Using Domain Specific Languages in Product Line Engineering

Markus Voelter
Indepenent/itemis
voelter@acm.org
http://www.voelter.de
itemis

Using Domain Specific Languages in Product Line Engineering

1. PLE Intro
2. Classifying Variability
3. Configuration
4. Customization
5. Binding Time
6. Architecture DSLs
7. Business Domain DSLs
8. Configuration for Customization
9. Variability in Generators
10. Summing Up

1 PLE Intro

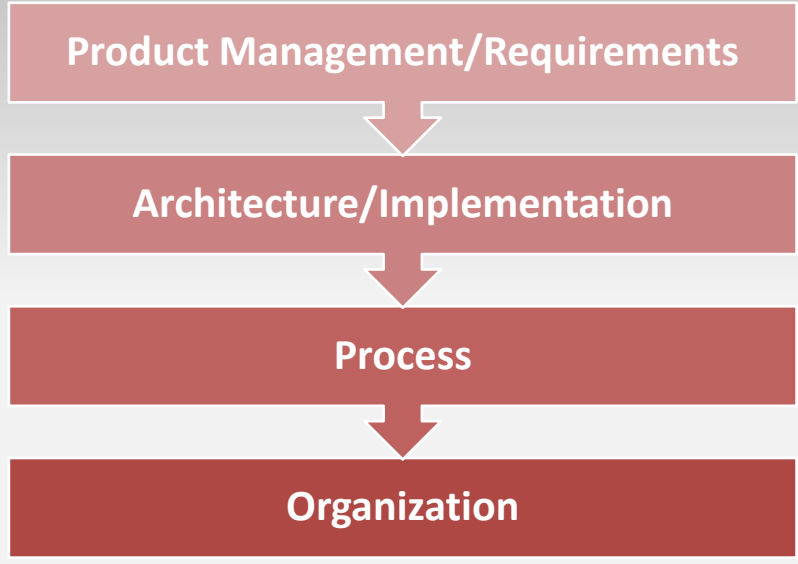
Systematically
analyse, design and build
families of products,
not single systems.

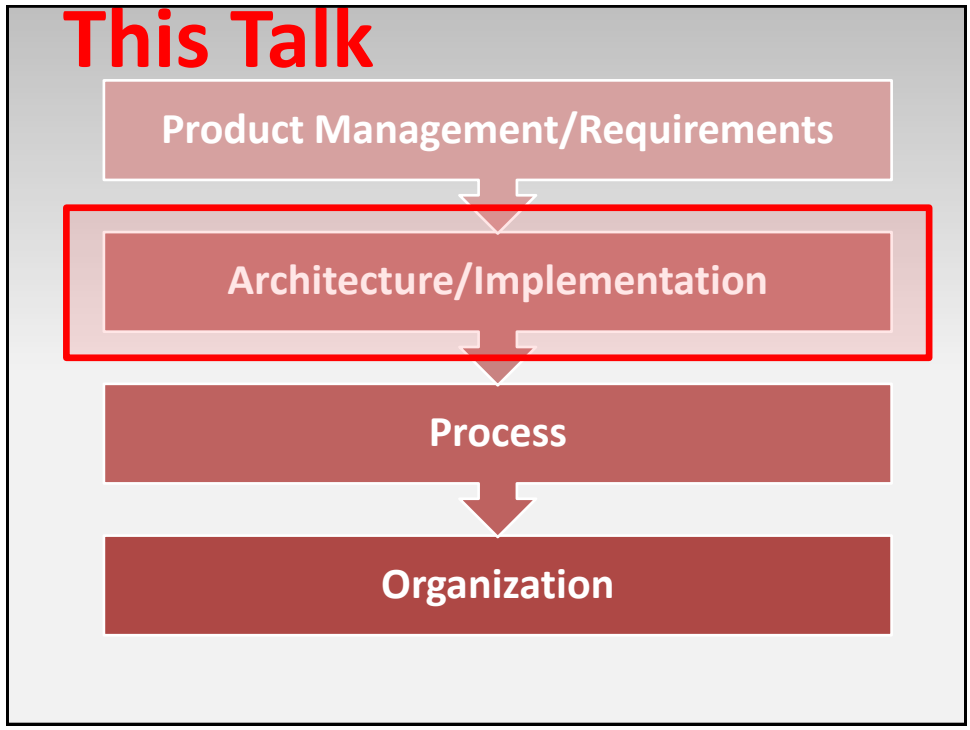
Identify

Commonalities

Manage

Differences





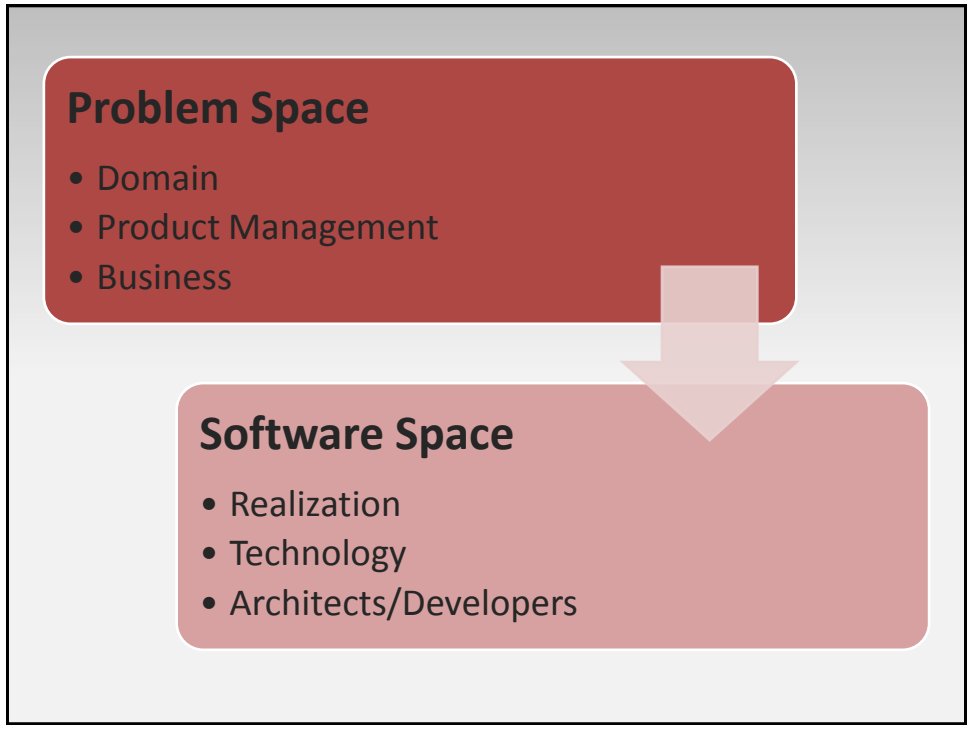
Domain Eng'g

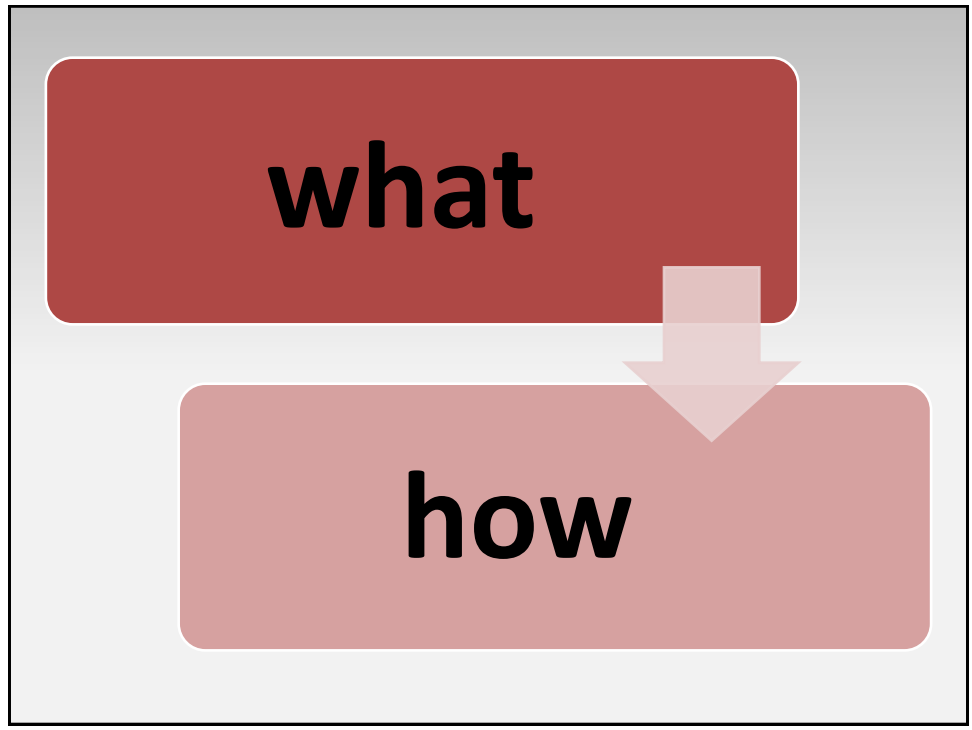
… analyse domain
… build common artifacts
… define/build way to build the products
(i.e. build/setup tools)

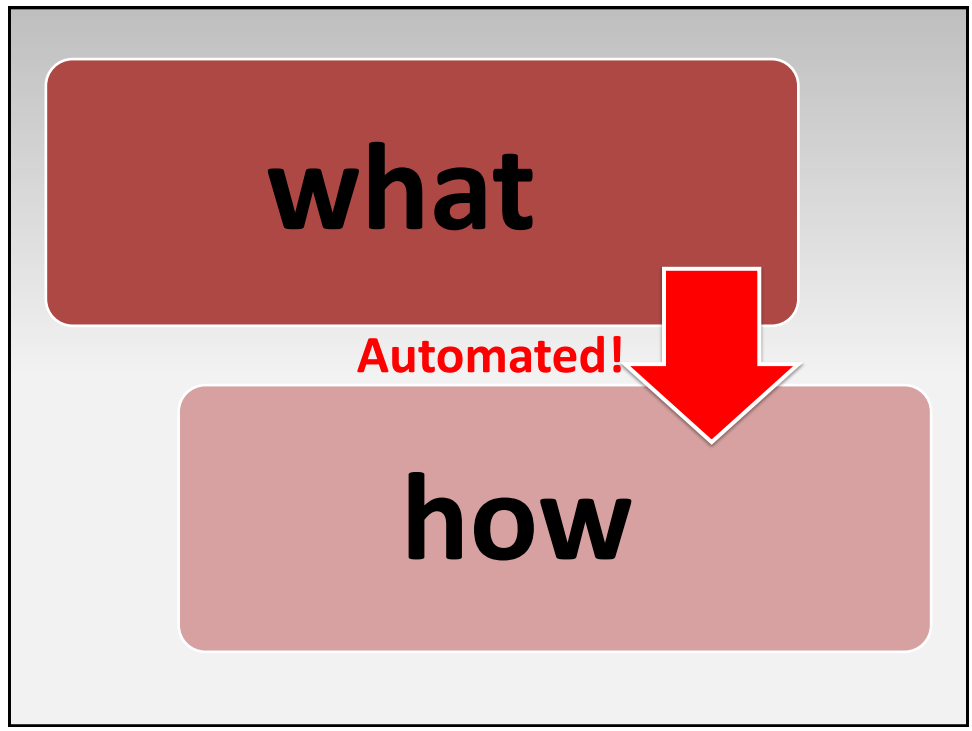
App Eng‘g

… analyze product req‘s
… use domain artifacts
… to build products

… evolve domain artifacts




what
how


what
Automated!
how

Variation Point

… a point where a variation can occur

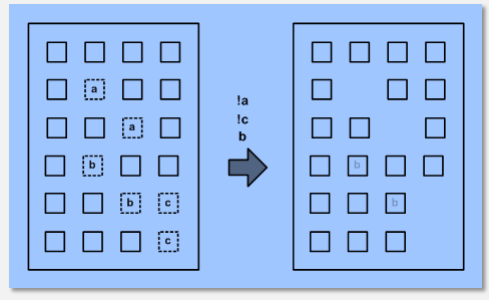
… must be bound for each product

… bind when?
… bind how?

Variability Mechanisms

Removal

… optionally take away from overall whole



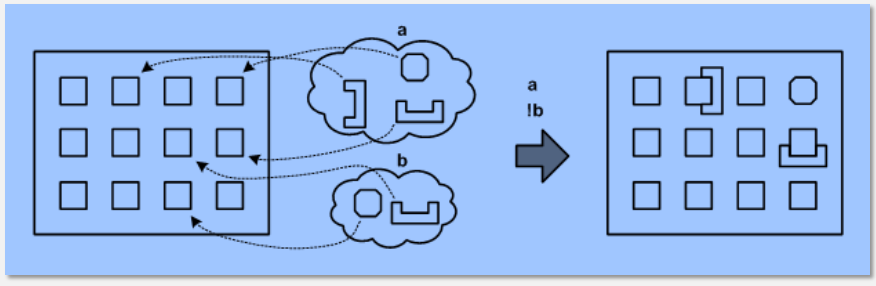
Variability Mechanisms
Removal

… 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

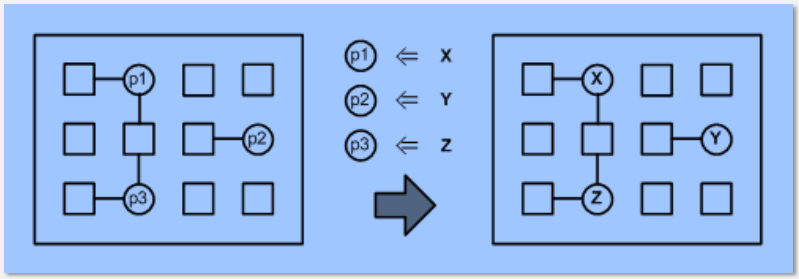
… optionally add to minimal core

Challenge:

how to point into the core and add something to it

Variability Mechanisms
Parametrization

… define values for predefined params



Variability Mechanisms
Parametrization

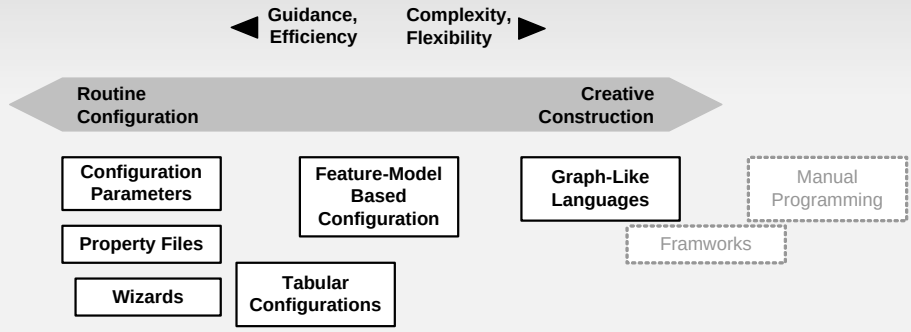
… define values for predefined params

Challenge:
types for parameters can be non trivial (DSLs)

2 Classifying Variability

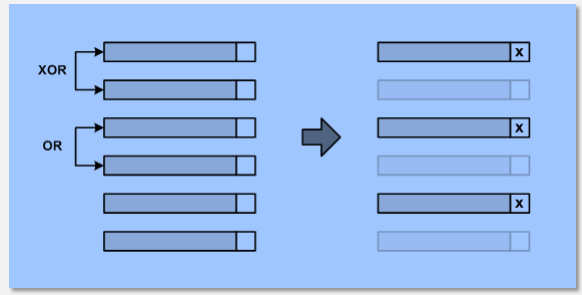
Configuration vs. Customization

Variability



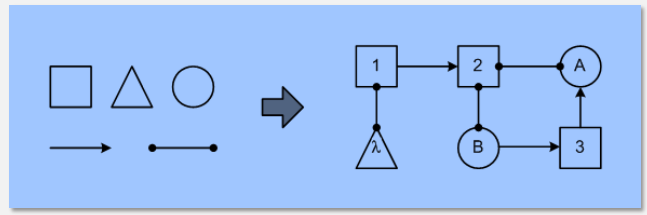
Configuration

… selecting options
… setting param values



Customization

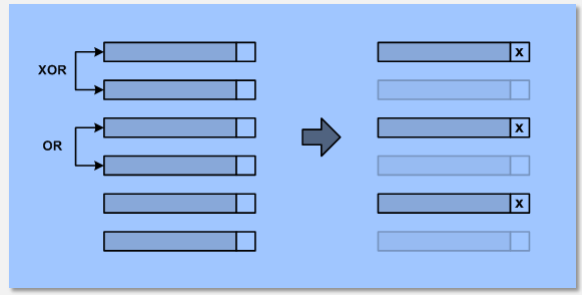
… „real languages“
… instantiation
… connections



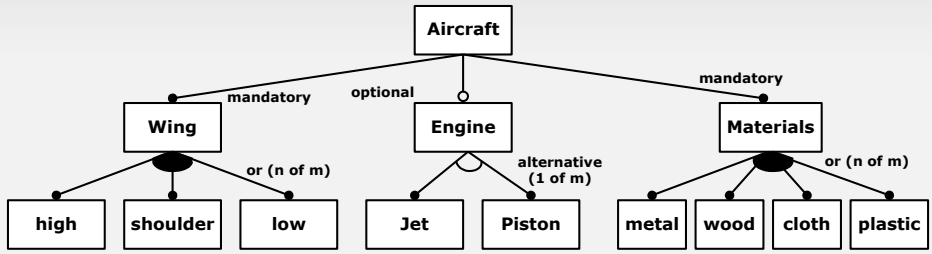
❸ Configuration

Configuration

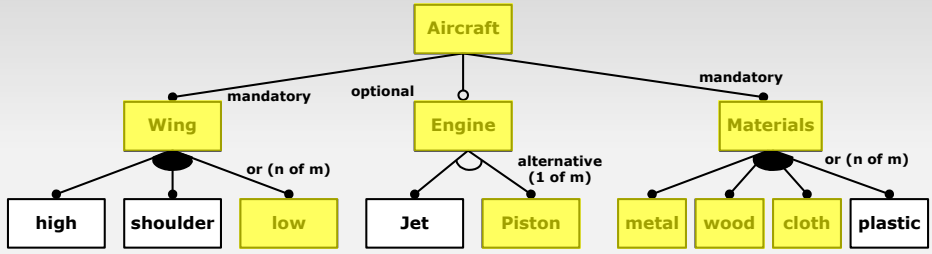
… selecting options
… setting param values



Configuration
Feature Models

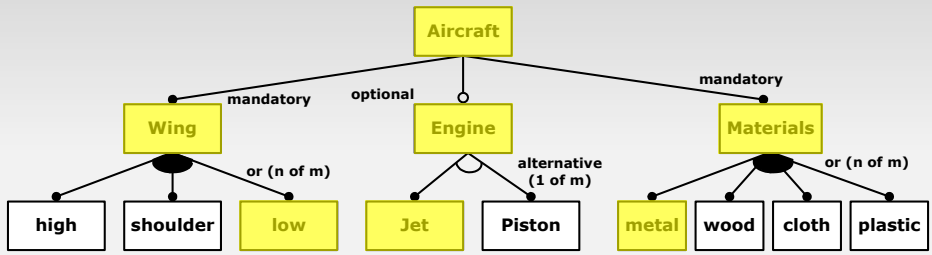


Configuration
Feature Models



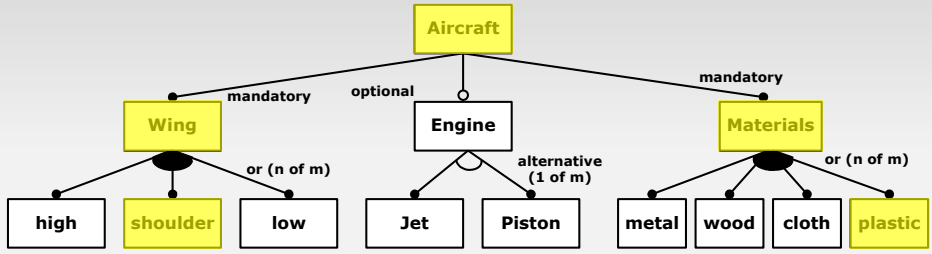
Robin DR-400
An aircraft with a low wing, piston engine and made of metal, wood and cloth

Configuration
Feature Models



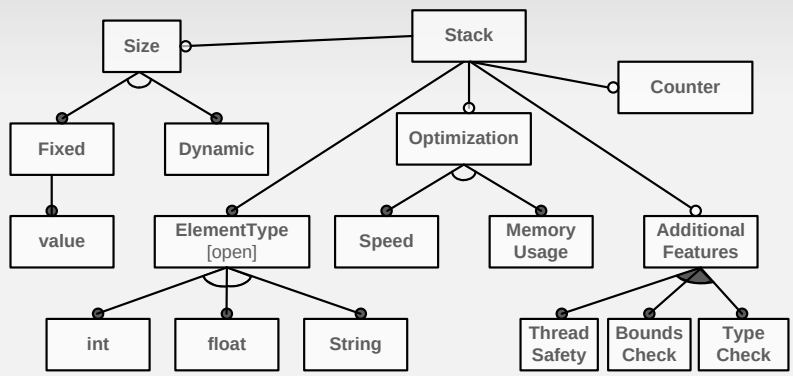
Airbus A 320
An aircraft with low wing, jet engine(s) and made of metal.

Configuration
Feature Models

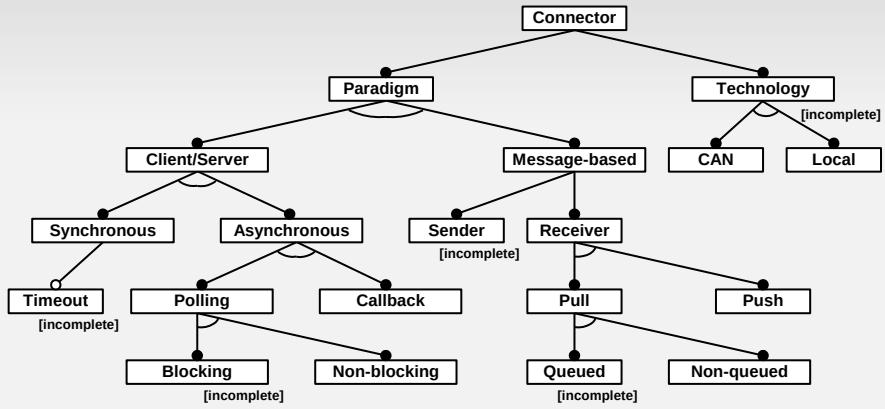


Schleicher ASW 27
An aircraft with shoulder wing, no engine and made of plastic

Configuration
Feature Models



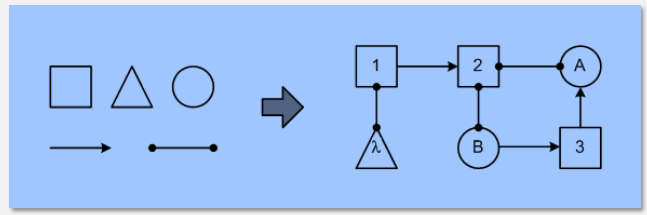
Configuration
Feature Models



4 Customization

Customization

… „real languages“
… instantiation
… connections



abstractions
~computing

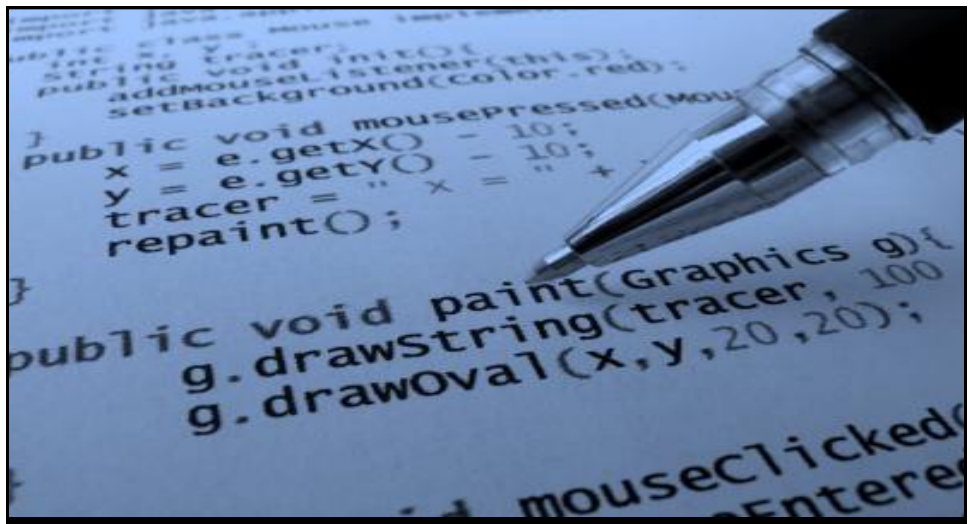
chips

abstractions
~computing

bits

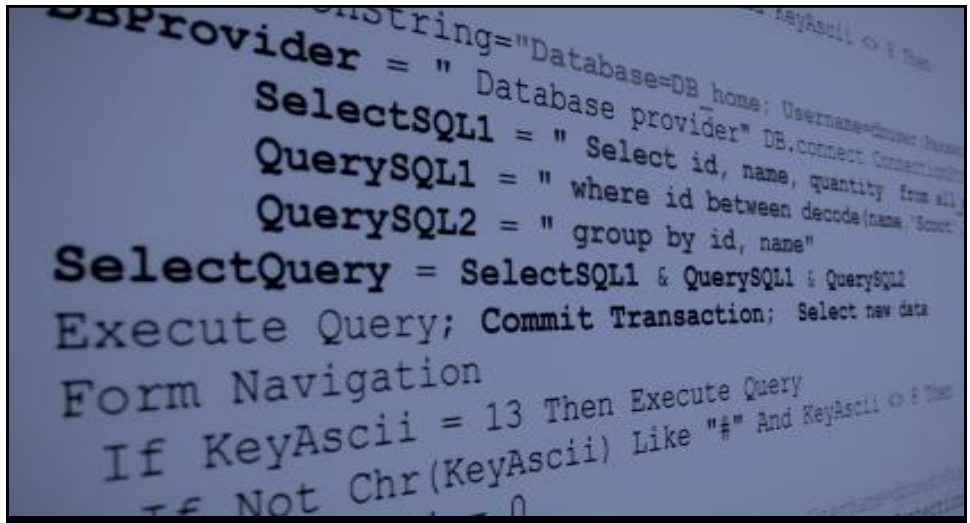
abstractions ~ computing

C



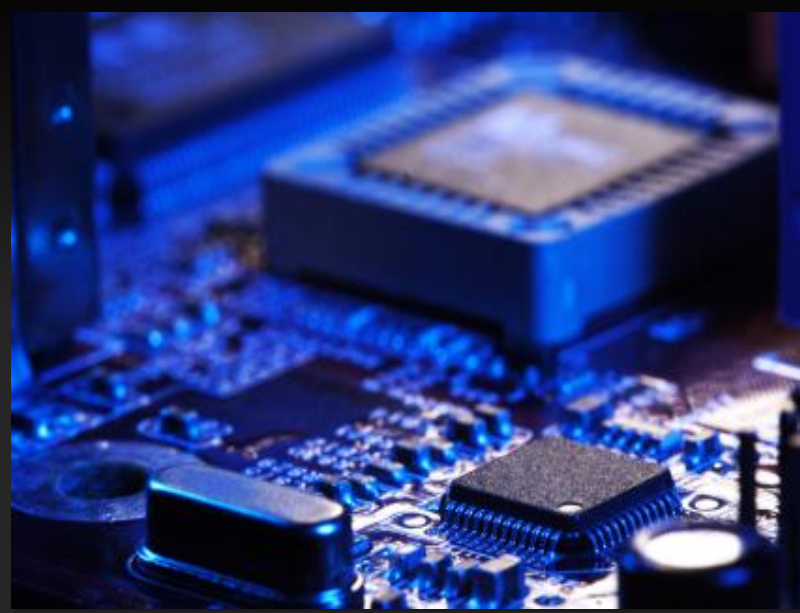
abstractions ~ computing?

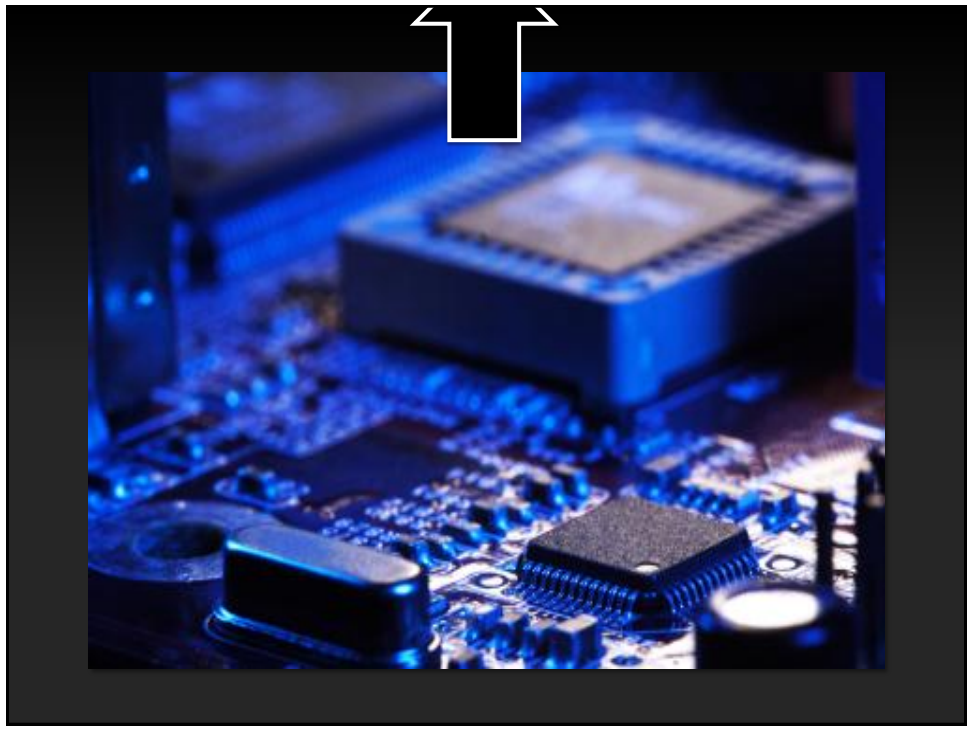
Java



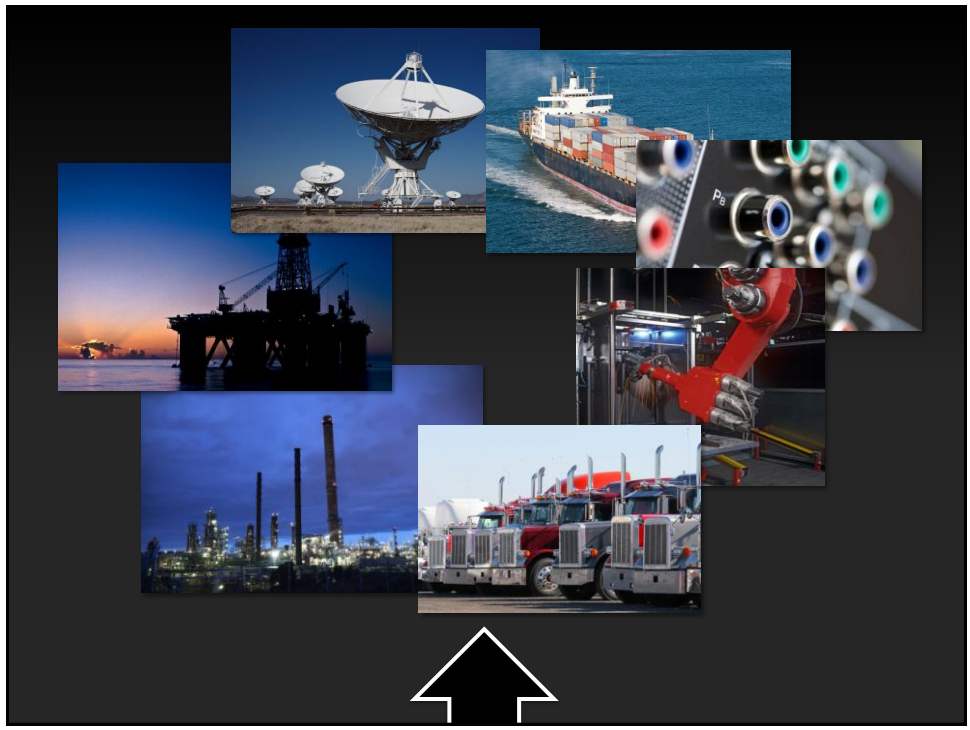
abstractions
~computing?

SQL





?



general purpose

domain specific

tailor made

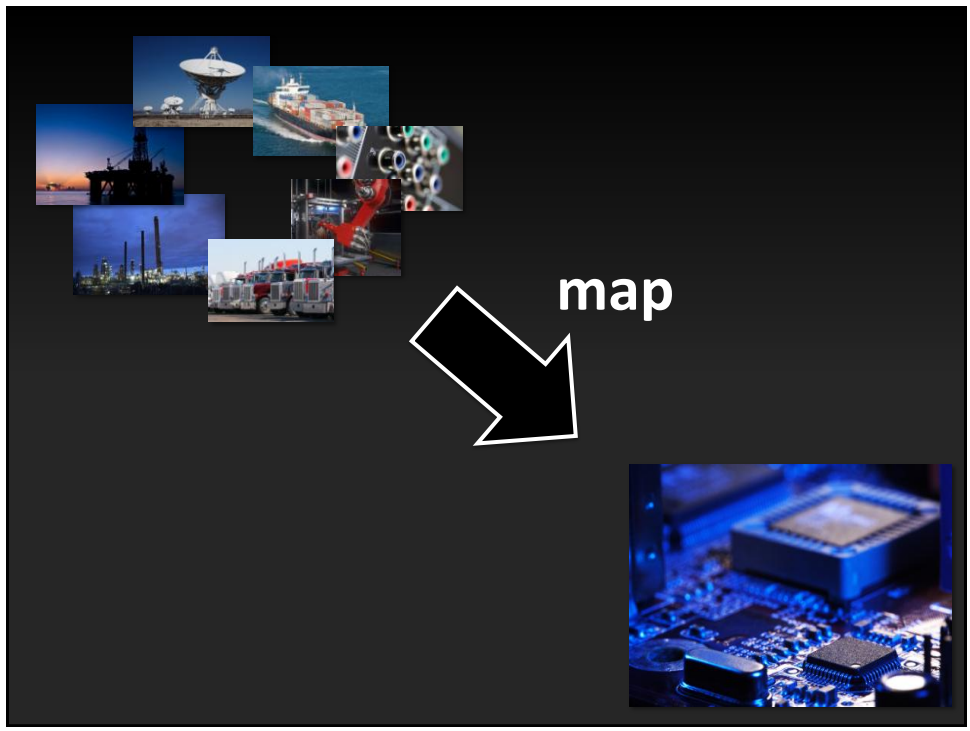
effective++

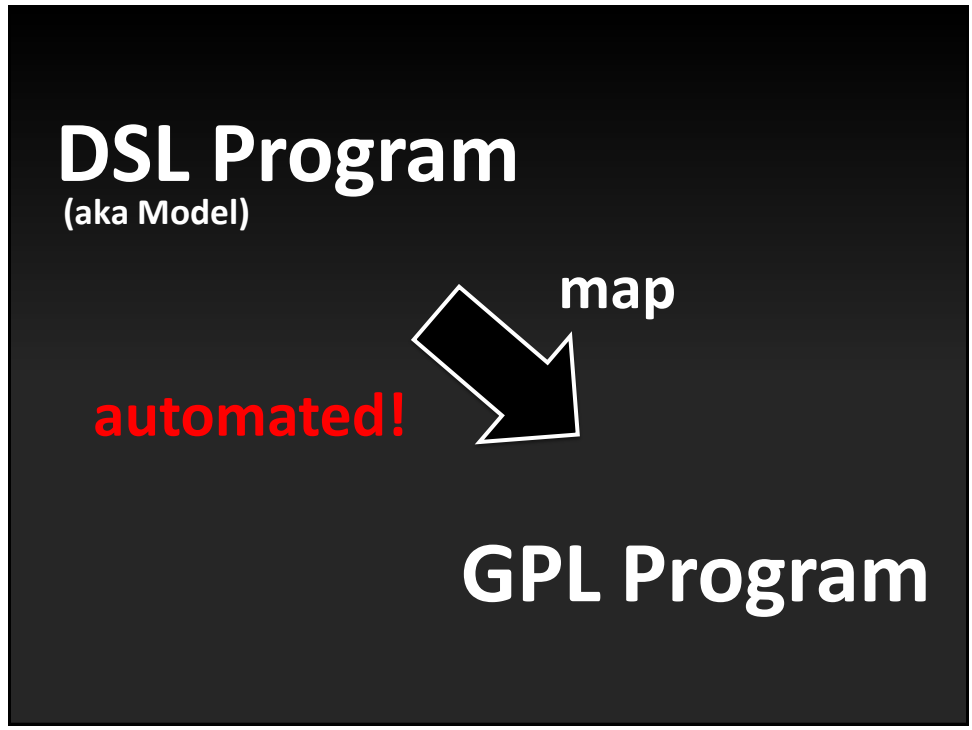
specialized, limited

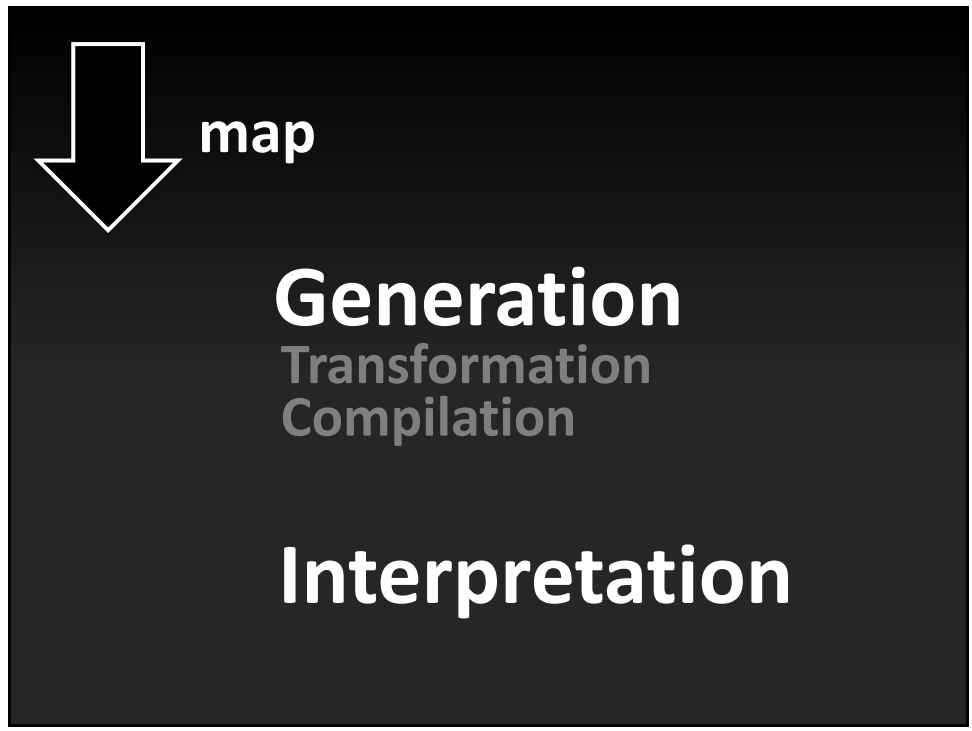
used by experts

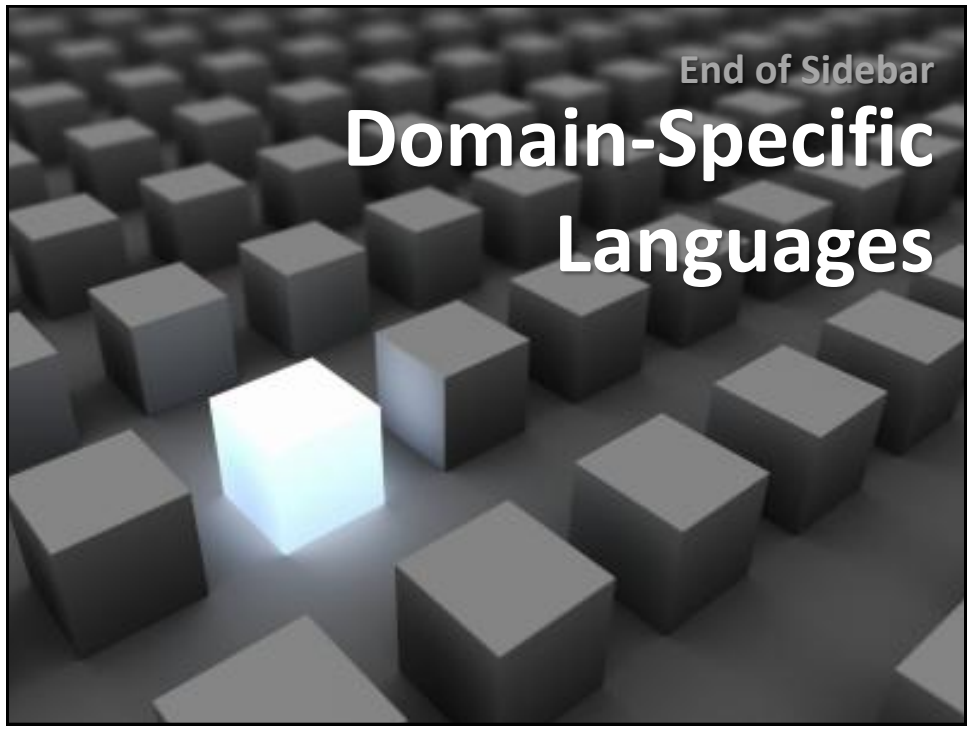
together with other specialized tools

execute?

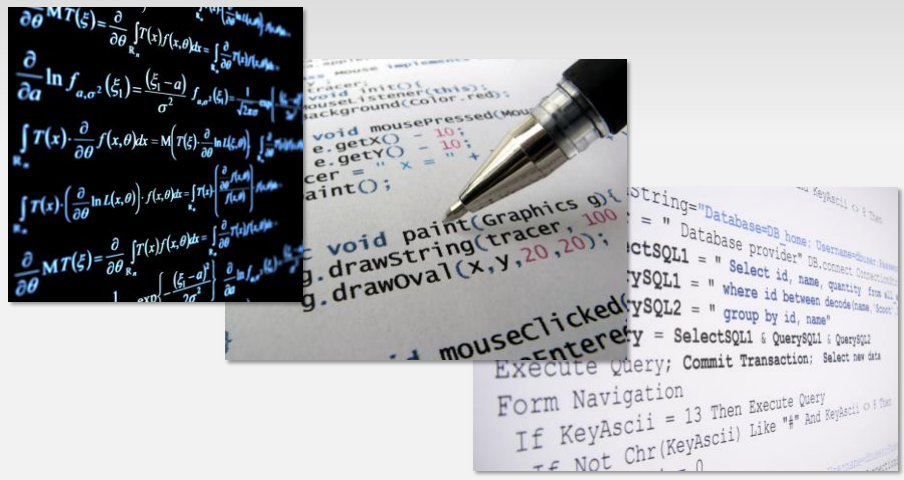

map


DSL Program
(aka Model)
map
automated!
GPL Program

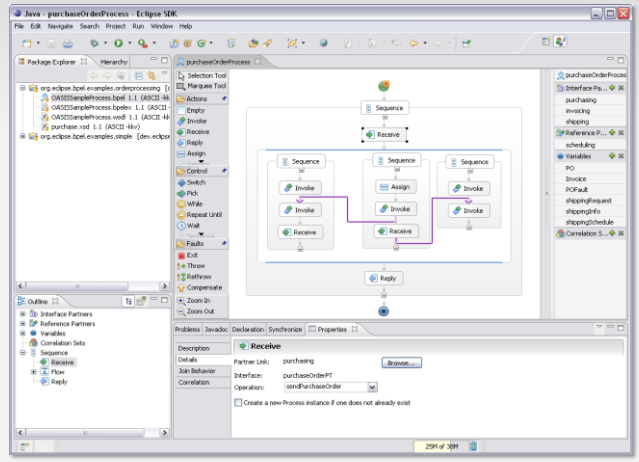

map
Generation
Transformation
Compilation
Interpretation



Customization
Languages



Customization
Languages

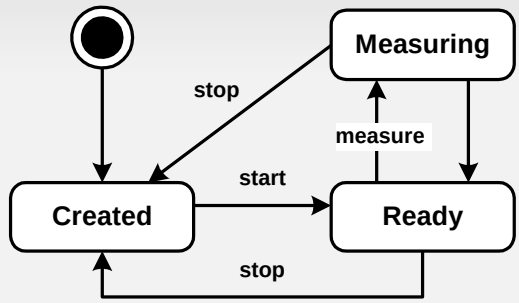


Customization
Languages

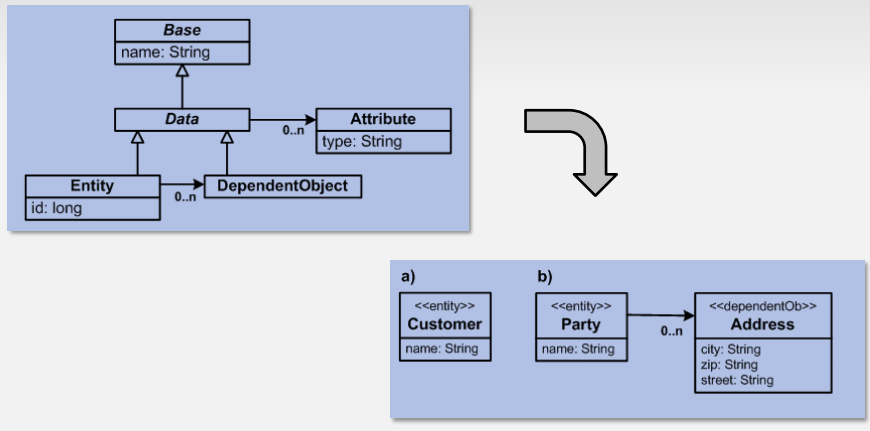
```
data {

    data Patient persistent {
        name: String
        age : int range 0..120
        firstName: String[20] regex "[a-z]*"
        birthDate: String
        insuranceName: String
        insuranceNumber: String
        -> adr: Adress
        -> tel: Telephone*
        -> stays: Stay*
    }
    data Adress persistent {

    data Telephone {
        countryCode: String
        cityCode: String
        localNumber: String
    }
    data Stay {
        beginDate: String
        -> stations: Station*
        endDate: String
    }
    data Station {
        date: String
        stationName: String
    }
}
```

```
records {

    record P001 {
        p: Patient
        1 -> p.name
        2 -> p.firstName
        3 -> p.birthDate


        record A001 {
            a: Adress
            1 -> a.street
            2 -> a.zip
            3 -> a.city
            p.adr = a
        }

        record V001 {
            1 -> p.insuranceName
            2 -> p.insuranceNumber
        }
        record T001 {
        record S001 {
    }
}
```

Customization
Languages



Customization
Languages



5 Binding Times for Customization

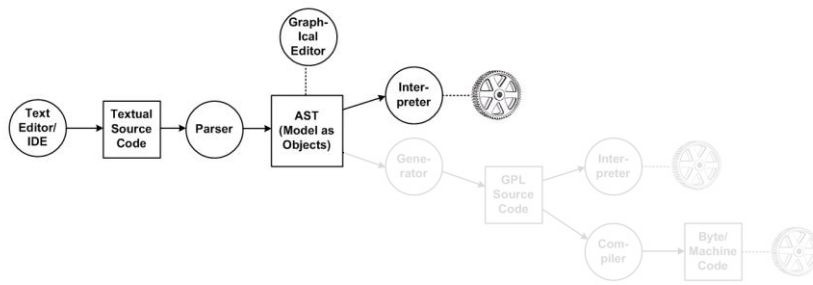
Binding Time

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

Binding Time
for DSLs?

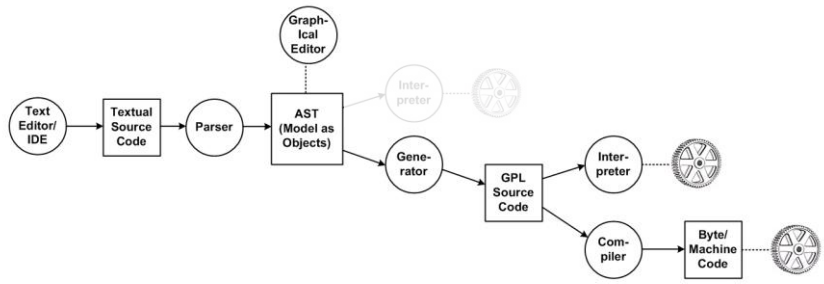
Interpretation vs. Generation

Binding Time
Interpretation



Interpretation vs. Generation

Binding Time
Generation



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

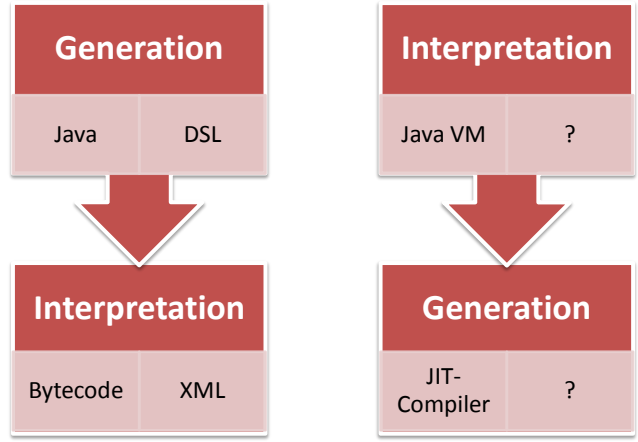
Interpretation

for platform indepenence
an interpreter might be

less porting effort

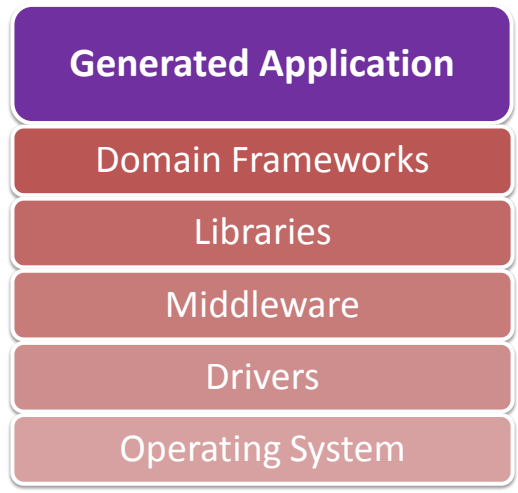
Interpretation vs. Generation

Combinations



Rich Platform

Platform



Rich Platform

Platform

Domain Frameworks

Libraries

Grown with the DSL!

Rich Platform

Platform
Extreme Case

Generated Application

populates

Domain Frameworks

6 Architecture DSLs

Architecture DSL

As you understand and develop your Platform Architecture…

Develop a language to express it!

Language resembles architectural concepts…

Express the applications with the language.

```
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

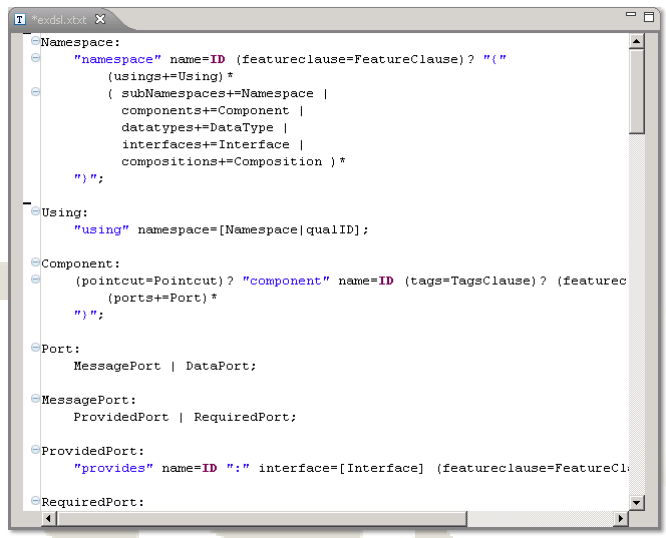
```
import "classpath:/twoworlds.compdsl"
import "classpath:/twoworlds.scenario"

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

TOOLING

eclipse modeling PROJECT

http://eclipse.org/modeling

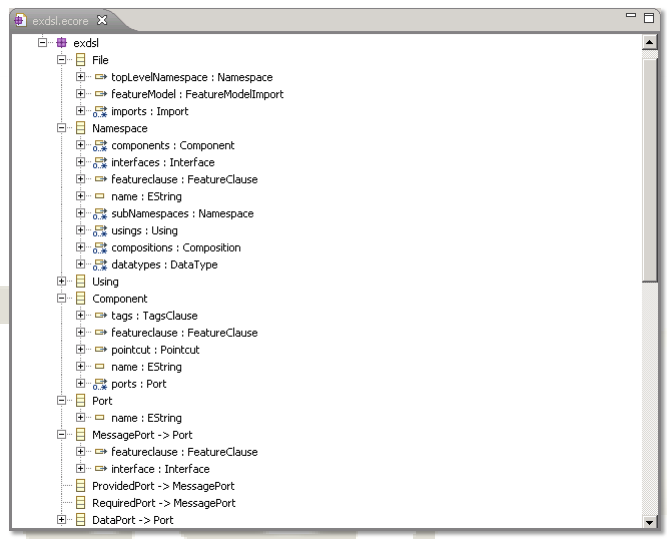
xtext Specify Grammar



xtext

Antlr Grammar and Parser is generated from this specification

xtext Gen. Metamodel



xtext Constraints

Checks.chk

```
import exdsl;

extension net::ample::adsl::exdsl::Extensions;
extension org::openarchitectureware::util::stdlib::io;

context Component ERROR "Qualified Name "+qualifiedName()+" must be unique"
   allComponents().select( c| c.qualifiedName() == qualifiedName() ).size

context DataType ERROR "Qualified Name "+qualifiedName()+" must be unique"
   allDataTypes().select( c| c.qualifiedName() == qualifiedName() ).size

context Namespace if !isEmpty() ERROR "Qualified Name "+qualifiedName()+"
   allNamespaces().select( c| c.qualifiedName() == qualifiedName() ).size

context emf::EObject if metaType.getProperty("name")!=null ERROR "name not
   metaType.getProperty("name").get(this) != "Unnamed";

context Interface ERROR "interface names must start with a capital I":
    name.startsWith("I");

context MessagePort ERROR "interface not defined. Missing a 'using'?: ":
    visibleInstancesOfType(this, Interface).contains(interface);

context Attribute ERROR "no type defined: "+type.name:
    visibleInstancesOfType(this, DataType).contains(type);

context DataPort ERROR "data not defined: "+type.name:
    visibleInstancesOfType(this, ComplexType).contains(type);
```

xtext Generated Editor

components.exdsl

```
namespace com {
    namespace airwizard  {

        using com.airwizard.domaintypes
        using com.airwizard.types

        namespace shared {
            struct aaa {
                x : FlightAIDI
                y : Flights
            }

            typedef String FlightID

            struct FlightStatus {
                f : FlightStatus
                id:
                eta:
            }

            interfac
                onew                                    tatus[]]
            }
```

FlightID -- com.airwizard.shared
FlightStatus -- com.airwizard.shared
Flights -- com.airwizard.shared
String -- com.airwizard.types
Time -- com.airwizard.domaintypes
aaa -- com.airwizard.shared
boolean -- com.airwizard.types
int -- com.airwizard.types

Task | Outlin

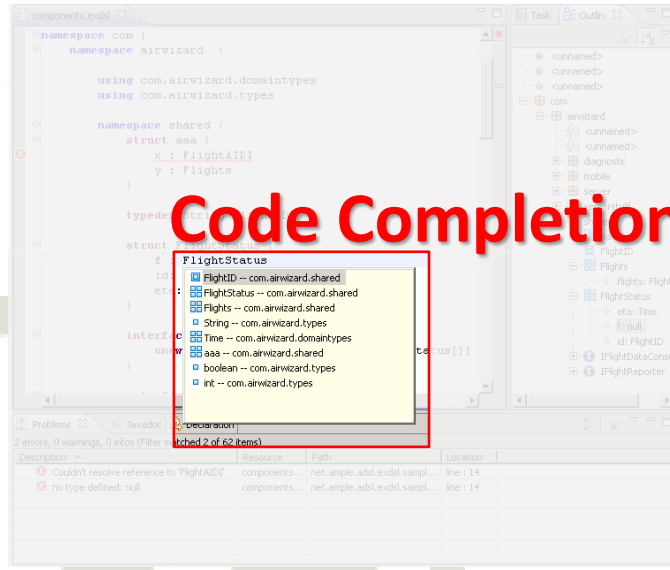
<unnamed>
<unnamed>
<unnamed>
com
airwizard
<unnamed>
<unnamed>
diagnostic
mobile
server
serverstuff
shared
aaa
FlightID
Flights
flights: Flight
FlightStatus
eta: Time
f: null
id: FlightID
IFlightDataConsu
IFlightReporter

Problems | Javadoc | Declaration

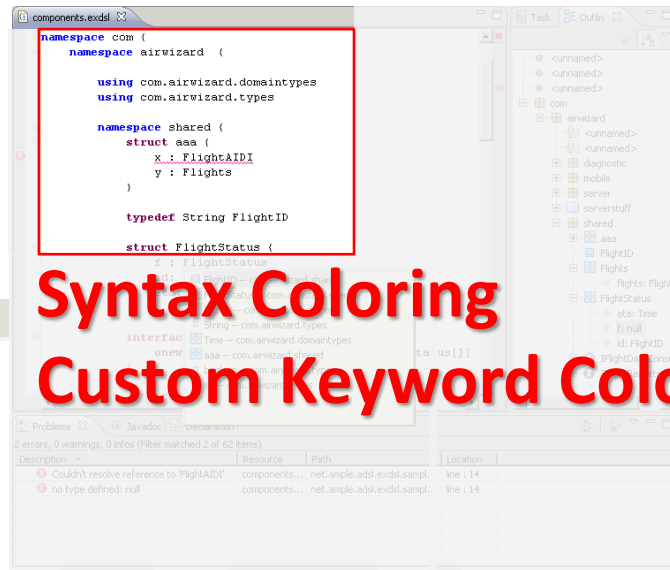
2 errors, 0 warnings, 0 infos (Filter matched 2 of 62 items)

Description	Resource	Path	Location
Couldn't resolve reference to 'FlightAIDI'	components...	net.ample.adsl.exdsl.sampl...	line : 14
no type defined: null	components...	net.ample.adsl.exdsl.sampl...	line : 14

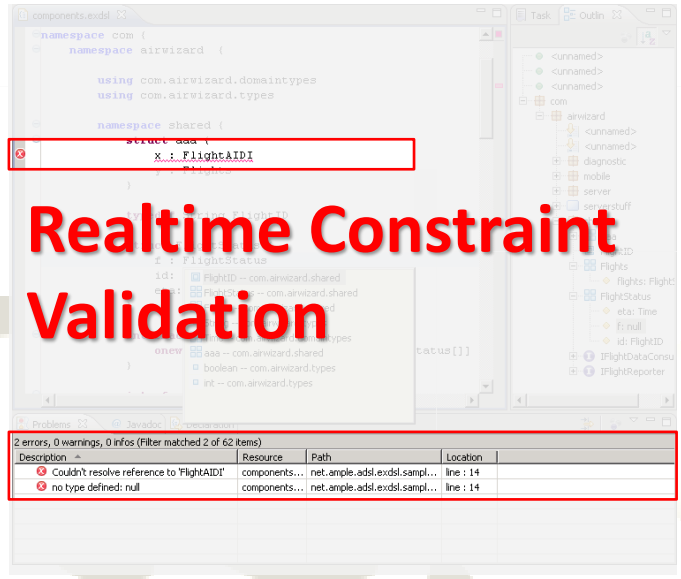
xtext Generated Editor



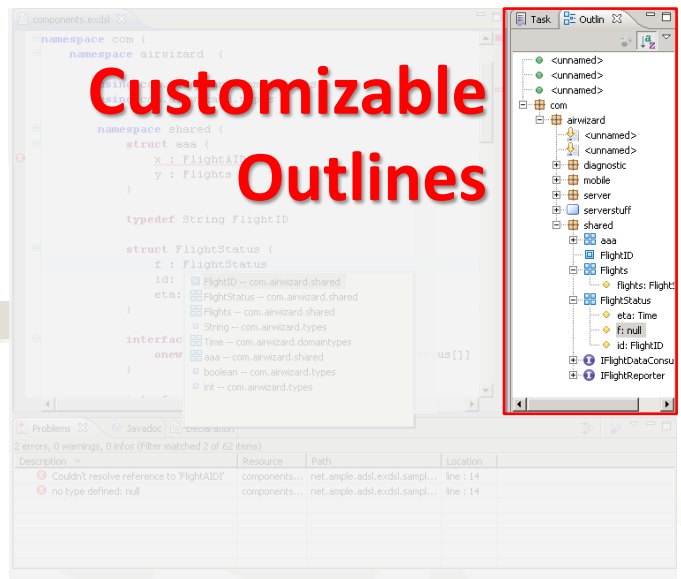
xtext Generated Editor



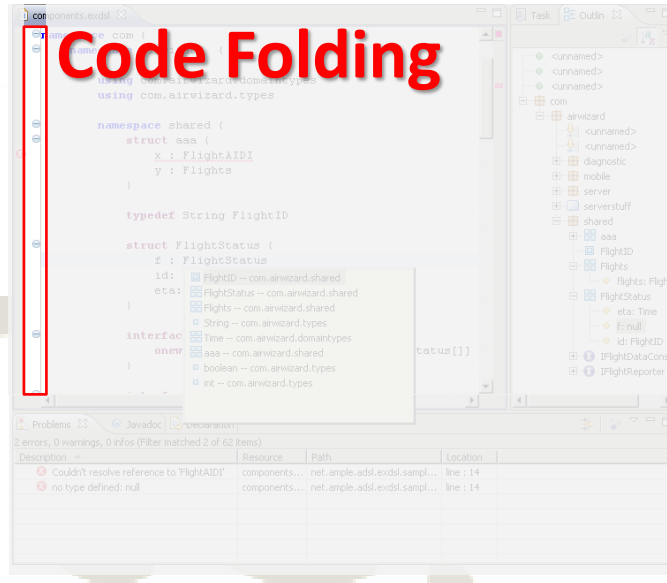
xtext Generated Editor



xtext Generated Editor

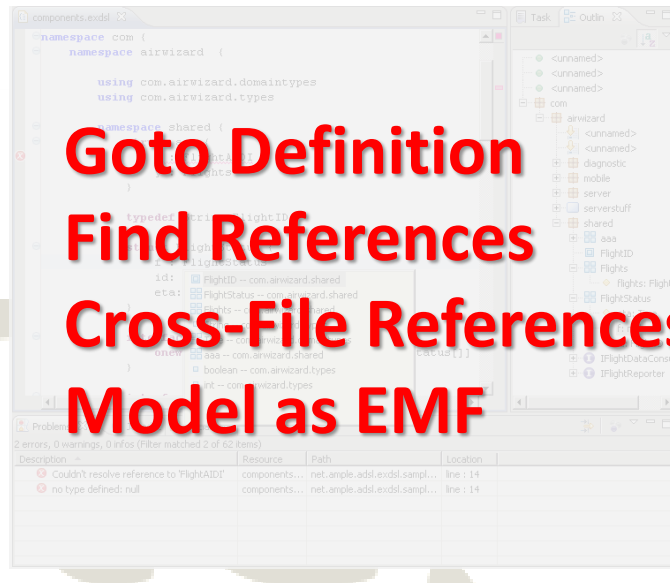


xtext Generated Editor



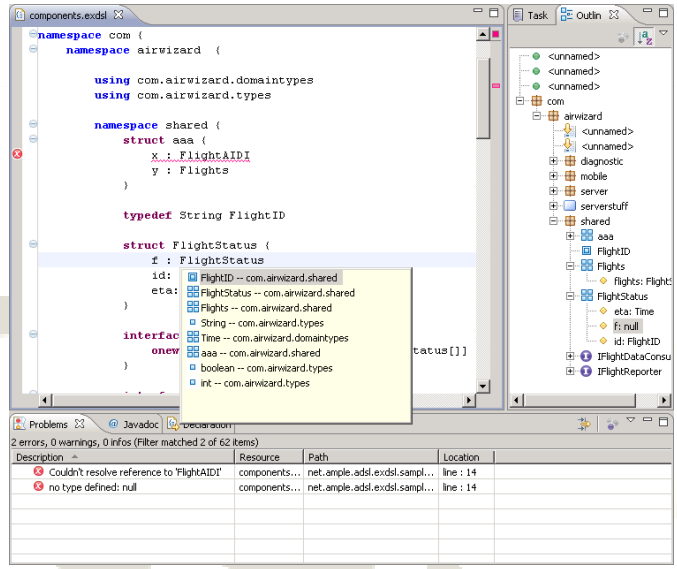
Code Folding

xtext Generated Editor



Goto Definition
Find References
Cross-File References
Model as EMF

xtext Generated Editor



TOOLING

JetBRAINS

Jetbrains‘ Meta Programming System

Language Extension Example

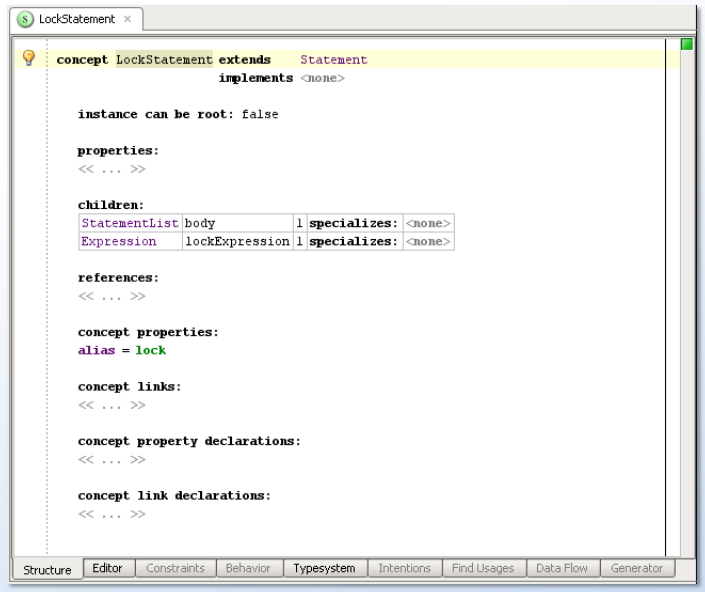
Old

```
ReadWriteLock l = …
l.readLock().lock();
try {
  //code
} finally {
  l.readLock().unlock();
}
```

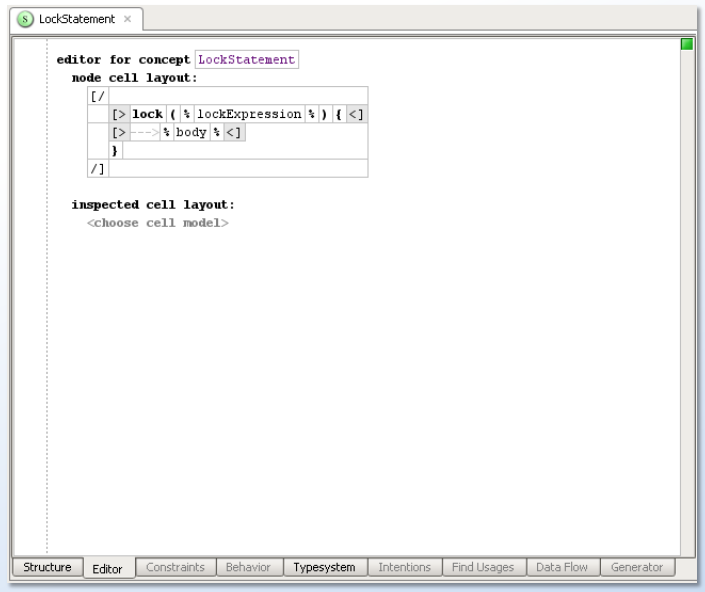
New

```
ReadWriteLock l = …
lock (l) {
  //code
}
```

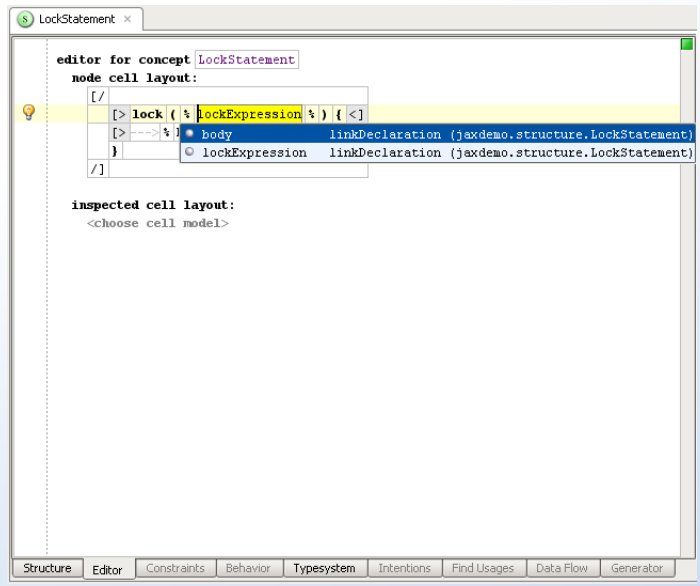
Structure ◆ Editor ◆ Typesystem ◆ Generator



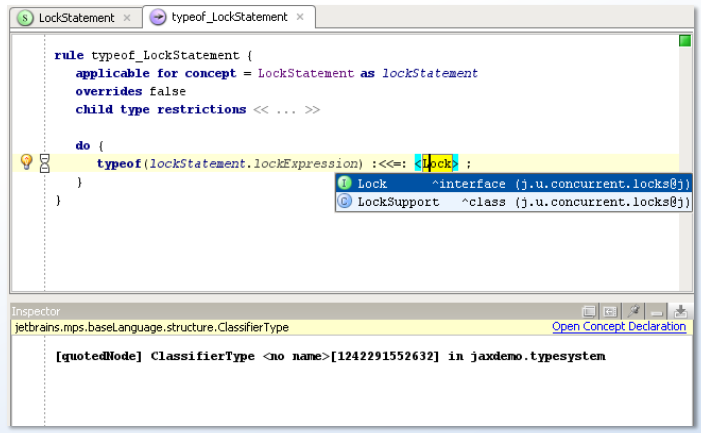
Structure ◆ Editor ◆ Typesystem ◆ Generator



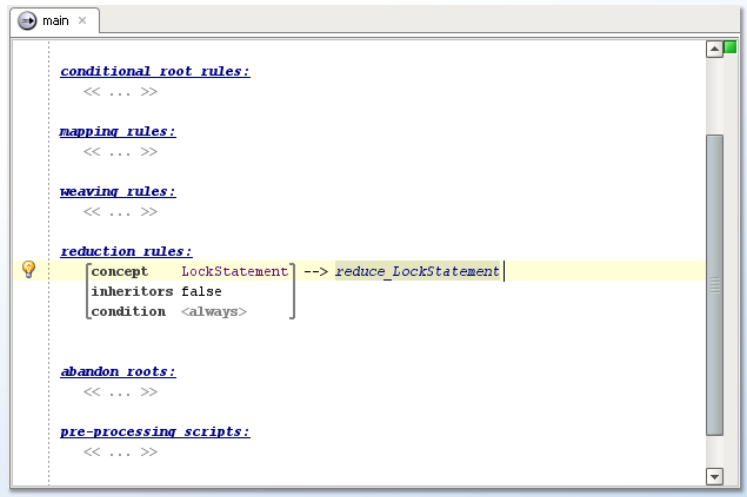
Structure ◆ Editor ◆ Typesystem ◆ Generator



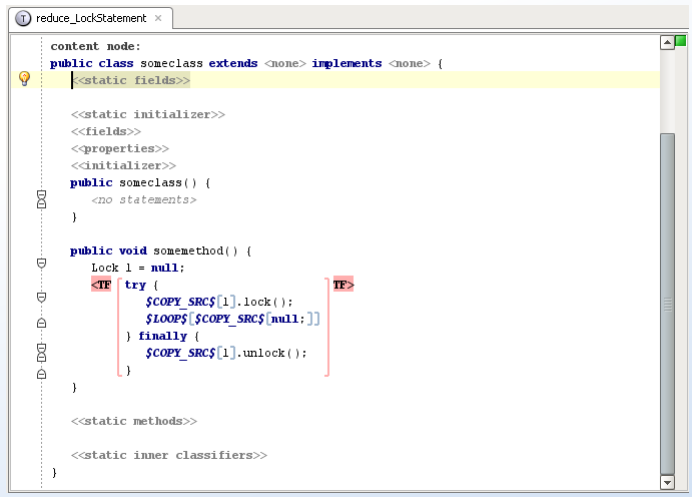
Structure ◆ Editor ◆ Typesystem ◆ Generator



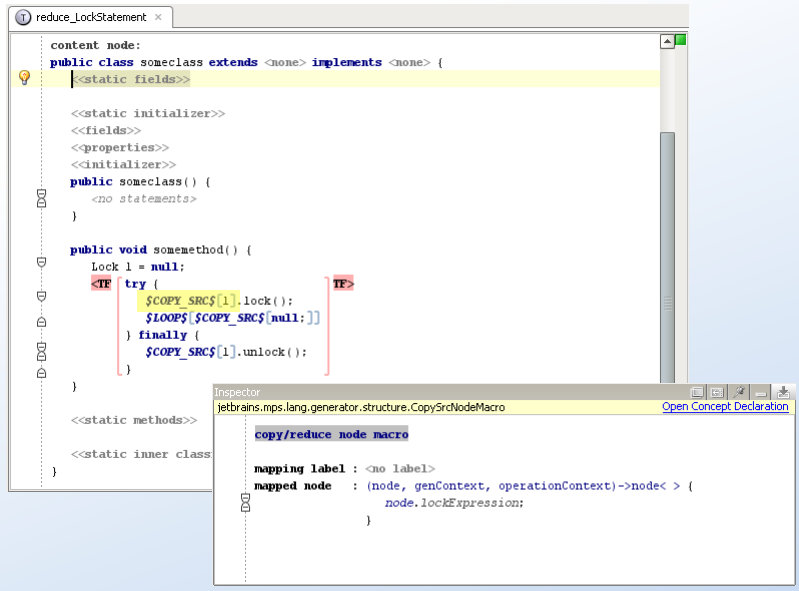
Structure ◆ Editor ◆ Typesystem ◆ Generator



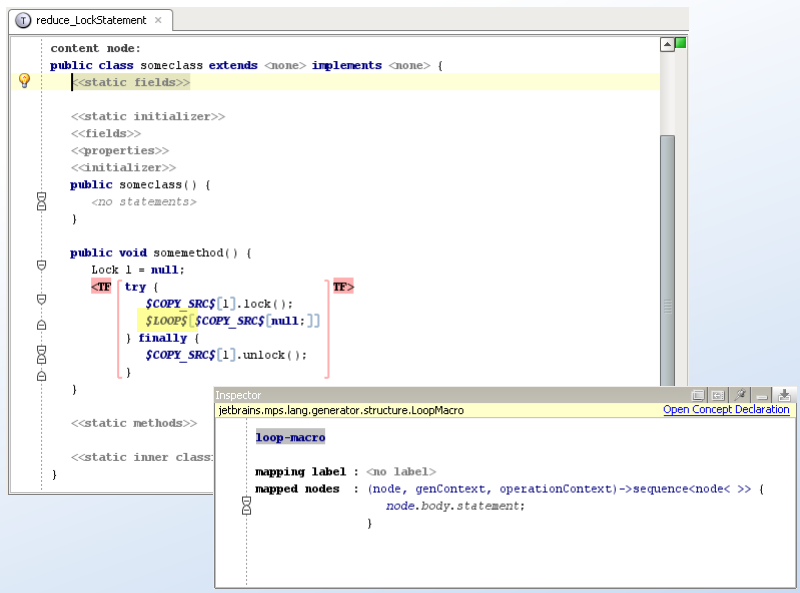
Structure ◆ Editor ◆ Typesystem ◆ Generator



Structure ◆ Editor ◆ Typesystem ◆ Generator

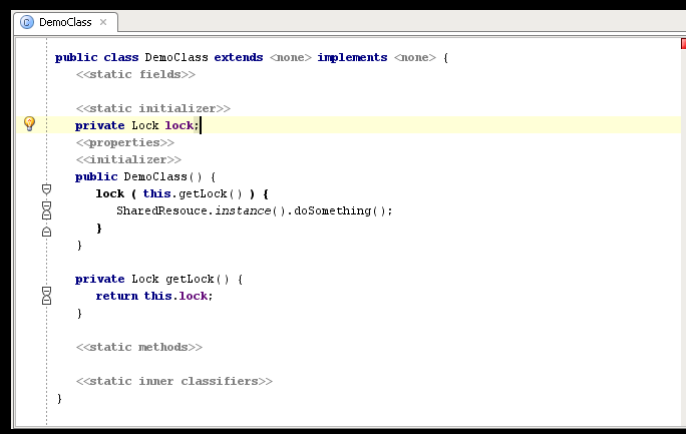


Structure ◆ Editor ◆ Typesystem ◆ Generator



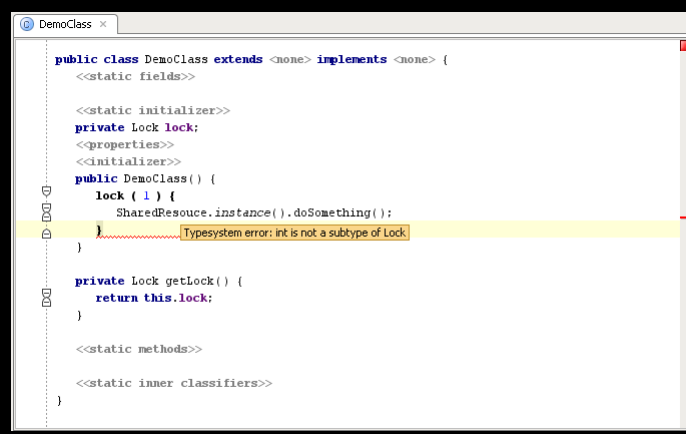
Language Extension Example

Result behaves like a native base language construct



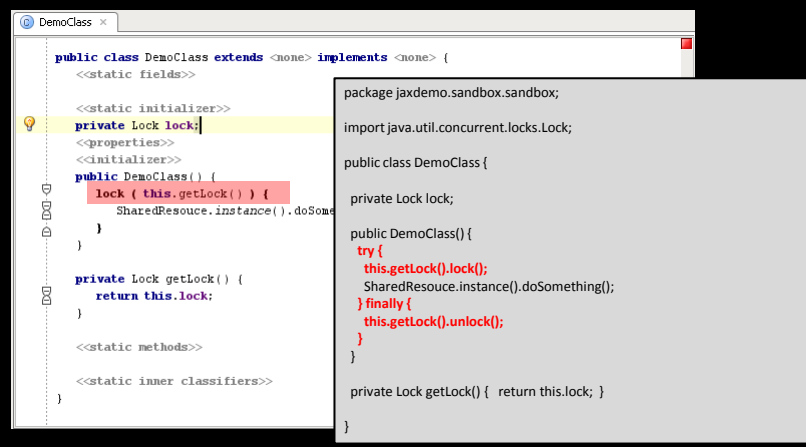
Language Extension Example

Result behaves like a native base language construct



Language Extension Example

Translated to regular Java code based on the generator



Example Languages

UI Language

```
component NewLanguageDialog

IDEDialog
  left: 100 top: 100 width: 600 height: 200
  title = New Language

  content pane:
  Grid
    Row
      [Label
        text : "Language Namespace:"]
    Row
      [TextField ( name )
        text : { this.languageNamespace }]
    Row
      [Label
        text : "Language Path:"]
    Row
      [PathField ( path )
        path : { this.languagePath }]

  buttons
  [text OK
  default true
  handler this.onOk()][text Cancel
  default false
  handler this.onCancel()]
```

Example Languages

HTML Templates

```
html template MessageHeader(Message message, boolean hideUser, boolean userLast)

is not refreshable: Only one root element allowed for refreshable template.

<< variables >>

rss links: << rss links >> << rss template uris>>

<< explicit js import >>

if (!(hideUser) && !(userLast)) {
  <span class: right-margin-span>
    [html include <visible>
      def[ ]: UserLink(user: message.getUser(), bold: true)]
  </span>
}
<span class: right-margin-span>
  [html include <visible>
    def[ ]: Date(date: message.created)]
</span>
if (null != message.updated) {
  <span class: right-margin-span>
    updated:
    [html include <visible>
      def[ ]: Date(date: message.updated)]
  </span>
}
```

Example Languages

Persistent Classes

```
public persistent class Forum extends <none> implements <none> {

  features
    save changes history if: false
    save changes history callback: no callback
    version mismatch resolution: default
    invariant: no invariant

  << static fields >>

  public simple string name opts;
  public unordered child FThread[0..n] threads opts;
  public unordered bidirectional association User[0..n] subscribers onDelete(clear);
  public unordered bidirectional association User[0..n] watchers onDelete(clear);

  public Forum(string name, User creator) {
    this.name = name;
    this.watchers.add(creator);
  }


  << destructor >>
```

7 Business Domain DSLs

Domain Users Programming?

Precision

!=

Programming

Domain Users

Programmers

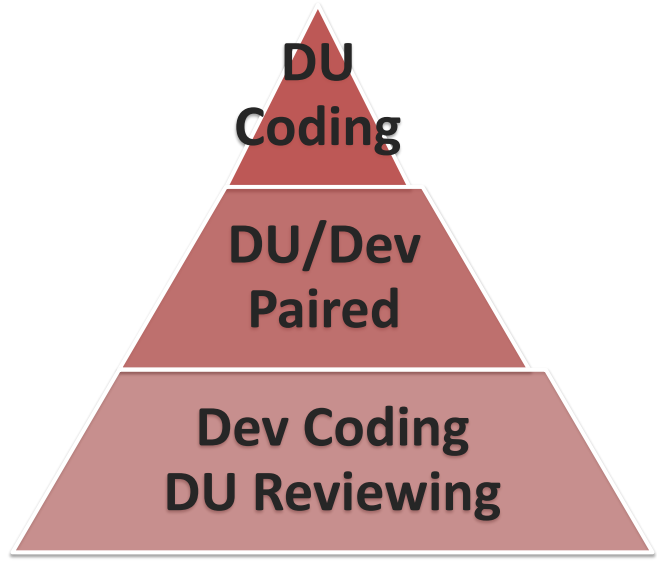
Programming = Precision + X

Domain Users Programming?

Precision

… Scientists
… Insurance Mathematicians
… Logisticians
… Medical Doctors

Domain Users Programming?



Domain Users Programming?

If Domain Users
don‘t get it
it might hint at a
problem with
the language!
… or the documentation!

Domain Users vs. Experts

Creating the Language

vs.

Using the Language

Domain Users vs. Experts

Creating: Domain Expert

Using: Domain User

Domain Users vs. Experts

Creating: Domain Expert

… senior
… complete
… big picture
… deep
… precise, formal
… guru

Domain Users vs. Experts

Using: Domain User

… not senior
… narrower
… shallow

Domain Users vs. Experts

Can Domain Users **understand** what the Experts put into the language?

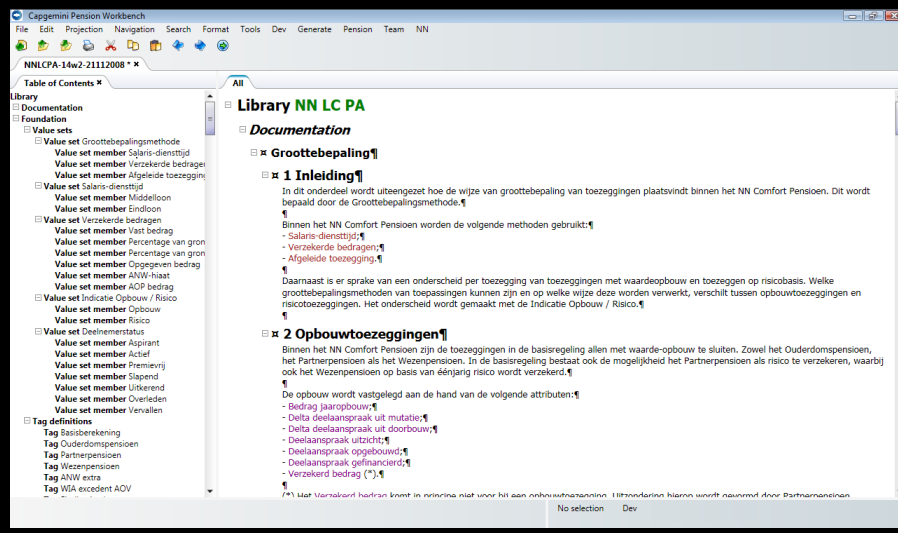
Verify early and often!

TOOLING

INTENTIONAL® SOFTWARE

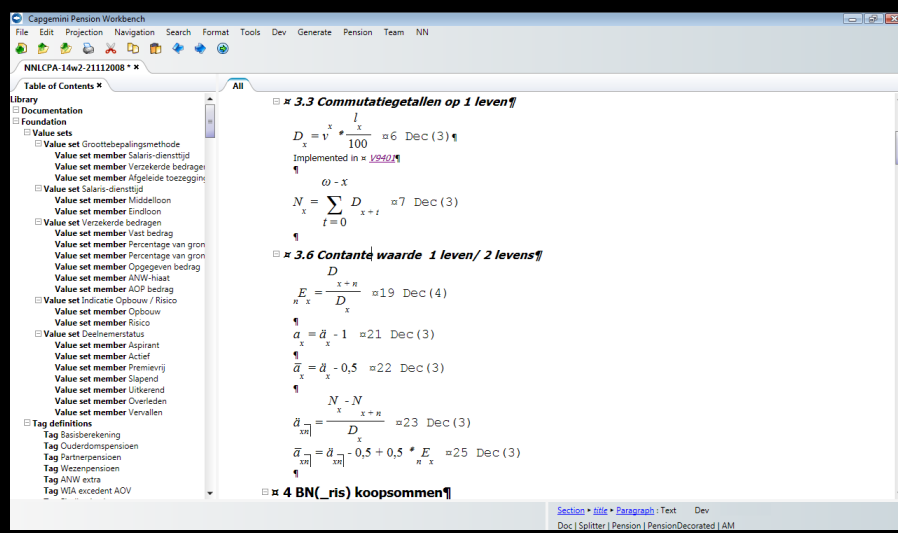
Intentional Software
Domain Workbench

Pension Workbench Example



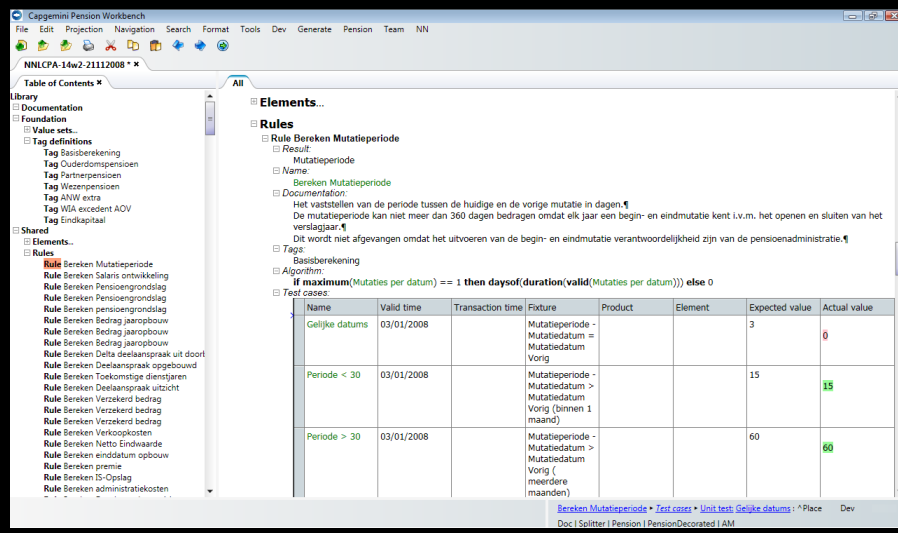
Text Editing Domain

Pension Workbench Example



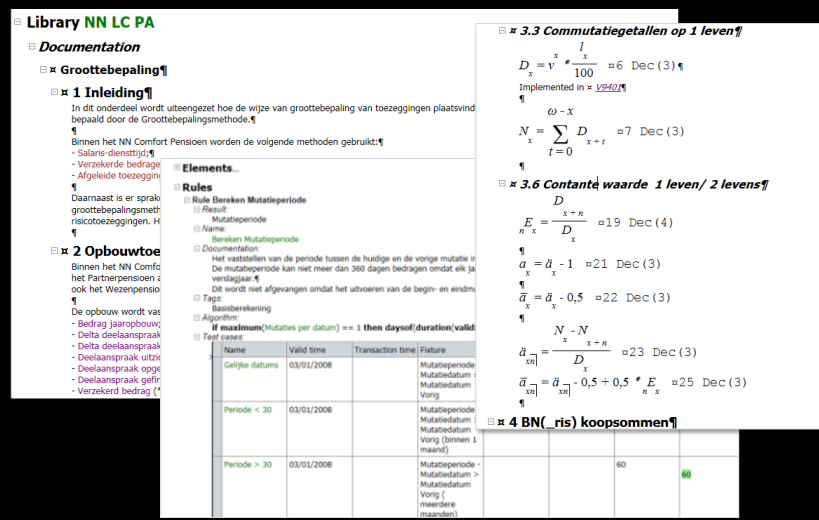
Insurance Mathematics Domain

Pension Workbench Example



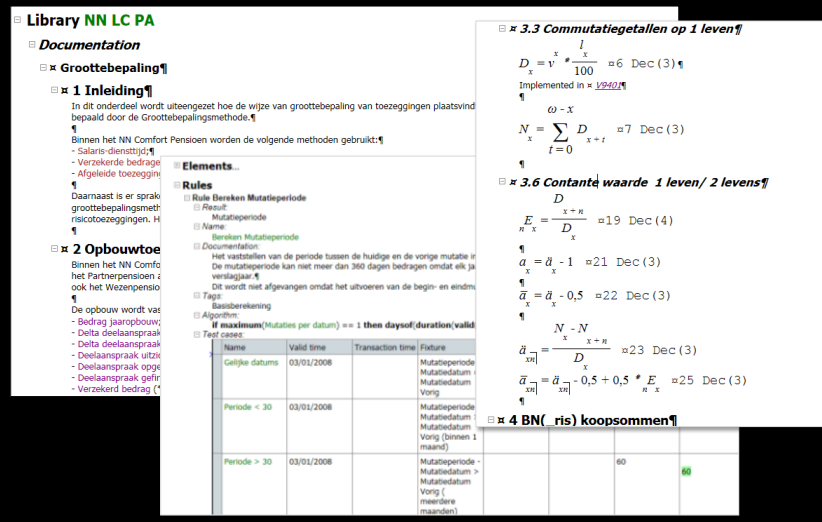
Pension Contract Rules Domain

Pension Workbench Example

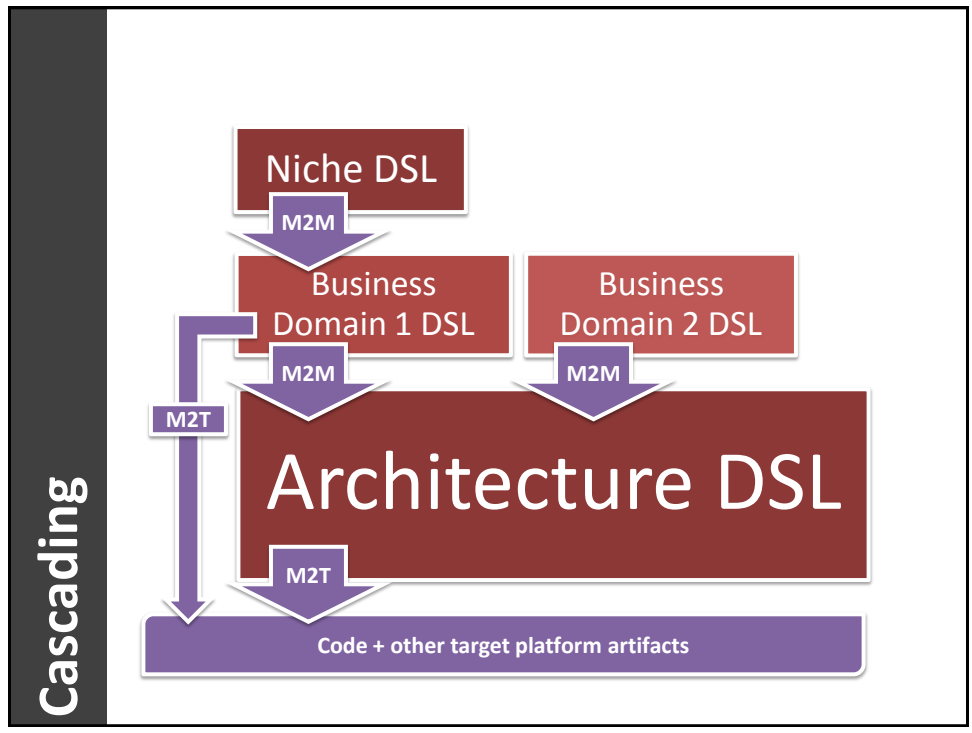


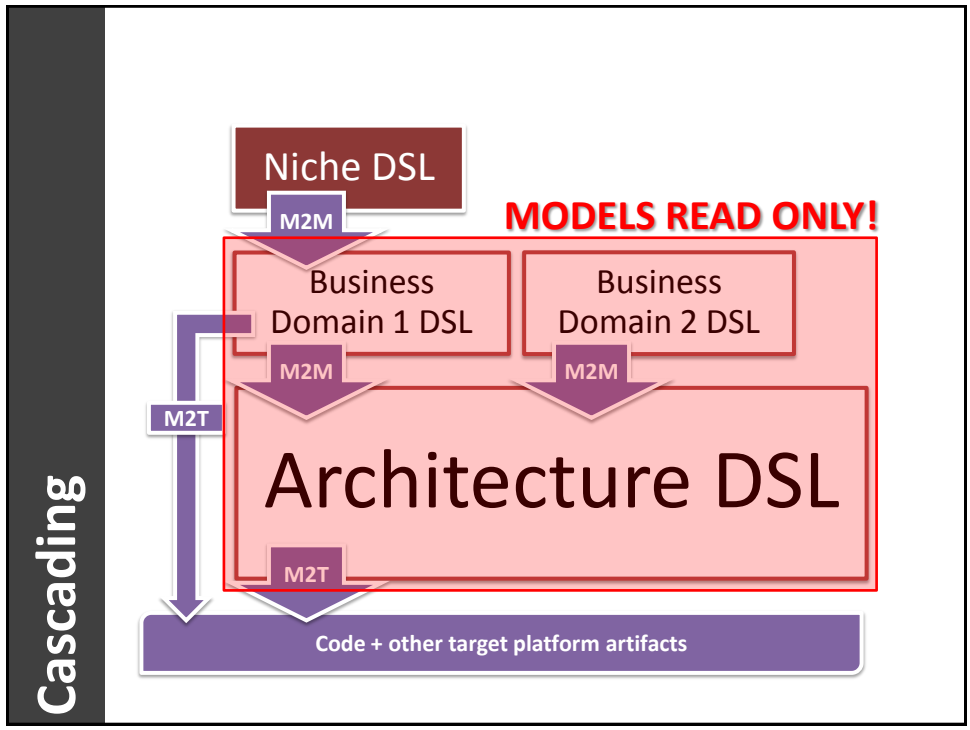
All in one Document

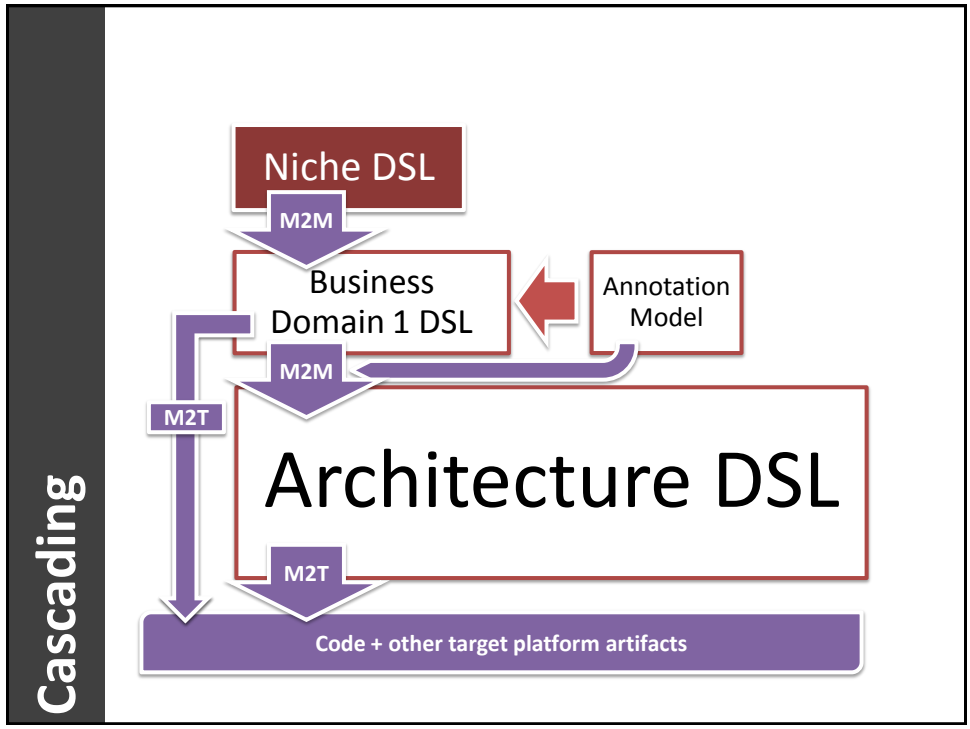
Pension Workbench Example



Symbolically integrated




Cascading
Niche DSL
M2M
MODELS READ ONLY!
Business Domain 1 DSL
Business Domain 2 DSL
M2M
M2M
M2T
Architecture DSL
M2T
Code + other target platform artifacts


Cascading
Niche DSL
M2M
Business Domain 1 DSL
Annotation Model
M2M
M2T
Architecture DSL
M2T
Code + other target platform artifacts

8 Configuration and Customization

8 Configuration and Customization

In-Language
Varying models
DSLs as Parameters

Reuse and Variations

The language provides

explicit

support!

(we can learn that from OO)

Reuse and Variations

Specialization

overriding, overwriting

Leaving Holes

for variant to fill in

Inject Stuff

several places at a time?

Reuse and Variations

Specialization

Inheritance

Leaving Holes

Template Method Pattern

Inject Stuff

Aspect Orientation

8

Configuration and Customization

In-Language

Varying models

DSLs as Parameters

Two Levels

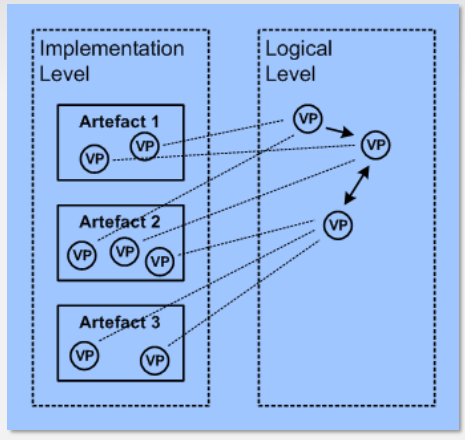
~ problem space
vs. software space

Problem Space:
Configuration

Software Space:
Customization

Two Levels

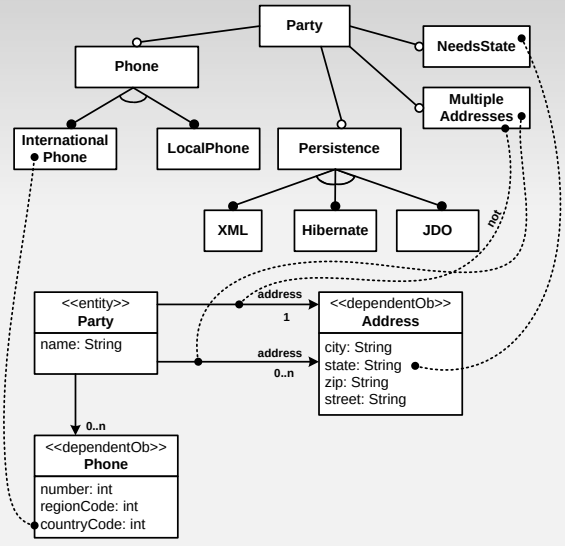
~ problem space
vs. software space



Two Levels Removal

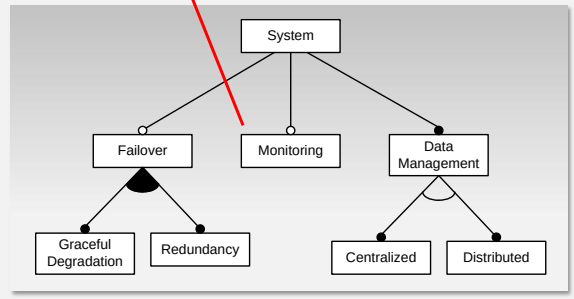
```
#if defined (ACE_HAS_TLI)
  static ssize_t t_snd_n (
         ACE_HANDLE handle,
         const void *buf,
         size_t len,
         int flags,
         ACE_Time_Value *timeout = 0,
         size_t *bytes_transferred = 0);
#endif /* ACE_HAS_TLI */
```

Two Levels Removal



Two Levels Removal

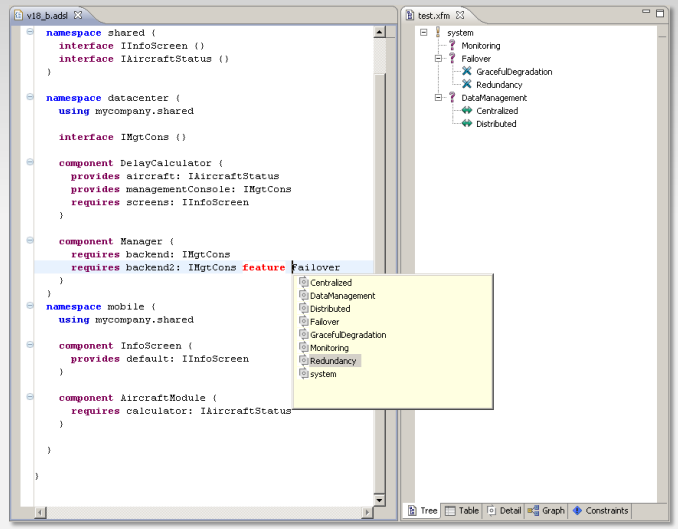
```
component DelayCalculator {
  provides default: IDelayCalculator
  requires screens[0..n]: IInfoScreen
  provides mon: IMonitoring feature monitoring
}
```



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
  }
}
```

Looks just like "normal" MPS Java code with state machine extensions

```
public statemachine class TrafficLights extends <none> implements <none> {
  <<static fields>>

  <<static initializer>>
  event go (  ) ;
  event stop (  ) ;
  state stop {
    call this.setRed(true)
    go --> transitioning
    go --> go
    call this.setRed(false)
  }
  state transitioning {
    call this.setYellow(true)
    after ( 3 ) --> go
    call this.setYellow(false)
  }
  state go {
    call this.setGreen(true)
    stop --> stop
    call this.setGreen(false)
  }
  private boolean green;
  private boolean red;
  private boolean yellow;
  <<properties>>
  <<initializer>>
  public TrafficLights() {
    <no statements>
  }

  protected void setRed(boolean b) {
    this.red = b;
  }

  protected void setGreen(boolean b) {
```

But contains "Feature Markup" that can be shown optionally

Feature Expressions are "real code", not comments

```
public statemachine class TrafficLights extends <none> implements <none> {
  <<static fields>>

  <<static initializer>>
  event go (  ) ;
  event stop (  ) ;
  state stop {
    call this.setRed(true)
    { car } go --> transitioning
    { pedestrian } go --> go
    call this.setRed(false)
  }
  { car } state transitioning {
    call this.setYellow(true)
    after ( 3 ) --> go
    call this.setYellow(false)
  }
  state go {
    call this.setGreen(true)
    stop --> stop
    call this.setGreen(false)
  }
  private boolean green;
  private boolean red;
  { car } private boolean yellow;
  <<properties>>
  <<initializer>>
  public TrafficLights() {
    <no statements>
  }

  protected void setRed(boolean b) {
    this.red = b;
  }

  protected void setGreen(boolean b) {
```

This is the code for the **pedestrian** configuration

```
public statemachine class TrafficLights extends <none> implements <none> {
  <<static fields>>

  <<static initializer>>
  event go (  ) ;
  event stop (  ) ;
  state stop {
    call this.setRed(true)
    go --> go
    call this.setRed(false)
  }
  state go {
    call this.setGreen(true)
    stop --> stop
    call this.setGreen(false)
  }
  private boolean green;
  private boolean red;
  <<properties>>
  <<initializer>>
  public TrafficLights() {
    <no statements>
  }

  protected void setRed(boolean b) {
    this.red = b;
  }

  protected void setGreen(boolean b) {
    this.green = b;
    for (int i = 0; i <= 10; <no iteration>) {
      int j = 7;
      j++;
    }
  }
```

This is the code for the **car** configuration

```
pu TrafficLights\TrafficLights s TrafficLights extends <none> implements <none> {
  <<static fields>>

  <<static initializer>>
  event go (  ) ;
  event stop (  ) ;
  state stop {
    call this.setRed(true)
    go --> transitioning
    call this.setRed(false)
  }
  state transitioning {
    call this.setYellow(true)
    after ( 3 ) --> go
    call this.setYellow(false)
  }
  state go {
    call this.setGreen(true)
    stop --> stop
    call this.setGreen(false)
  }
  private boolean green;
  private boolean red;
  private boolean yellow;
  <<properties>>
  <<initializer>>
  public TrafficLights() {
    <no statements>
  }

  protected void setRed(boolean b) {
    this.red = b;
  }

  protected void setGreen(boolean b) {
    this.green = b;
```

8 Configuration and Customization

In-Language Varying models

DSLs as Parameters

Feature Model w/ Paramters

… parameters with simple types

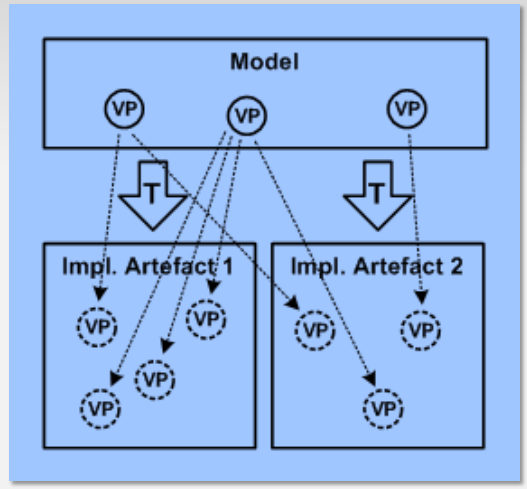
… complex types: DSL/metamodel

Model-Based Implementation

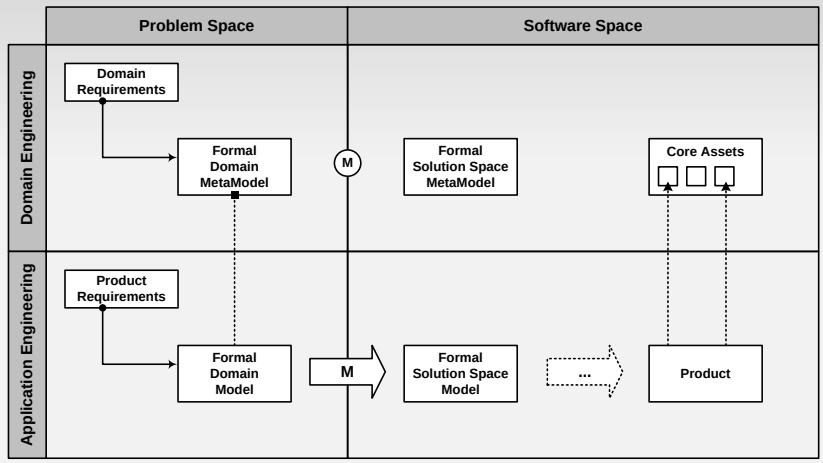
… customization in problem space

… Problem-Space DSL

Model-Based Implementation



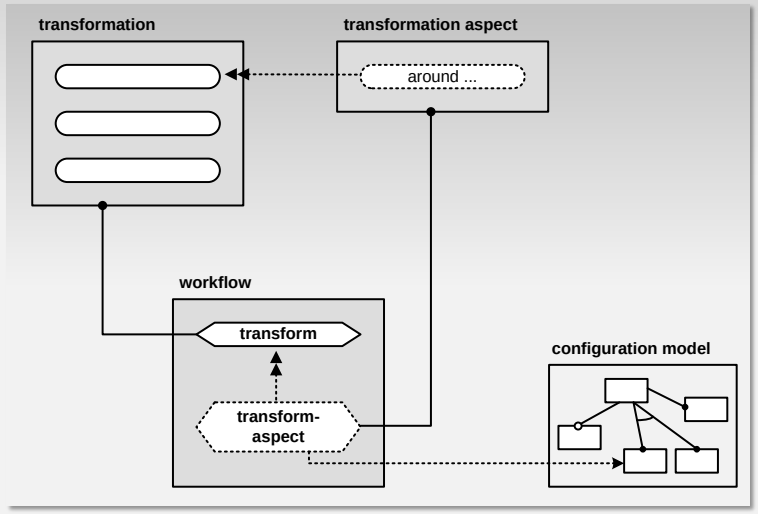
Model-Based Implementation



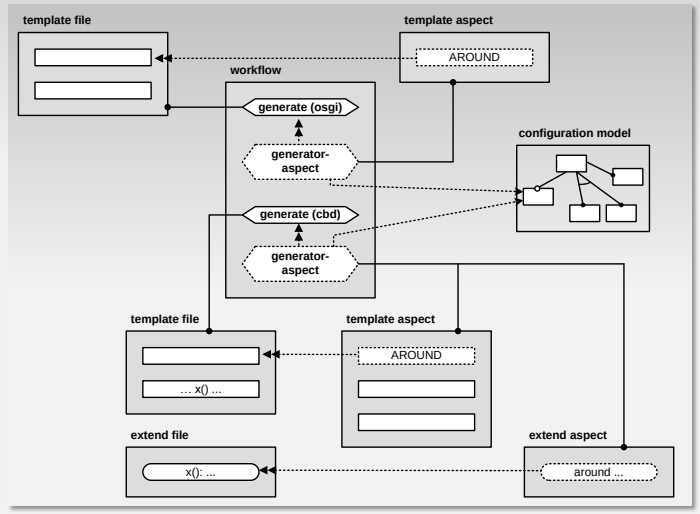
Transformation
Variability

```
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



⑩ Summing Up

THE END.

Markus Voelter
Indepenent/itemis
voelter@acm.org
http://www.voelter.de

itemis