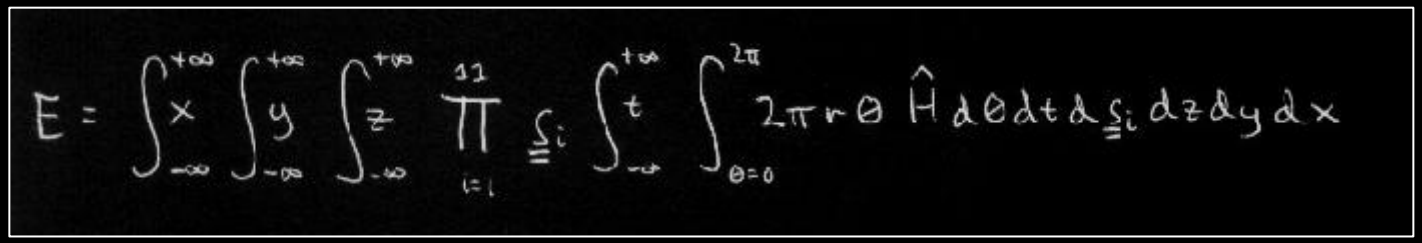
Models, DSLs
and Code Generation



Markus Voelter
independent/itemis
voelter@acm.org

www.voelter.de
voelterblog.blogspot.de
@markusvoelter
+Markus Voelter

Models, Languages, DSLs

Model

A schematic description of a system, theory, or phenomenon that accounts for its known or inferred properties and may be used for further study of its characteristics

www.answers.com/topic/model

Model

an abstraction or simplification of reality

ecosurvey.gmu.edu/glossary.htm

Model

which ones?

an abstraction or simplification of reality

what should we leave out?

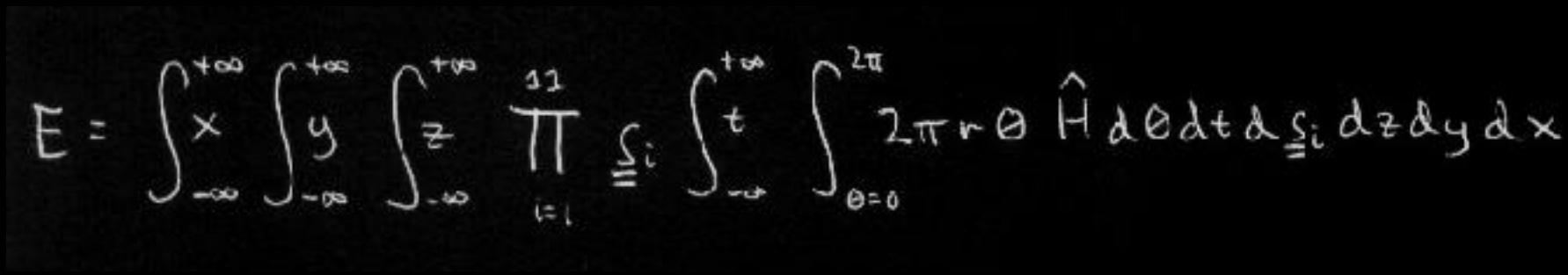
ecosurvey.gmu.edu/glossary.htm

Model Purpose

… code generation
… analysis and checking
… platform independence
… stakeholder integration
…

drives design of language!

A Domain Specific Language is a **focussed**, **processable language** for describing software in a specific **domain**. The **abstractions** and **notations** used are suitable for the **stakeholders** who use the language.



This is a DSL!

```
namespace com.mycompany {
  namespace datacenter {
    component DelayCalculator {
      provides aircraft: IAircraftStatus
      provides console: IManagementConsole
      requires screens[0..n]: IInfoScreen
    }
    component Manager {
      requires backend[1]: IManagementConsole
    }
    public interface IInfoScreen {
      message expectedAircraftArrivalUpdate
              (id: ID, time: Time)
      message flightCancelled(flightID: ID)
    }
    public interface IAircraftStatus …
    public interface IManagementConsole …
  }
}
```

Example

Components

```
namespace com.mycompany.test {
  system testSystem {
    instance dc: DelayCalculator
    instance screen1: InfoScreen
    instance screen2: InfoScreen
    connect dc.screens to
      (screen1.default, screen2.default)
  }
}
```

Example

Components

```
appliance KIR {

    compressor compartment cc {
        static compressor c1
        fan ccfan
    }

    ambient tempsensor at

    cooling compartment RC {
        light rclight
        superCoolingMode
        door rcdoor
        fan rcfan
        evaporator tempsensor rceva
    }

}
```

Example

Refrige rators

```
parameter t_abtaustart: int
parameter t_abtaudauer: int
parameter T_abtauEnde: int

var tuerNachlaufSchwelle: int = 0

start:
    entry { state noCooling }

state noCooling:
    check ( (RC->needsCooling) && (cc.c1->stehzeit > 333) ) {
        state rccooling
    }
    on isDown ( RC.rcdoor->open ) {
        set RC.rcfan->active = true
        set RC.rclight->active = false
        perform rcfanabschalttask after 10 {
            set RC.rcfan->active = false
        }
    }


state rccooling:
    entry { set RC.rcfan->active = true }
    check ( !(RC->needsCooling)  ) {
        state noCooling
    }
    on isDown ( RC.rcdoor->open ) {
        set RC.rcfan->active = true
        set RC.rclight->active = false
        set tuerNachlaufSchwelle = currStep + 30
    }
    exit {
        perform rcfanabschalttask after max( 5, tuerNachlaufSchwelle-currStep ) {
            set RC.rcfan->active = false
        }
    }
```

Example

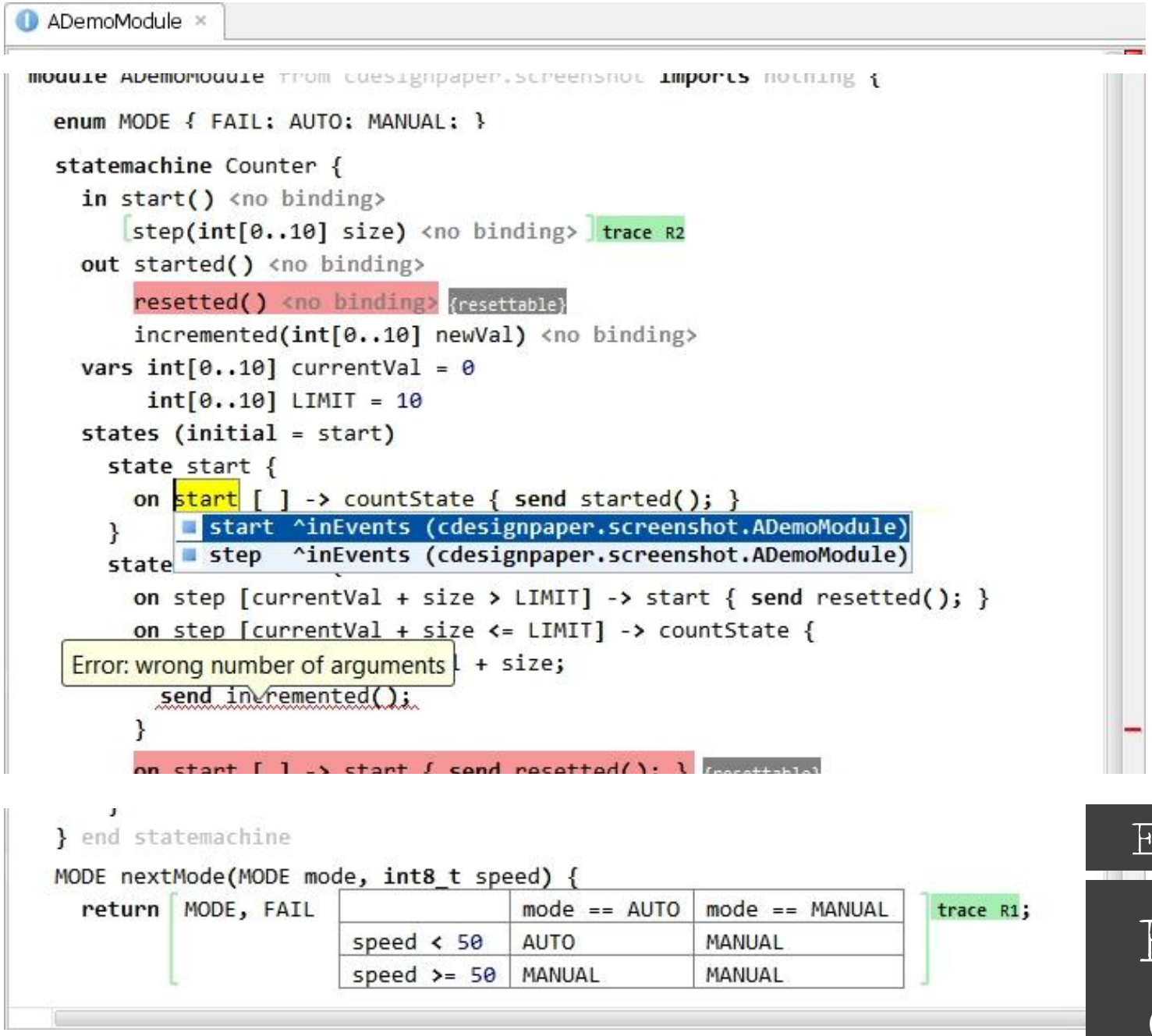
Refrige rators

```
parameter t_abtaustart: int
parameter t_abtaudauer: int
parameter T_abtauEnde: int

var tuerNachlaufSchwelle: int = 0

start:
    entry { state noCooling }

state noCooling:
    check ( (RC->needsCooling) && (cc.c1->stehz
        state rccooling
    }
    on isDown ( RC.rcdoor->open ) {
        set RC.rcfan->active = true
        set RC.rclight->active = false
        perform rcfanabschalttask after 10 {
            set RC.rcfan->active = false
        }
    }


state rccooling:
    entry { set RC.rcfan->active = true }
    check ( !(RC->needsCooling)  ) {
        state noCooling
    }
    on isDown ( RC.rcdoor->open ) {
        set RC.rcfan->active = true
        set RC.rclight->active = false
        set tuerNachlaufSchwelle = currStep + 30
    }
    exit {
        perform rcfanabschalttask after max( 5, tuerNachlaufSchwelle-currStep ) {
            set RC.rcfan->active = false
        }
    }
```

```
prolog {
    set RC->accumulatedRuntime = 80
}

step 10
assert-currentstate-is noCooling

mock: set RC->accumulatedRuntime = 110
step


mock: set RC.rceva->evaTemp = 10
assert-currentstate-is abtauen
assert-value cc.c1->active is false
mock: set RC->accumulatedRuntime = 0
step 5
assert-currentstate-is abtauen
assert-value cc.c1->active is false
step 15
assert-currentstate-is noCooling
```

Example

Refrige rators



Example

Exten
ded C

```
void testBasicUnits() {
  int8_t/N n = 3 N;
                              -2
  int8_t/N n3 = 3 kg m s  ;

  int8_t/m length;
  int8_t/s time;
              -1
  int8_t/[m s  ] speed = length / time;

  int8_t/v thisShouldNotWork = length + time;

  length l1 = length + length;
  length l2 = length * 33;
  length l3 = length / 3;
  length l4 = length + 3;
} testBasicUnits (function)
```

Example

Extended C

BasisHausrat

simple product **BasisHausrat**

attributes

groesse : **int16_t**
alter : **int16_t**
dritterParam : **boolean**

calculate

int16_t, 0

	groesse < 150	groesse >= 150
alter < 18	10	20
alter >= 18	30	40

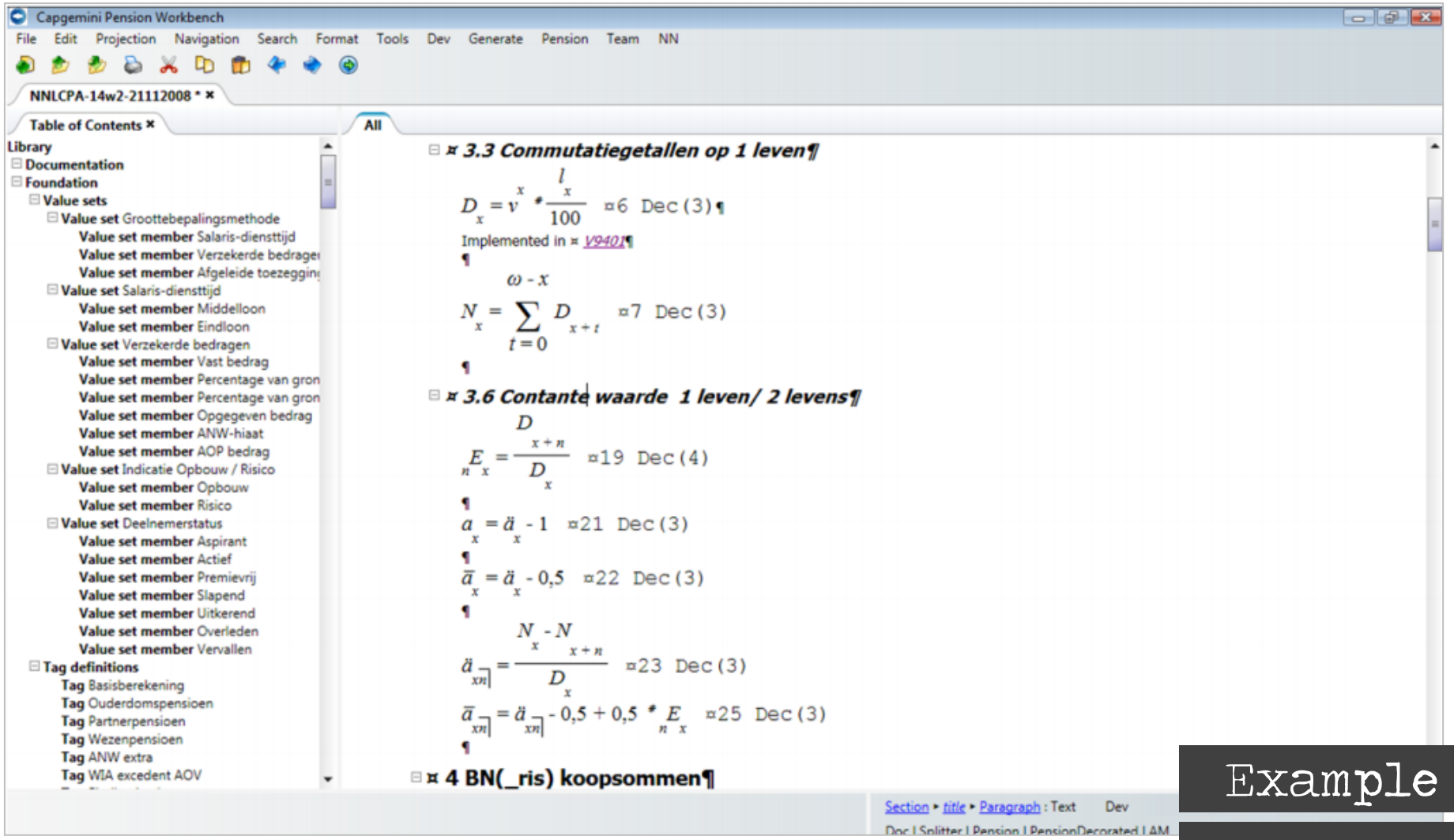
tests

groesse	alter	dritterParam		
100	2	true	20	10
200	30	true	40	40
150	20	true	40	40

evaluate

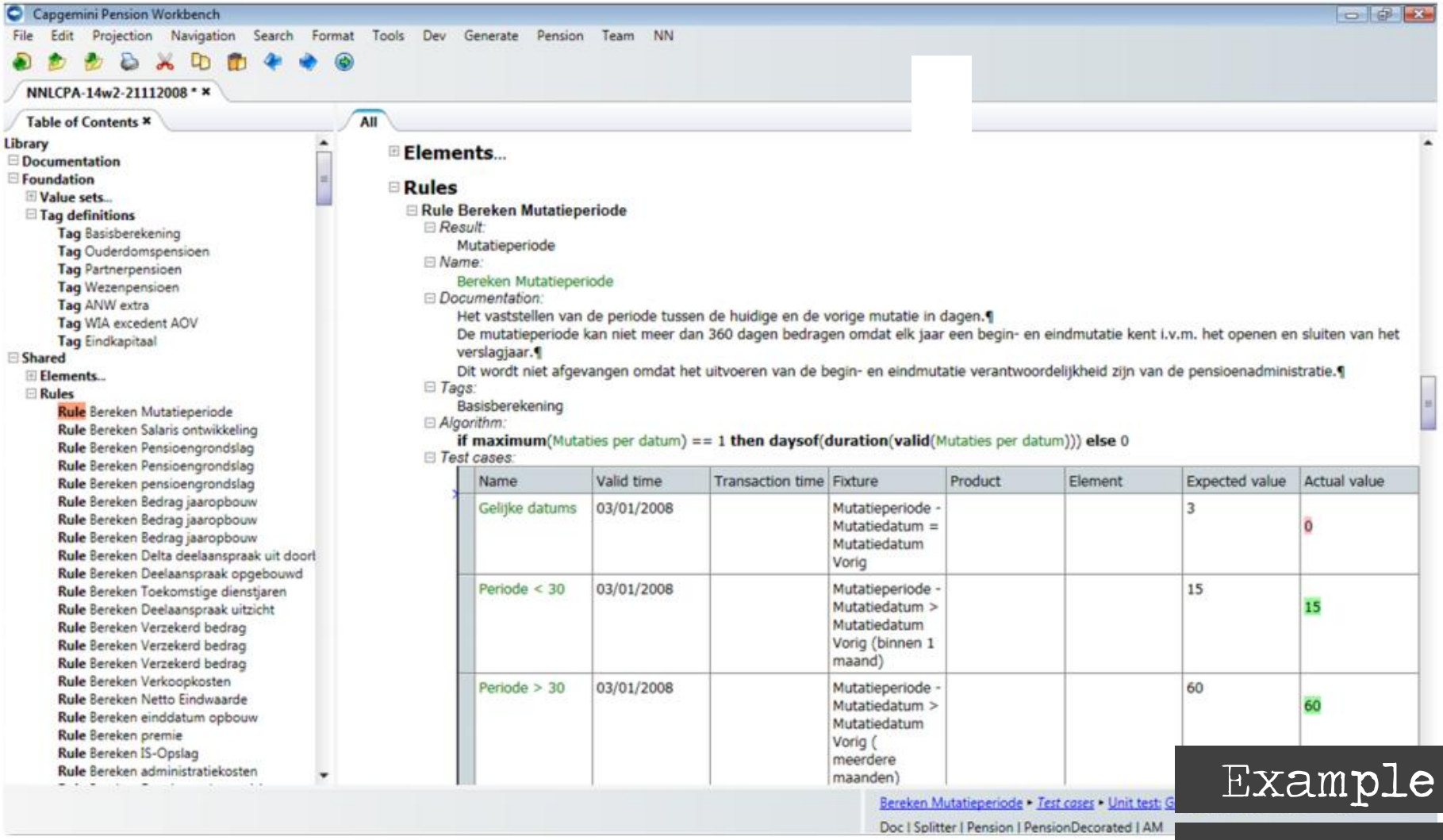
Example

Exten
ded C



Example

Pension Plans



Example

Pension Plans

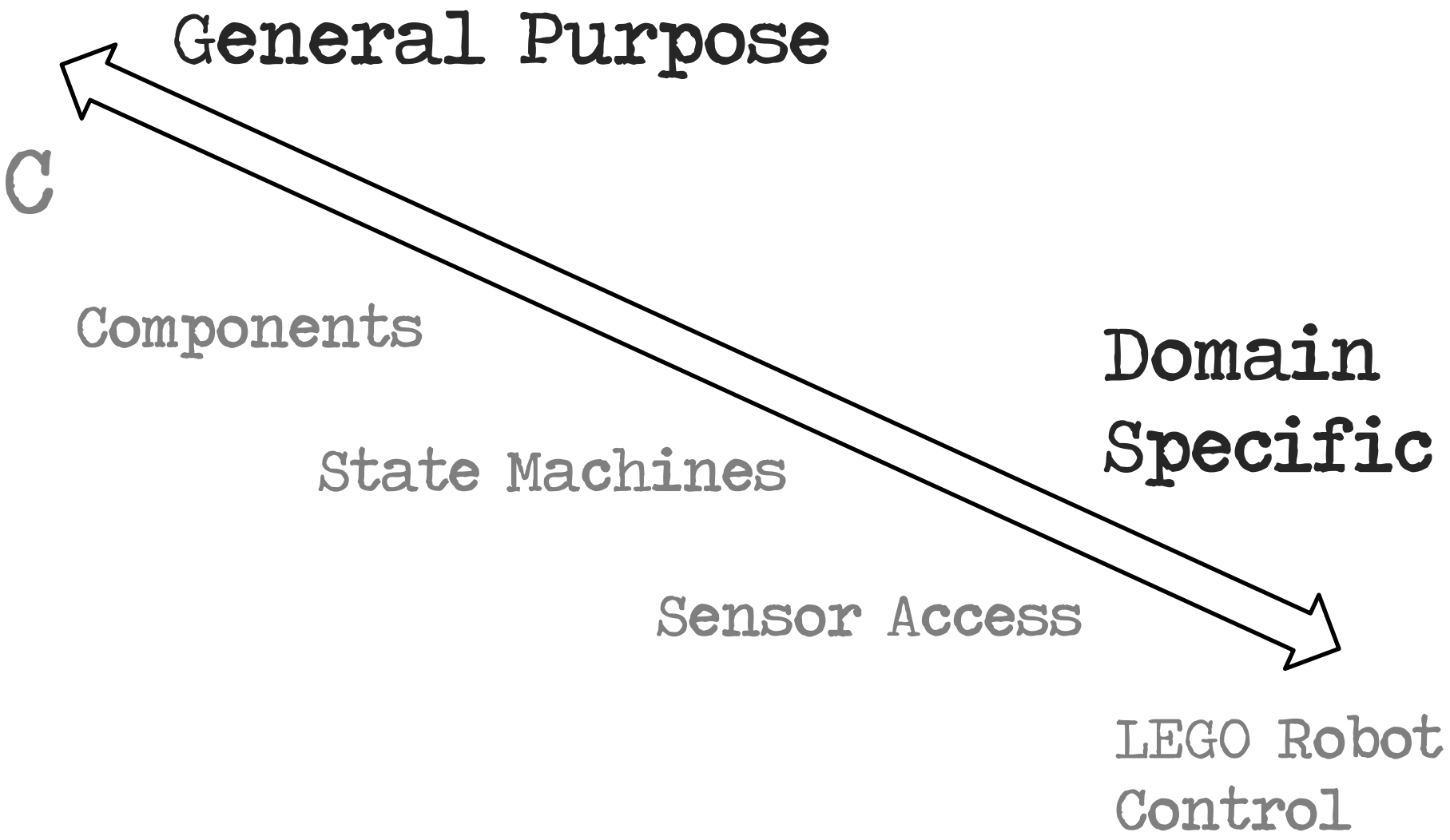
Concepts (abstract syntax)

(concrete) Syntax

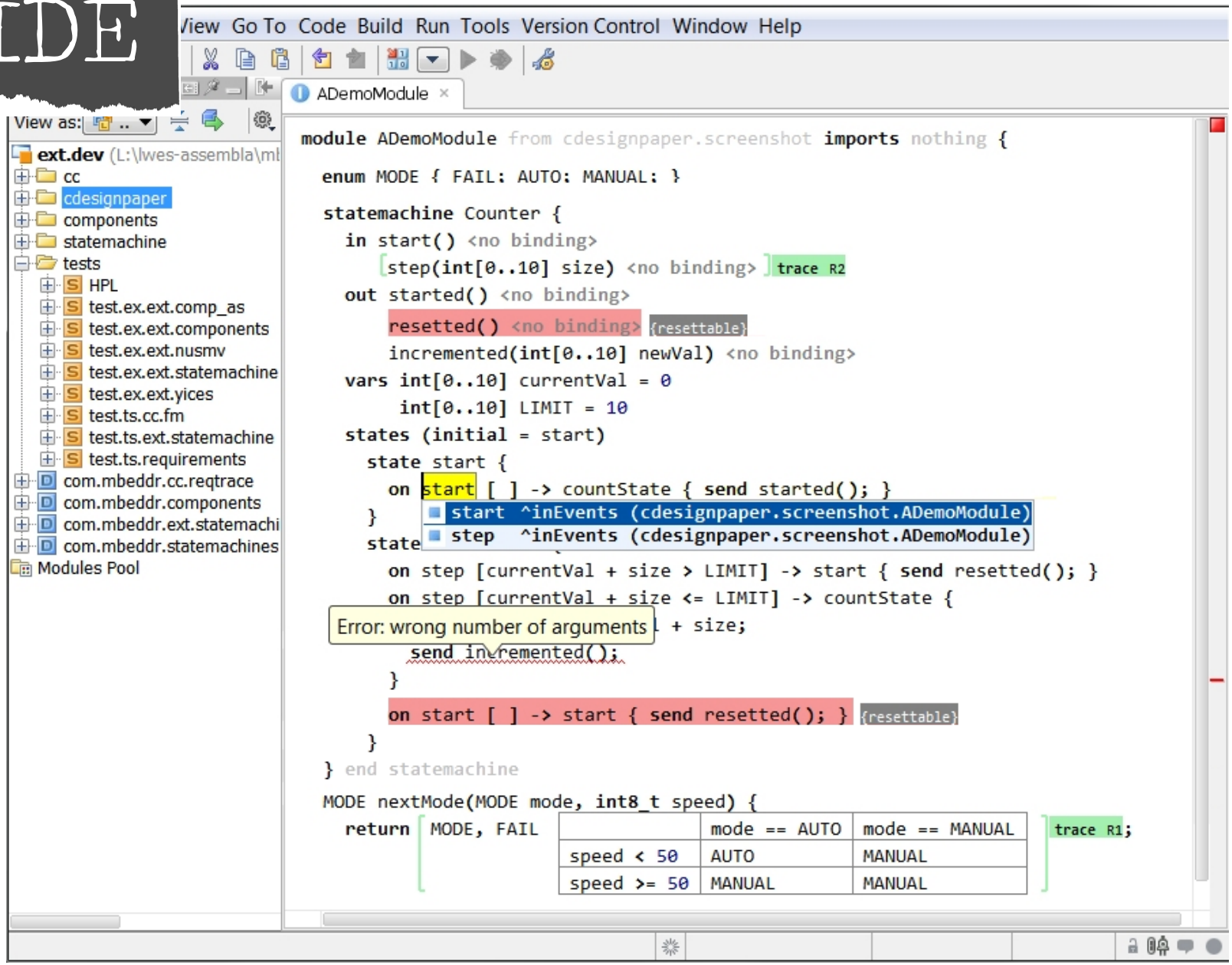
semantics (generators)

Tools and IDE

	more in GPLs	more in DSL
Domain Size	large and complex	smaller and well-defined
Designed by	guru or committee	a few engineers and domain experts
Language Size	large	small
Turing-completeness	almost always	often not
User Community	large, anonymous and widespread	small, accessible and local
In-language abstraction	sophisticated	limited
Lifespan	years to decades	months to years (driven by context)
Evolution	slow, often standardized	fast-paced
Incompatible Changes	almost impossible	feasible


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

IDE


View Go To Code Build Run Tools Version Control Window Help
ADemoModule
View as:
ext.dev (L:\lwes-assembla\ml
cc
cdesignpaper
components
statemachine
tests
HPL
test.ex.ext.comp_as
test.ex.ext.components
test.ex.ext.nusmv
test.ex.ext.statemachine
test.ex.ext.yices
test.ts.cc.fm
test.ts.ext.statemachine
test.ts.requirements
com.mbeddr.cc.reqtrace
com.mbeddr.components
com.mbeddr.ext.statemachi
com.mbeddr.statemachines
Modules Pool
module ADemoModule from cdesignpaper.screenshot imports nothing {
enum MODE { FAIL: AUTO: MANUAL: }
statemachine Counter {
in start() <no binding>
step(int[0..10] size) <no binding>
trace R2
out started() <no binding>
resetted() <no binding>
{resettable}
incremented(int[0..10] newVal) <no binding>
vars int[0..10] currentVal = 0
int[0..10] LIMIT = 10
states (initial = start)
state start {
on start [ ] -> countState { send started(); }
start ^inEvents (cdesignpaper.screenshot.ADemoModule)
step ^inEvents (cdesignpaper.screenshot.ADemoModule)
on step [currentVal + size > LIMIT] -> start { send resetted(); }
on step [currentVal + size <= LIMIT] -> countState {
Error: wrong number of arguments
+ size;
send incremented();
}
on start [ ] -> start { send resetted(); }
{resettable}
}
} end statemachine
MODE nextMode(MODE mode, int8_t speed) {
return MODE, FAIL
trace R1;
mode == AUTO
mode == MANUAL
speed < 50
AUTO
MANUAL
speed >= 50
MANUAL
MANUAL

IDE for Lang Developer

Language Workbench

Define, use, evolve and debug a set of integrated langauges

xtext

Design
Concerns
Expressivity

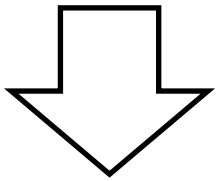
Shorter
Programs

More
Accessible
Semantics

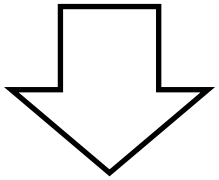
For a limited Domain!

Domain Knowledge encapsulated in language

Smaller Domain

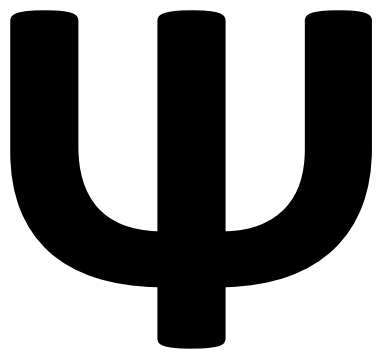


More Specialized Language



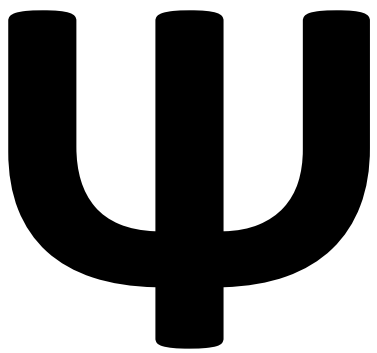
Shorter Programs

The do-what-I-want language



Single Program vs. Class/Domain

No Variability!



Design
Concerns
Analyzability

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

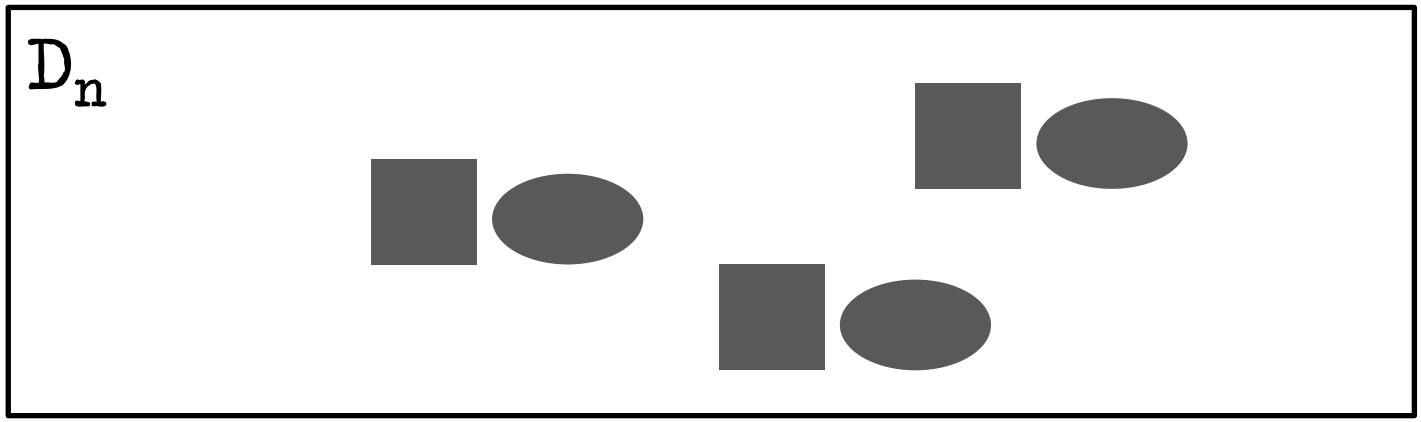
```
int[] arr = ...
List<int> l = ...
for (int i=0;  i<arr.size(); i++) {
    l.add( arr[i] );
}
```

Overspecification!
Requires Semantic Analysis!

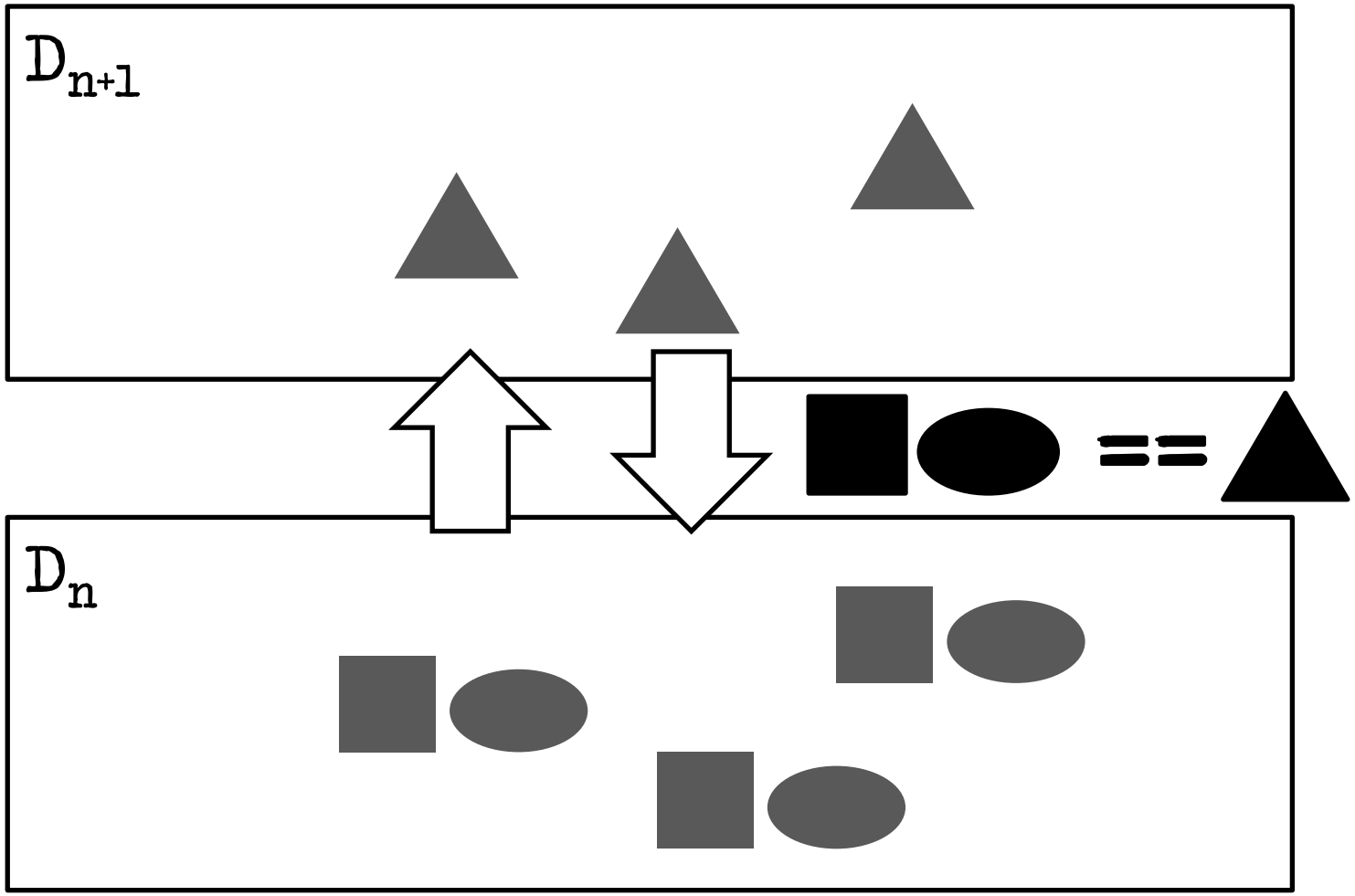
```
int[] arr = ...
for (int i=0; i<arr.size(); i++) {
    sum += arr[i];
}
```

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

Reification

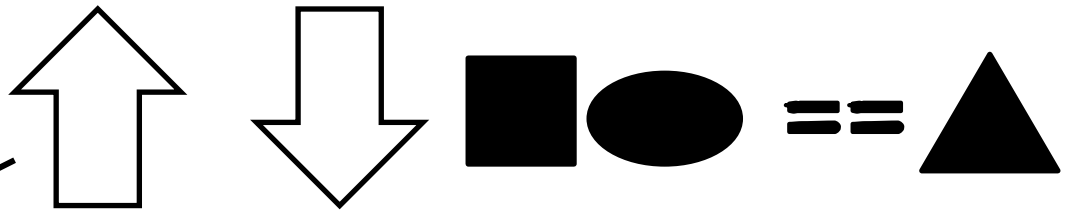


Reification



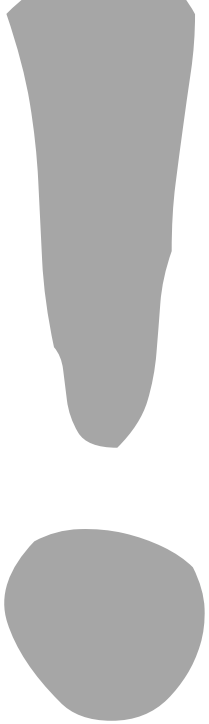
Reification

Transformation/
Generation



Language
Definition

Linguistic Abstraction



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

Declarative!

Directly represents Semantics.

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

Def: DSL cont'd

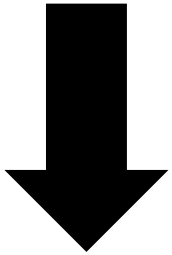
A good DSL does **not** require the use of patterns and idioms to express **semantically interesting** concepts in D. Processing tools do not have to do "semantic recovery" on D programs.

Declarative!

Reduced Expressiveness

bad? maybe.

good? maybe!

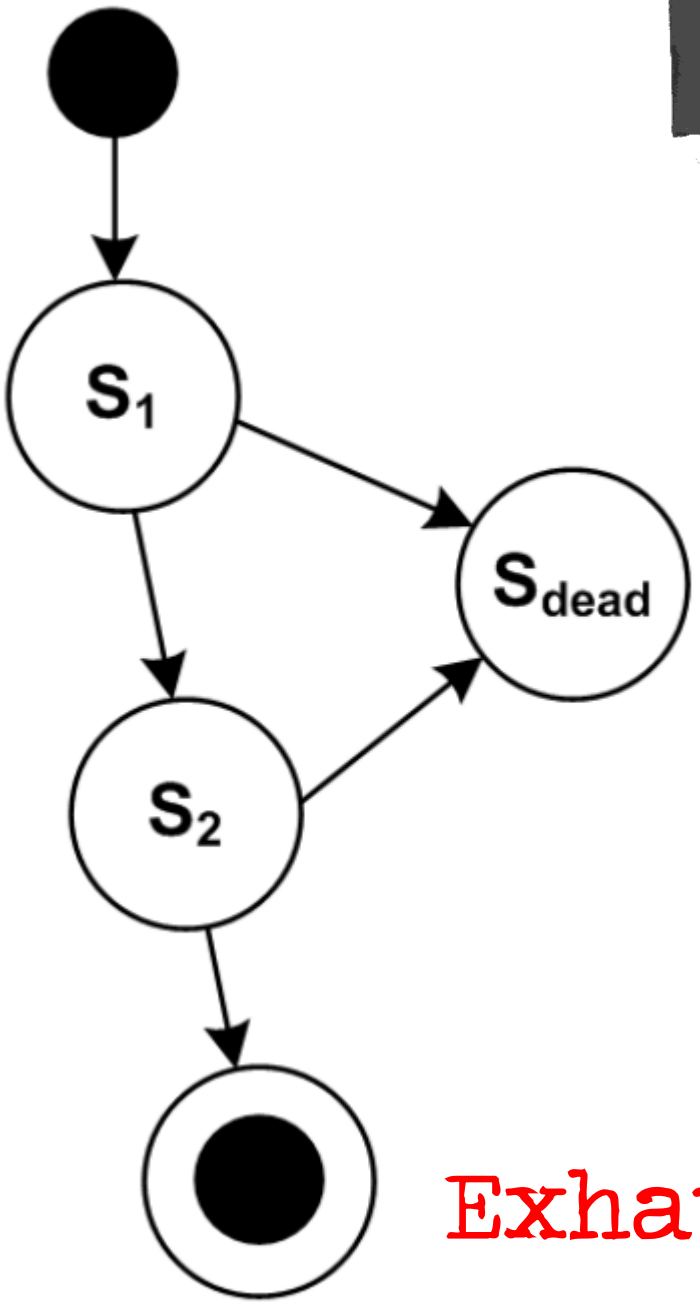


Model Checking

SAT Solving

Exhaustive Search, Proof!

Model Checking



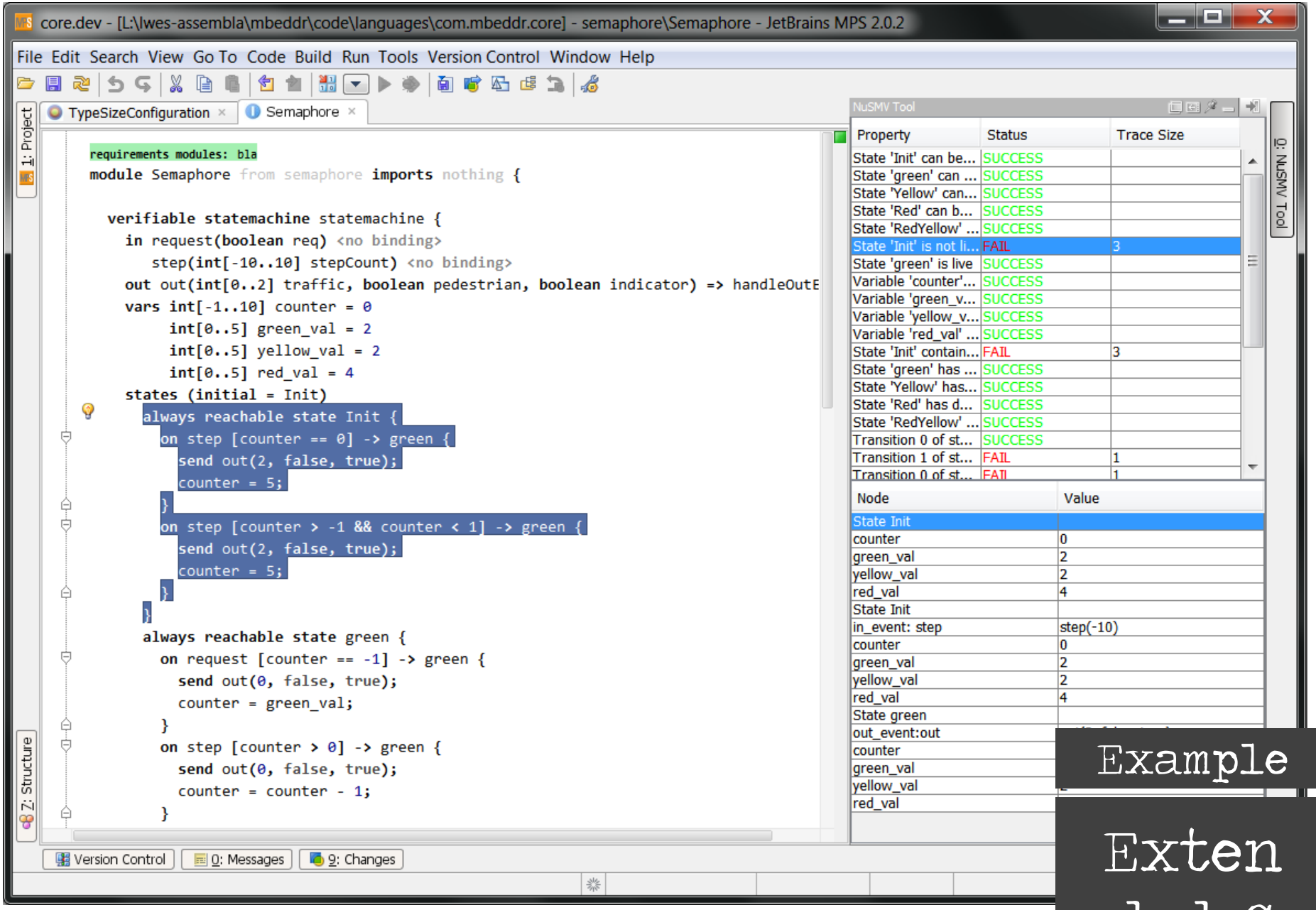
Unreachable States

Dead End States

Guard Decidability

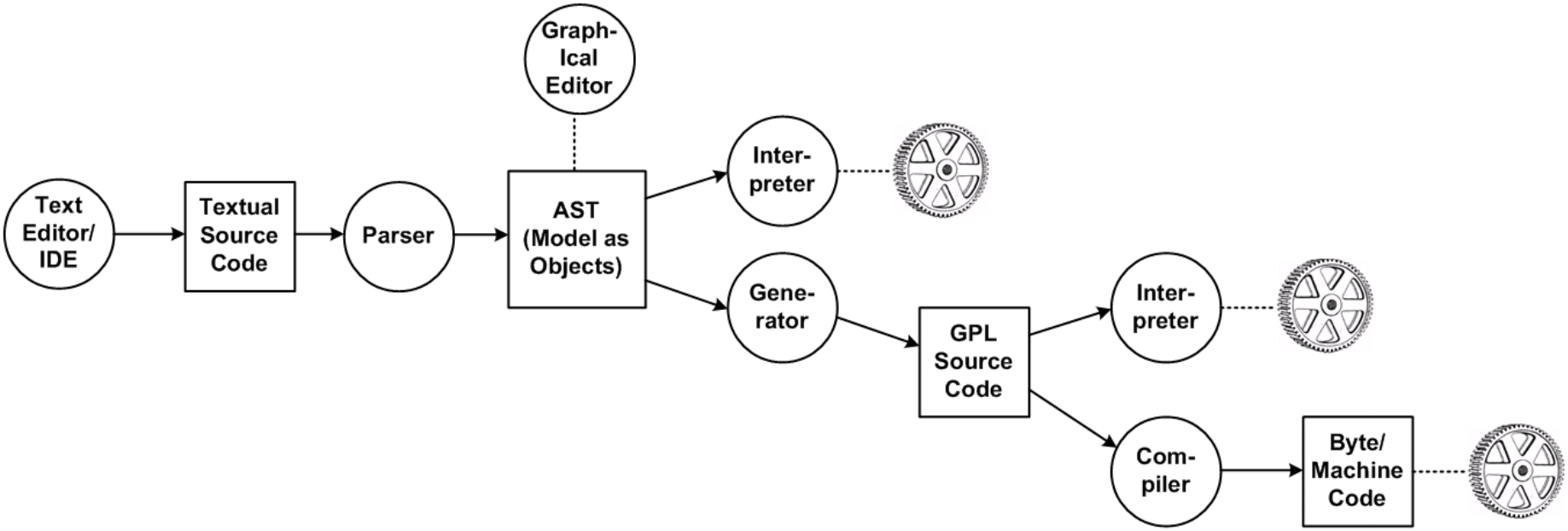
Reachability

Exhaustive Search, Proof!


core.dev - [L:\lwes-assembla\mbeddr\code\languages\com.mbeddr.core] - semaphore\Semaphore - JetBrains MPS 2.0.2
File Edit Search View Go To Code Build Run Tools Version Control Window Help
TypeSizeConfiguration
Semaphore
requirements modules: bla
module Semaphore from semaphore imports nothing {
verifiable statemachine statemachine {
in request(boolean req) <no binding>
step(int[-10..10] stepCount) <no binding>
out out(int[0..2] traffic, boolean pedestrian, boolean indicator) => handleOutE
vars int[-1..10] counter = 0
int[0..5] green_val = 2
int[0..5] yellow_val = 2
int[0..5] red_val = 4
states (initial = Init)
always reachable state Init {
on step [counter == 0] -> green {
send out(2, false, true);
counter = 5;
}
on step [counter > -1 && counter < 1] -> green {
send out(2, false, true);
counter = 5;
}
}
always reachable state green {
on request [counter == -1] -> green {
send out(0, false, true);
counter = green_val;
}
on step [counter > 0] -> green {
send out(0, false, true);
counter = counter - 1;
}
NuSMV Tool
Property
Status
Trace Size
State 'Init' can be... SUCCESS
State 'green' can ... SUCCESS
State 'Yellow' can... SUCCESS
State 'Red' can b... SUCCESS
State 'RedYellow' ... SUCCESS
State 'Init' is not li... FAIL 3
State 'green' is live SUCCESS
Variable 'counter'... SUCCESS
Variable 'green_v... SUCCESS
Variable 'yellow_v... SUCCESS
Variable 'red_val' ... SUCCESS
State 'Init' contain... FAIL 3
State 'green' has ... SUCCESS
State 'Yellow' has... SUCCESS
State 'Red' has d... SUCCESS
State 'RedYellow' ... SUCCESS
Transition 0 of st... SUCCESS
Transition 1 of st... FAIL 1
Node
Value
State Init
counter 0
green_val 2
yellow_val 2
red_val 4
State Init
in_event: step step(-10)
counter 0
green_val 2
yellow_val 2
red_val 4
State green
out_event:out
counter
green_val
yellow_val
red_val
1: Project
7: Structure
0: NuSMV Tool
Version Control
0: Messages
9: Changes
Example
Exten
ded C

Design
Concerns
Execution

Execution


Graph-
Ical
Editor
Text
Editor/
IDE
Textual
Source
Code
Parser
AST
(Model as
Objects)
Inter-
preter
Gene-
rator
GPL
Source
Code
Inter-
preter
Com-
piler
Byte/
Machine
Code

Transformation

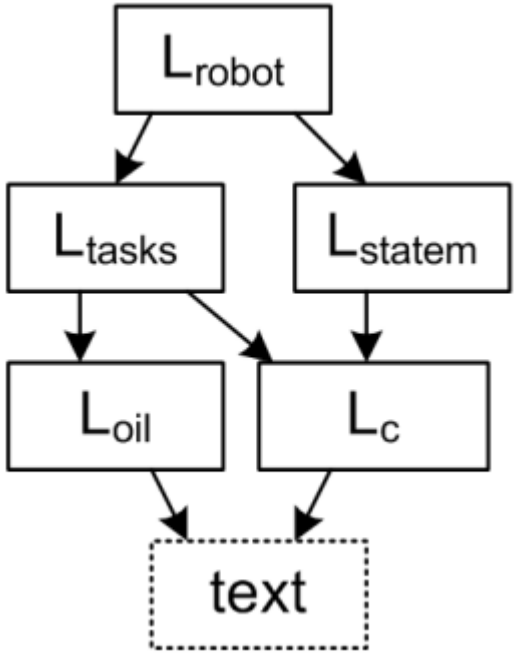
+ Code Inspection
+ Debugging
+ Performance & Optimization
+ Platform Conformance

Interpretation

+ Turnaround Time
+ Runtime Change

Transformation

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

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

  robot script stopAndGo
    block main on bump
      accelerate to 12 + speed(12) within 3000
      drive on for 2000
      turn left for 200
      decelerate to 0  within 3000
      stop

}
```



Example

Exten
ded C

```
parameter t_abtaustart: int
parameter t_abtaudauer: int
parameter T_abtauEnde: int

var tuerNachlaufSchwelle: int = 0

start:
    entry { state noCooling }

state noCooling:
    check ( (RC->needsCooling) && (cc.c1->stehz
        state rccooling
    }
    on isDown ( RC.rcdoor->open ) {
        set RC.rcfan->active = true
        set RC.rclight->active = false
        perform rcfanabschalttask after 10 {
            set RC.rcfan->active = false
        }
    }


state rccooling:
    entry { set RC.rcfan->active = true }
    check ( !(RC->needsCooling)  ) {
        state noCooling
    }
    on isDown ( RC.rcdoor->open ) {
        set RC.rcfan->active = true
        set RC.rclight->active = false
        set tuerNachlaufSchwelle = currStep + 30
    }
    exit {
        perform rcfanabschalttask after max( 5, tuerNachlaufSchwelle-currStep ) {
            set RC.rcfan->active = false
        }
    }
```

```
prolog {
    set RC->accumulatedRuntime = 80
}

step 10
assert-currentstate-is noCooling

mock: set RC->accumulatedRuntime = 110
step


mock: set RC.rceva->evaTemp = 10
assert-currentstate-is abtauen
assert-value cc.c1->active is false
mock: set RC->accumulatedRuntime = 0
step 5
assert-currentstate-is abtauen
assert-value cc.c1->active is false
step 15
assert-currentstate-is noCooling
```

Example

Refrige rators

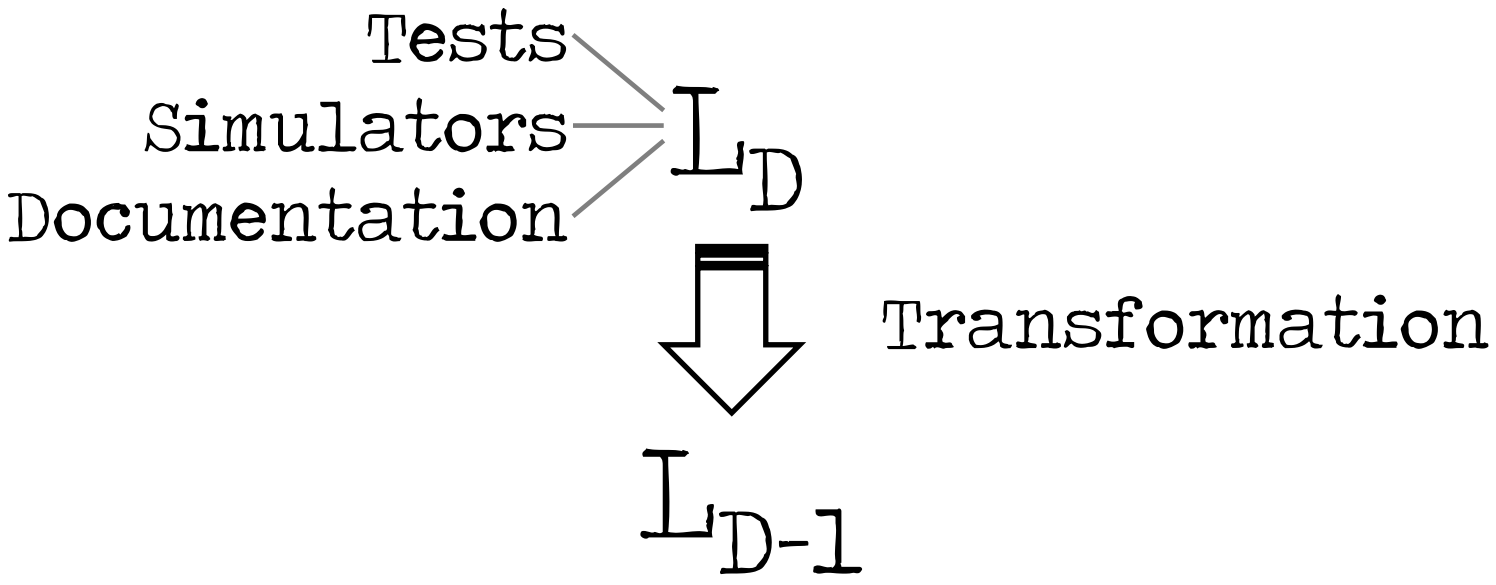
Transformation

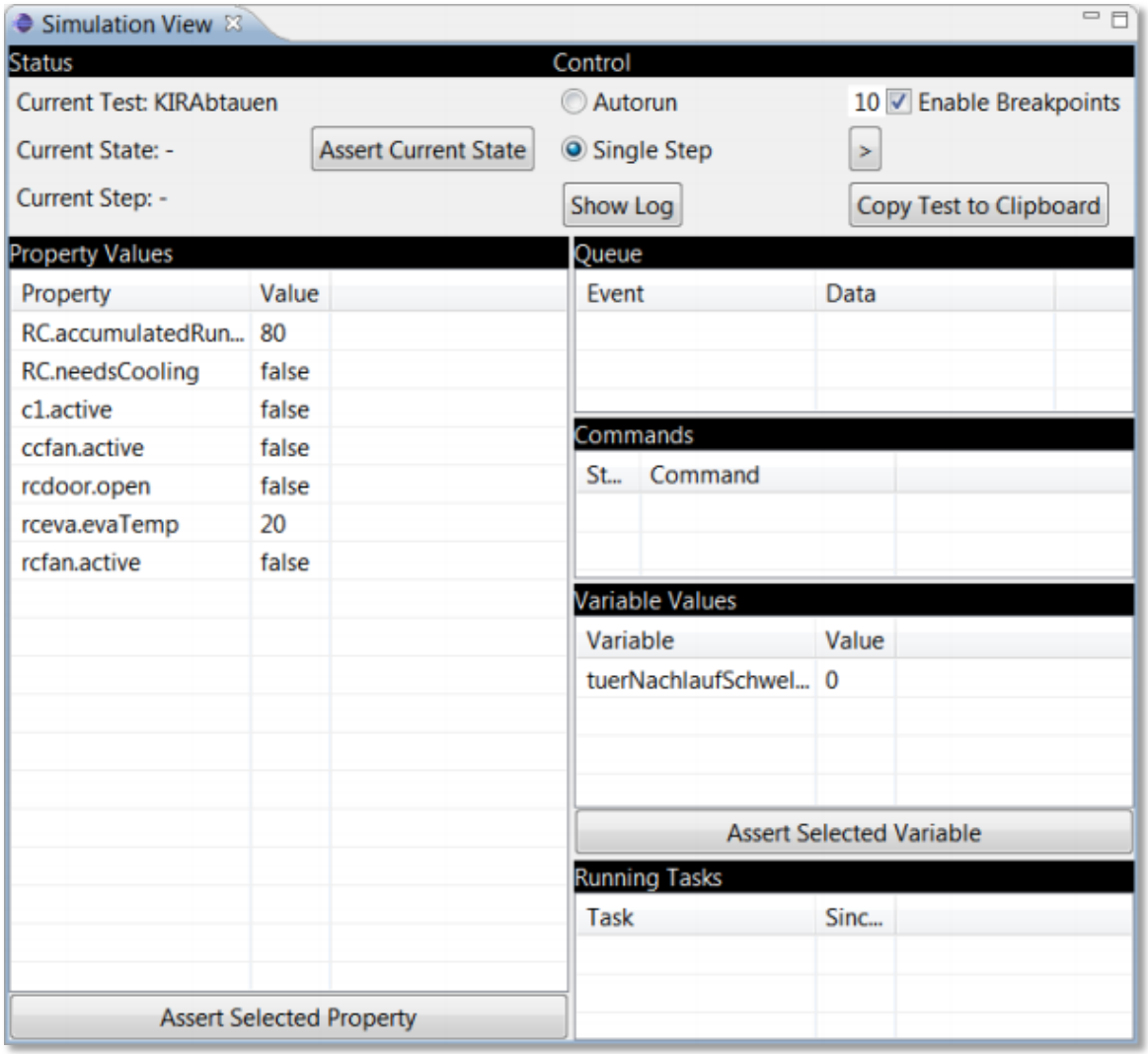
Name	Documentation	Tags	Valid time	Transaction time	Fixture	Product	Element	Expected value	Actual value
Accrued right at retireme			2006-12-31	2007-9-24	Jan De Jong	Old Age Pension	Accrued right	761.0402	761.0402
Accrued Right last final pay			2004-1-1	2007-9-24	Jan De Jong	Old Age Pension	Accrued right	705.0589	705.0589
premiun last year			2006-1-1	2007-9-24	Jan De Jong	Old Age Pension	Premium old age pension	329.0625	329.0625
Accrued right at retireme 2)			2006-12-31	2007-9-24	Piet Van Dijk	Old Age Pension	Accrued right	740.94	724.7658
			1985-12-31	2007-9-24	Jan De Jong	Old Age Pension	Accrued Right in service period	73.661	73.661
			1985-12-31	2007-9-24	Jan De Jong	Old Age Pension	Years of service in service period	3.7534	3.7534
			1987-12-31	2007-9-24	Jan De Jong	Old Age Pension	Pension base average FP	7750	
			1998-12-31	2007-9-24	Jan De Jong	Old Age Pension	Accrued Right in service period	387.7449	
			1998-12-31	2007-9-24	Jan De Jong	Old Age Pension	Years of service in service period	10.8082	
			1998-12-31	2007-9-24	Jan De Jong	Old Age Pension	Pension base average FP	8250	

Example

Pension Plans

Transformation





Example

Refrige
rators

Design
Concerns
Completeness

Can you generate 100%
of the code from the DSL
program?

Manually written code?

Call "black box" code (foreign functions)

Embed code in DSL program

Use composition mechanisms of L_{D-1} (inheritance, patterns, aspects, …)

Use protected regions (if you really have to…) DON'T!

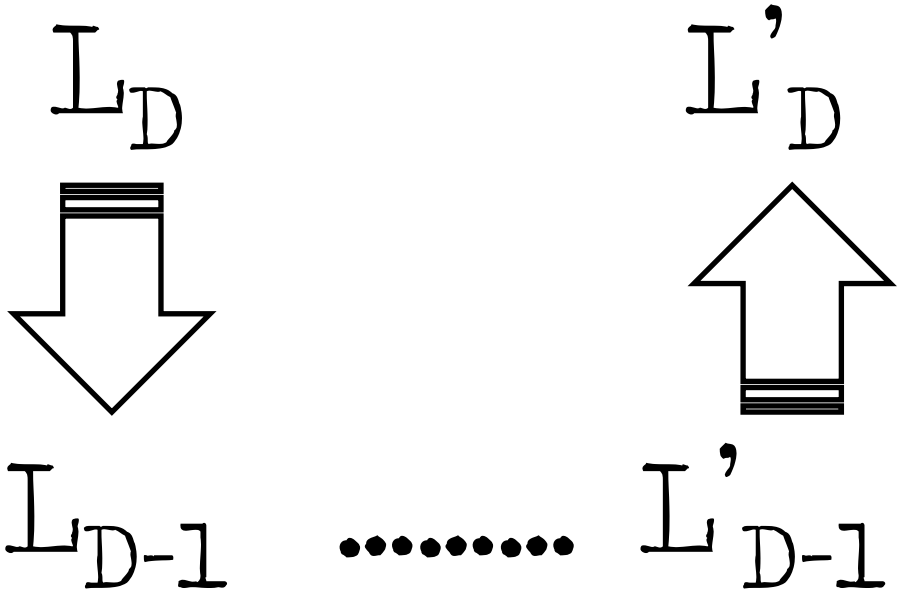
Incomplete: When?

Good for technical DSLs: Devs write L_{D-1} code

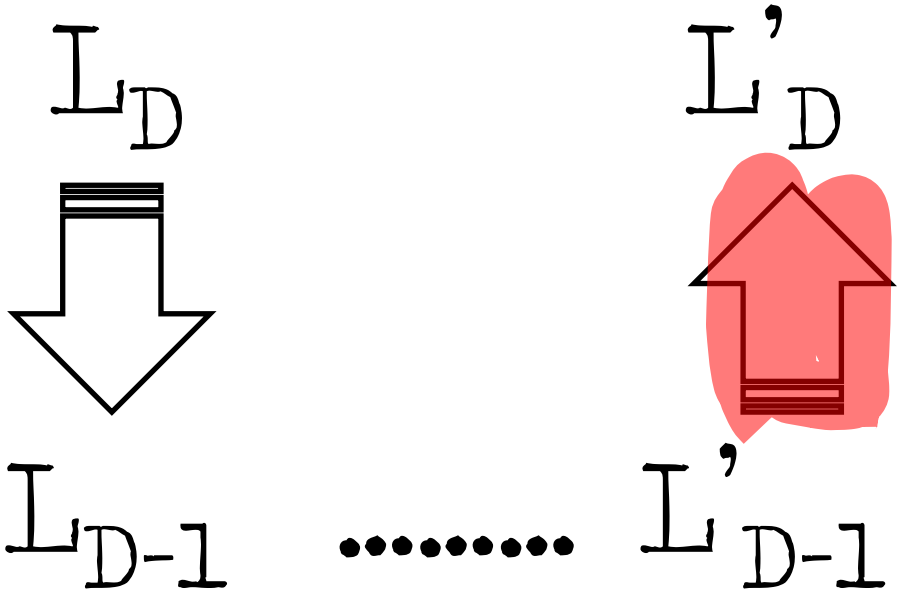
Bad for business DSLs.

Maybe use a L_{D-1} std lib that L_D code can call into?

Roundtripping



Roundtripping - Don't!



Semantic Recovery!

Design
Concerns
Modularity

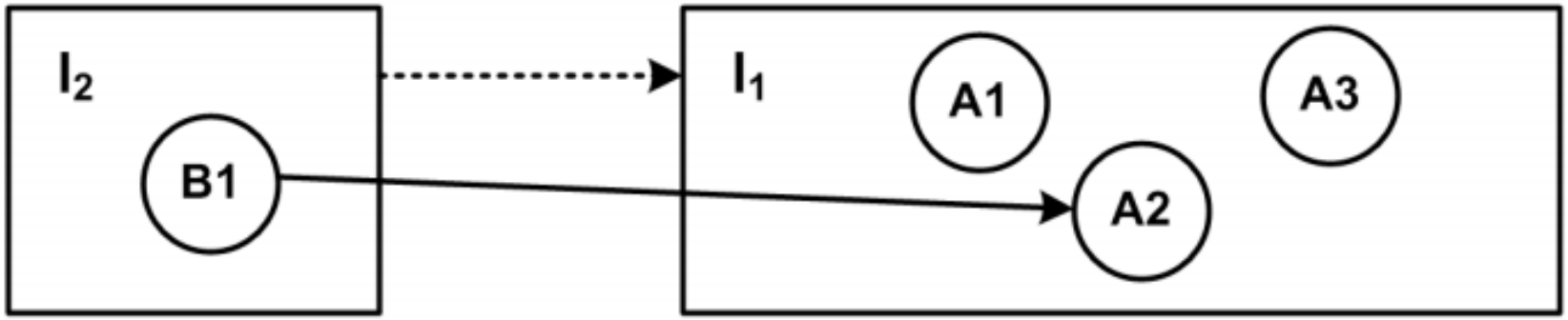
Language Modularity, Composition and Reuse (LMR&C)

increase efficiency
of DSL development

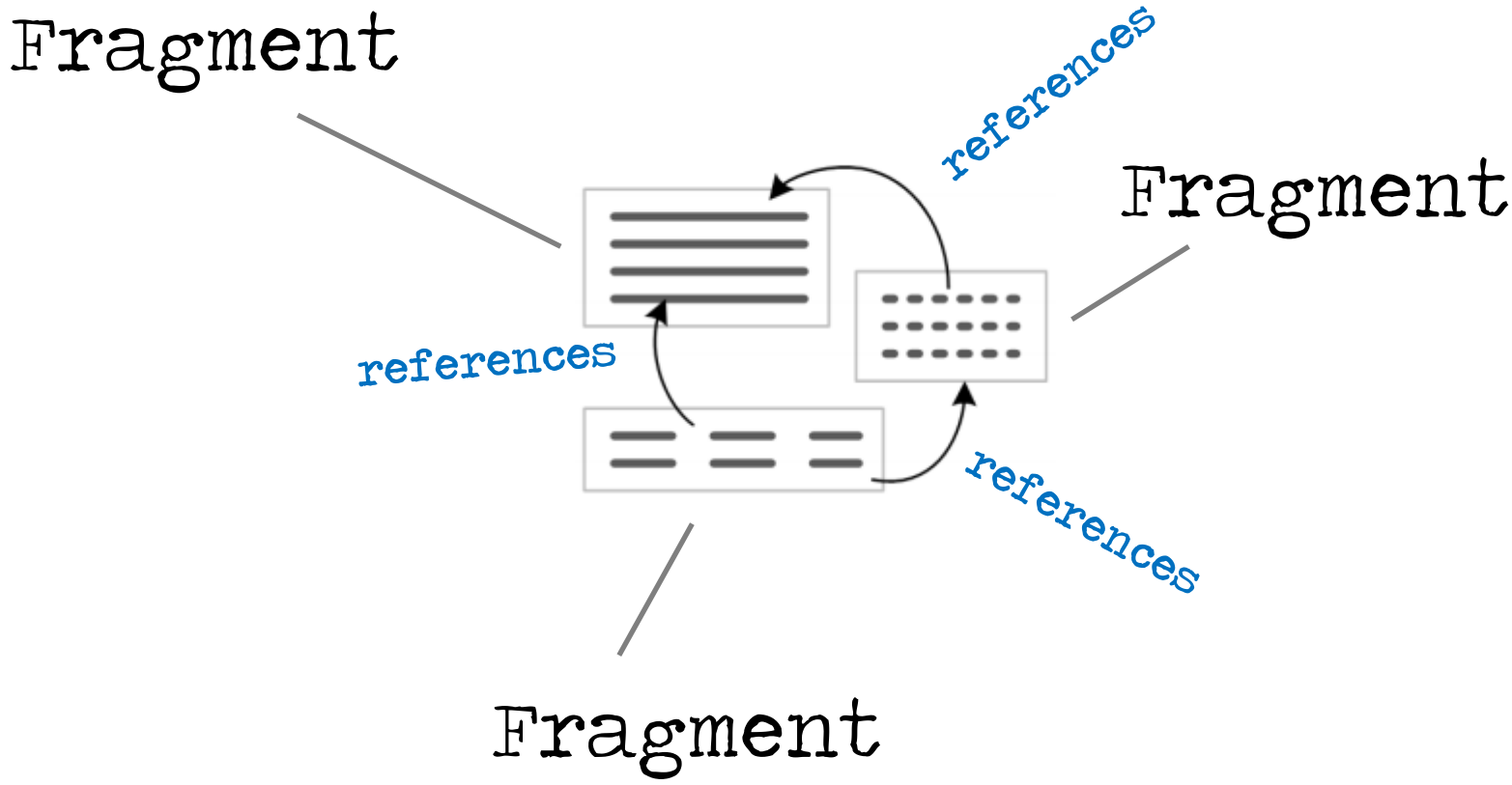
Referencing

languages dependencies	homogeneous	heterogeneous
independent	**Reuse**	**Embedding**
dependent	**Referencing**	**Extension**

fragment structure



Referencing



Referencing

```
parameter t_abtaustart: int
parameter t_abtaudauer: int
parameter T_abtauEnde: int

var tuerNachlaufSchwelle: int = 0

start:
    entry { state noCooling }

state noCooling:
    check ( (RC->needsCooling) && (cc.c1->stehz
        state rccooling
    }
    on isDown ( RC.rcdoor->open ) {
        set RC.rcfan->active = true
        set RC.rclight->active = false
        perform rcfanabschalttask after 10 {
            set RC.rcfan->active = false
        }
    }


state rccooling:
    entry { set RC.rcfan->active = true }
    check ( !(RC->needsCooling)  ) {
        state noCooling
    }
    on isDown ( RC.rcdoor->open ) {
        set RC.rcfan->active = true
        set RC.rclight->active = false
        set tuerNachlaufSchwelle = currStep + 30
    }
    exit {
        perform rcfanabschalttask after max( 5, tuerNachlaufSchwelle-currStep ) {
            set RC.rcfan->active = false
        }
    }
```

```
prolog {
    set RC->accumulatedRuntime = 80
}

step 10
assert-currentstate-is noCooling

mock: set RC->accumulatedRuntime = 110
step


mock: set RC.rceva->evaTemp = 10
assert-currentstate-is abtauen
assert-value cc.c1->active is false
mock: set RC->accumulatedRuntime = 0
step 5
assert-currentstate-is abtauen
assert-value cc.c1->active is false
step 15
assert-currentstate-is noCooling
```

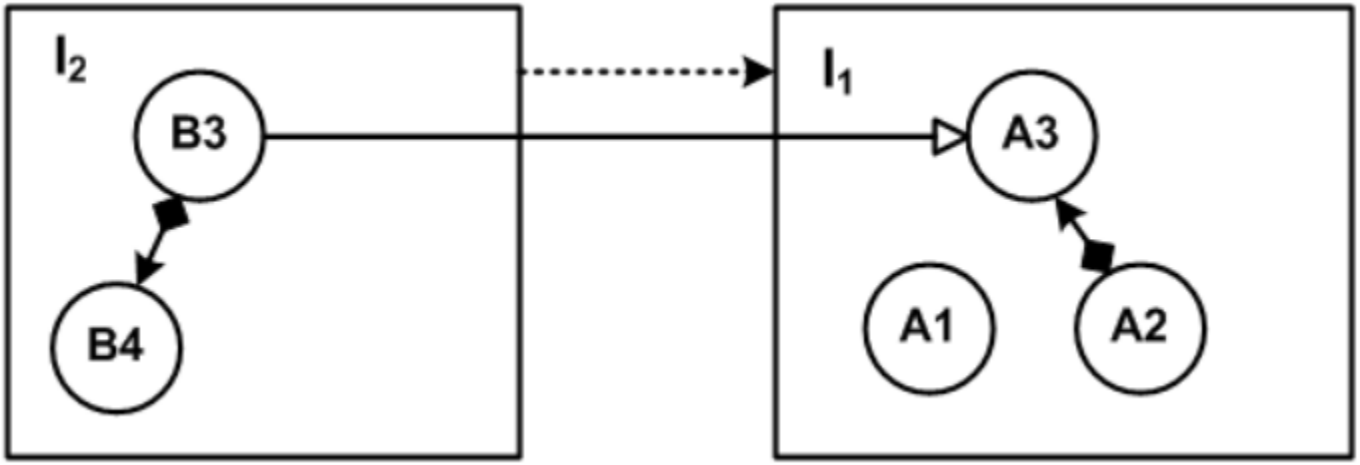
Example

Refrige
rators

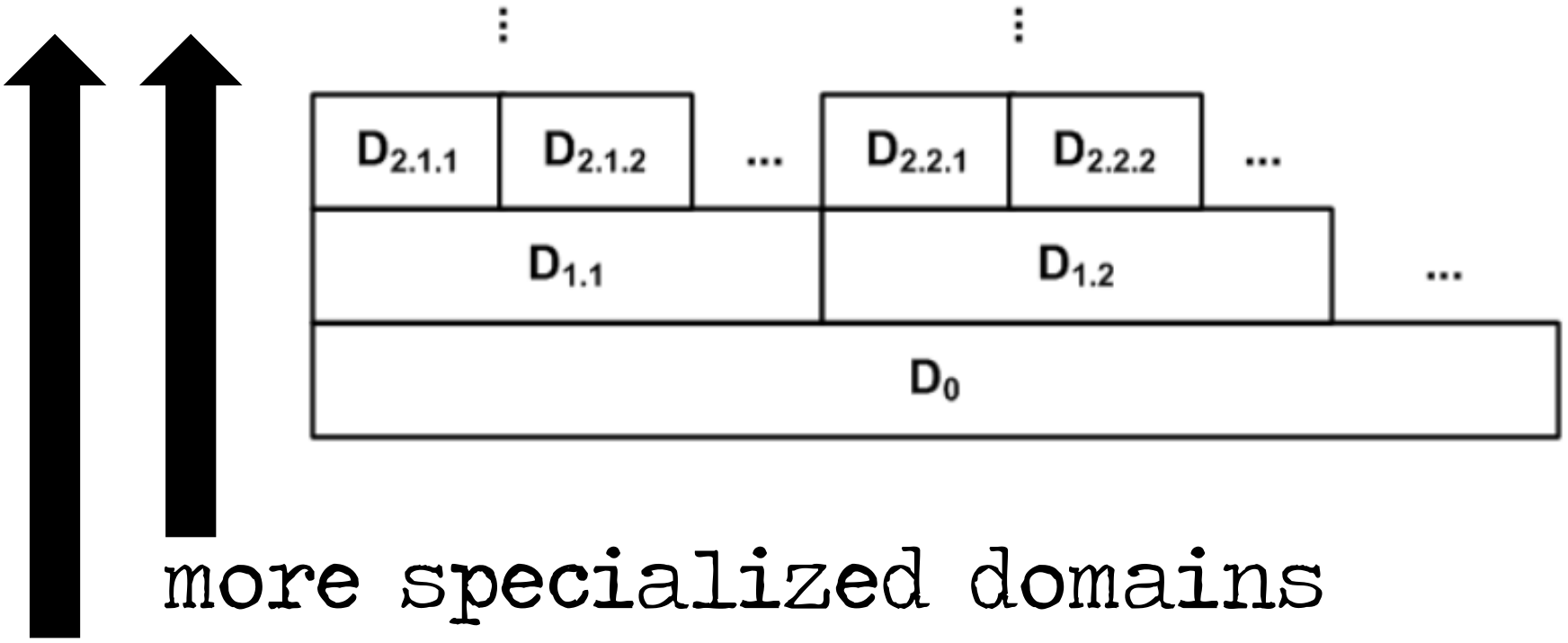
Extension

languages dependencies	homogeneous	heterogeneous
independent	**Reuse**	**Embedding**
dependent	**Referencing**	**Extension**

fragment structure



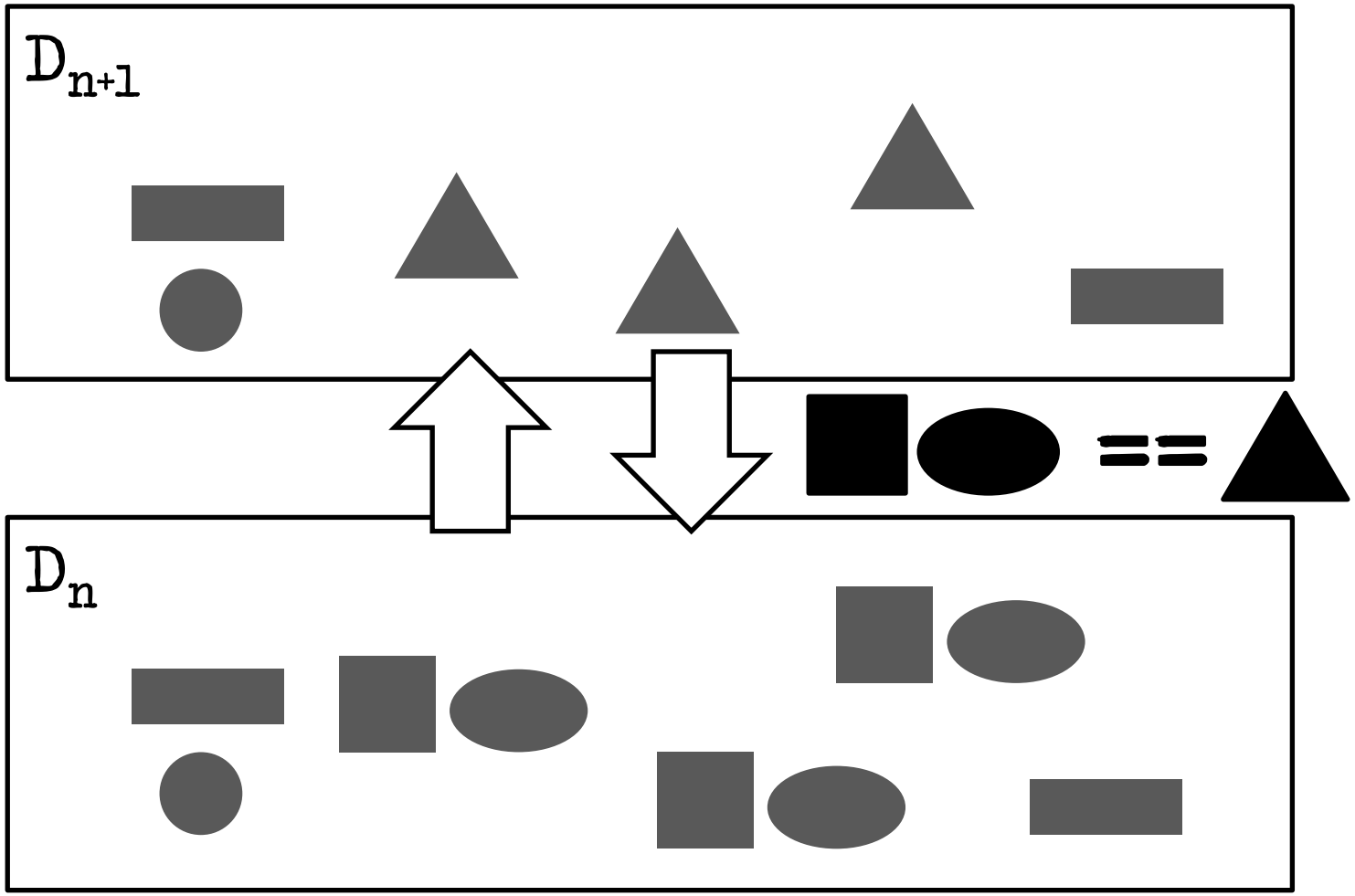
Extension



more specialized domains

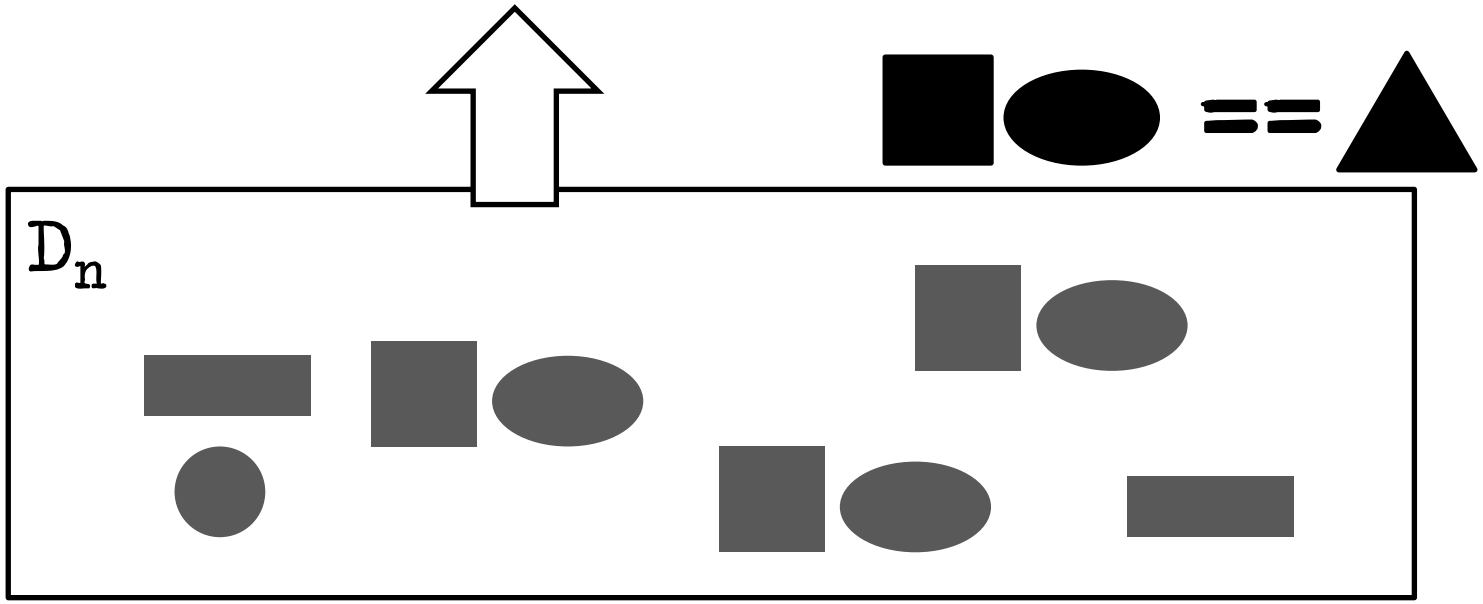
more specialized languages

Extension



Extension

Good for **bottom-up** (inductive) domains, and for use by **technical** DSLs (people)



Extension

```
module main imports OsekKernel, EcAPI, BitLevelUtilies {

  constant int WHITE = 500;
  constant int BLACK = 700;
  constant int SLOW = 20;
  constant int FAST = 40;

  statemachine linefollower {
    event initialized;
    initial state initializing {
      initialized [true] -> running
    }
    state running { }
  }

  initialize {
    ecrobot_set_light_sensor_active
                (SENSOR_PORT_T::NXT_PORT_S1);
    event linefollower:initialized
  }

  terminate {
    ecrobot_set_light_sensor_inactive
                (SENSOR_PORT_T::NXT_PORT_S1);
  }
```

```
task run cyclic prio = 1 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>;
}

void updateMotorSettings( int left, int right ) {
  nxt_motor_set_speed(MOTOR_PORT_T::NXT
  nxt_motor_set_speed(MOTOR_PORT_T::NXT
}
```

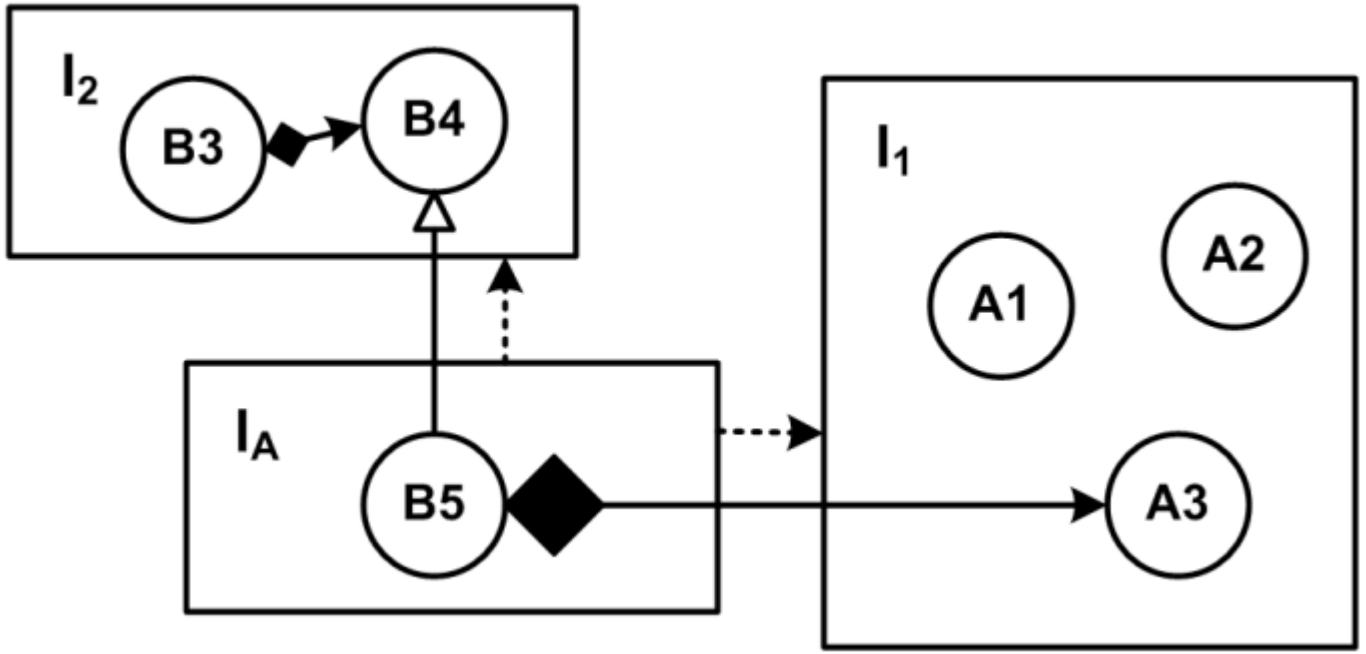
Example

Exten
ded C

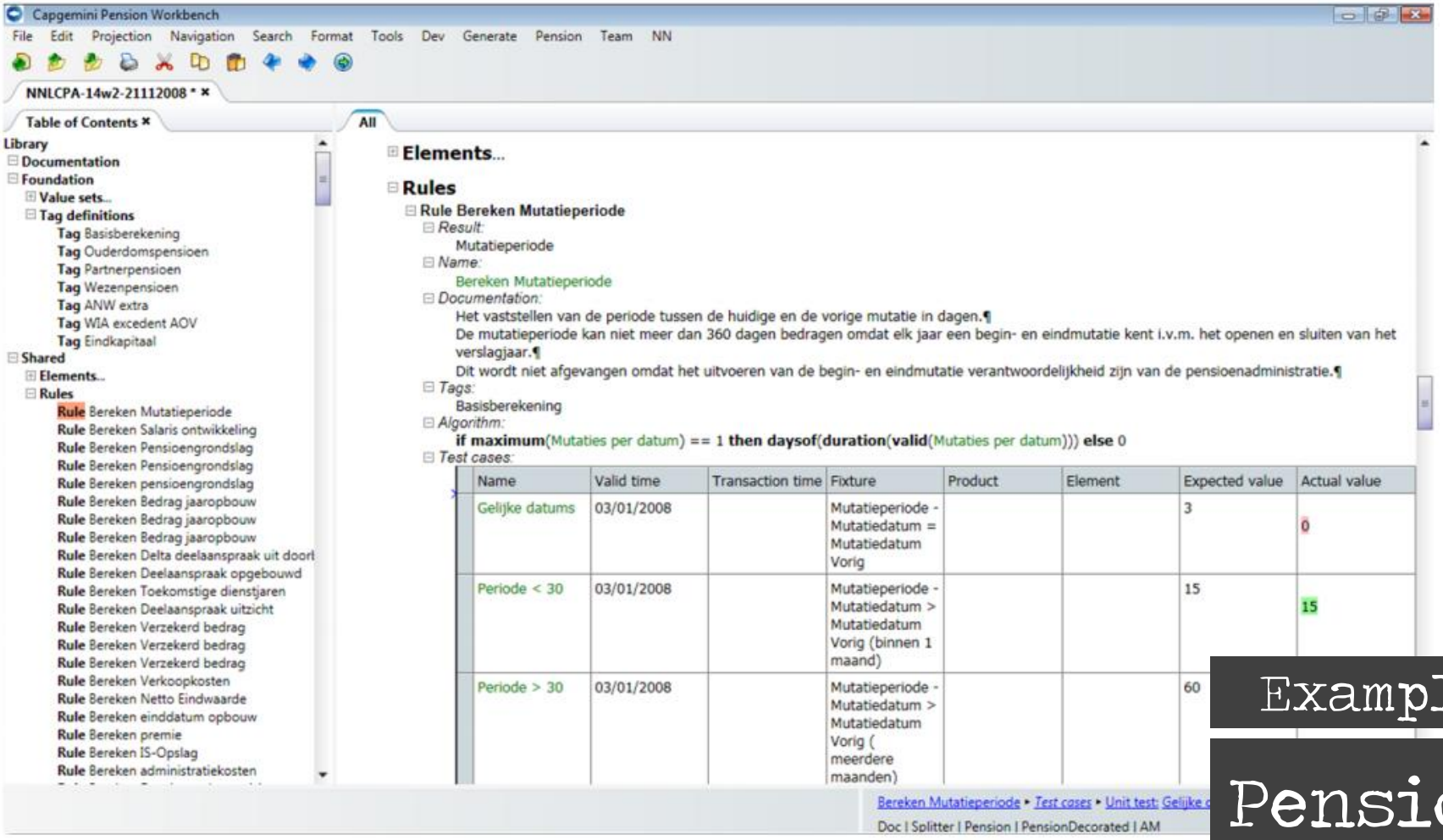
Embedding

languages dependencies	homogeneous	heterogeneous
independent	**Reuse**	**Embedding**
dependent	**Referencing**	**Extension**

fragment structure



Embedding



Example

Pension Plans

Design
Concerns
Concrete
Syntax

UI for the language!

Important for acceptance by users!

- Textual
- Symbolic
- Tabular
- Graphical

Reuse existing syntax of domain, if any!

Tools let you freely combine all kinds.

Default: Text

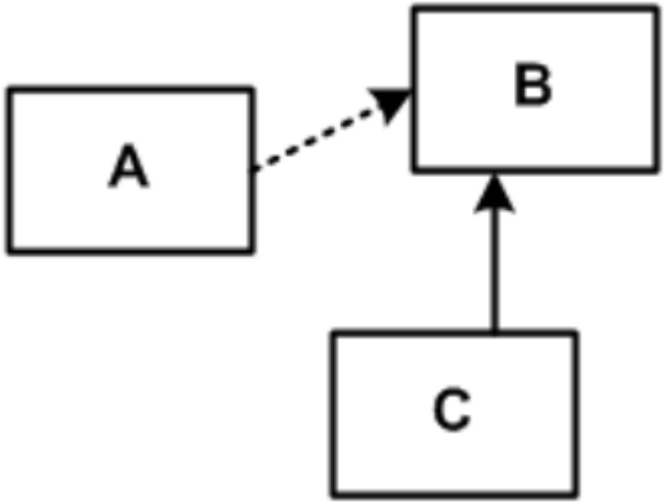
- Editors simple to build
- Productive
- Easy to integrate w/ tools
- Easy to evolve programs

Default: Text

Editors simple to build
Productive
Easy to integrate w/ tools
Easy to evolve programs

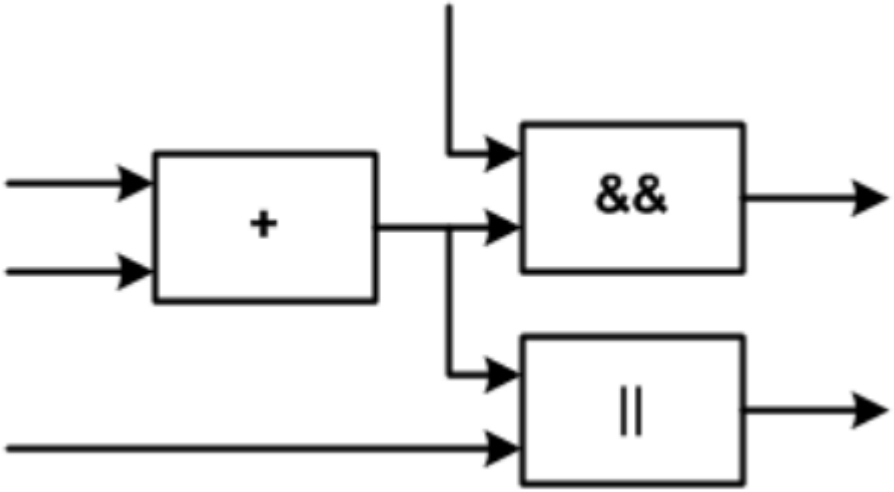
… then add other forms,
if really necessary

Graphical in case…



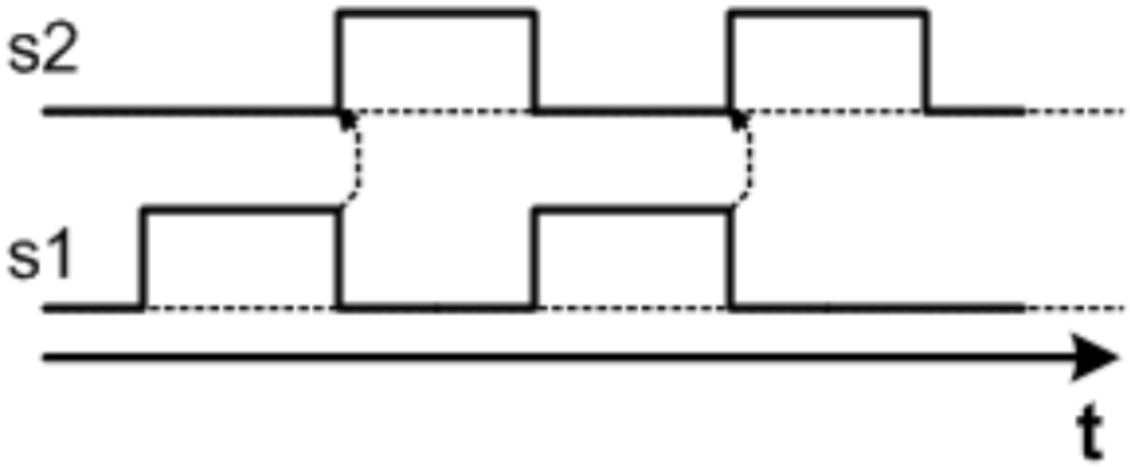
Relationships

Graphical in case…



Flow and
Depenency

Graphical in case…



Causality
and Timing

Symbolic

Either Mathematical, or often highly inspired by domain

¤ ***3.3 Commutatiegetallen op 1 leven¶***

$$D_x = v^x * \frac{l_x}{100}$$ ¤6 Dec(3)¶

Implemented in ¤ V9401¶

¶

$$N_x = \sum_{t=0}^{\omega - x} D_{x+t}$$ ¤7 Dec(3)

¶

¤ ***3.6 Contante waarde 1 leven/ 2 levens¶***

$$E_{n\ x} = \frac{D_{x+n}}{D_x}$$ ¤19 Dec(4)

¶

$$a_x = \ddot{a}_x - 1$$ ¤21 Dec(3)

¶

$$\bar{a}_x = \ddot{a}_x - 0{,}5$$ ¤22 Dec(3)

¶

$$\ddot{a}_{xn\rceil} = \frac{N_x - N_{x+n}}{D_x}$$ ¤23 Dec(3)

$$\bar{a}_{xn\rceil} = \ddot{a}_{xn\rceil} - 0{,}5 + 0{,}5 * E_{n\ x}$$ ¤25 Dec(3)

¶

Tables

Name	Documentation	Tags	Valid time	Transaction time	Fixture	Product	Element	Expected value	Actual value
Accrued right at retireme			2006-12-31	2007-9-24	Jan De Jong	Old Age Pension	Accrued right	761.0402	761.0402
Accrued Right last final pay			2004-1-1	2007-9-24	Jan De Jong	Old Age Pension	Accrued right	705.0589	705.0589
premium last year			2006-1-1	2007-9-24	Jan De Jong	Old Age Pension	Premium old age pension	329.0625	329.0625
Accrued right at retireme 2)			2006-12-31	2007-9-24	Piet Van Dijk	Old Age Pension	Accrued right	740.94	724.7658
			1985-12-31	2007-9-24	Jan De Jong	Old Age Pension	Accrued Right in service period	73.661	73.661
			1985-12-31	2007-9-24	Jan De Jong	Old Age Pension	Years of service in service period	3.7534	3.7534
			1987-12-31	2007-9-24	Jan De Jong	Old Age Pension	Pension base average FP	7750	7750
			1998-12-31	2007-9-24	Jan De Jong	Old Age Pension	Accrued Right in service period	387.7449	387.7449
			1998-12-31	2007-9-24	Jan De Jong	Old Age Pension	Years of service in service period	10.8082	10.8082
			1998-12-31	2007-9-24	Jan De Jong	Old Age Pension	Pension base average FP	8250	8250

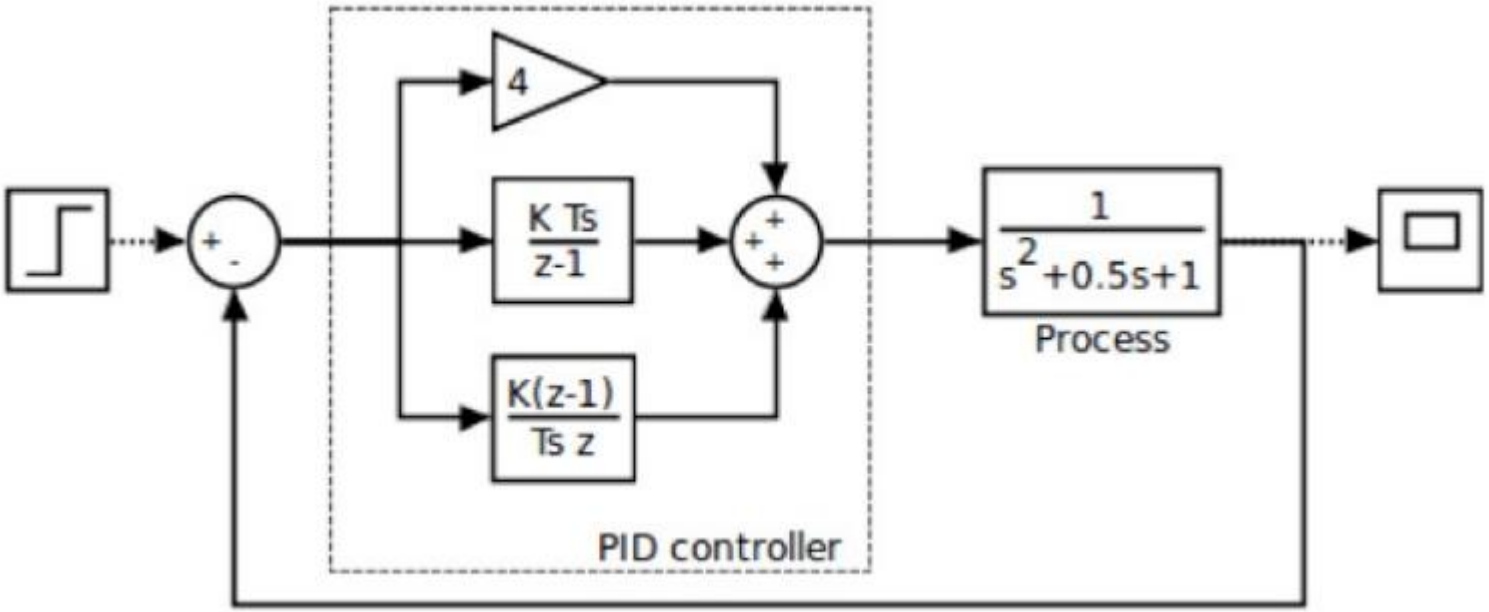
Combinations

```
c/s interface Decider {
  int decide(int x, int y) pre
}

component AComp extends nothing {
  ports:
    provides Decider decider
  contents:
    int decide(int x, int y) <- op decider.decide {
      return int, 0
    }
}
```

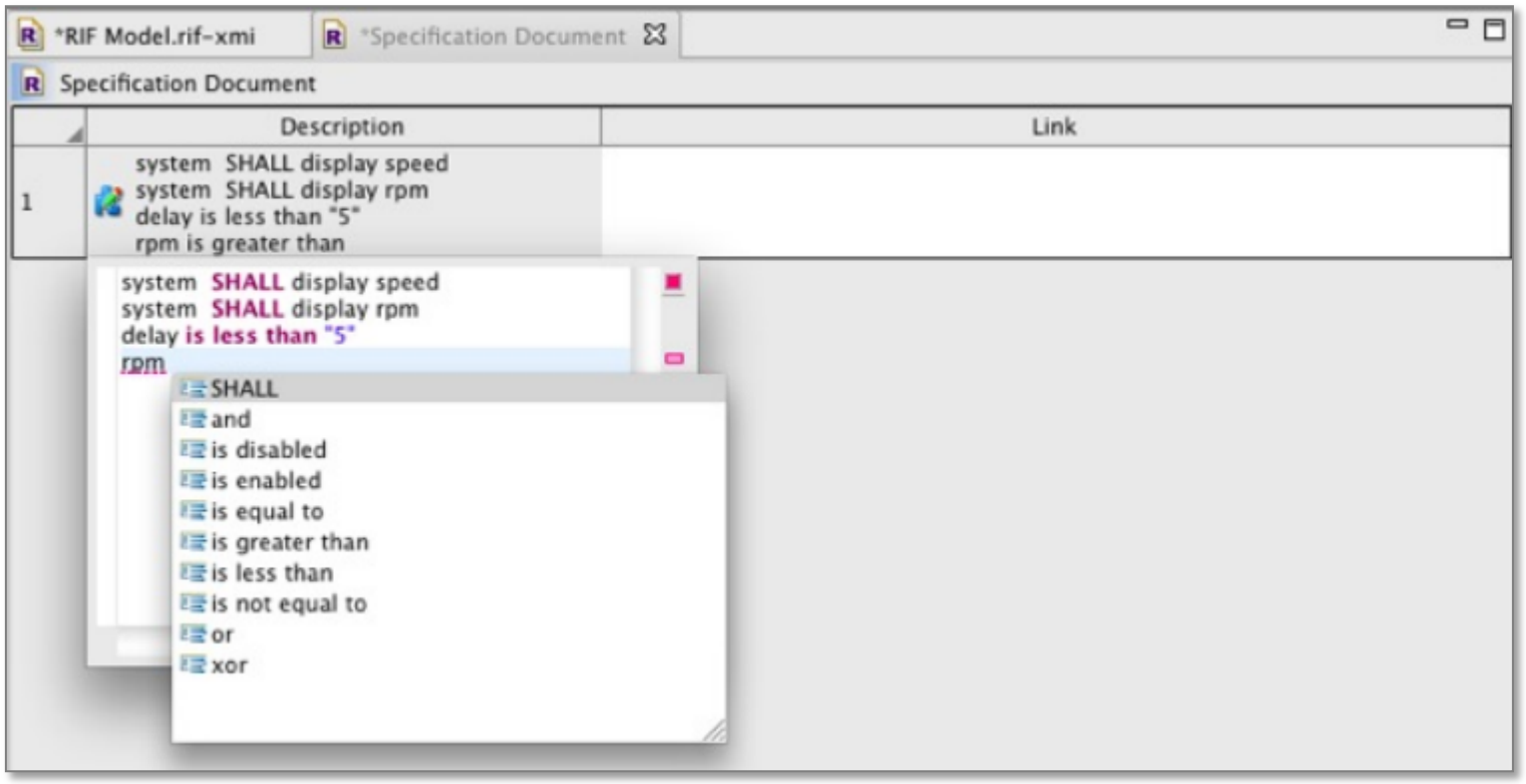
	x == 0	x > 0
y == 0	0	1
y > 0	1	2

;

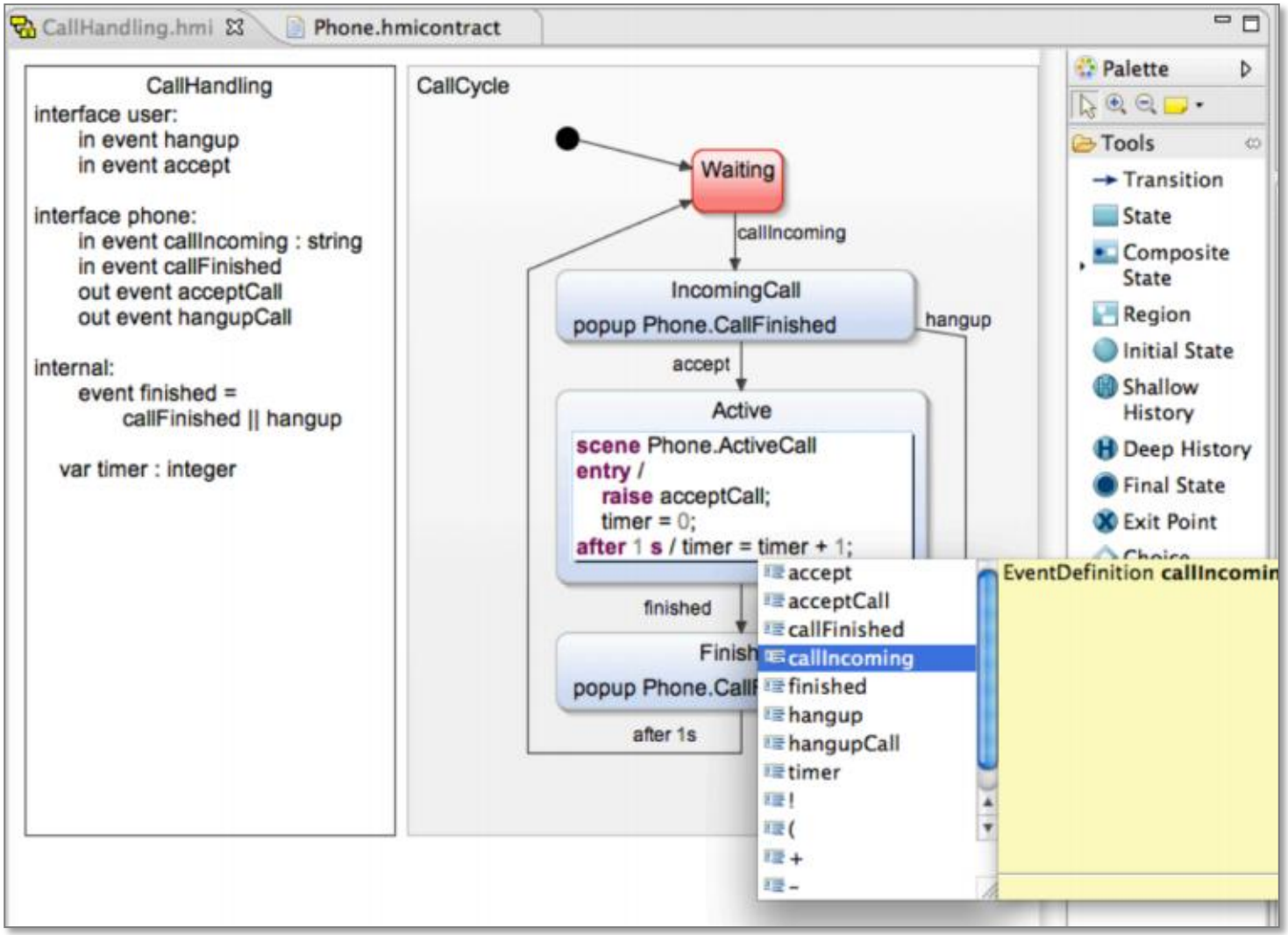
Combinations



Combinations



Combinations



The End.

This material is part of my upcoming (early 2013) book

DSL Engineering with Language Workbenches

Stay in touch; it will be cheap or maybe even free :-)

www.voelter.de/dslbook

@markusvoelter
+Markus Voelter