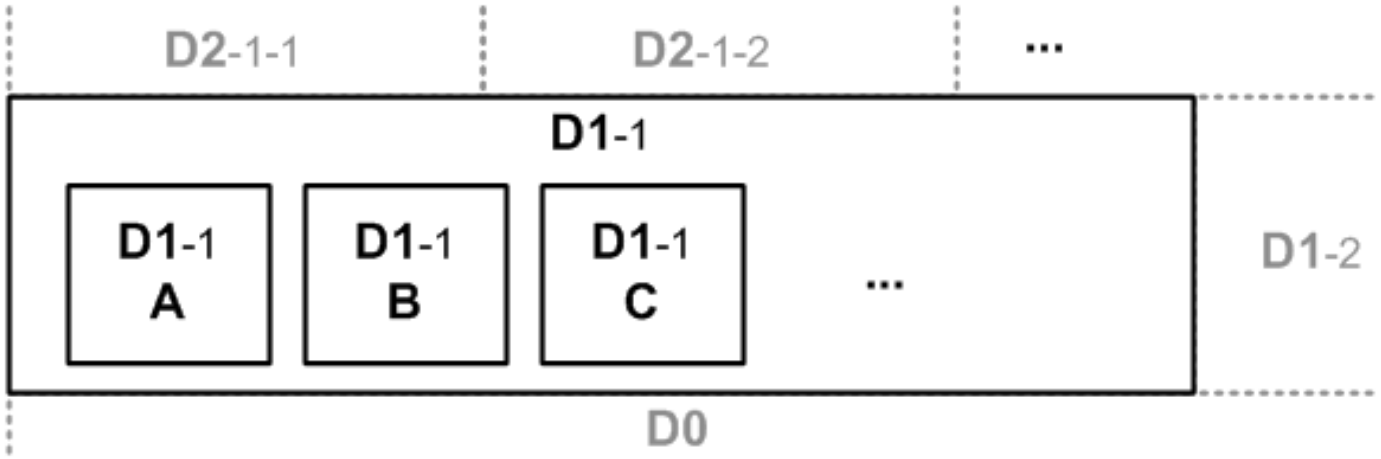
but with syntax
and IDE support



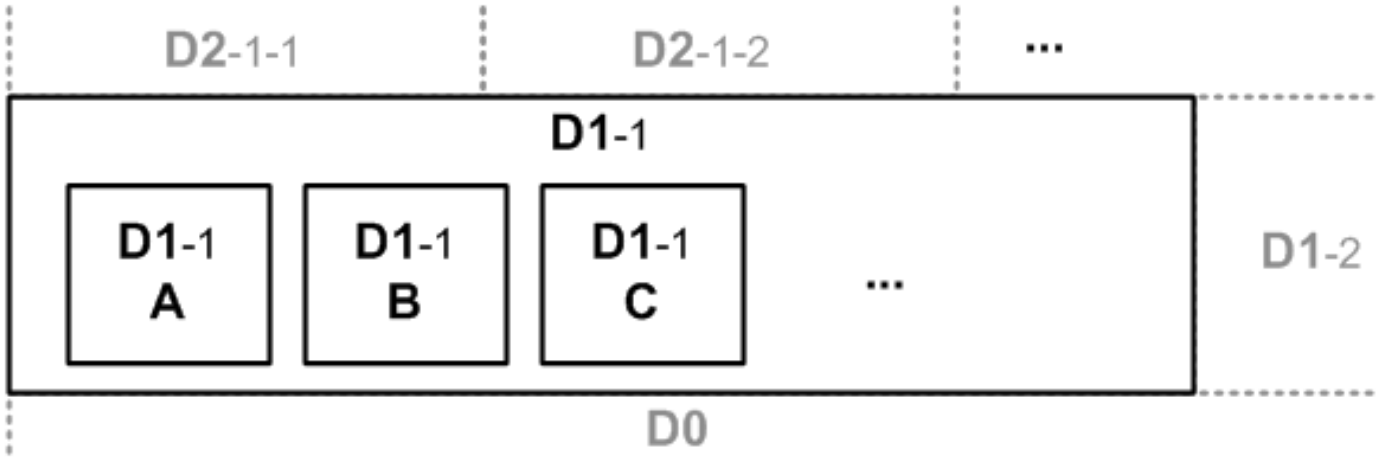
Several Concerns in a Domain

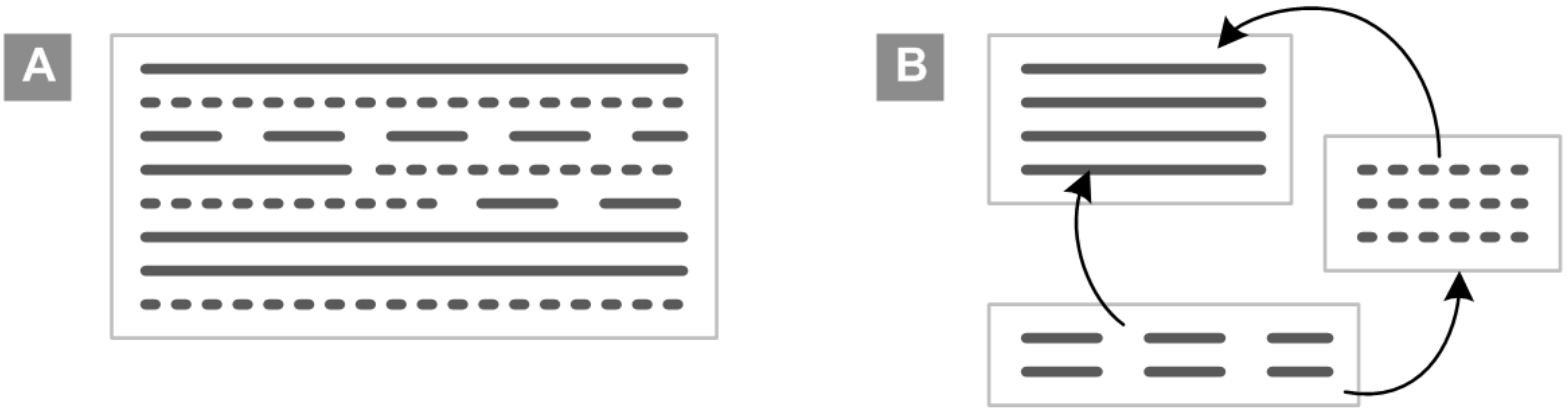


Viewpoints



Viewpoints

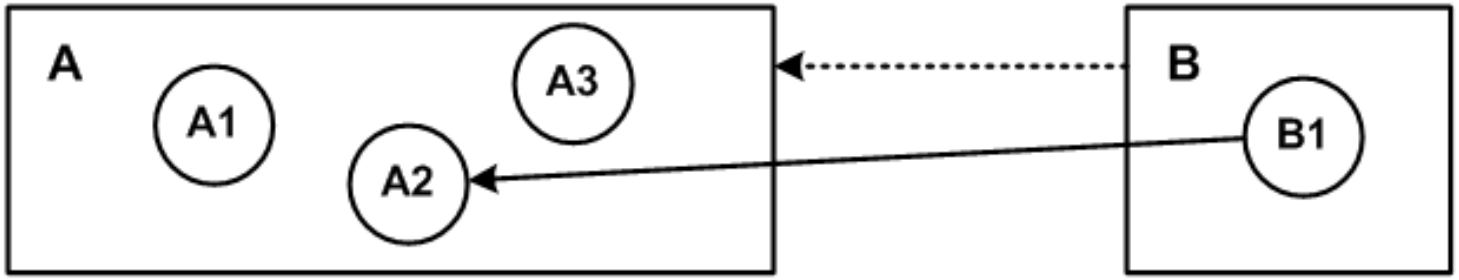




A: mixed

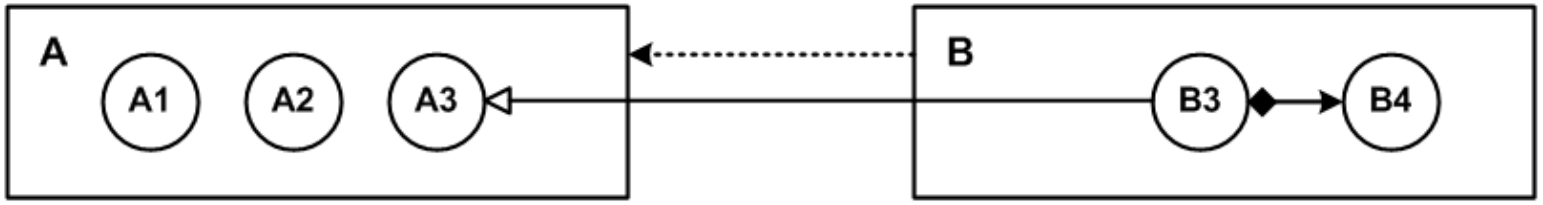
B: separate Viewpoints

Langugae Combination



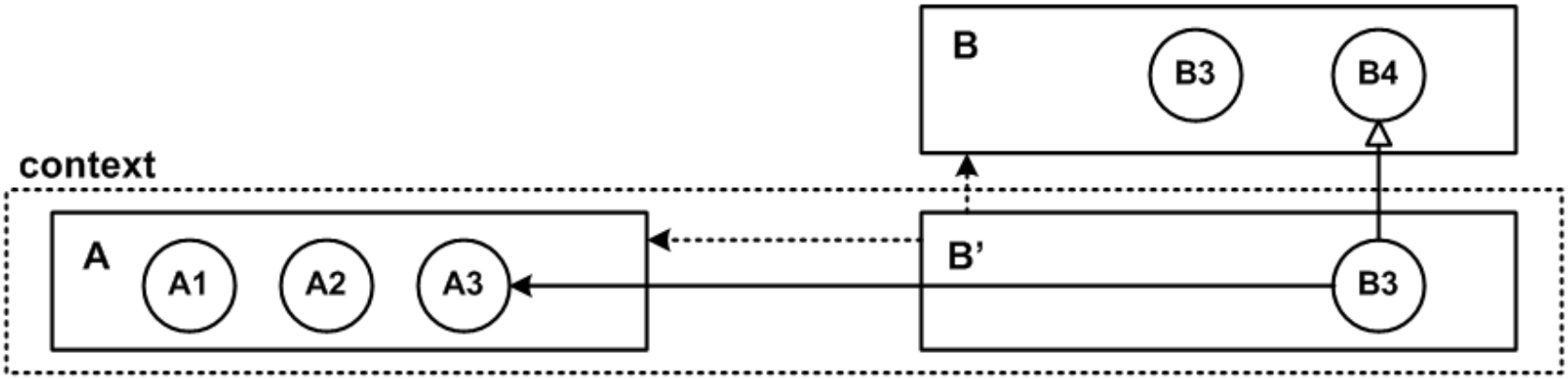
When DSL (for several concerns) are developed from scratch, as a group, then dependencies between the concerns can be materialized as dependencies between the languages and the language concepts

Langugae Extension



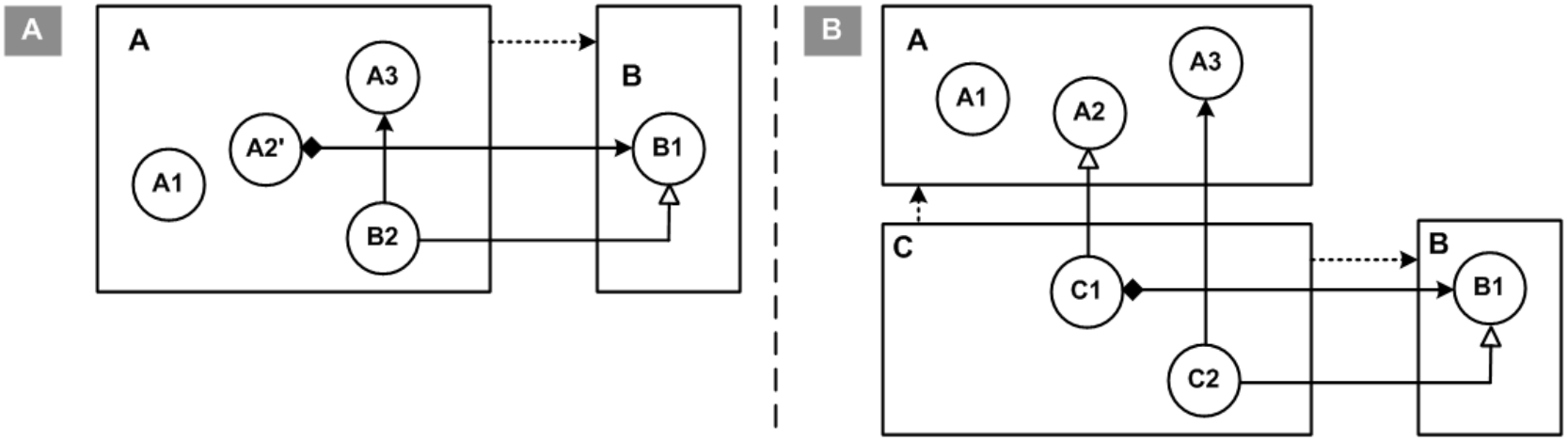
A language B extends another language A if B contains additional language concepts. This means that for programs written in B, all concepts from A are available, plus those defined in B.

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.

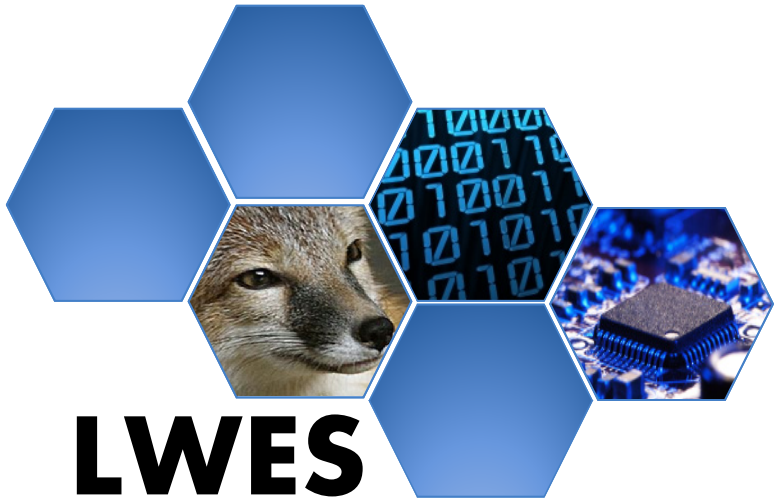
Langugae Embedding



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

8

The LWES Project

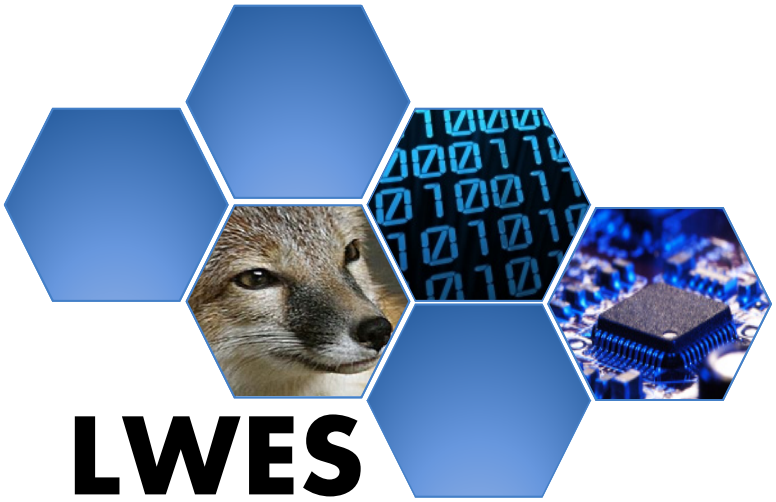


LWES

Language Workbenches *for* Embedded Systems

Bundesministerium für Bildung und Forschung

http://mbeddr.com

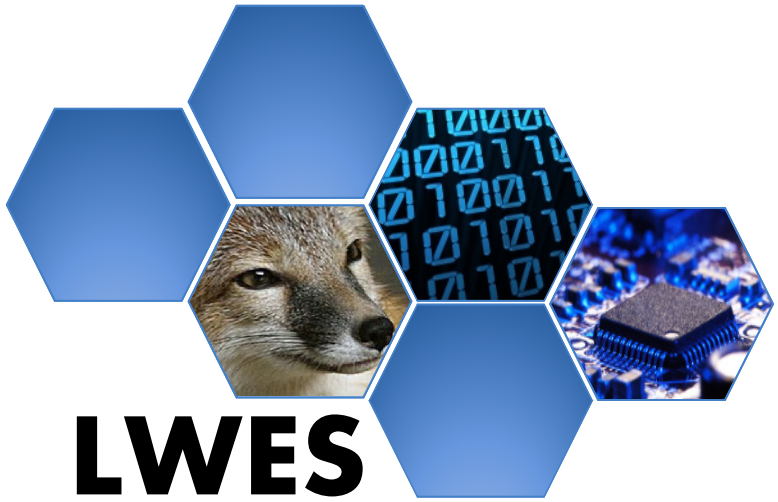


LWES
Language Workbenches
for Embedded Systems

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

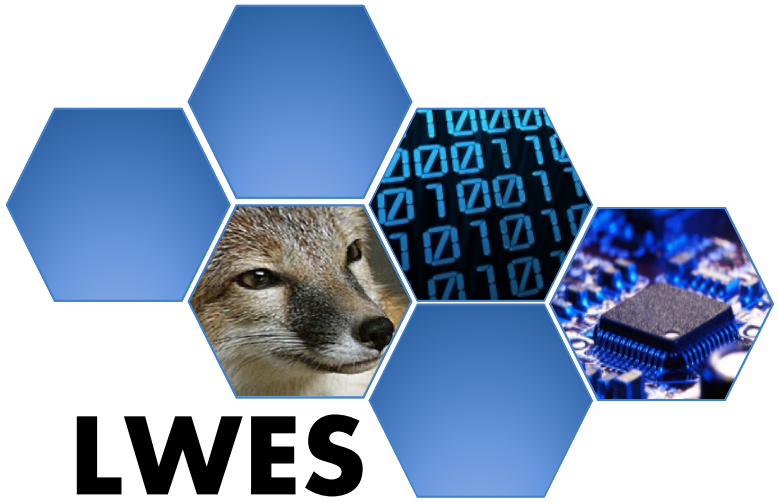


LWES

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

itemis fortiss SICK LEAR

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] -> running
    }
    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

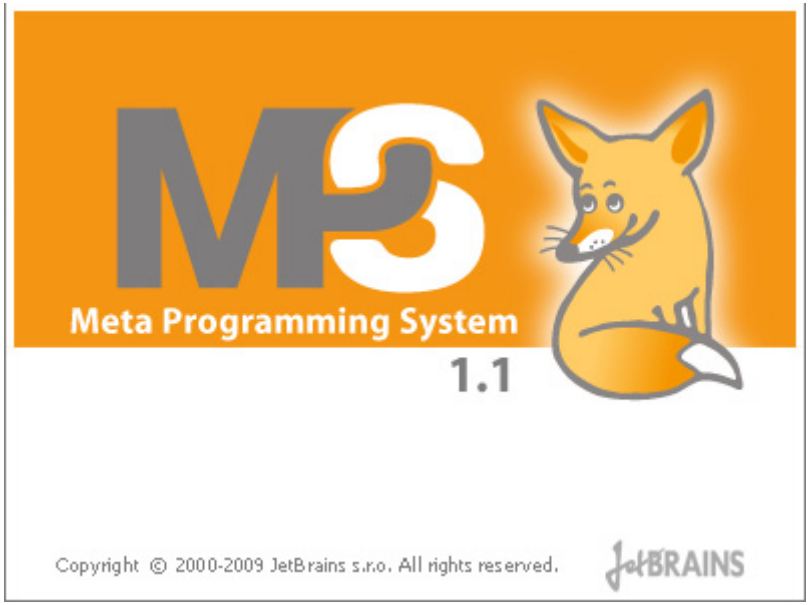
```
exported interface MotorControl {
  void stop(   );
  void setLeftSpeed( int8 speed );
  void setRightSpeed( int8 speed );
}
```

```
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

}
```



JetBrains MPS

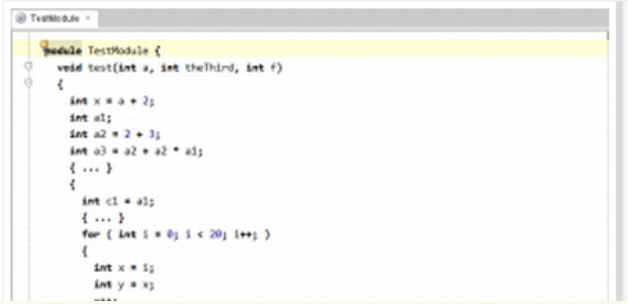
TOOLS

mbeddr.com

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

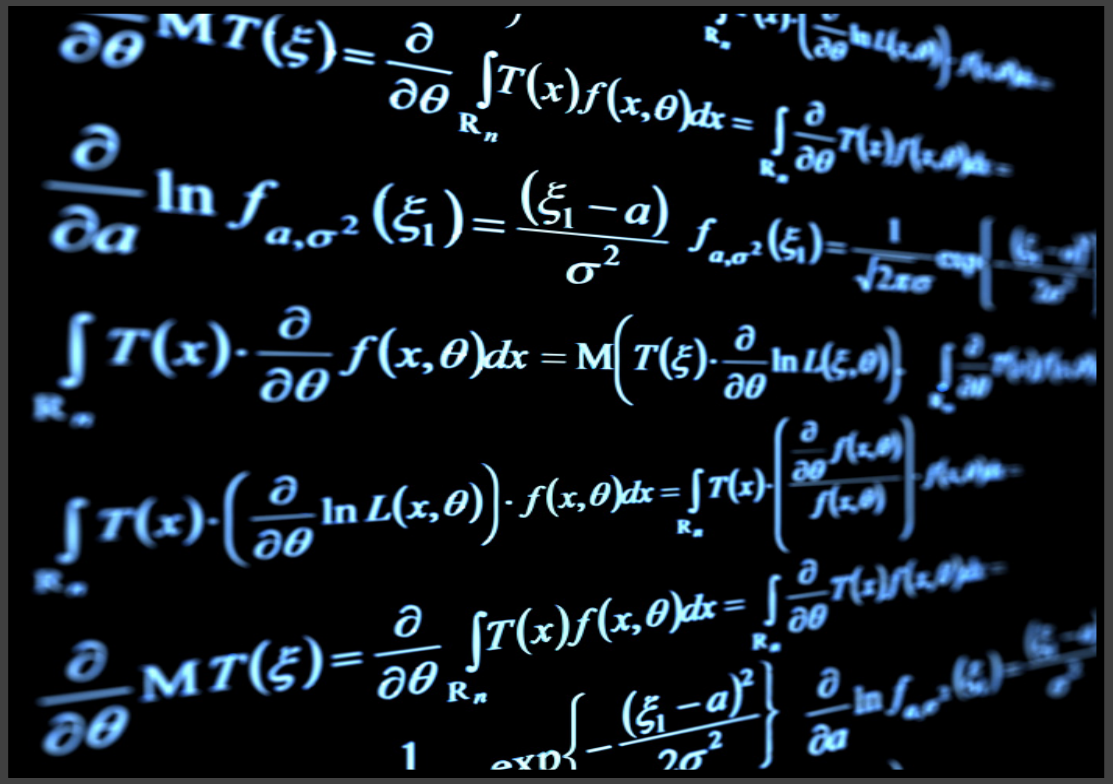
code (3)
demos'n'stuff (4)
dev progress (1)
news (4)
Uncategorized (2)

Pages


9
Bonus:
Best Practices

Limit Expressiveness

Notation, Notation, Notation

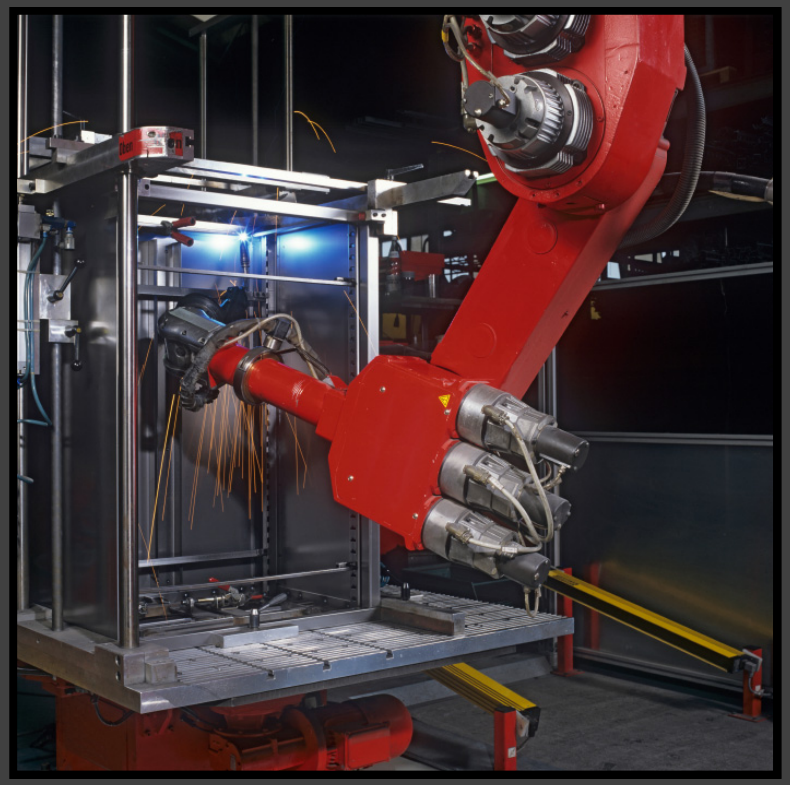


Graphical vs. Textual

Viewpoints

Teamwork Support

Interpretation
vs.
Generation



Rich Domain Specific Platform

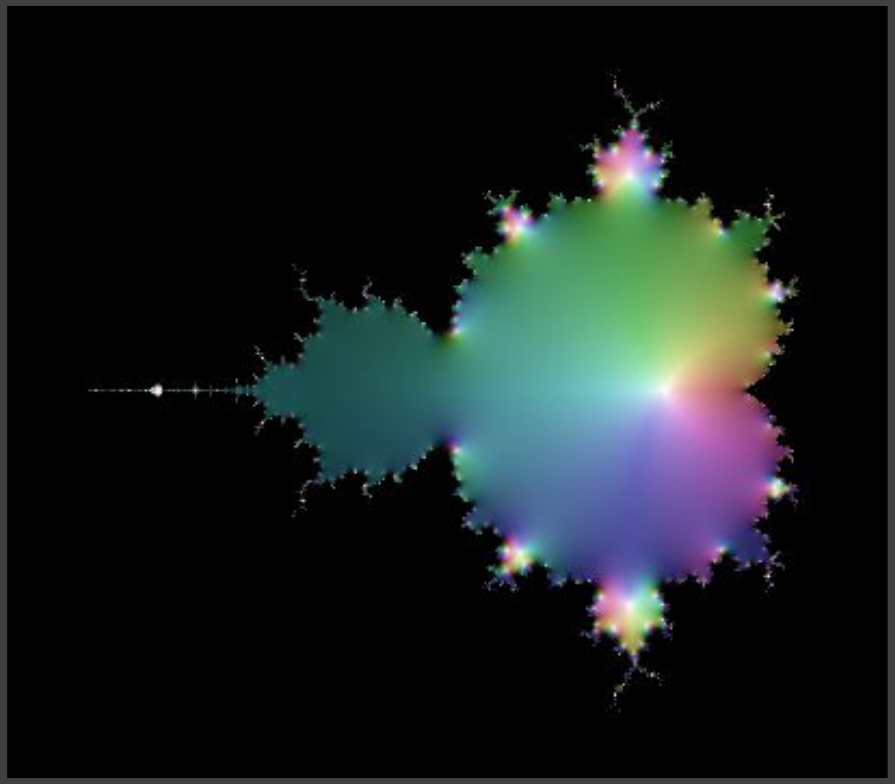
Checks First and Separate

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