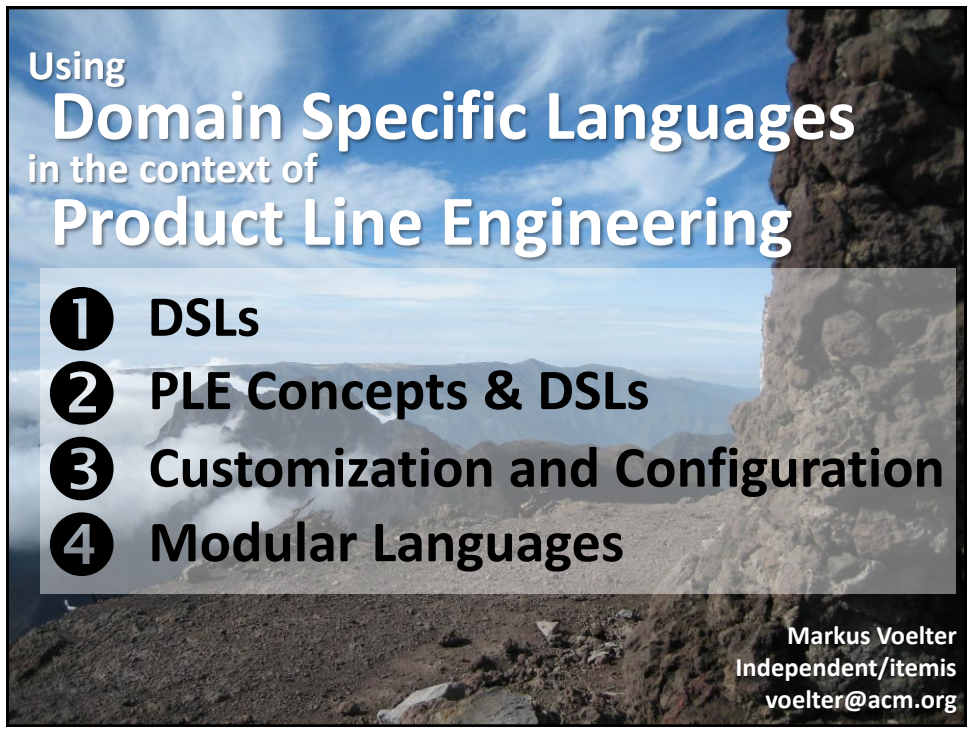
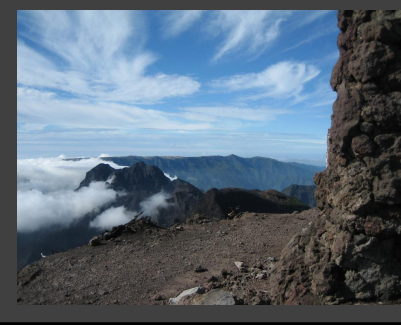
Using
Domain Specific Languages
in the context of
Product Line Engineering
Markus Voelter
Independent/itemis
voelter@acm.org


Using
Domain Specific Languages
in the context of
Product Line Engineering
1 DSLs
2 PLE Concepts & DSLs
3 Customization and Configuration
4 Modular Languages
Markus Voelter
Independent/itemis
voelter@acm.org

1 DSLs



abstractions
~computing

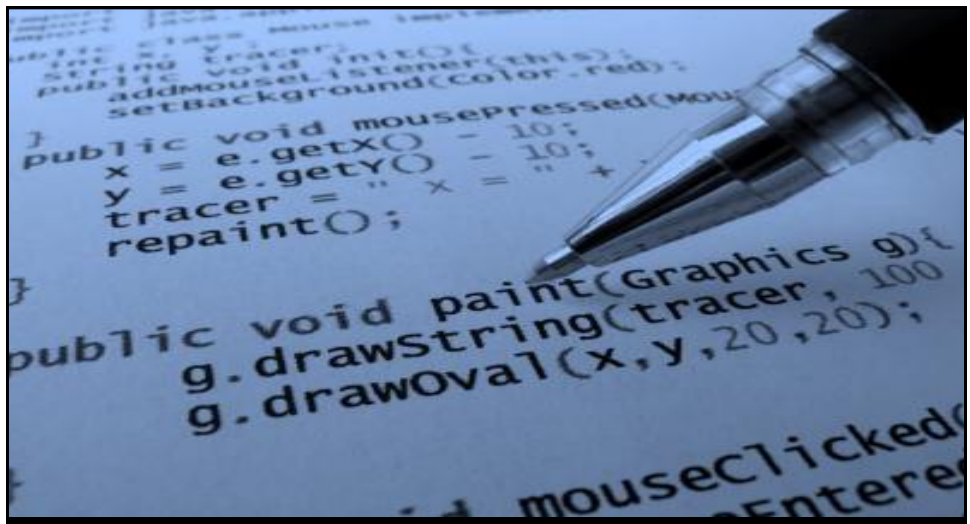
chips

abstractions
~computing

bits

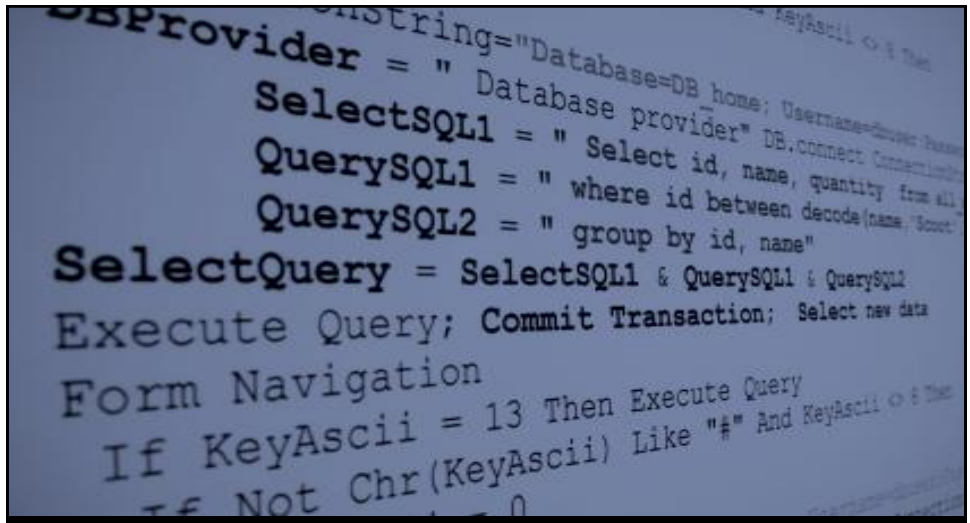
abstractions ~computing

C



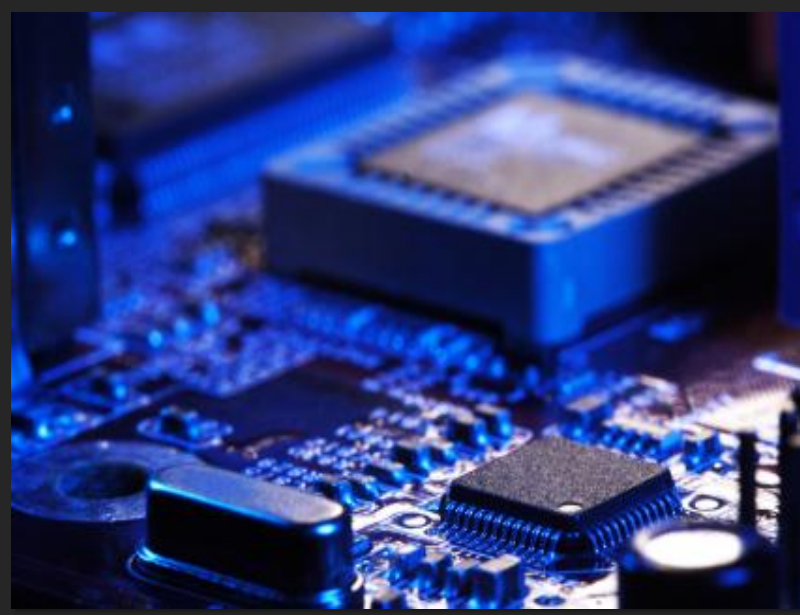
abstractions ~computing?

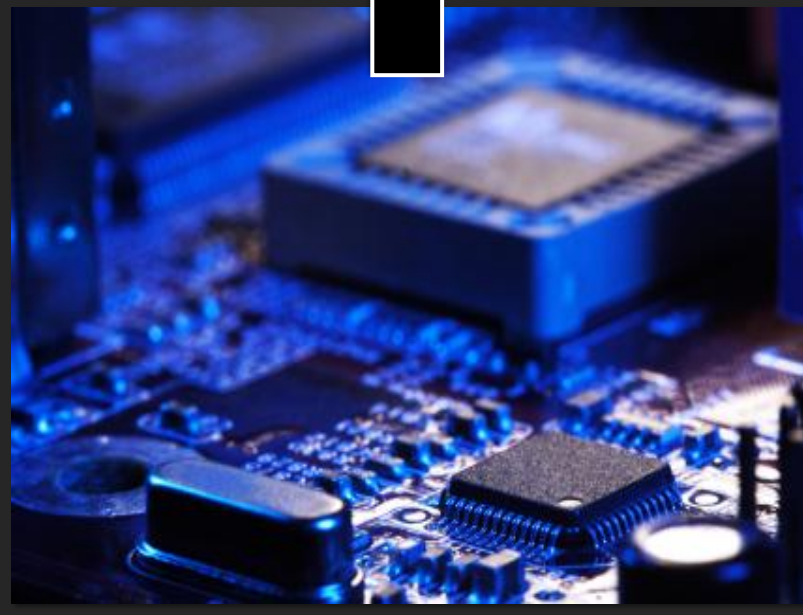
Java



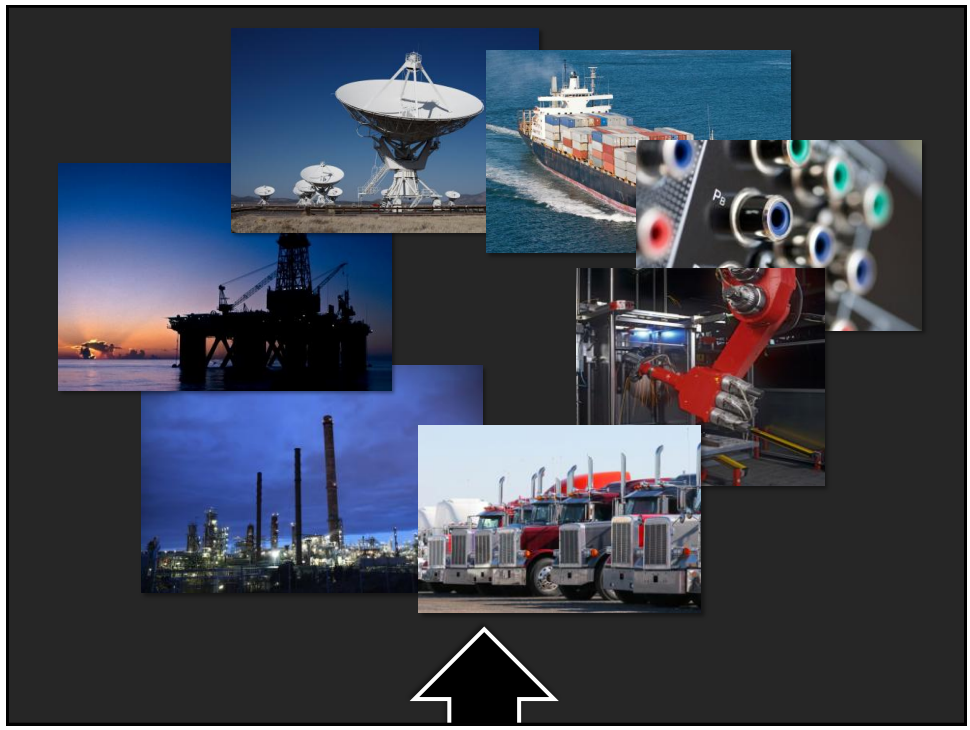
abstractions ~computing?

SQL





?



general purpose

domain specific

tailor made

effective++

specialized, limited

used by experts

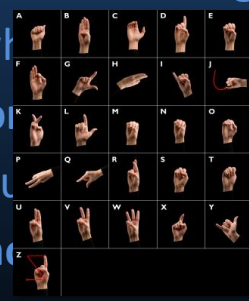
together with other specialized tools

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

A DSL is a **focussed**, **processable** **language** for des ic **concern** when b 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 **conce** stem in a **specifi** **actions** and **notati** /suitable for the **sta** ify that particular concern.

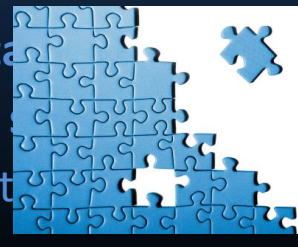
A DSL is a **focussed**, **processable** **language** for describing a specific **concern** wh a system in a specific **do** **stractions** and **notations** u ural/suitable for the **stakeh** pecify that particular concern.



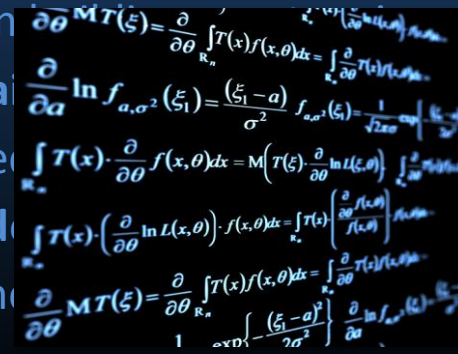
A DSL is a **focussed**, **processable** **language** for describing a specific **concern** when building a system in a specific **doma** nd **notations** use for the **stakehold** particular con

A DSL is a **focussed**, **processable language** for describing a specific **concern** when building a system in a specific **domain**. The **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 who specify that part



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



specific **domai**

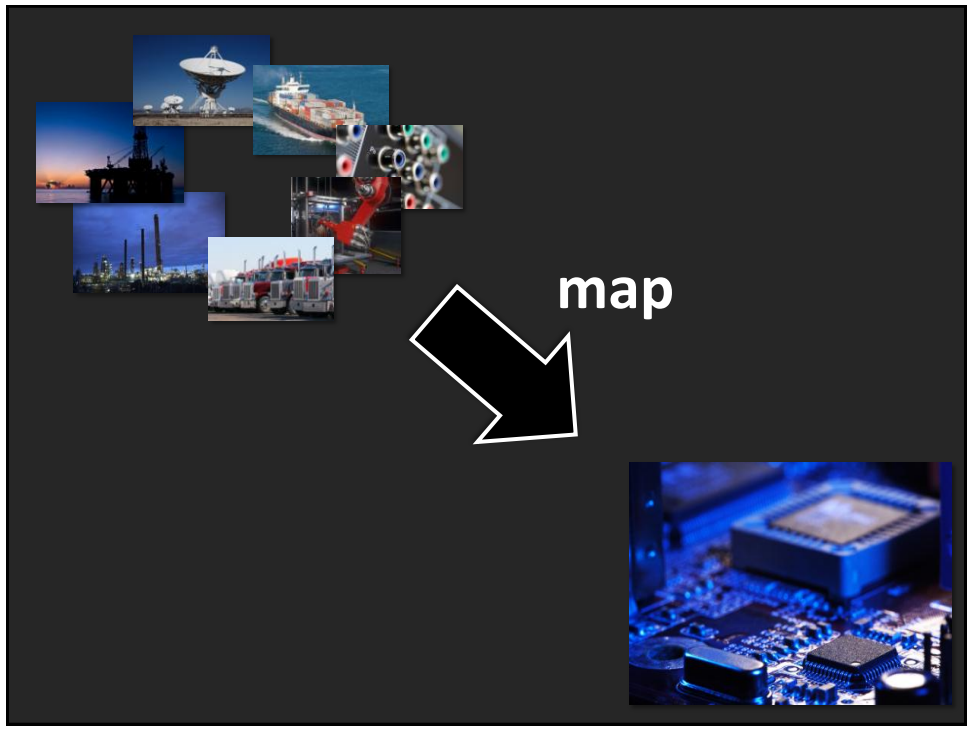
notations use

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 and **notations** used are na le for the **stakeholders** who t particular concern.

execute?

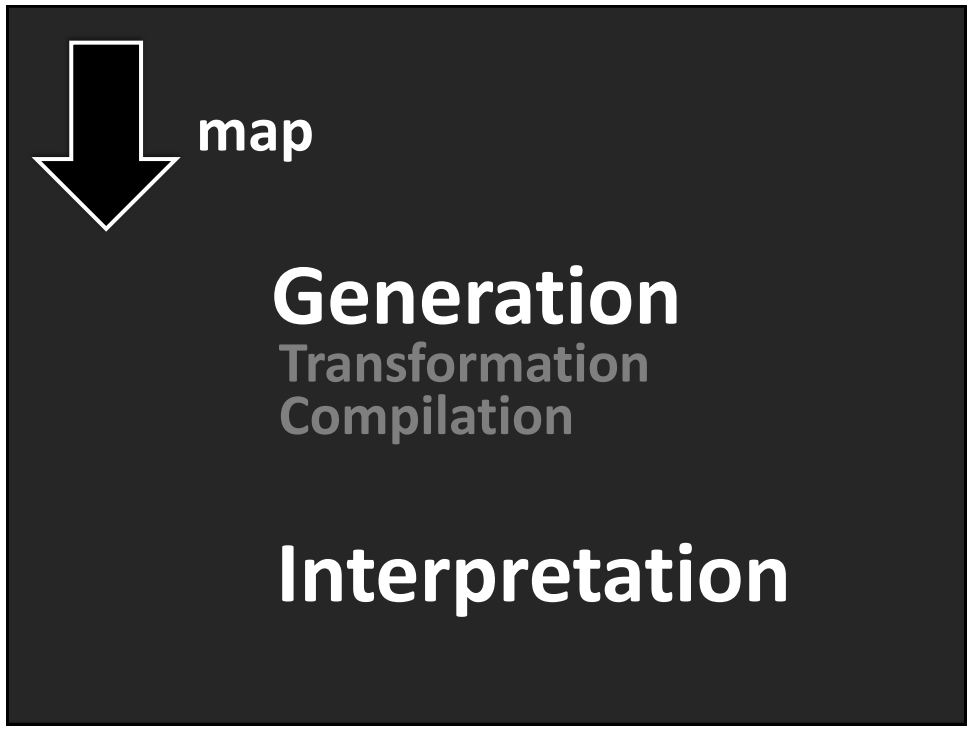


DSL Program
(aka Model)

map

automated!

GPL Program


map
Generation
Transformation
Compilation
Interpretation

Activities

- Analysing Domains
- Defining Languages
  - Adapting/Selecting
- Building Editors
- Transforming Models
- Building Generators
- Building Frameworks

Activities

Analysing Domains

Defining Languages

Adapting/Selecting

Building Editors

Transforming Models

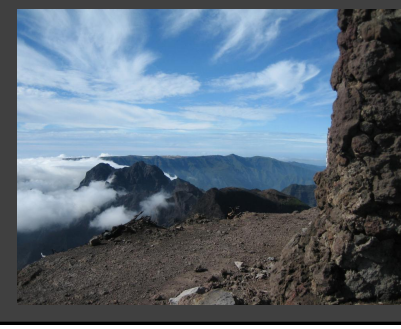
Building Generators

Building Frameworks

… and using all of that to build apps

1 DSLs

Some Examples



Embedded Protocol Handler

```
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;
    };
};
```

OSGI-based Enterprise System

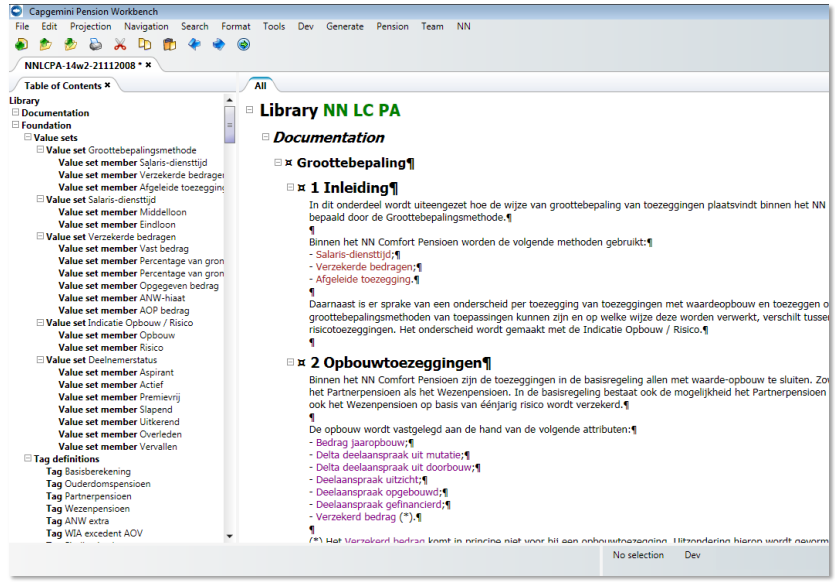
```
subsystem the.world.scenarios {

    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 onceUponS
            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"
    }

}
```

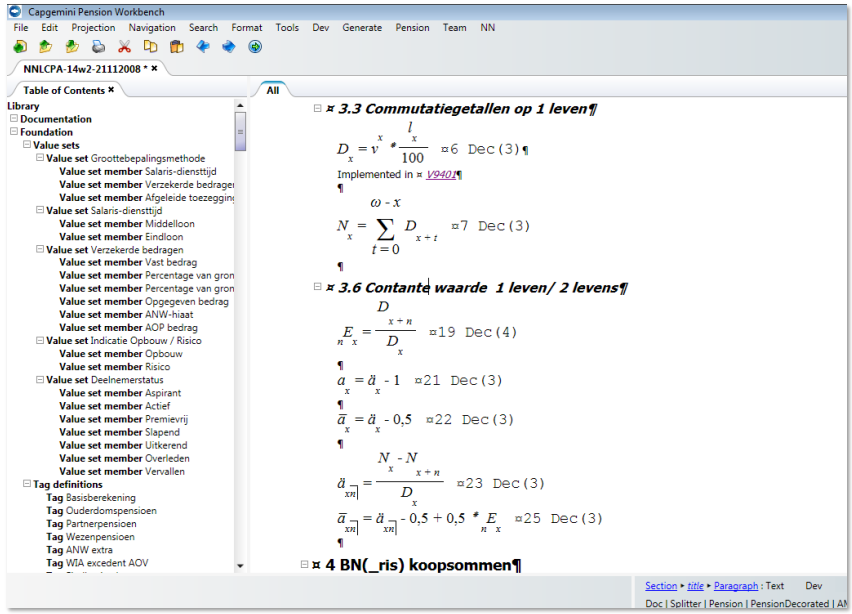
647x551

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

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

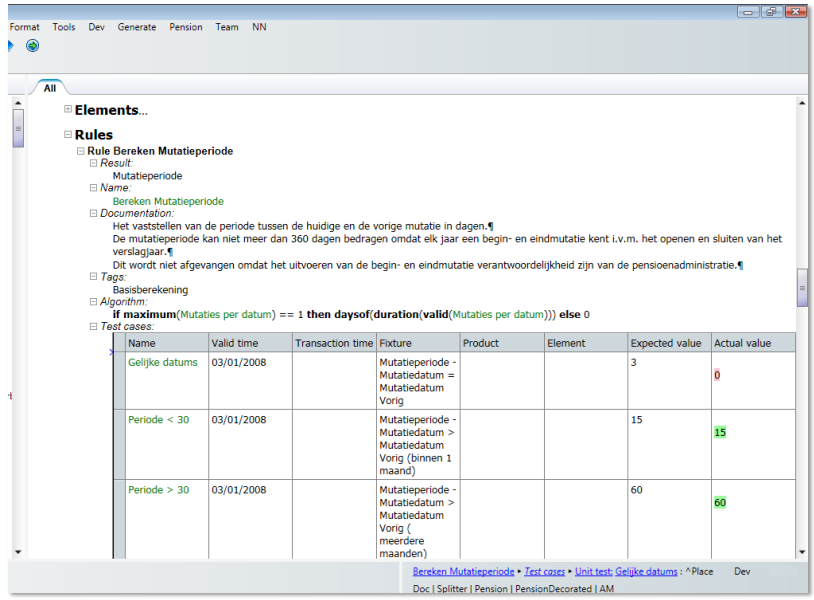
Insurance Contracts



Insurance Contracts



Insurance Contracts



Alarm System Menus

```
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
```

Alarm System Menus

```
message TurnOffAlarm
message TurnOnAlarm
message AlarmLevel
message UnlockNow

component AlarmManager {
    receives TurnOffAlarm
    receives TurnOnAlarm
    receives AlarmLevel
}

component MasterSwitch {
    receives UnlockNow
}
```

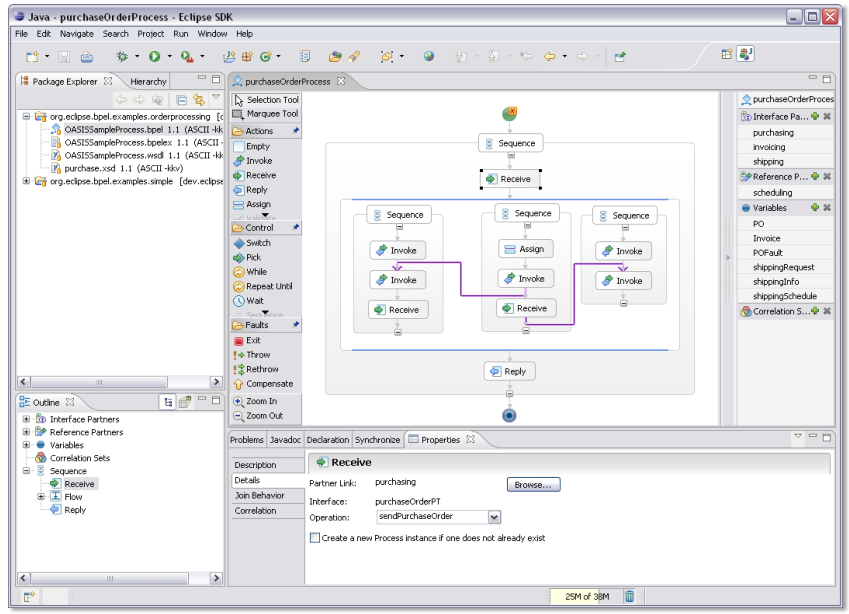
Hearing Aids

A DSL for Hearing Aid Configuration

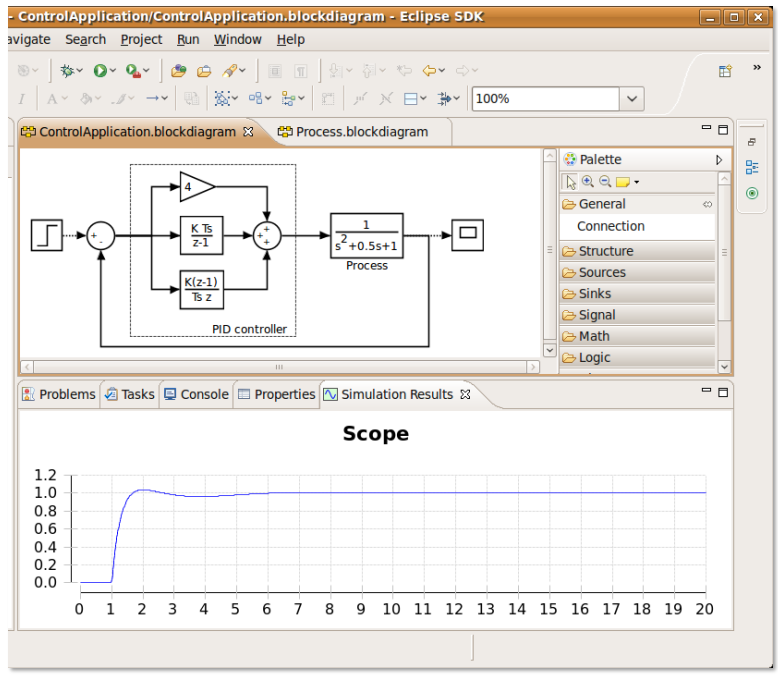
Refrigerators

A DSL for Refridgerator Configuration

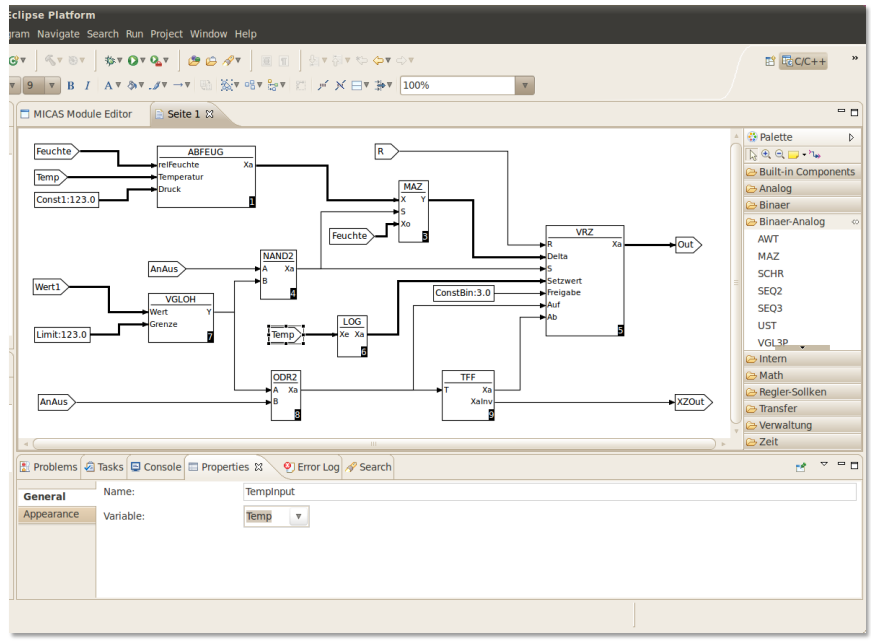
BPEL Designer



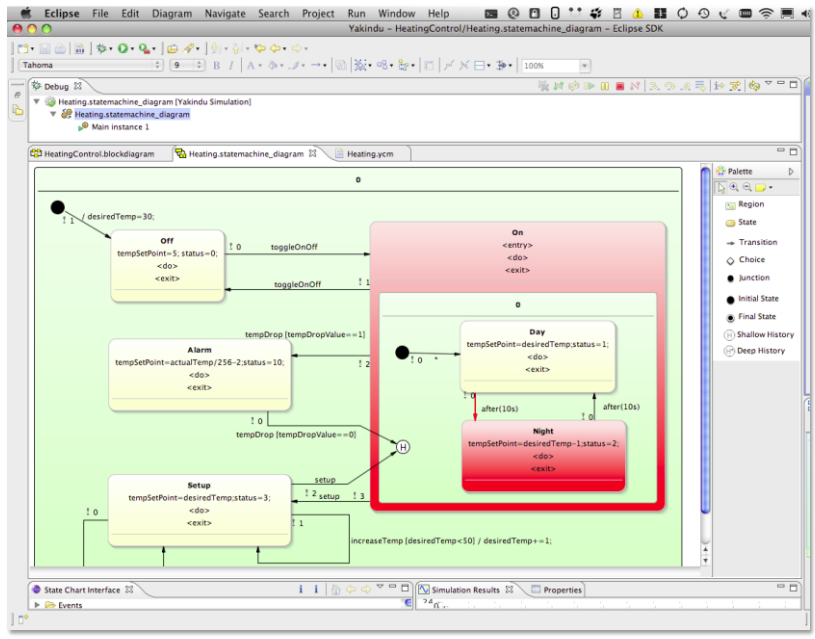
Block Diagrams



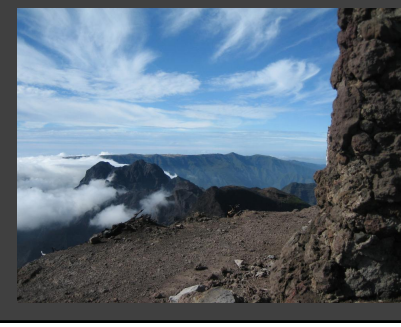
PLC Programming



State Charts



❷ 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

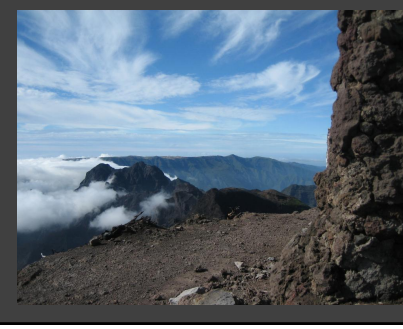
… languages
… generators
… editors

variation point

… bind how
… kind of varibility
… bind when

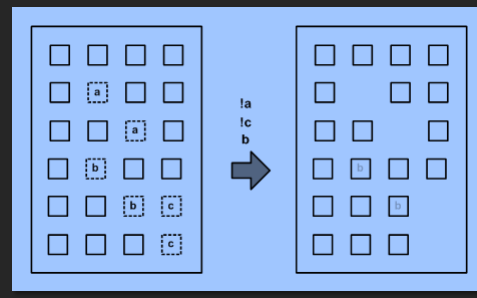
❷ PLE Concepts & DSLs

Bind How
Kinds of Variability
Bind When



Variability Mechanisms
Removal

… optionally take away from



Variability Mechanisms
Removal

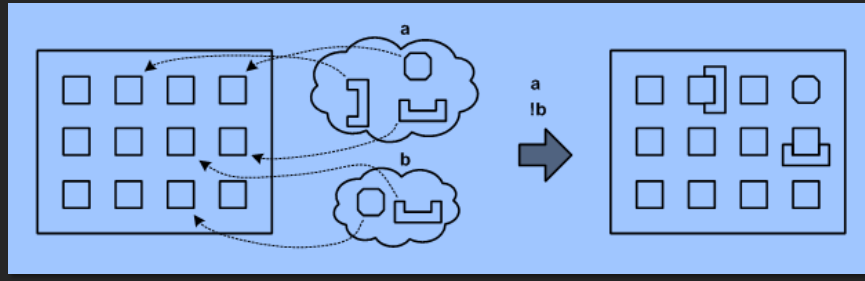
… optionally take away from overall whole

Challenge:

overall whole can get big and unwieldy

Variability Mechanisms
Injection

… optionally add to minimal core



Variability Mechanisms
Injection

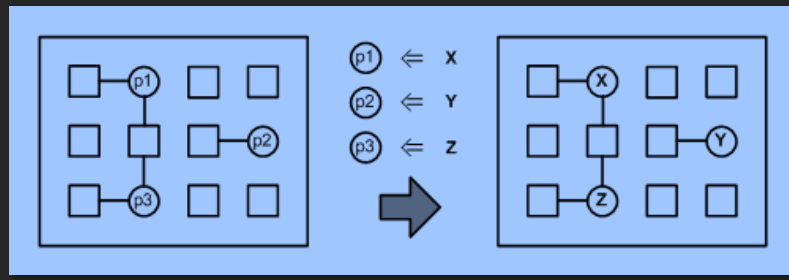
… optionally add to minimal core

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:

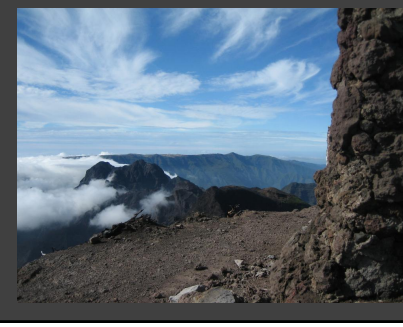
types for parameters can be non trivial (DSLs)

❷ PLE Concepts & DSLs

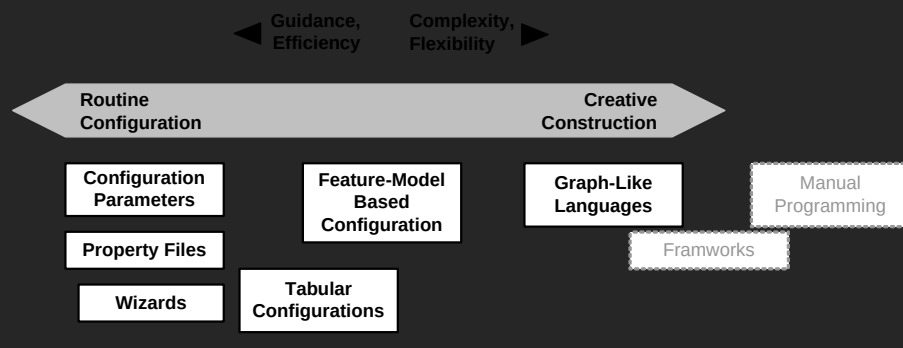
Bind How

Kinds of Variability

Bind When

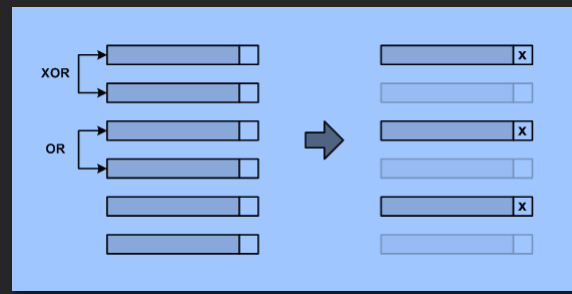


Configuration vs. Customization
Variability



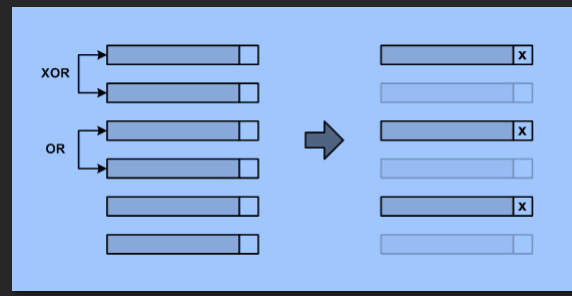
Configuration

… selecting options
… setting param values

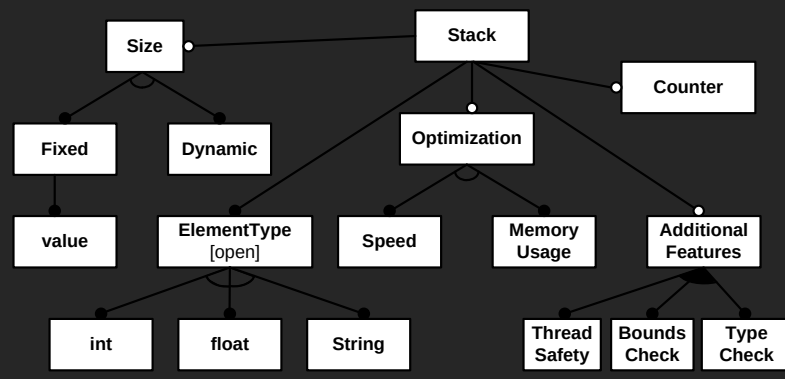


Configuration

… selecting options
… setting param values

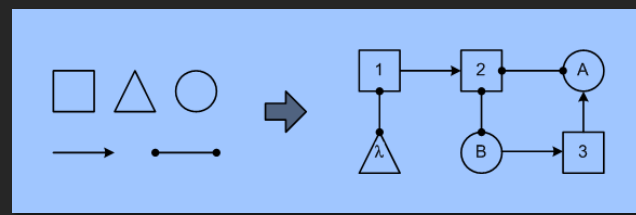


Configuration
Feature Models



Customization

… DSLs
… instantiation
… connections

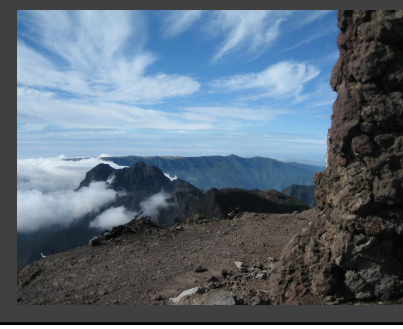


DEMO I

Example DSL – Fountains

❷ 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

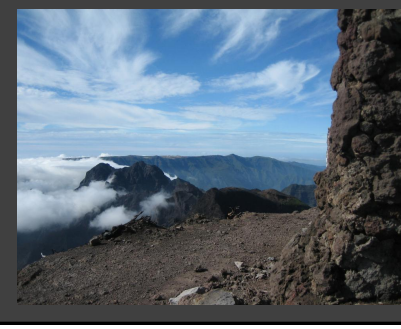
possibly direct feedback from

in-IDE interpreter

DEMO II

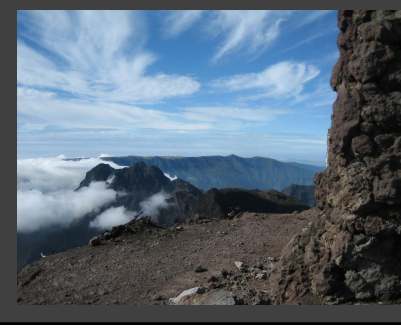
In-IDE Interpreter
in Fountains

3 Customization and Configuration



3 Customization and Configuration

In-Language
Varying models
DSLs as Parameters



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

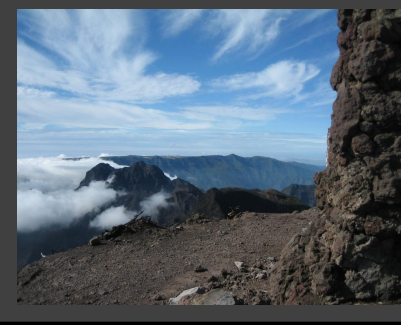
AOP

DEMO III

A DSL with
“variant blocks”

❸ Customization and Configuration

In-Language
Varying models
DSLs as Parameters



Two Levels

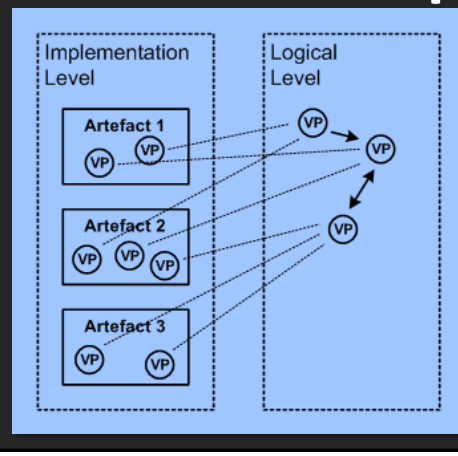
~ problem space
vs. software space

Problem Space:
Configuration

Solution Space:
Customization

Two Levels

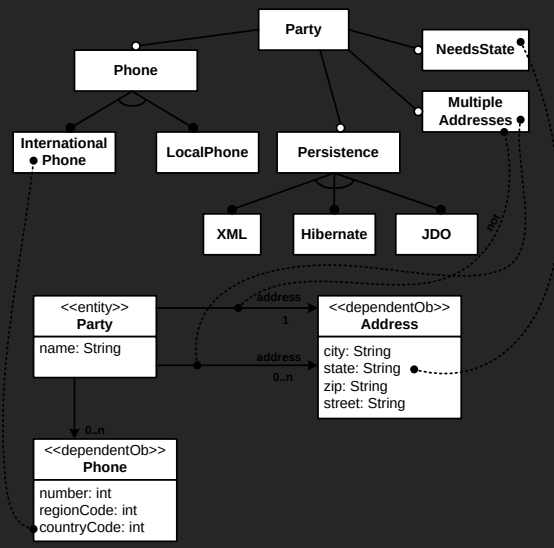
~ problem space vs. software space



Two Levels Removal

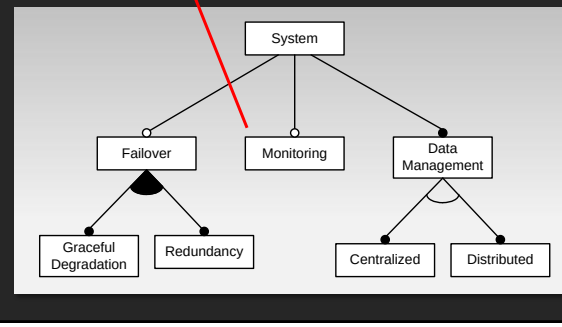
```
#if defined (ACE_HAS_TLI)
  static ssize_t t_snd_n (
         ACE_HANDLE handle,
         const void *buf,
         size_t len,
         int flags,
         ACE_Time_Value *timeout = 0,
         size_t *bytes_transferred = 0);
#endif /* ACE_HAS_TLI */
```

Two Levels Removal



Two Levels Removal

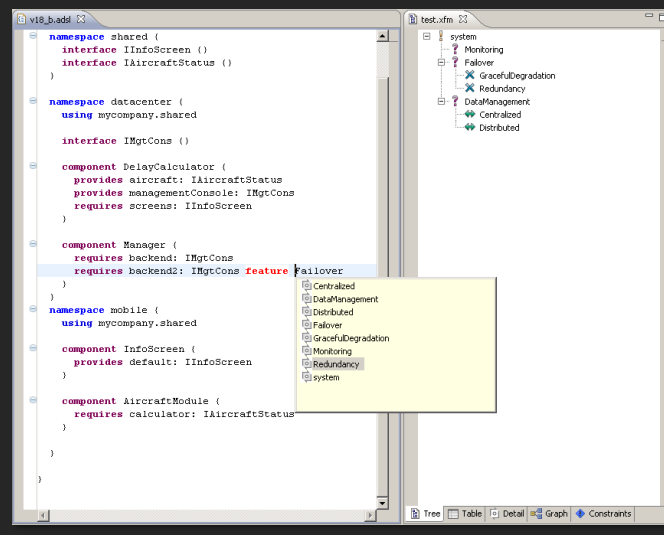
```
component DelayCalculator {
  provides default: IDelayCalculator
  requires screens[0..n]: IInfoScreen
  provides mon: IMonitoring feature monitoring
}
```



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

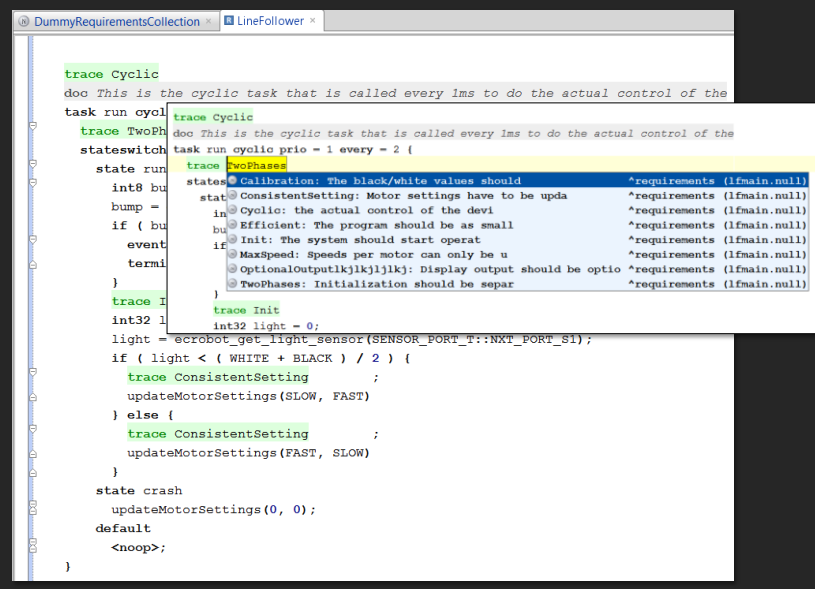
Two Levels

~ varying models, not text

… simplifies traceability

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

… simplifies traceability

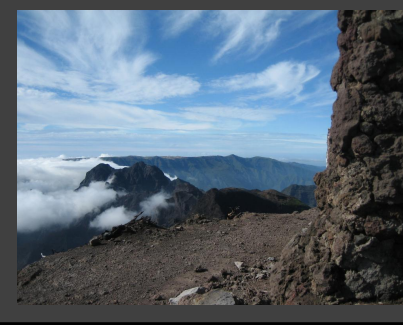


DEMO IV

Feature Annotations

❸ Customization and Configuration

In-Language
Varying models
DSLs as Parameters



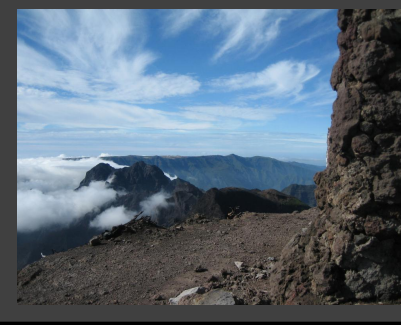
Feature Model
w/ Paramters

… parameters with simple types

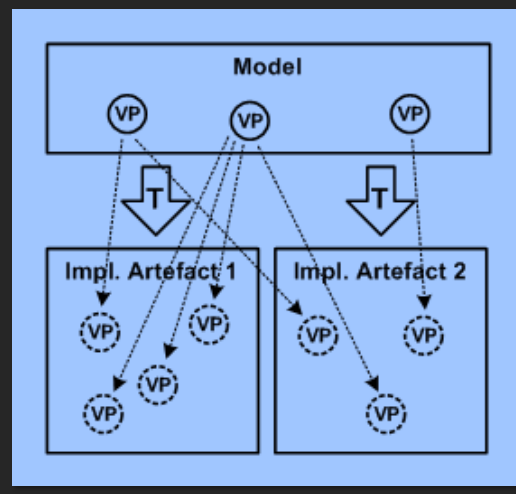
… complex types: DSL/metamodel

3 Customization and Configuration

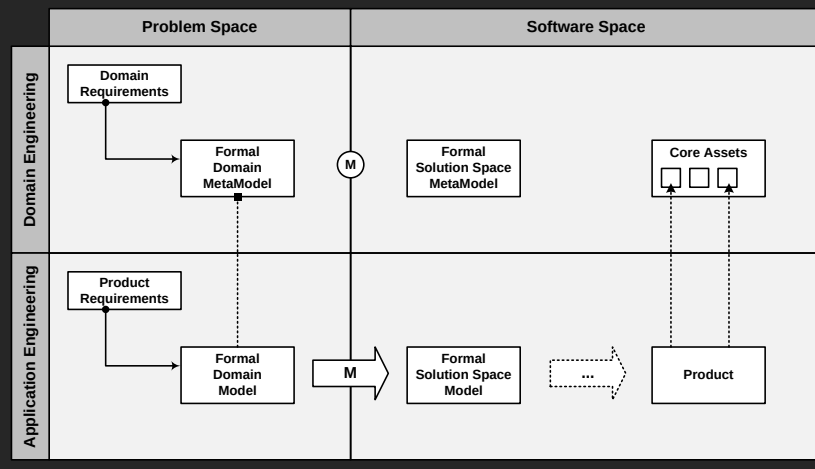
Trafo Variability



Model-Based Implementation



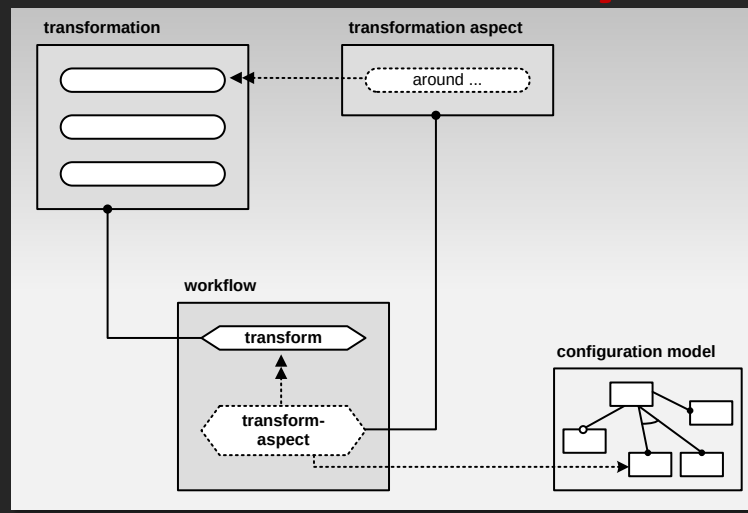
Model-Based Implementation



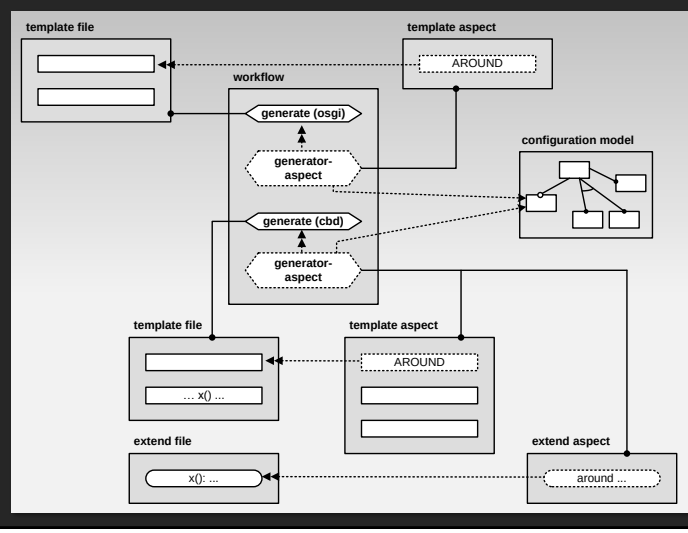
Transformation Variability

```
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
);
```

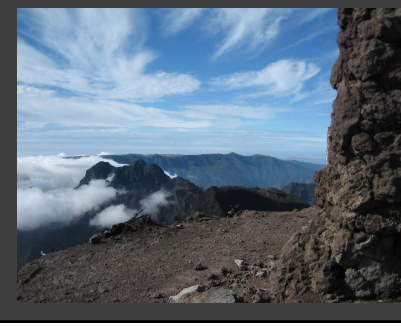
Transformation Variability



Generator Variability



4 Modular Languages



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!

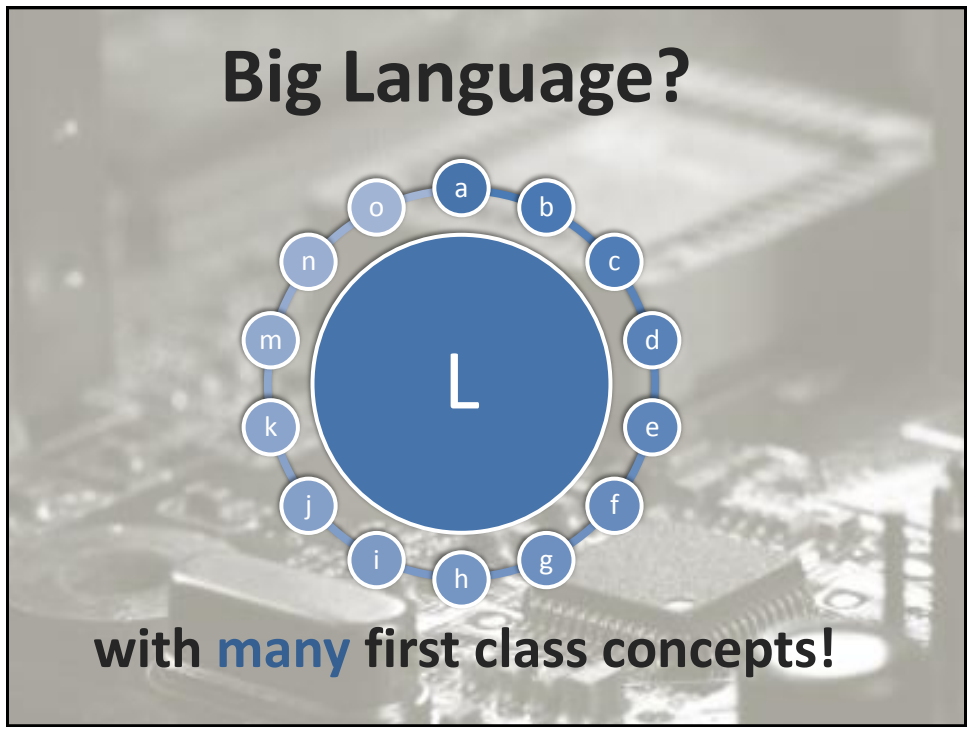
… with different degrees of **domain-specificity**
… with suitable **notations**
… with suitable **expressiveness**

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

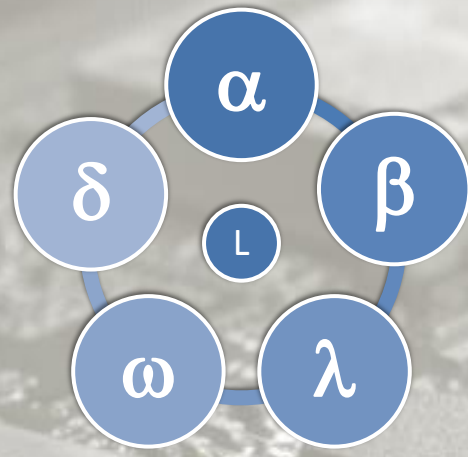
And always:
precise and tool processable

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

PLE for DSLs
suitable to related domains

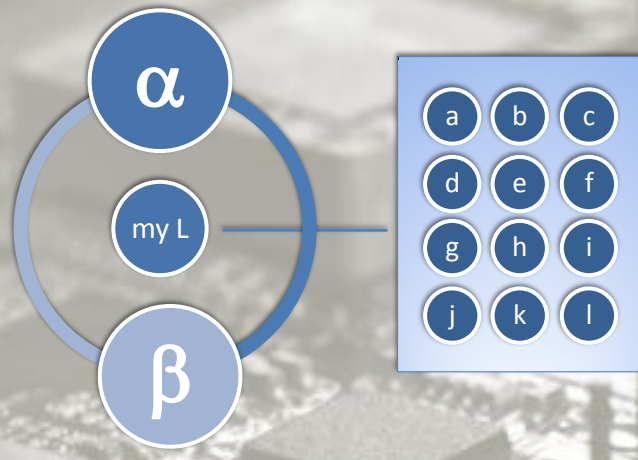

Big Language?
a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
L
with many first class concepts!

Small Language?



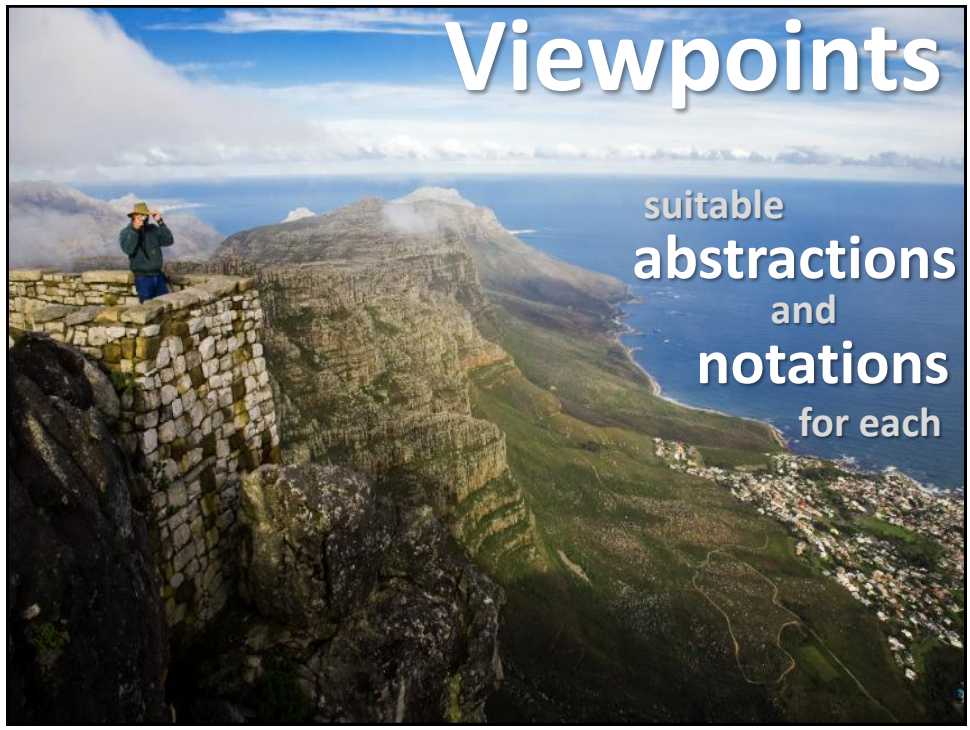
with a few, orthogonal and poweful concepts

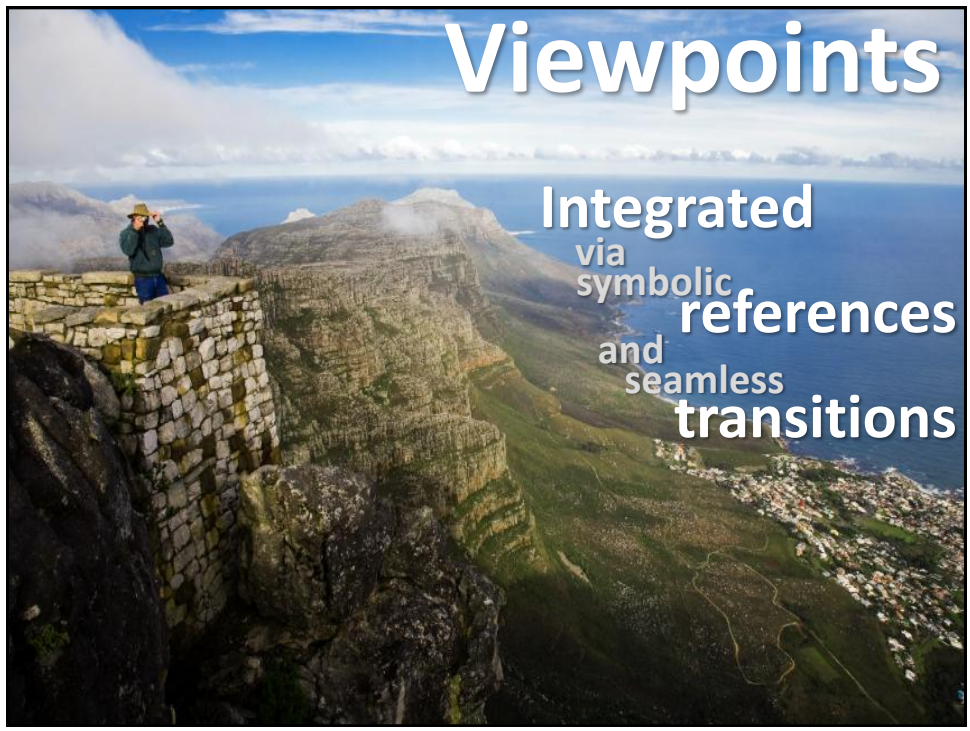
Modular Language

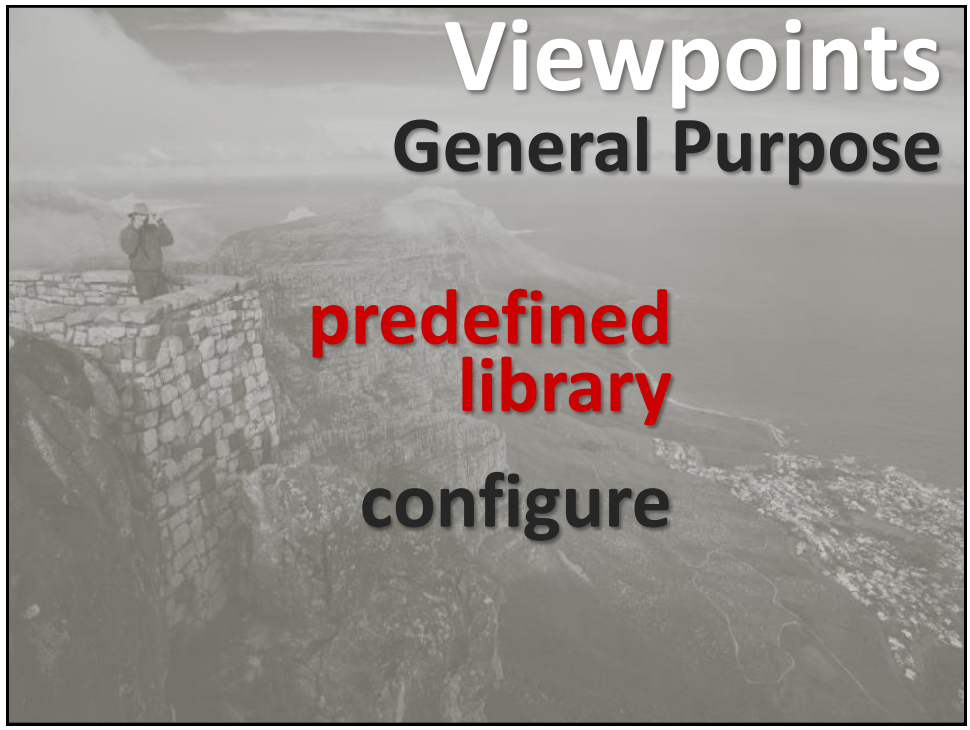


with many optional, composable concepts

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

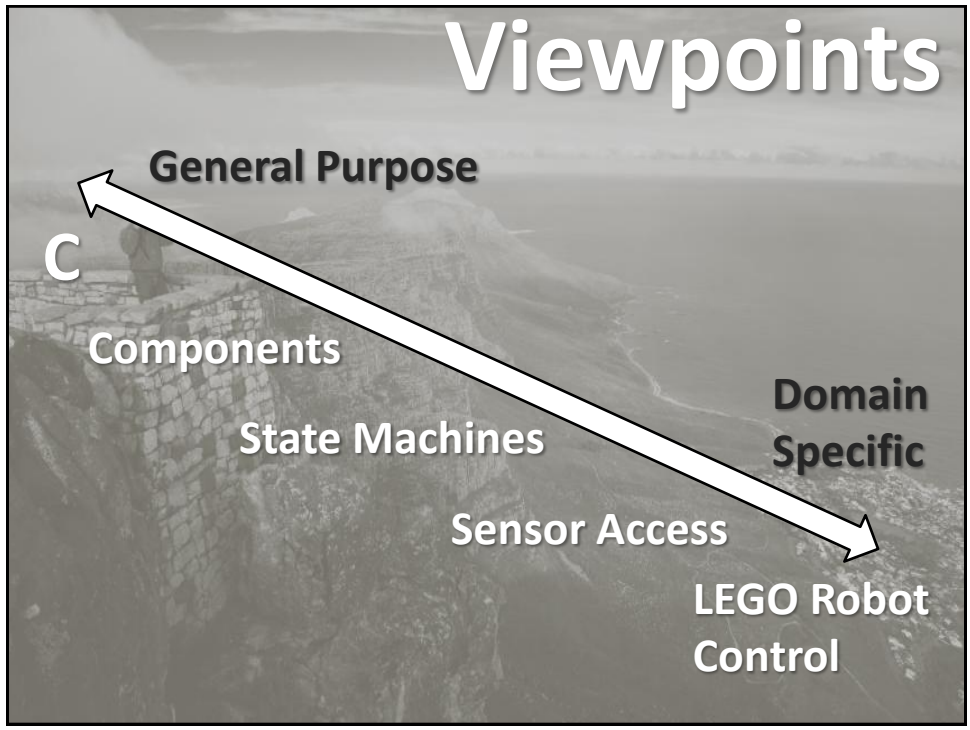
Viewpoints

General Purpose

C

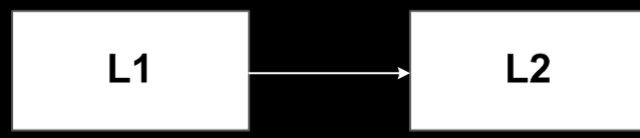
Domain Specific

LEGO Robot Control


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

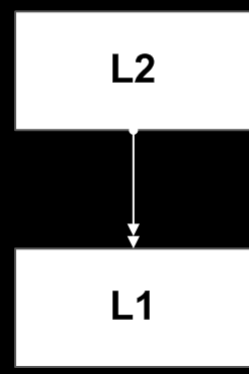
Language Integration by…

Referencing



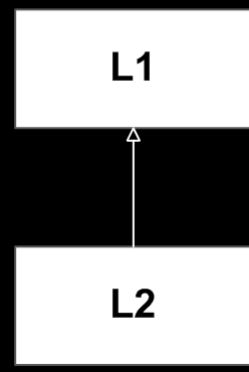
Language Integration by…

Cascading



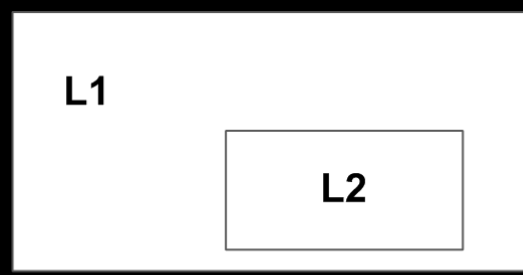
Language Integration by…

Extension



Language Integration by…

Embedding



Language Integration by…

Contributing

DEMO IV

A Modular Language

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

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

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

THE END.
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

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