

# DSL Design

## A conceptual framework for building good DSLs

Markus Voelter  
independent/itemis  
[voelter@acm.org](mailto:voelter@acm.org)

based on material  
from a paper written  
with Eelco Visser

[www.voelter.de](http://www.voelter.de)  
[voelterblog.blogspot.de](http://voelterblog.blogspot.de)  
[@markusvoelter](https://twitter.com/markusvoelter)  
+Markus Voelter

[E.Visser@tudelft.nl](mailto:E.Visser@tudelft.nl)  
<http://eelcovisser.org/>

# Introduction

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.

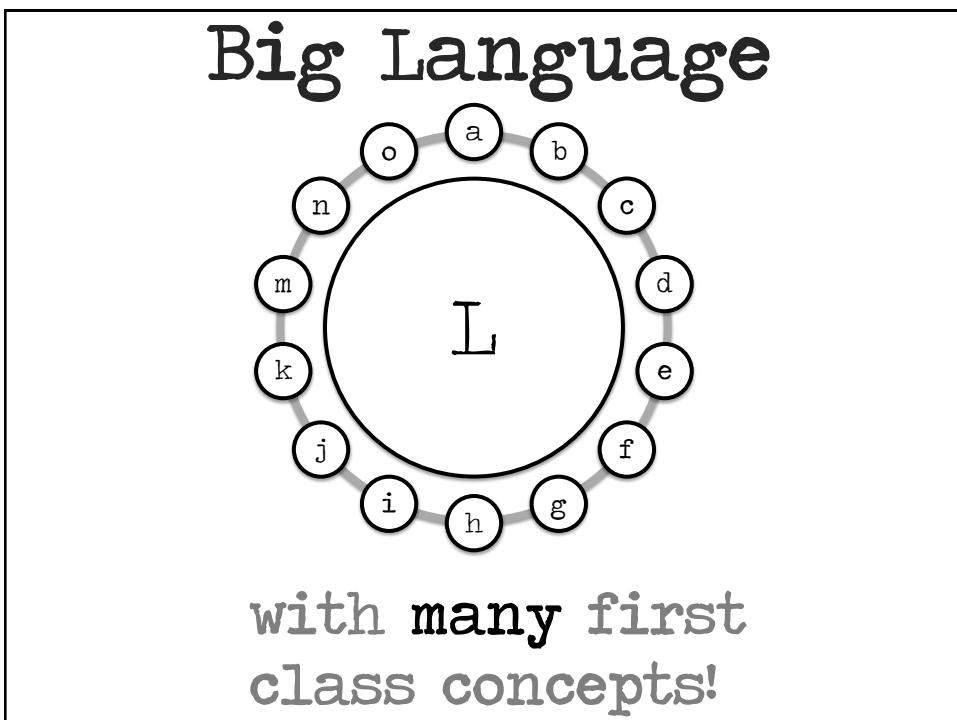
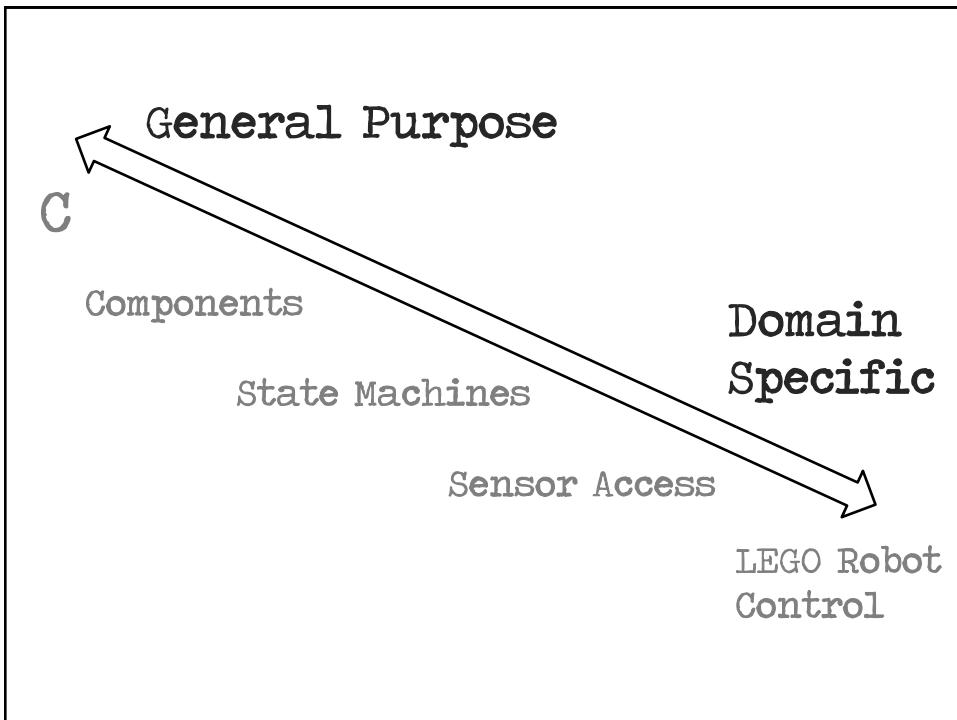
# Concepts (abstract syntax)

## (concrete) Syntax

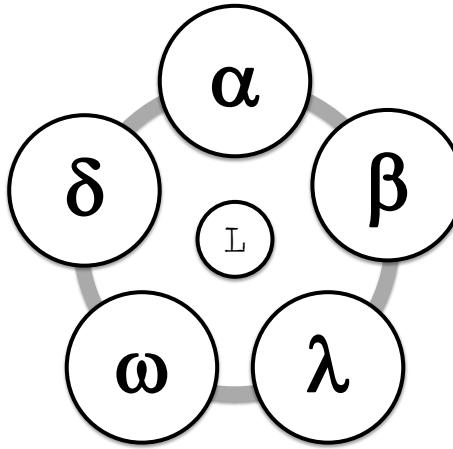
## semantics (generators)

## Tools and IDE

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

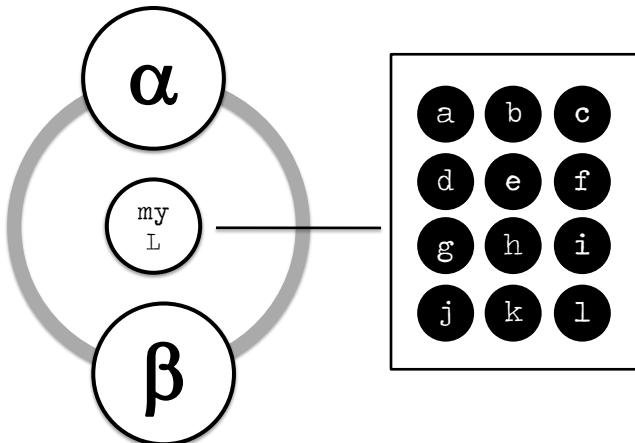


## Small Language



with a few, orthogonal  
and powerful concepts

## Modular Language



with many optional,  
composable modules

# Case Studies

```

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.rcfea->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

```

module main imports OsekKernel, EcAPI, BitLevelUtilities {

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_PORT_C, left, 1);
    nxt_motor_set_speed(MOTOR_PORT_T::NXT_PORT_B, right, 1);
}

```

Example  
Exten  
ded C

The screenshot shows a window titled "Capgemini Pension Workbench" with the file path "NNLCPA-14w2-21112008.x". The left sidebar contains a "Table of Contents" with sections like "Library", "Documentation", "Value sets", "Tag definitions", "Shared", and "Rules". The main pane displays a document with several mathematical formulas:

- 3.3 Commutatiegetallen op 1 levensjaar**

$$D_x = v \cdot \frac{l}{100} \approx 6 \text{ Dec (3)}$$

Implemented in = 1990.1

$$N_x = \sum_{t=0}^{\infty} D_{x+t} \approx 7 \text{ Dec (3)}$$
- 3.6 Contante waarde 1 levensjaar / 2 levensjaar**

$$E_x = \frac{l+n}{D_x} \approx 19 \text{ Dec (4)}$$

$$a_x = d - 1 \approx 21 \text{ Dec (3)}$$

$$\bar{a}_x = \bar{d}_x - 0,5 \approx 22 \text{ Dec (3)}$$

$$\bar{a}_{x|n} = \frac{N_x - N_{x+n}}{D_x} \approx 23 \text{ Dec (3)}$$

$$\bar{a}_{x|n} = \bar{a}_x - 0,5 + 0,5 \cdot E_x \approx 25 \text{ Dec (3)}$$
- 4 BN(\_ris) koopsommen**

At the bottom right of the main pane, there is a "Section" menu with options like "Edit", "Paragraph", "Text", "Dev", and "Doc | Splitter | Pension | PensionDecorated | AM".

Example  
Pension  
Plans

The screenshot shows a window titled "Capgemini Pension Workbench" with the file path "NNLCPA-14w2-21112008.x". The left sidebar contains a "Table of Contents" with sections like "Library", "Documentation", "Value sets", "Tag definitions", "Shared", and "Rules". The main pane displays a document with a table titled "Elements..." under the "Rules" section:

Name	Valid time	Transaction time	Fixture	Product	Element	Expected value	Actual value
Gelijke datums	03/01/2008		Mutatieperiode - Mutatiedatum = Mutatiedatum Vorig			3	0
Periode < 30	03/01/2008		Mutatieperiode - Mutatiedatum > Mutatiedatum Vorig (binnen 1 maand)			15	15
Periode > 30	03/01/2008		Mutatieperiode - Mutatiedatum > Mutatiedatum Vorig (mee erdere maanden)			60	60

Below the table, there is a note: "Bereken Mutatieperiode • Test cases • Unit test: Gelijke datums : ^Place Dev Doc | Splitter | Pension | PensionDecorated | AM".

Example  
Pension  
Plans

```
entity Post {
    key      :: String (id)
    blog     → Blog
    urlTitle :: String
    title    :: String (searchable)
    content  :: WikiText (searchable)
    public   :: Bool (default=false)
    authors  → Set<User>
    function isAuthor(): Bool {
        return principal() in authors
    }
    function mayEdit(): Bool {
        return isAuthor();
    }
    function mayView(): Bool {
        return public || mayEdit();
    }
}

access control rules
rule page post(p: Post, title: String) {
    p.mayView()
}
rule template newPost(b: Blog) {
    b.isAuthor()
}
section posts
define page post(p: Post, title: String) {
    title{ output(p.title) }
    bloglayout(p.blog){
        placeholder view { postView(p) }
        postComments(p)
    }
}
define permalink(p: Post) {
    navigate post(p, p.urlTitle) { elements }
}
```

### Example

Web  
DSL

# Terms & Concepts

## 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](http://www.answers.com/topic/model)

## Model

A representation of a set of components of a process, system, or subject area, generally developed for understanding, analysis, improvement, and/or replacement of the process

[www.ichnet.org/glossary.htm](http://www.ichnet.org/glossary.htm)

## Model

an abstraction or simplification of reality

[ecosurvey.gmu.edu/glossary.htm](http://ecosurvey.gmu.edu/glossary.htm)

## Model

which ones?

an abstraction or  
simplification of  
/ reality

what should  
we leave out?

[ecosurvey.gmu.edu/glossary.htm](http://ecosurvey.gmu.edu/glossary.htm)

## Model Purpose

- ... code generation
- ... analysis and checking
- ... platform independence
- ... stakeholder integration
- ...
- drives design of language!

## Model Purpose

- ... code generation
- ... analysis and checking
- ... platform independence
- ... stakeholder integration

Example

Components

## Model Purpose

- ... code generation
- ... analysis and checking
- ... platform independence
- ... stakeholder integration

Example

Refrige  
rators

## Model Purpose

- ... code generation
- ... analysis and checking
- ... platform independence
- ... stakeholder integration

Example

Exten  
ded C

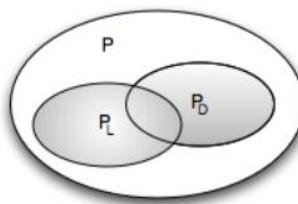
## Model Purpose

- ... code generation
- ... analysis and checking
- ... platform independence
- ... stakeholder integration

Example

Pension  
Plans

# Programs Languages Domains



body of  
knowledge  
in the real  
world

deductive  
top down

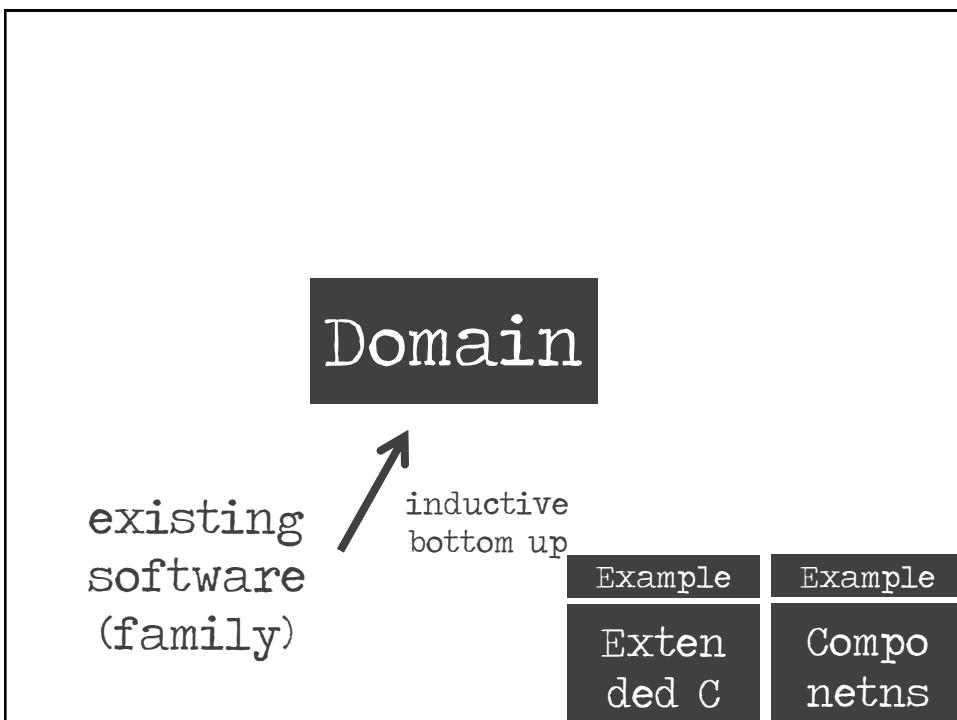
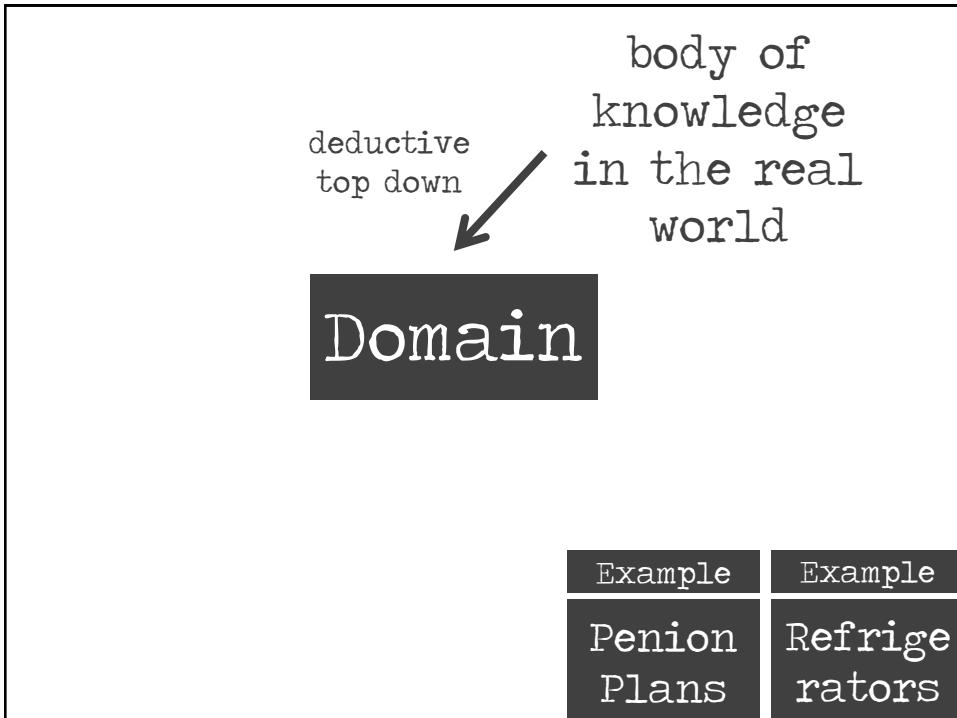
↓

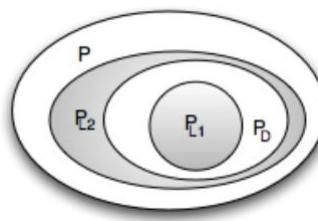
Domain

existing  
software  
(family)

inductive  
bottom up

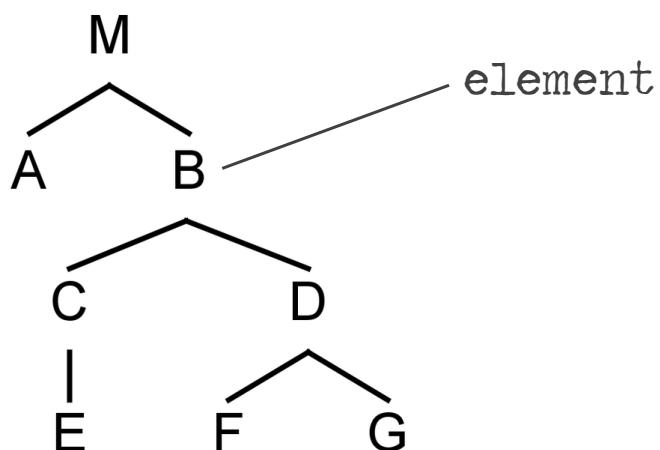
↑



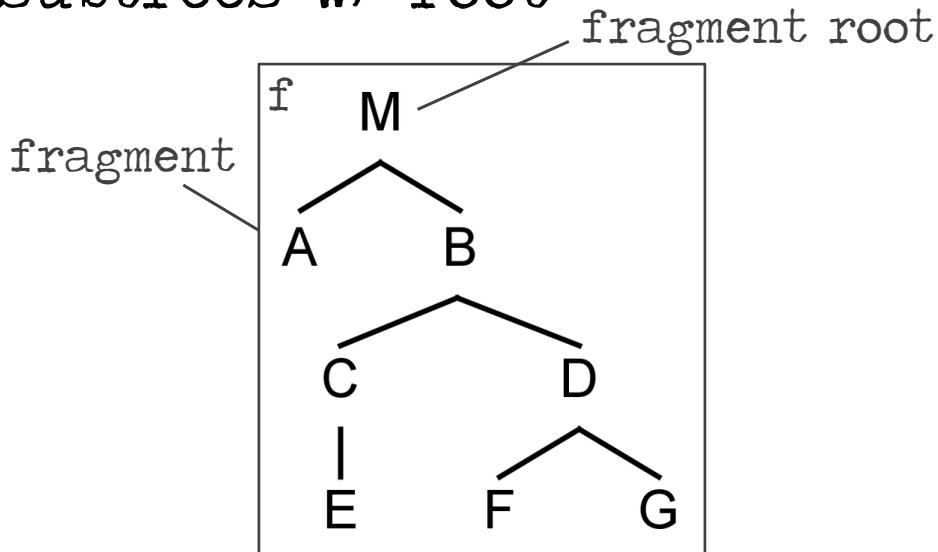


A DSL  $L_D$  for  $D$  is a language that is specialized to encoding  $P_D$  programs.  
more efficient  
smaller

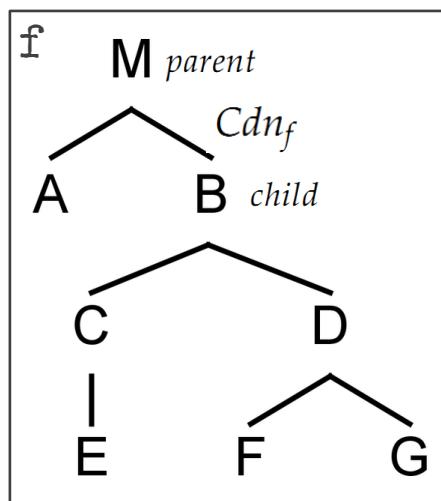
Programs  
are trees



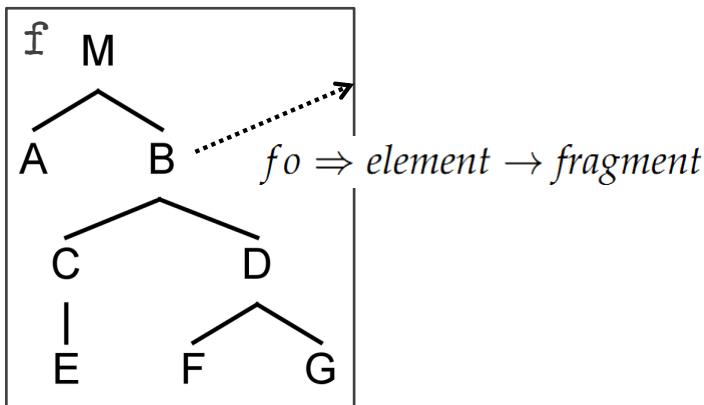
Fragments are subtrees w/ root



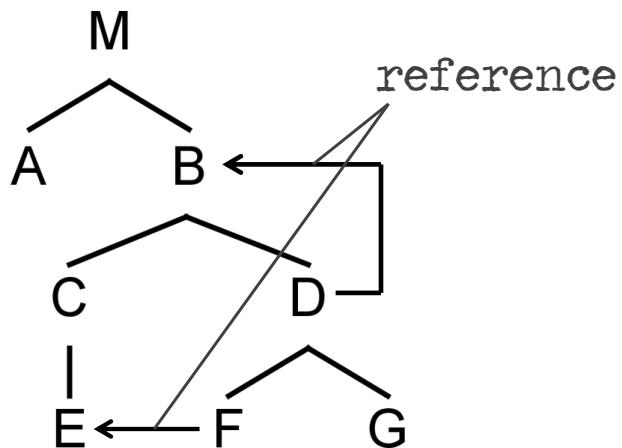
Parent-Child Relation



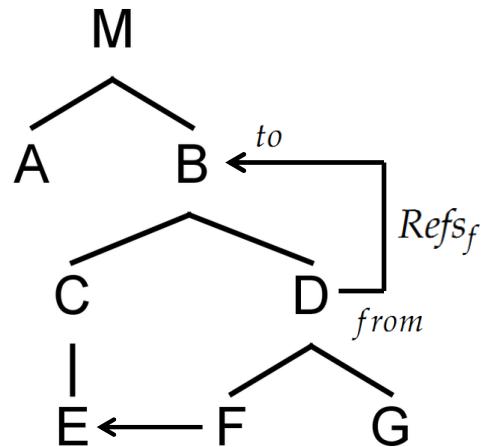
# Programs and Fragments



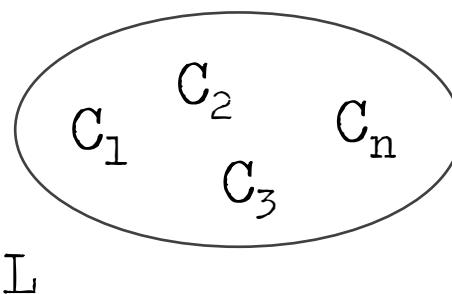
Programs are  
graphs, really.



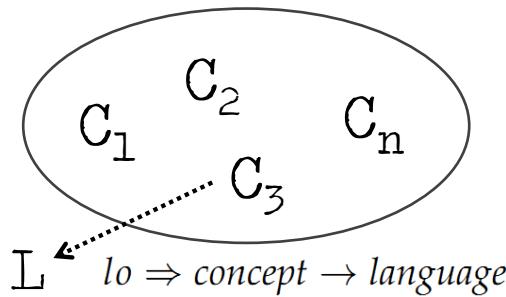
Programs are graphs, really.



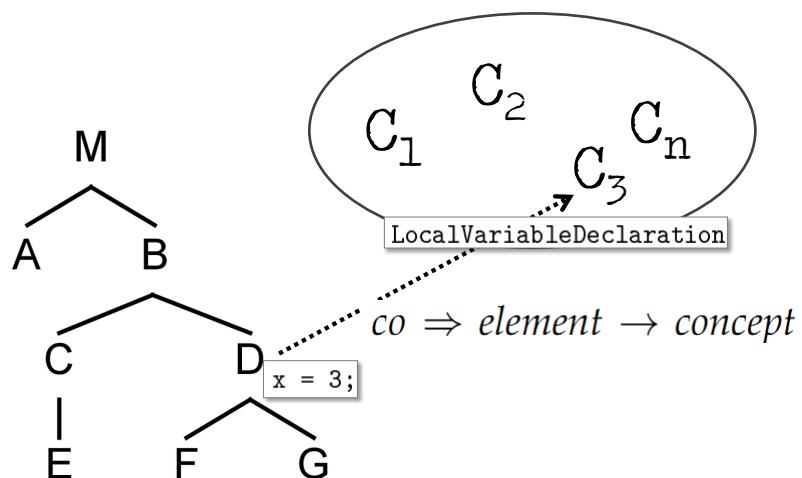
Languages are sets of concepts



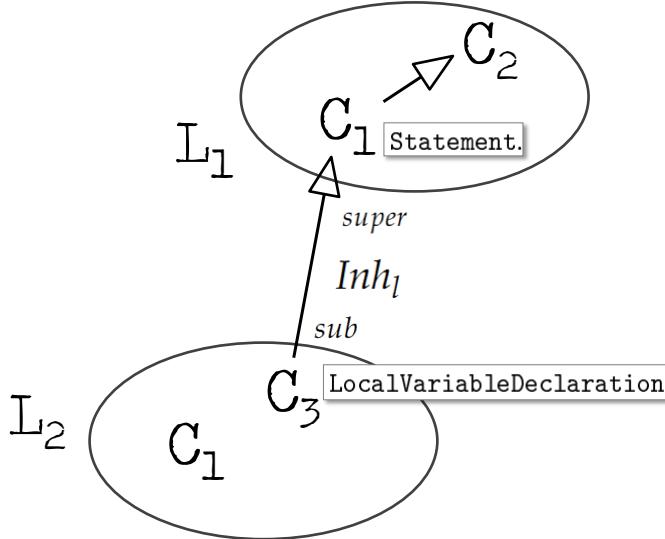
Languages are sets of concepts



Programs  
and languages



# Language: concept inheritance



Language  
does not depend on  
any other language

$$\begin{aligned}\forall r \in Refs_l \mid lo(r.to) = lo(r.from) = l \\ \forall s \in Inh_l \mid lo(s.super) = lo(s.sub) = l \\ \forall c \in Cdn_l \mid lo(c.parent) = lo(c.child) = l\end{aligned}$$

Independence

Fragment  
does not depend on  
any other fragment

$$\begin{aligned}\forall r \in Refs_f \mid fo(r.to) = fo(r.from) = f \\ \forall e \in E_f \mid lo(co(e)) = l\end{aligned}$$

# Independence

**Hardware:**

```
compressor compartment cc {
    static compressor c1
    fan ccfan
}
```



**Cooling Algorithm**

```
macro kompressorAus {
    set cc.c1->active = false
    perform ccfanabschalttask after 10 {
        set cc.ccfan->active = false
    }
}
```

**Example**

Refrige  
rators

# Homogeneous

**Fragment**  
everything expressed  
with one language

$$\forall e \in E_f \mid lo(e) = l$$

$$\forall c \in Cdn_f \mid lo(c.parent) = lo(c.child) = l$$

**Heterogeneous**

```

module CounterExample from countered imports nothing {

    var int theI;

    var boolean theB;

    var boolean hasBeenCalled;

    statemachine Counter {
        in start() <no binding>
            step(int[0..10] size) <no binding>
        out someEvent(int[0..100] x, boolean b) <no binding>
            resetted() <no binding>
        vars int[0..10] currentVal = 0
            int[0..100] LIMIT = 10
        states (initial = initialState)
            state initialState {
                on start [ ] -> countState { send someEvent(100, true && false || true); }
            }
            state countState {
                on step [currentVal + size > LIMIT] -> initialState { send resetted(); }
                on step [currentVal + size <= LIMIT] -> countState { currentVal = currentVal + size; }
                on start [ ] -> initialState [ ]
            }
        } end statemachine

        var Counter c1;

        exported test case test1 {
            init(c1);
            assert(0) isInState<c1, initialState>;
            trigger(c1, start);
            assert(1) isInState<c1, countState>;
        } test1(test case)
    }
}

```

**Example**

**Extended C**

**Heterogeneous**

**C**

**Statemachines**

**Testing**

```

module CounterExample from countered imports nothing {

    var int theI;

    var boolean theB;

    var boolean hasBeenCalled;

    statemachine Counter {
        in start() <no binding>
            step(int[0..10] size) <no binding>
        out someEvent(int[0..100] x, boolean b) <no binding>
            resetted() <no binding>
        vars int[0..10] currentVal = 0
            int[0..100] LIMIT = 10
        states (initial = initialState)
            state initialState {
                on start [ ] -> countState { send someEvent(100, true && false || true); }
            }
            state countState {
                on step [currentVal + size > LIMIT] -> initialState { send resetted(); }
                on step [currentVal + size <= LIMIT] -> countState { currentVal = currentVal + size; }
                on start [ ] -> initialState [ ]
            }
        } end statemachine

        var Counter c1;

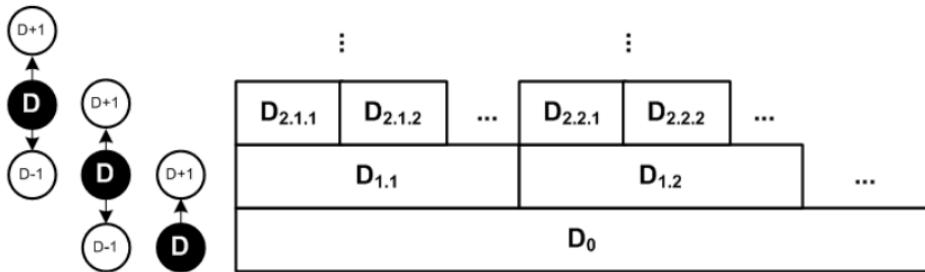
        exported test case test1 {
            init(c1);
            assert(0) isInState<c1, initialState>;
            trigger(c1, start);
            assert(1) isInState<c1, countState>;
        } test1(test case)
    }
}

```

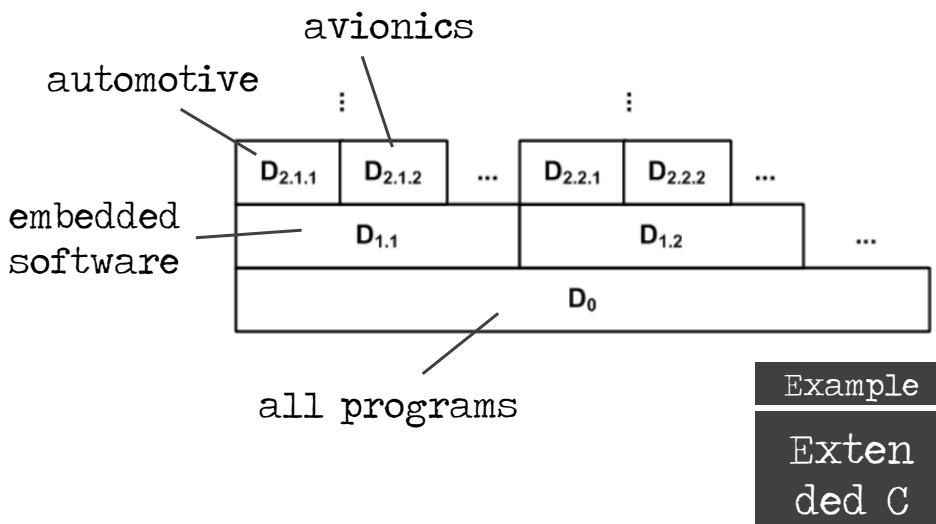
**Example**

**Extended C**

# Domain Hierarchy



# Domain Hierarchy



# Design Dimensions

expressivity  
coverage  
semantics  
separation of  
concerns

completeness  
paradigms  
modularity  
concrete  
syntax

---

process

# Expressivity

expressivity  
coverage  
semantics  
separation of  
concerns

completeness  
paradigms  
modularity  
concrete  
syntax

process

## Shorter Programs

---

More  
Accessible  
Semantics

For a limited  
Domain!

---

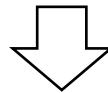
Domain Knowledge  
encapsulated in  
language

Def: Expressivity

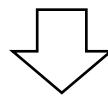
A language  $L_1$  is *more expressive in domain D*  
than a language  $L_2$

if for each  $p \in P_D \cap P_{L_1} \cap P_{L_2}$ ,  $|p_{L_1}| < |p_{L_2}|$

Smaller Domain

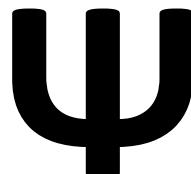


More Specialized  
Language



Shorter Programs

The  
do-what-I-want  
language

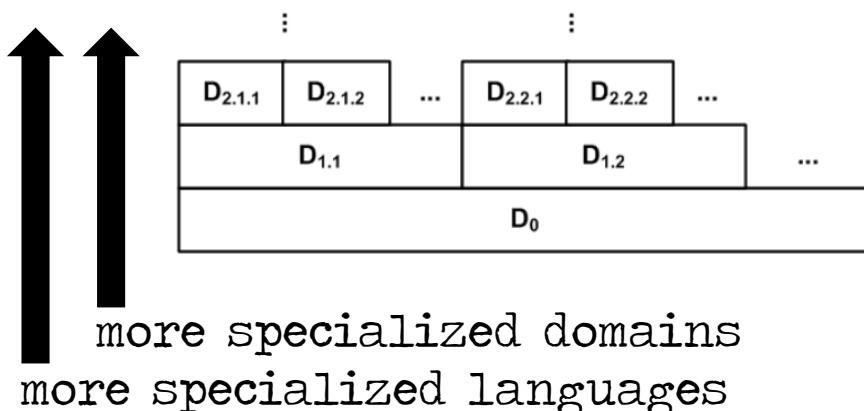


# Single Program vs. Class/Domain

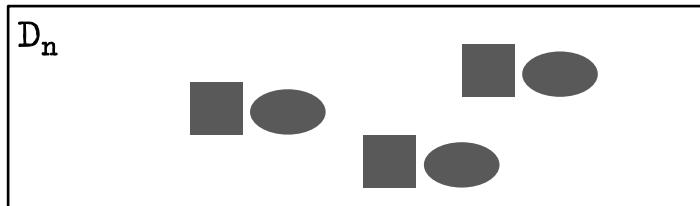
## No Variability!

Ψ

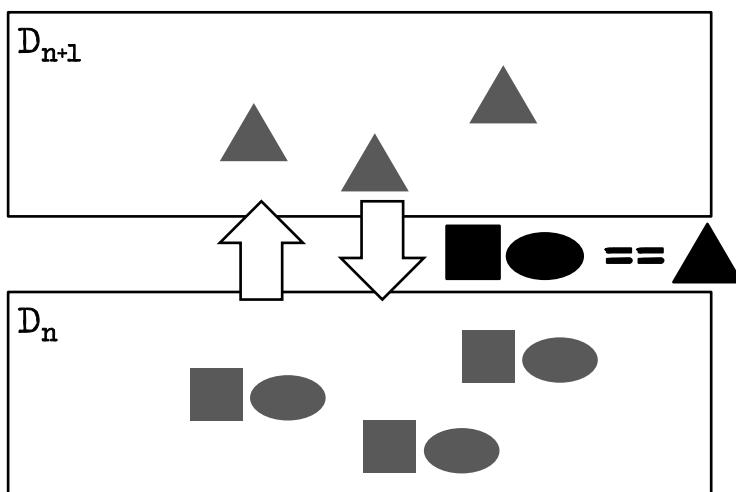
### Domain Hierarchy



# Reification

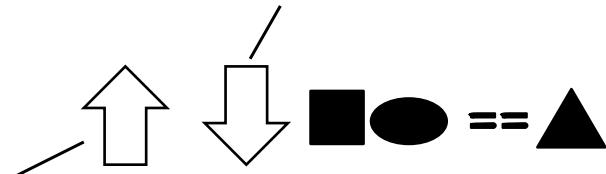


# Reification



# Reification

Transformation/  
Generation



Language  
Definition

```
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] );
}
```

## Linguistic Abstraction

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



**Declarative!**  
**Directly represents Semantics.**

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

## Def: DSL

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

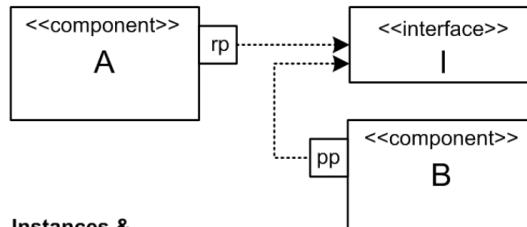
## Def: DSL cont'd

A good DSL does **not** require the use of patterns and idioms to express **semantically interesting** concepts in D.

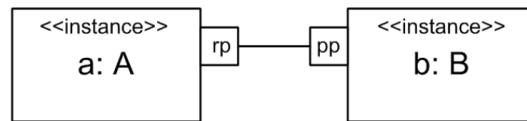
Processing tools do not have to do "semantic recovery" on D programs.

**Declarative!**

## Another Example



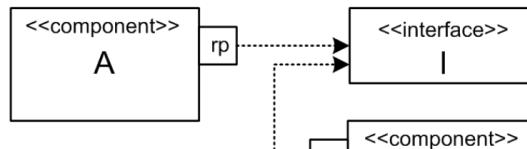
Instances &  
Connectors



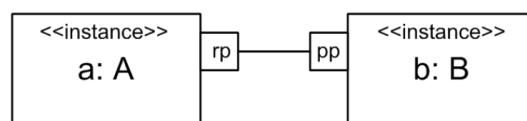
```
if (isConnected(port)) {
    port.doSomething();
}
```

Example  
Exten  
ded C

## Another Example



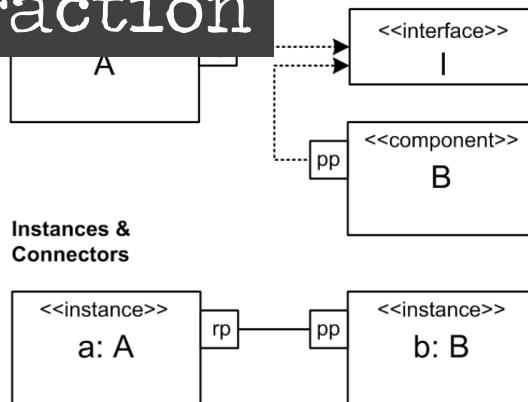
Instances &  
Connectors



```
if (isConnected(port) || true) {
    port.doSomething();
}
Turing Complete!
Requires Semantic Analysis!
```

Example  
Exten  
ded C

# Linguistic Abstraction



Instances &amp; Connectors

```
with_port (port) {
    port.doSomething();
}
```

Example  
Extended C

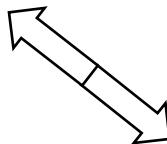
# Linguistic Abstraction

```
exported component AnotherDriver extends Driver {
    ports:
        requires optional ILogger logger
        provides IDriver cmd
    contents:
        field int count = 0

        int setDriverValue(int addr, int value) <- op cmd.setDriverValue {
            with port (logger) {
                logger.log("some error message");
            } with port
            return 0;
        }
}
```

Example  
Extended C

# Linguistic Abstraction



## In-Language Abstraction

Libraries  
Classes  
Frameworks

# Linguistic Abstraction

Analyzable  
Better IDE Support

## In-Language Abstraction

User-Definable  
Simpler Language

# Linguistic Abstraction

Analyzable  
Better IDE Support

Special Treatment!



# In-Language Abstraction

User-Definable  
Simpler Language

# Linguistic Abstraction

Std Lib

# In-Language Abstraction

# Std Lib

```
lib stdlib {  
    command compartment::coolOn  
    command compartment::coolOff  
    property compartment::totalRuntime: int readonly  
    property compartment::needsCooling: bool readonly  
    property compartment::couldUseCooling: bool readonly  
    property compartment::targetTemp: int readonly  
    property compartment::currentTemp: double readonly  
    property compartment::isCooling: bool readonly  
}
```

Example  
Refrige  
rators

# Language Evolution Support

Customization  
vs.  
Configuration

Precision  
vs.  
Algorithmics

# Coverage

expressivity  
coverage  
semantics  
separation of  
concerns

completeness  
paradigms  
modularity  
concrete  
syntax

---

process

Domain  $D_L$  defined  
inductively by  $L$

(the domain that can be expressed by  $L$ )

$C_L(L) == 1$  (by definition)

not very interesting!

Def: Coverage

to what extend can a  
language  $L$  cover a domain  $D$

$$C_D(L) = \frac{\text{number of } P_D \text{ programs expressable by } L}{\text{number of programs in domain } D}$$

## Def: Coverage

why would  $C_D(L)$  be != 1?

- 1) L is deficient
- 2) L is intended to cover only a subset of D,  
corner cases may make L too complex

Rest must be expressed in  $D_{-1}$

## Def: Coverage

Coverage is full.  
You call always write C.

Example

Exten  
ded C

## Def: Coverage

Only a particular style of web apps are supported.

Many more are conceivable.

Example

WebDSL

## Def: Coverage

DSLs are continuously evolved so the relevant parts of the deductive domain are supported.

Example

Example

Example

Components

Refrigerators

Pension Plans

# Semantics & Execution

expressivity  
coverage  
semantics  
separation of  
concerns

completeness  
paradigms  
modularity  
concrete  
syntax

process

Static Semantics

---

Execution Semantics

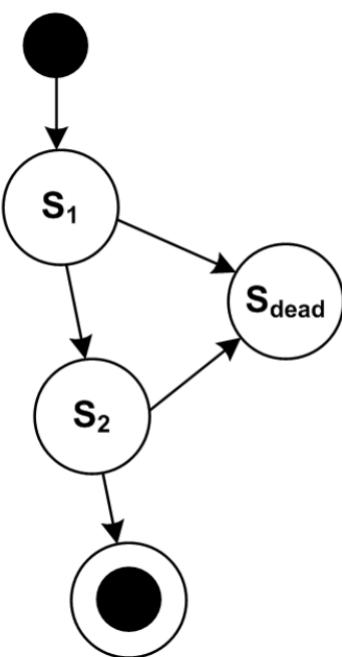
Static Semantics

---

Execution Semantics

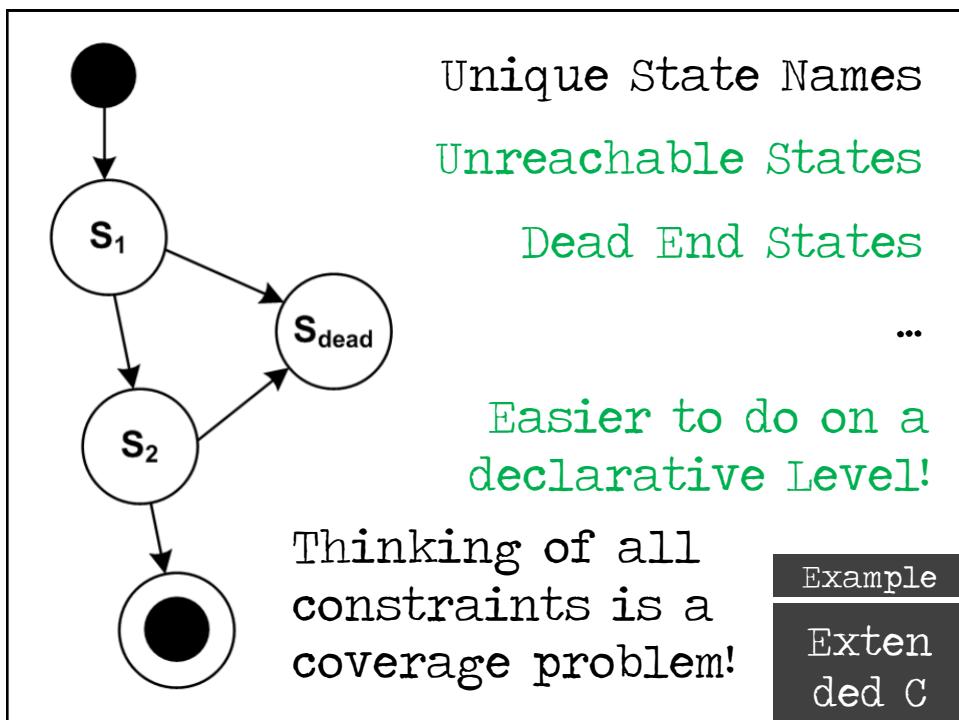
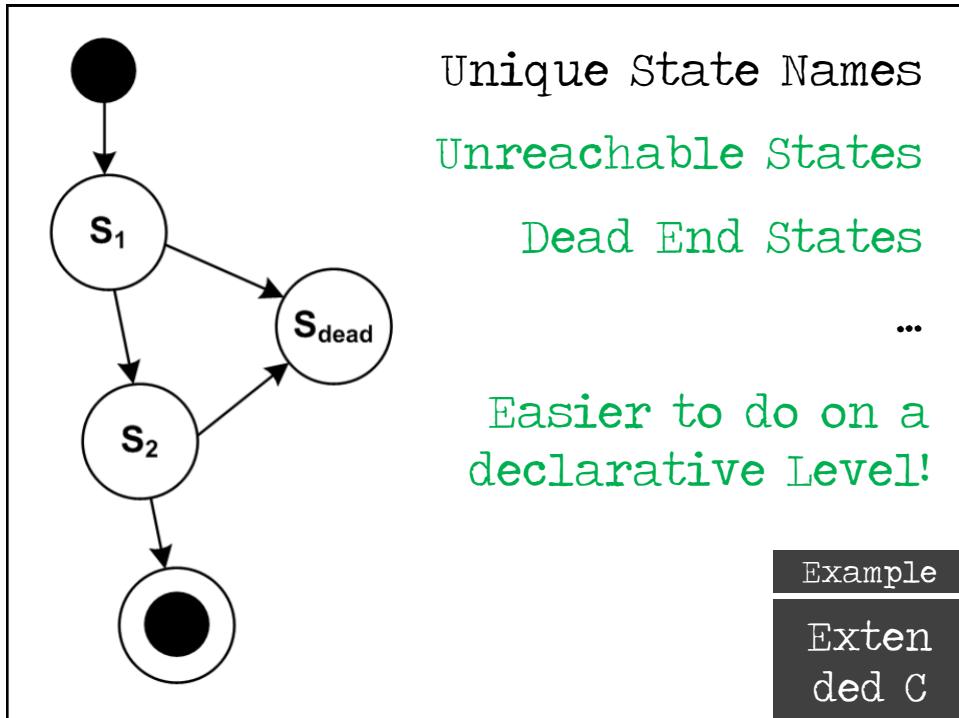
## Static Semantics

Constraints  
Type Systems



Unique State Names  
Unreachable States  
Dead End States  
...

Example  
Extended C



```
var int x = 2 * someFunction(sqrt(2));
```

Assign fixed types

What does a type system do?

```
var int x = 2 * someFunction(sqrt(2));
```

Assign fixed types

Derive Types

What does a type system do?

```
var int x = 2 * someFunction(sqrt(2));
```

Derive Types

Assign fixed types

Calculate Common Types

What does a type system do?

```
var int x = 2 * someFunction(sqrt(2));
```

Derive Types

Assign fixed types

Calculate Common Types

Check Type Consistency

What does a type system do?

## Intent + Check

```
var int x = 2 *  
    someFunction(sqrt(2));
```

More code

Better error messages

Better Performance

## Derive

```
var x = 2 * some  
    Function(sqrt(2));
```

More convenient

More complex checkers

Harder to understand for users

```
macro kompressorAus {  
    set cc.c1->active = "aString"  
    perform ccfanausschalten  
        set cc.ccfan->active  
    }  
}
```

Incompatible type BoolType|COMPARABLE and StringType (on a AssignmentStatement)

### Example

Refrigerators

What does it  
all mean?

## Execution Semantics

### Def: Semantics

... via mapping to lower level

$$\text{semantics}(p_{L_D}) := q_{L_{D-1}}$$

where  $OB(p_{L_D}) == OB(q_{L_{D-1}})$

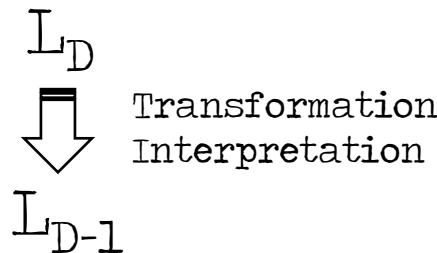
OB: Observable Behaviour (Test Cases)

## Def: Semantics

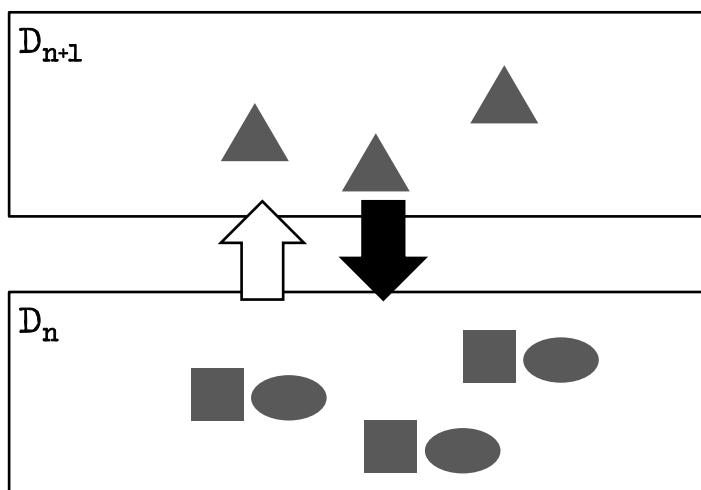
... via mapping to lower level

$$\text{semantics}(p_{L_D}) := q_{L_{D-1}}$$

$$\text{where } OB(p_{L_D}) == OB(q_{L_{D-1}})$$



## Transformation

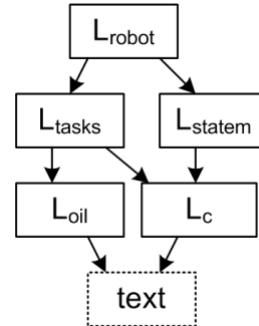


# Transformation

```
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

# Transformation

$L_D$

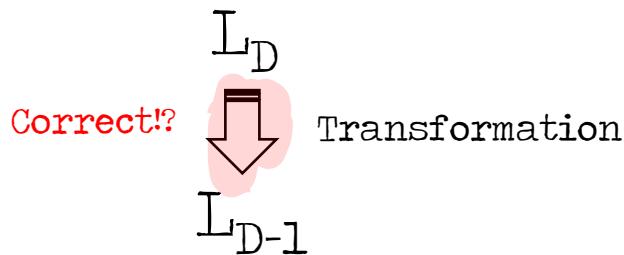


Transformation

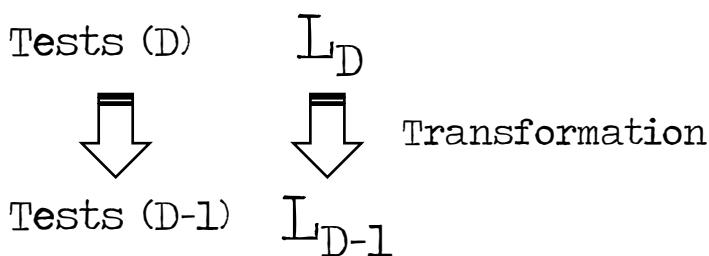
$L_{D-1}$

Known Semantics!

## Transformation



## Transformation



Run tests on both levels; all pass.  
Coverage Problem!

```

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.rcfan->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

```

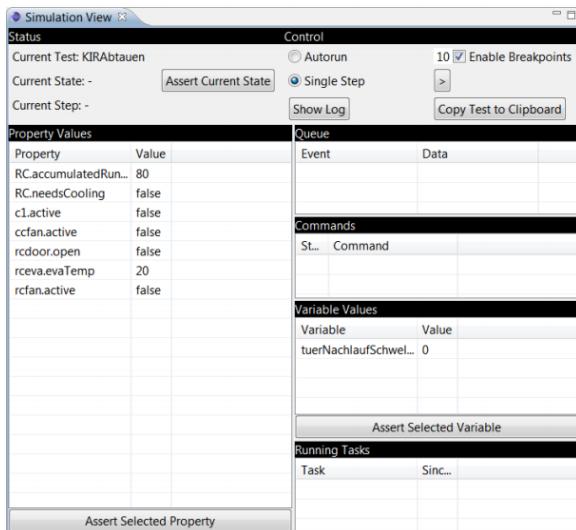
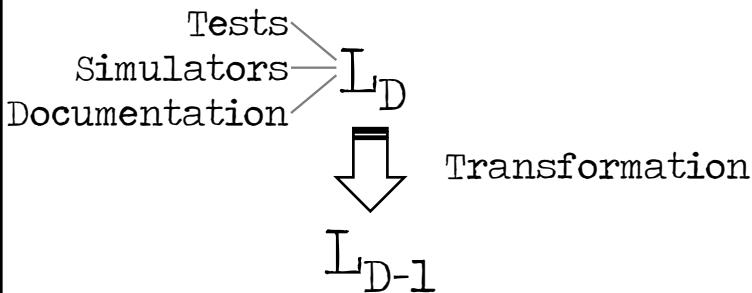
Example  
Refrige  
rators

## Transformation

Name	Documentation	Tags	Valid time	Transaction time	Fixture	Product	Element	Expected value	Actual value
Accrued right at retirement		⊕	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 retirement 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	

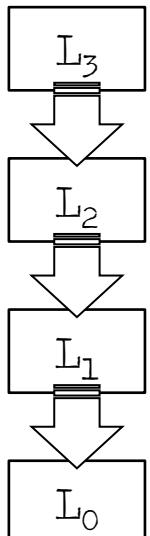
Example  
Pension  
Plans

# Transformation



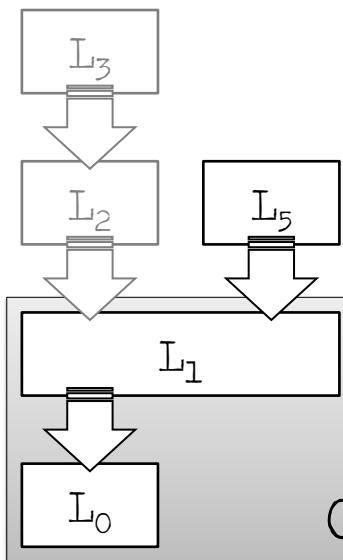
Example  
Refrige  
rators

## Multi-Stage



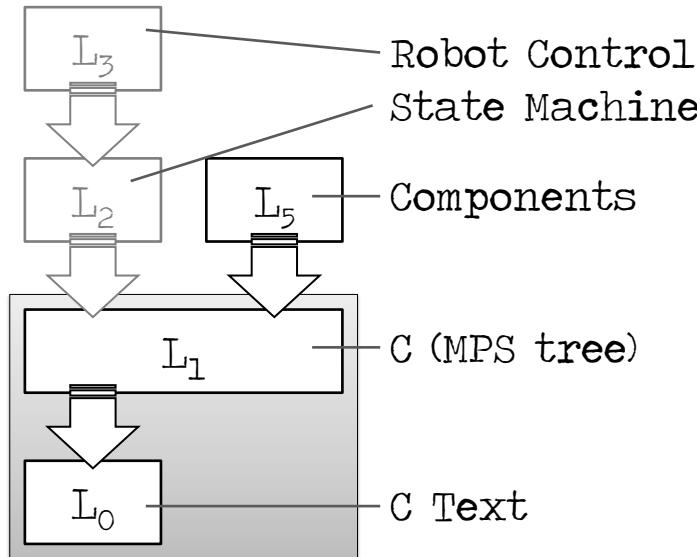
Modularization

## Multi-Stage: Reuse



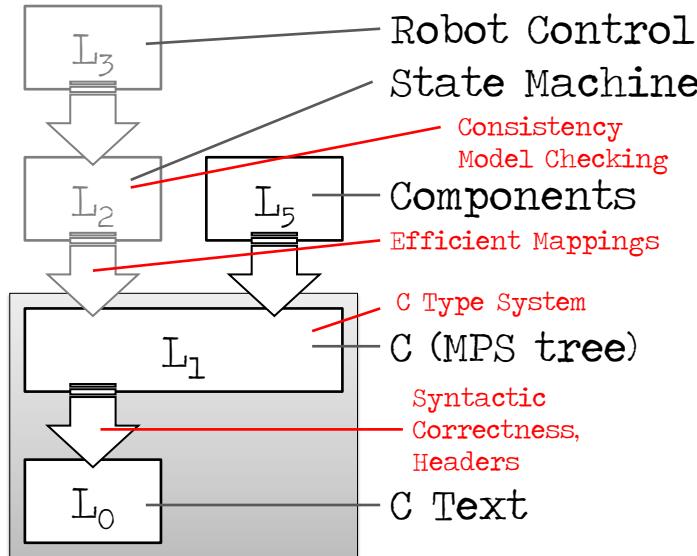
Reusing  
Later Stages  
Optimizations!

## Multi-Stage: Reuse



Example  
Extended C

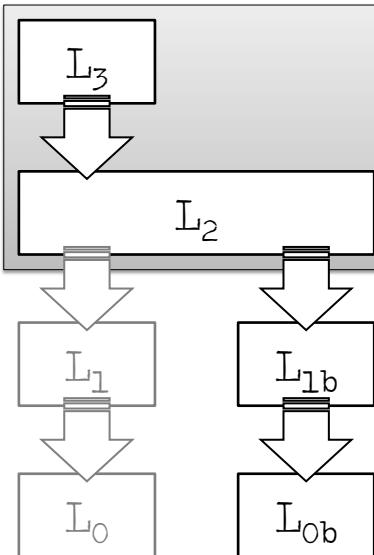
## Multi-Stage: Reuse



Example  
Extended C

## Multi-Stage: Reuse

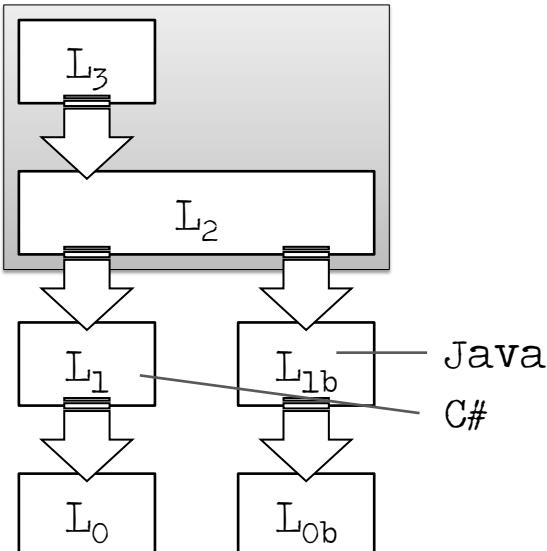
Reusing  
Early Stages  
Portability



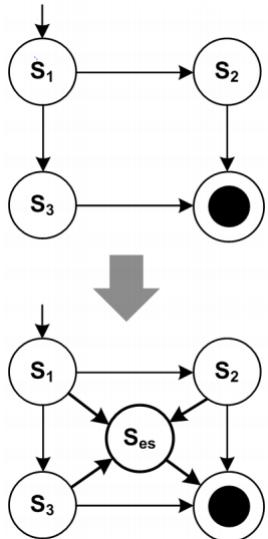
## Multi-Stage: Reuse

Java  
C#

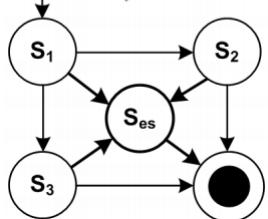
Example  
Extended C



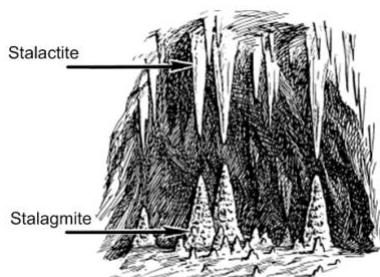
## Multi-Stage: Preprocess



Adding an optional, modular emergency stop feature



## Platform



Generated Application

Domain Frameworks

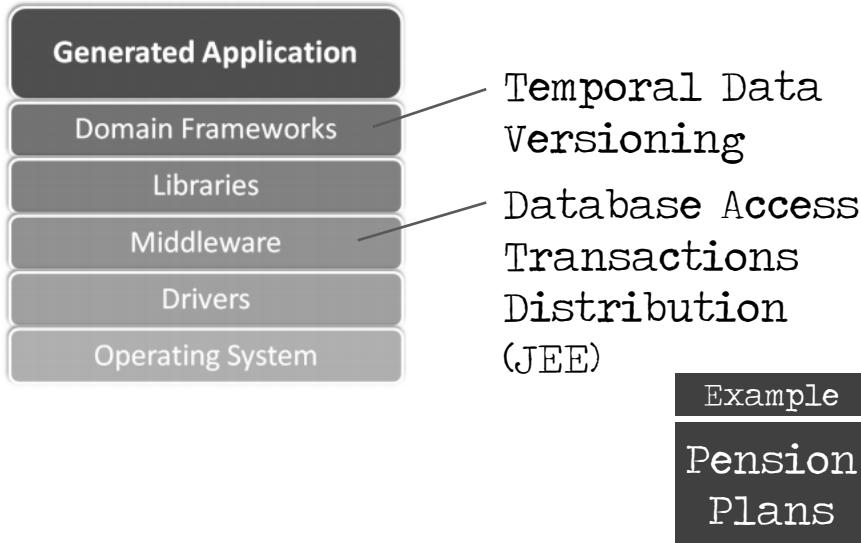
Libraries

Middleware

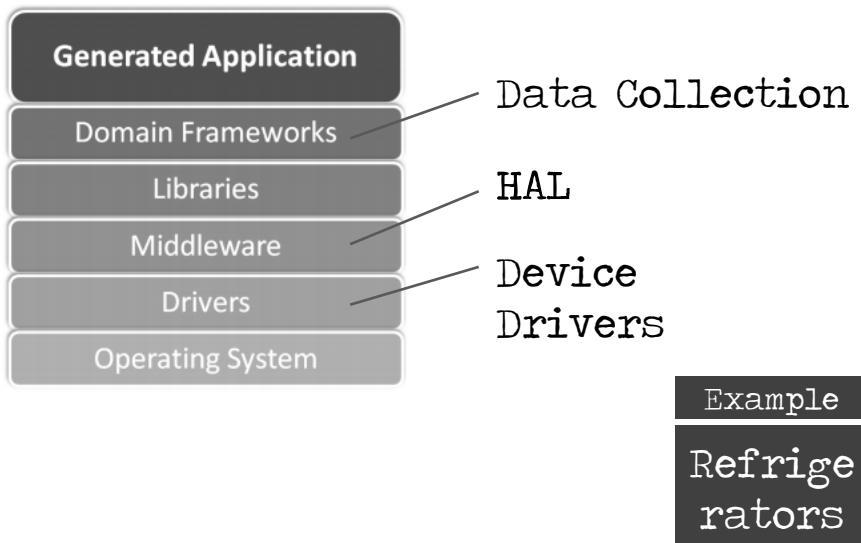
Drivers

Operating System

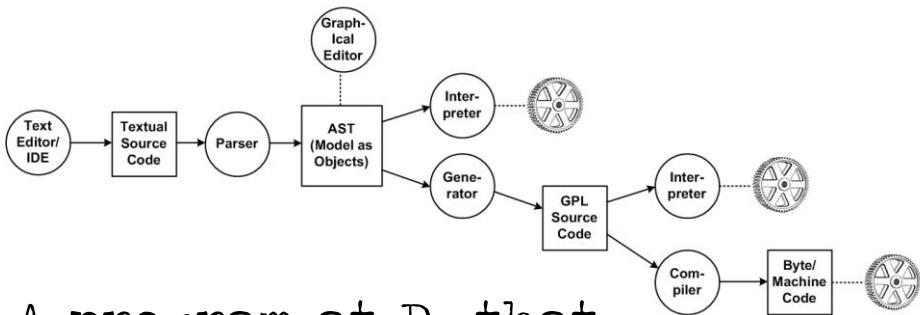
# Platform



# Platform



# Interpretation



A program at  $D_0$  that acts on the structure of an input program at  $D_{>0}$

# Interpretation

A program at  $D_0$  that acts on the structure of an input program at  $D_{>0}$

imperative → step through  
 functional → eval recursively  
 declarative → ? solver ?

Name	Documentation	Tags	Valid time	Transaction time	Fixture	Product	Element	Expected value	Actual value
Accrued right at retirement			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 retirement 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

```

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

```

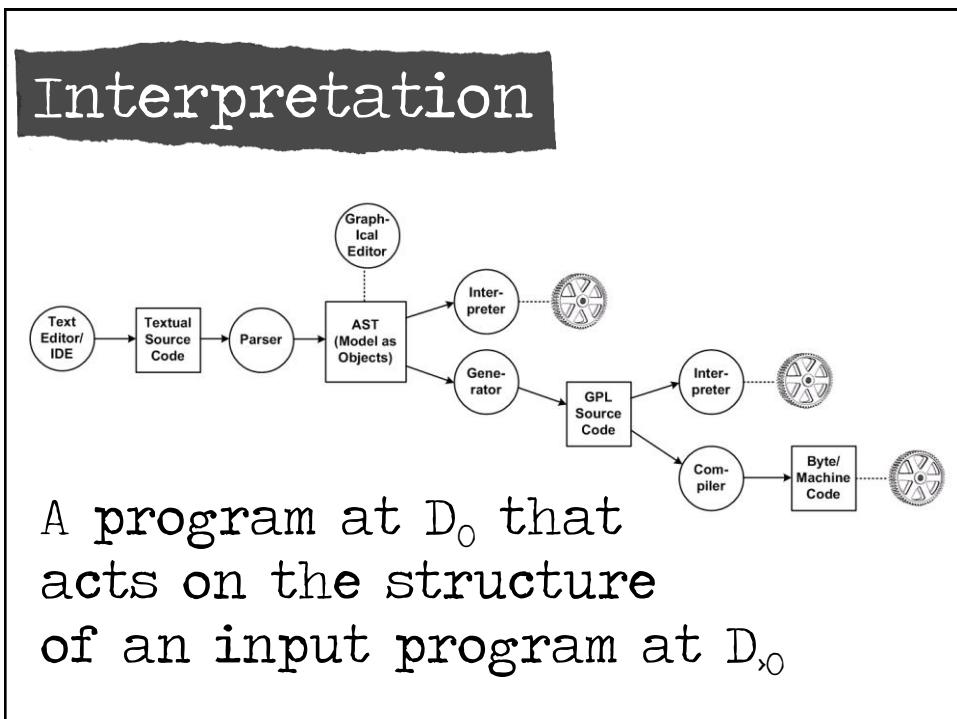
## Example Refrige rators

The screenshot shows the 'Simulation View' window with the following interface elements:

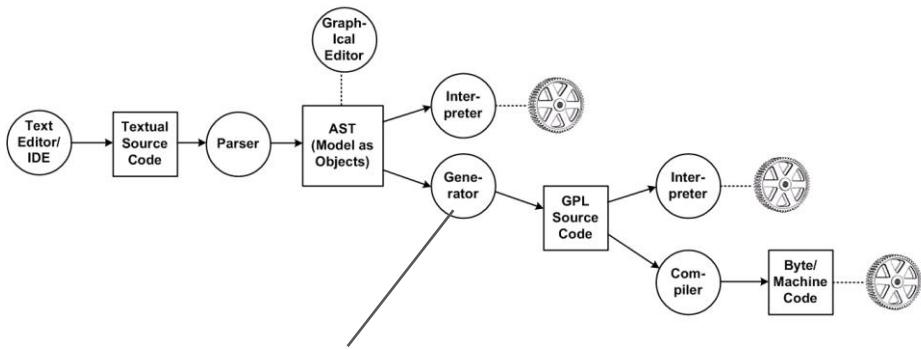
- Status:** Current Test: KIRAbtauern
- Control:**
  - Radio buttons: Autorun (unchecked), Single Step (checked), 10 (unchecked), Enable Breakpoints (checked).
  - Buttons: Assert Current State, Show Log, Copy Test to Clipboard.
- Property Values:** A table showing properties and their values:
 

Property	Value
RC.accumulatedRun...	80
RC.needsCooling	false
c1.active	false
ccfan.active	false
rctdoor.open	false
rceva.evaTemp	20
rcfan.active	false
- Queue:** A table with columns Event and Data, currently empty.
- Commands:** A table with columns St... and Command, currently empty.
- Variable Values:** A table with columns Variable and Value, currently empty.
- Assert Selected Variable:** A button.
- Running Tasks:** A table with columns Task and Sinc..., currently empty.
- Buttons at the bottom:** Assert Selected Property, Assert Selected Variable.

**Example Refrigerators**



# Interpretation



An interpreter :-)

Transformation

Interpretation



Transformation	Interpretation
+ Code Inspection + Debugging	

Transformation	Interpretation
+ Code Inspection + Debugging Platform Interactions	
	Example Refrigerators

Transformation	Interpretation
<ul style="list-style-type: none"><li>+ Code Inspection</li><li>+ Debugging</li><li>+ Performance &amp; Optimization</li></ul>	

Transformation	Interpretation
+ Code Inspection	
+ Debugging	
+ Performance & Optimization	
Efficiency for Real-Time S/w	
Memory Use	Example Extended C

Transformation	Interpretation
<ul style="list-style-type: none"><li>+ Code Inspection</li><li>+ Debugging</li><li>+ Performance &amp; Optimization</li><li>+ Platform Conformance</li></ul>	

Transformation	Interpretation
<ul style="list-style-type: none"><li>+ Code Inspection</li><li>+ Debugging</li><li>+ Performance &amp; Optimization</li><li>+ Platform Conformance</li></ul> <p>Web Frameworks and Standards</p>	<p>Example</p> <p>Web DSL</p>

Transformation	Interpretation
<ul style="list-style-type: none"><li>+ Code Inspection</li><li>+ Debugging</li><li>+ Performance &amp; Optimization</li><li>+ Platform Conformance</li></ul>	<ul style="list-style-type: none"><li>+ Turnaround Time</li></ul>

Transformation	Interpretation				
<ul style="list-style-type: none"><li>+ Code Inspection</li><li>+ Debugging</li><li>+ Performance &amp; Optimization</li><li>+ Platform Conformance</li></ul>	<ul style="list-style-type: none"><li>+ Turnaround Time</li></ul>				
	<p>Testing</p> <table border="1"><thead><tr><th>Example</th><th>Example</th></tr></thead><tbody><tr><td>Pension Plans</td><td>Refrigerators</td></tr></tbody></table>	Example	Example	Pension Plans	Refrigerators
Example	Example				
Pension Plans	Refrigerators				

Transformation	Interpretation
<ul style="list-style-type: none"><li>+ Code Inspection</li><li>+ Debugging</li><li>+ Performance &amp; Optimization</li><li>+ Platform Conformance</li></ul>	<ul style="list-style-type: none"><li>+ Turnaround Time</li><li>+ Runtime Change</li></ul>

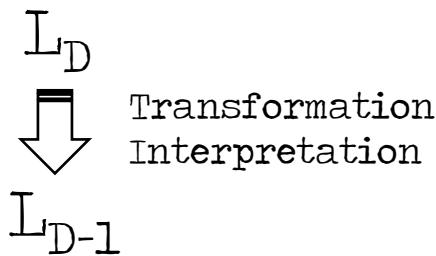
Transformation	Interpretation
<ul style="list-style-type: none"><li>+ Code Inspection</li><li>+ Debugging</li><li>+ Performance &amp; Optimization</li><li>+ Platform Conformance</li></ul>	<ul style="list-style-type: none"><li>+ Turnaround Time</li><li>+ Runtime Change</li><li>Change Business Rules without Redeployment</li></ul>

Example

Pension  
Plans

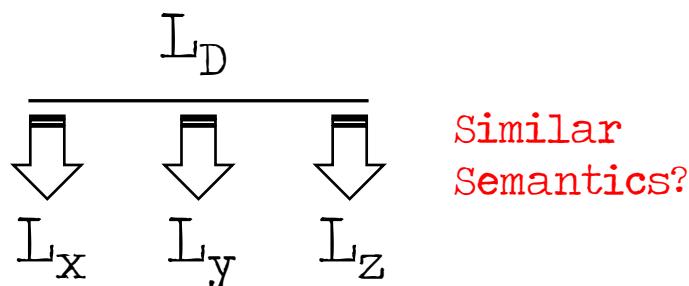
## Def: Semantics

... via mapping to lower level



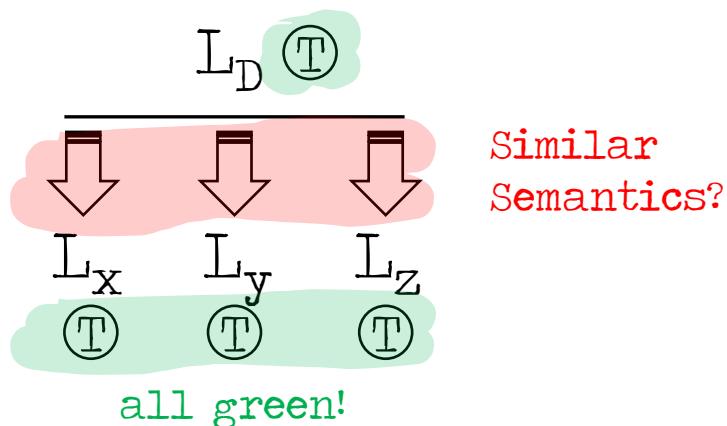
## Multiple Mappings

... at the same time



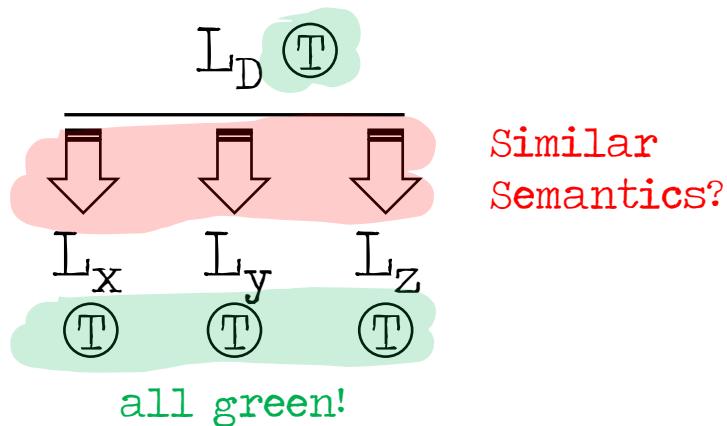
# Multiple Mappings

... at the same time



# Multiple Mappings

... at the same time



Name	Documentation	Tags	Valid time	Transaction time	Fixture	Product	Element	Expected value	Actual value
Accrued right at retirement			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 retirement (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	

Example

## Pension Plans

```

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 abtauend
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

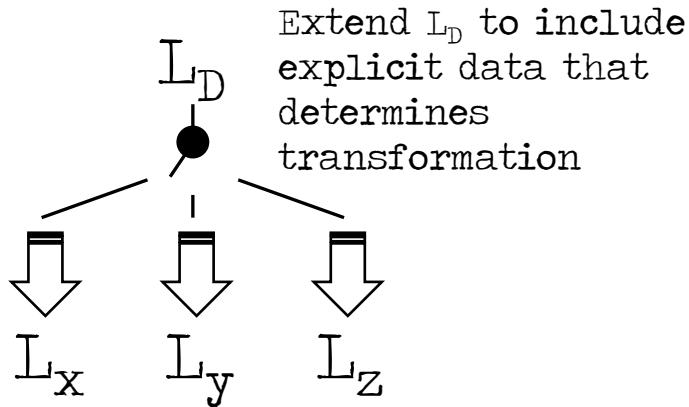
```

Example

## Refrigerators

# Multiple Mappings

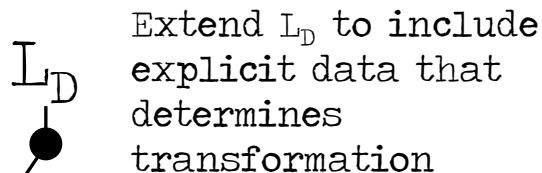
... alternatively, selectively



Extend  $L_D$  to include explicit data that determines transformation

# Multiple Mappings

... alternatively, selectively



Extend  $L_D$  to include explicit data that determines transformation

```
exported component AnotherDriver extends Driver {
    ports:
        provides IDiagnostic diag
        requires optional ILogger logger
        requires ILowLevel lowlevel restricted to LowLevelCode.ll
    contents:
        field int8_t count = 0
```

$x$        $y$        $z$

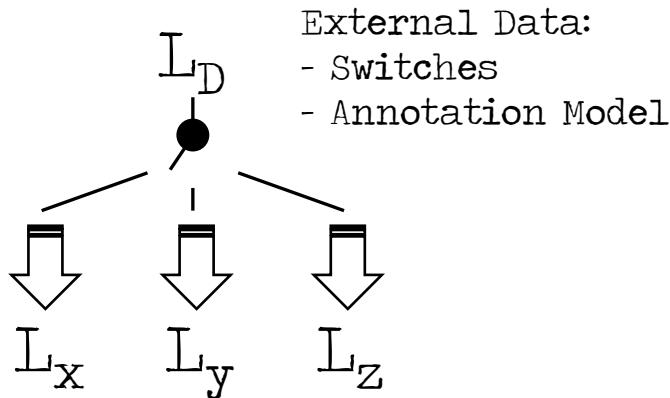
Example

Restricted Port leads  
to reduced overhead C

Exten  
ded C

# Multiple Mappings

... alternatively, selectively

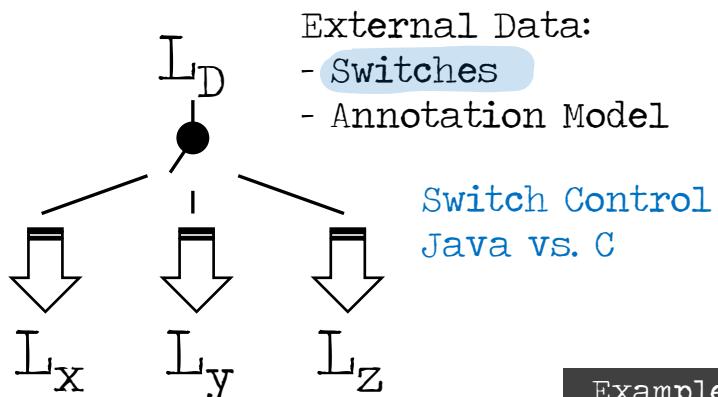


External Data:

- Switches
- Annotation Model

# Multiple Mappings

... alternatively, selectively



External Data:

- **Switches**
- Annotation Model

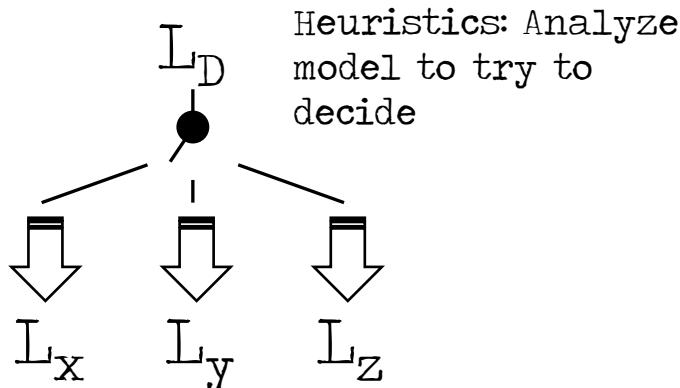
Switch Control  
Java vs. C

Example

Pension  
Plans

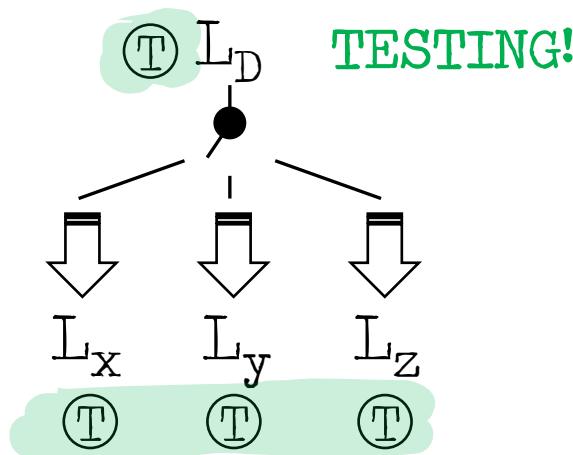
## Multiple Mappings

... alternatively, selectively



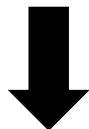
## Multiple Mappings

... alternatively, selectively



## Reduced Expressiveness

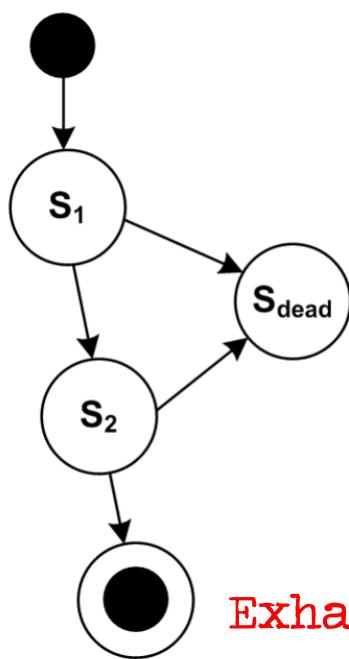
bad?	maybe.
good?	maybe!



Model Checking

SAT Solving

**Exhaustive Search, Proof!**



Unique State Names

Unreachable States

Dead End States

Guard Decidability

Reachability

**Exhaustive Search, Proof!**

**core.dev - [L:\wes-assembla\mbeddr\code\languages\com.mbeddr.core] - semaphoreSemaphore - JetBrains MPS 2.0.2**

File Edit Search View Go To Code Build Run Tools Version Control Window Help

TypeSizeConfiguration Semaphor

```

14 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;
            }
        }
    }
}

```

NUML Tool

Property	Status	Trace Size
State 'Init' can be live	SUCCESS	
State 'green' can be live	SUCCESS	
State 'Yellow' can be live	SUCCESS	
State 'Red' can be live	SUCCESS	
State 'green' has value	SUCCESS	
State 'Yellow' has value	SUCCESS	
State 'Red' has value	SUCCESS	
Variable 'counter' has value	SUCCESS	
Variable 'green_val' has value	SUCCESS	
Variable 'yellow_val' has value	SUCCESS	
Variable 'red_val' has value	SUCCESS	
Step 'step(-10..10)' has value	FAIL	3
State 'Init' is live	SUCCESS	
State 'green' can be live	SUCCESS	
State 'Yellow' can be live	SUCCESS	
State 'Red' can be live	SUCCESS	
State 'green' has value	SUCCESS	
State 'Yellow' has value	SUCCESS	
State 'Red' has value	SUCCESS	
Variable 'counter' has value	SUCCESS	
Variable 'green_val' has value	SUCCESS	
Variable 'yellow_val' has value	SUCCESS	
Variable 'red_val' has value	SUCCESS	
Transition 0 of state	SUCCESS	
Transition 1 of state	FAIL	1
Transition 0 of state	FAIL	1

Node Value

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	

Example  
Extended C

```

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     |


        }
}

```

Example  
Extended C

# Separation of Concerns

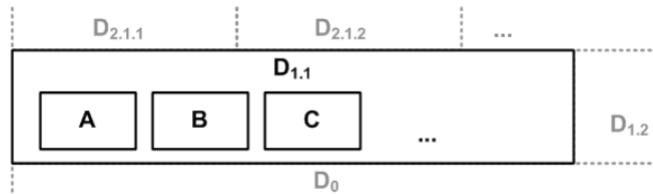
expressivity  
coverage  
semantics  
separation of  
concerns

completeness  
paradigms  
modularity  
concrete  
syntax

process

# Several Concerns

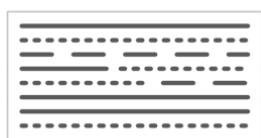
... in one domain



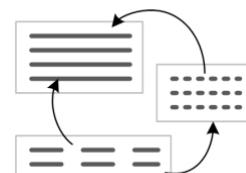
# Several Concerns

... in one domain

integrated into  
one fragment

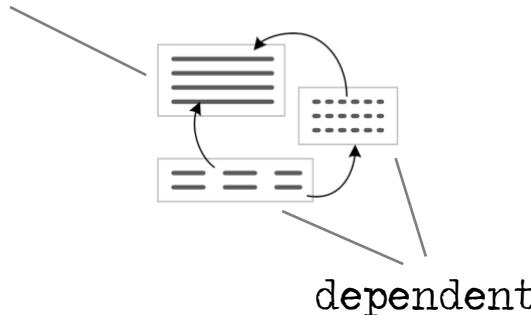


separated into  
several fragments



# Viewpoints

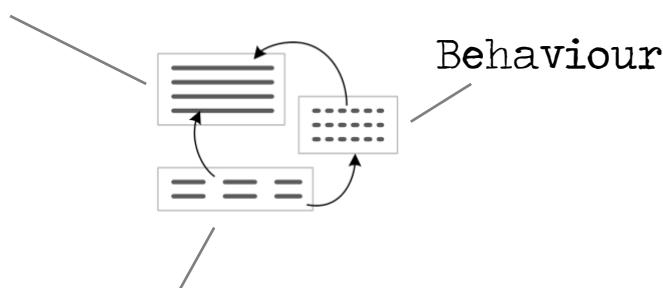
**independent**       $\forall r \in Refs_f \mid fo(r.to) = fo(r.from) = f$   
 $\forall e \in E_f \mid lo(co(e)) = l$



**dependent**

# Viewpoints

**Hardware**



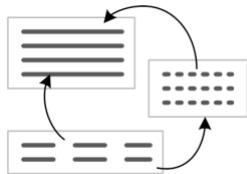
**Tests**

**Behaviour**

Example

Refrige  
rators

## Viewpoints: Why?



Sufficiency

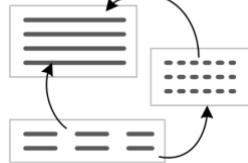
Different Stakeholders

Different Steps in  
Process - VCS unit!

## Viewpoints

Hardware

Product  
Management



Behaviour

Thermo-  
dynamics-  
Experts

Tests

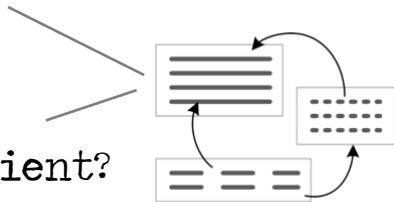
Example

Refrige-  
rators

# Viewpoints

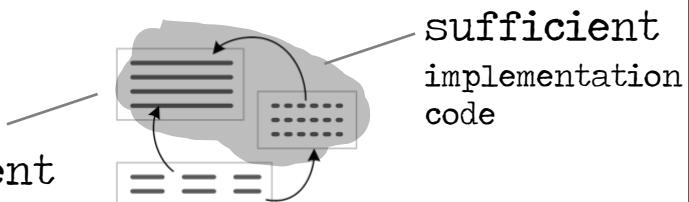
independent

sufficient?  
contains all  
the data for  
running a  
meaningful  
transformation



# Viewpoints

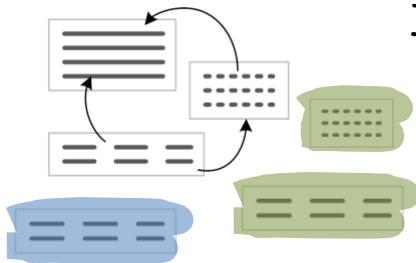
sufficient  
Hardware structure  
documentation



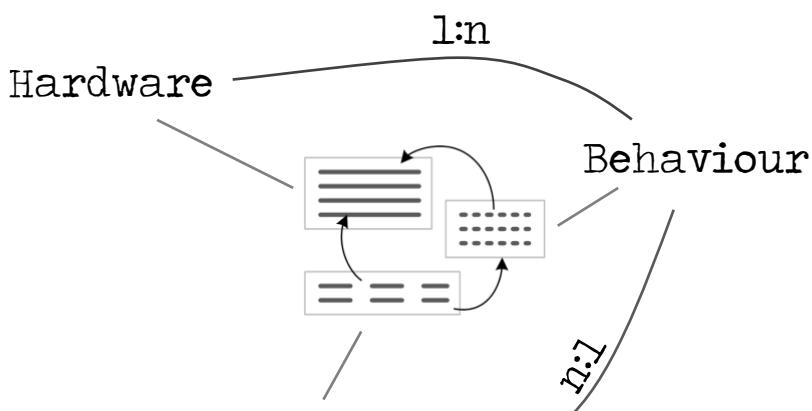
Example  
Refrige  
rators

## Viewpoints: Why?

l:n Relationships

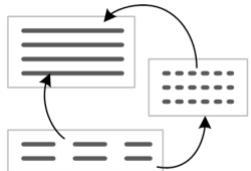


## Viewpoints



Example  
Refrige  
rators

## Viewpoints



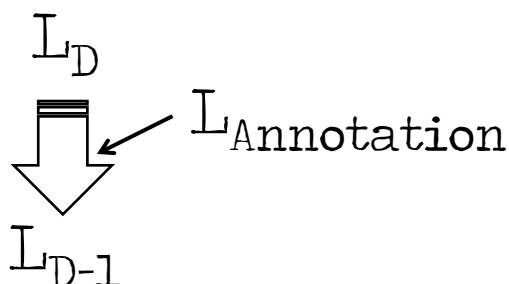
Well-defined  
Dependencies

No Cycles!

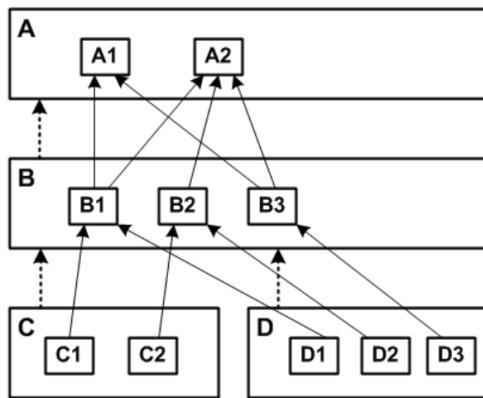
Avoid Synchronization!

(unless you use a projectional editor)

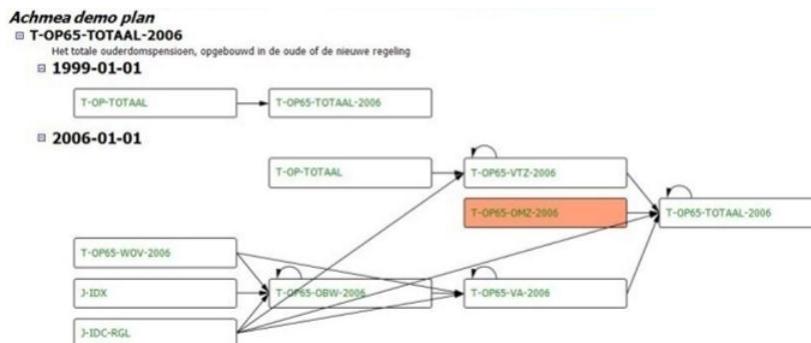
## Viewpoints: Why?



# Progressive Refinement



# Views on Programs



Example  
Pension  
Plans

# Completeness

expressivity  
coverage  
semantics  
separation of  
concerns

completeness  
paradigms  
modularity  
concrete  
syntax

process

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

More generally: all of  $D_{-1}$

Semantics:

$$\text{semantics}(p_{L_D}) := q_{L_{D-1}}$$

$$\text{where } OB(p_{L_D}) == OB(q_{L_{D-1}})$$

Introduce  $G$  ("generator"):

$$OB(p) == OB(G(p)) == OB(q) \text{ complete}$$

$$OB(G(p)) \subset OB(p) \quad \text{incomplete}$$

Incomplete: What to do?

$$\begin{array}{c} F_D \\ \downarrow \\ OB(F_D) \neq F_{D-1} \end{array}$$

Incomplete: What to do?

Manually written code!

$$\begin{array}{c} F_D \\ \downarrow \\ OB(F_D) == F_{D-1} + F_{D-1, \text{man}} \end{array}$$

Manually written code?

Call "black box" code  
(foreign functions)

Manually written code?

Call "black box" code  
(foreign functions)

Embed  $L_{D-1}$  code in  $L_D$  program

Manually written code?

Call "black box" code  
(foreign functions)

Embed  $L_{D-1}$  code in  $L_D$  program

Embed C statements  
in components, state machines  
or decision tables

Example

Refrige  
rators

Manually written code?

Call "black box" code  
(foreign functions)

Embed  $L_{D-1}$  code in  $L_D$  program

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

## Manually written code?

Call "black box" code  
(foreign functions)

Embed  $L_{D-1}$  code in  $L_D$  program

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

Generate base classes  
with abstract methods;  
implement in subclass

Example  
Components

## Manually written code?

Call "black box" code  
(foreign functions)

Embed  $L_{D-1}$  code in  $L_D$  program

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

Use protected regions (if you  
really have to...)

## Manually written code?

Call "black box" code  
(foreign functions)

Embed  $L_{D-1}$  code in  $L_D$  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?

## Incomplete: When?

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

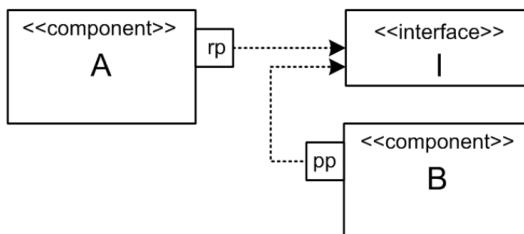
Example	Example
Extended C	Components

Bad for business DSLs.

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

Example	Example	Example
Refrigerators	Pension Plans	Web DSL

## Prevent Breaking Promises!

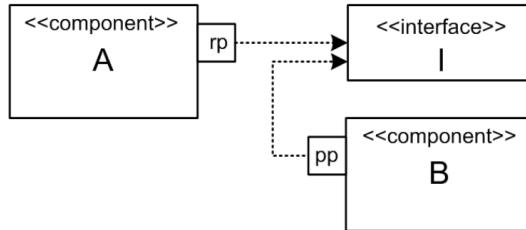


```

class B extens BBase {

    public void doSomething() {
        Registry.get("A").someMethod();
    }
}
    
```

# Prevent Breaking Promises!

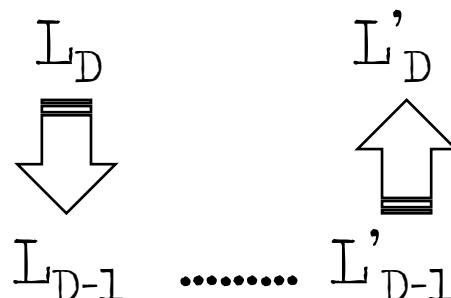


**Better:**

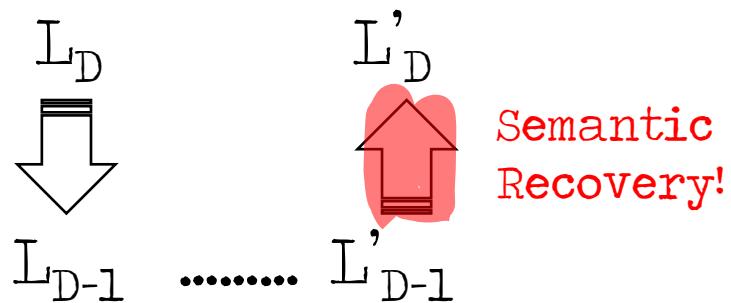
Dependency Injection  
Static analysis tools

Example  
Components

# Roundtripping



## Roundtripping - **Don't!**



# Fundamental Paradigms

expressivity  
coverage  
semantics  
separation of  
concerns

completeness  
paradigms  
modularity  
concrete  
syntax

process

## Structure

### Modularization, Visibility

Namespaces,  
public/private  
importing

## Structure

### Modularization, Visibility

Namespaces,  
public/private  
importing

→ divide & conquer  
reuse  
stakeholder integration

## Structure

### Partitioning (Files)

VCS Unit  
Unit of sharing  
Unit of IP

!= logical modules

may influence language design

## Structure

# Modularization, Visibility

```
module Module1 from HPL.main imports Module2 {

    exported var int aReallyGlobalVar;

    struct aLocallyVisibleStruct {
        int x;
        int y;
    };

    exported int anExportedFunction() {
        return anImportedFunction/Module2();
    } anExportedFunction (function)
}
```

Example

Extended C

## Structure

# Modularization, Visibility

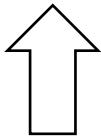
```
namespace com.mycompany.test {
    system testSystem {
        instance dc: DelayCalculator
        instance screen1: InfoScreen
        instance screen2: InfoScreen
        connect dc.screens to
            (screen1.default, screen2.default)
    }
}
```

Example

Components

## Structure

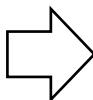
### Partitioning (Files)



change impact  
link storage  
model organization  
tool integration

## Structure

### Spec vs. Implementation



plug in different Impls  
different stakeholders

# Structure

## Spec vs. Impl.

```

exported component AnotherDriver extends Driver {
    ports:
        provides IDiagnostic diag
        requires optional ILogger logger
        requires ILowLevel lowlevel restricted to LowLevelCode
    contents:
        field int8_t count = 0

        override int8_t setDriverValue(int8_t addr, int8_t value) <- op cmd.setDriverValue {
            with port (logger) { logger.log("SomeMessage"); } with port
            lowlevel.doSomeLowlevelStuff(10);
            count++;
            return 1;
        }

        int8_t diag_getCount() <- op diag.getCount {
            return count;
        }
    }
}

```

```

exported c/s interface ITrafficLights {
    int8_t setColor(ILCommand cmd) ;
}

c/s interface IDriver {
    int8_t setDriverValue(int8_t addr, int8_t value) ;
}

c/s interface IDiagnostic {
    int8_t getCount() ;
}

c/s interface ILogger {
    void log(string message) ;
}

c/s interface ILowLevel {
    int8_t doSomeLowlevelStuff(int8_t y) ;
}

```

Example  
Extended C

# Structure

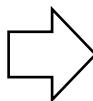
## Specialization

Liskov substitution P  
leaving holes ("abstract")

## Structure

### Specialization

Liskov substitution P  
leaving holes ("abstract")



variants (in space)  
evolution (over time)

## Structure

### Specialization

Pension Plans can inherit  
from other plans.

Rules can be abstract;

Plans with abstract  
rules are abstract

Example

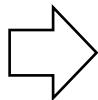
Components

## Structure

### Superposition, Aspects

merging  
overlay

AOP



modularize cross-cuts

## Structure

### Superposition, Aspects

```
component DelayCalculator {
    ...
}

component AircraftModule {
    ...
}

component InfoScreen {
    ...
    aspect (*) component {
        provides mon: IMonitoring
    }
}
```

```
    component DelayCalculator {
        ...
        provides mon: IMonitoring
    }

    component AircraftModule {
        ...
        provides mon: IMonitoring
    }

    component InfoScreen {
        ...
        provides mon: IMonitoring
    }
```

Example

Compo  
nents

## Behavior

Not all DSLs specify behavior

Some just **declare** behavior

This section is  
not for those!

## Behavior

### Imperative

sequence of statements  
changes program state

write	understand	debug	analyze	performance
simple	simple -	simple	hard (step)	good

# Behavior

## Imperative

sequence of statements  
changes program state

```
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

# Behavior

## Functional

functions call other functions.  
no state. No aliasing.

write	understand	debug	analyze	performance
simple -	simple	simple (tree)	good	good -

# Behavior

**Functional**  
functions call other functions.  
no state. No aliasing.

Elements...

Rules

- Rule Bereken Mutatieperiode
  - Result: Mutatieperiode
  - Name: Bereken Mutatieperiode
  - Documentation: Het vaststellen van de periode tussen de huidige en de vorige mutatie in dagen. De mutatieperiode kan niet meer dan 360 dagen bedragen omdat elk jaar een begin- en eindmutatie kent i.v.m. het openen en sluiten van het verdelingsjaar. Dit wordt niet aangevallen omdat het uitvoeren van de begin- en eindmutatie verantwoordelijkheid zijn van de pensioenadministratie.
  - Tags: Basisberekening
  - Algorithm:
 

```
if maximum(Mutates per datum) == 1 then daysof(duration(valid(Mutates per datum))) else 0
```
- Test cases

Name	Valid time	Transaction time	Fixture	Product	Element	Expected value	Actual
Gelijke datums	03/01/2008		Mutatieperiode - Mutatiedatum =			3	0

Example  
Pension  
Plans

# Behavior

**Functional**

```
rule for <years of service in service period> ( ) at 01/01/1986
  for p in valid(Service Period) sum(Part time % * yearsof(duration(valid(Part time % over p)))) at end(p);
    valid(Service Period)
    Service Period
    04/01/1984 ... 01/01/1990
    sum(Part time % * yearsof(duration(valid(Part time % over p)))) at end(p)
    sum(Part time % * yearsof(duration(valid(Part time % over p))))
    Part time %
    yearsof(duration(valid(Part time % over p)))
    Value:
    TemporalBranch
    01/01/2001 Schilder X 01/01/1984 ...
```

Example  
Pension  
Plans

## Behavior

### Functional

pure expressions are  
a subset of functional  
(operators hard-wired)

guards  
preconditions  
derived attributes

## Behavior

### Declarative

only facts and goals.  
no control flow.  
eval engine, solver (several)

write	understand	debug	analyze	performance
simple	simple -	hard	depends	often bad

## Behavior

**Declarative**

concurrency  
 constraint programming  
 solving  
 logic programming

## Behavior

**Declarative**

only facts and goals.  
 no control flow.  
 eval engine, solver (several)

```
section posts
  define page post(p: Post, title: String) {
    title(output(p.title))
    bloglayout(p.blog){
      placeholder view { postView(p) }
      postComments(p)
    }
  }
  define permalink(p: Post) {
    navigate post(p, p.urlTitle) { elements }
  }
```

Example

Web  
 DSL

# Behavior

## Declarative

```

synchronous blockType org::eclipselabs::damos::library::base:::_discrete:::DiscreteDerivative

input u
output y

parameter initialCondition = 0
parameter gain = 1(s) // normalized

behavior {
    stateful func main<initialCondition, gain, fs>(u) -> y {
        check<0, 1(s), 1(1/s)>(real) -> real

        static assert u is real() :
            error "Input value must be numeric"

        static assert initialCondition is real() :
            error "Initial condition must be numeric"

        static assert initialCondition is real() && u is real() => unit(initialCondition) == unit(u) :
            error "Initial condition and input value must have same unit"

        static assert gain is real() :
            error "Gain value must be numeric"

        eq u{-1} = initialCondition
        eq y{n} = fs * gain * (u{n} - u{n-1})
    }
}

```

Example

Extended C

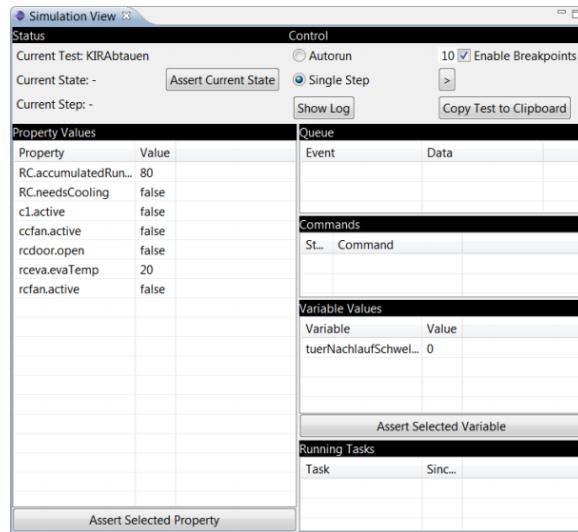
# Behavior

## Reactive

reactions to events, more events are produced.  
Communication via events and channels/queues

write	understand	debug	analyze	performance
simple	simple/hard	hard	simple	can be good

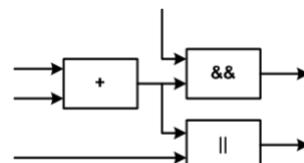
# Behavior      Reactive



Example  
Refrige  
rators

# Behavior

## Data Flow

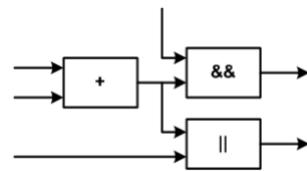


chained blocks consume  
continuous data that flows  
from block to block

write	understand	debug	analyze	performance
simple -	simple/hard	hard	simple	can be good

## Behavior

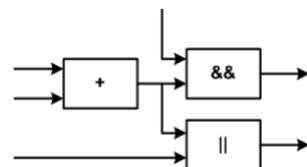
### Data Flow



continuous, calc on change  
quantized, calc on new data  
time triggered, calc every x

## Behavior

### Data Flow



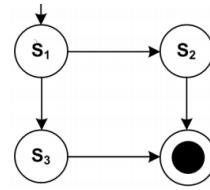
Embedded Programming  
Enterprise ETL & CEP

# Behavior

## State Based

states, transitions,  
guards, reactions

event driven, timed



write	understand	debug	analyze	performance
simple -	simple/hard	s/h	simple +	can be good

# Behavior

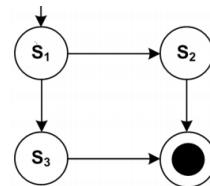
## State Based

```

start:
  entry { state noCooling }

state noCooling:
  check ( (RC->needsCooling) && (cc.cl->stehzeit > 333) ) {
    state rccooling
  }
  on isDown ( RC.rctdoor->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.rctdoor->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
    }
  }
}
  
```

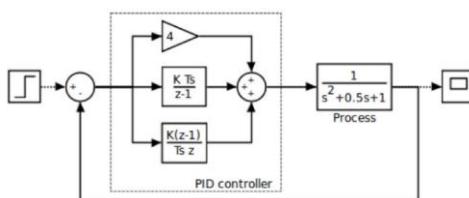


Example  
Refrige  
rators

# Behavior

## Combinations

data flow uses functional,  
imperative or declarative lang  
inside block



```

synchronous blockType org:redipuelabs:domes::library::base::discrete::DiscreteDerivative

input u
output y
parameter initialCondition = 0
parameter gain = 10 // normalized
behavior {
    stateful func main(initialCondition, gain, Ts:real) -> y {
        check(u, Ts, IC:real) -> real
        static assert (u is real);
        error "Input value must be numeric";
        static assert (initialCondition is real);
        error "Initial condition must be numeric";
        static assert (gain is real & u is real => unit(initialCondition) == unit(u));
        error "Initial condition and input value must have same unit";
        static assert gain is real;
        error "Gain value must be numeric";
        eq y(1) = initialCondition
        eq y(2) = u * gain * (x(0) - u(n-1))
    }
}

```

# Behavior

## Combinations

state machines use expressions  
in guards and often an  
imperative lang in actions

# Behavior

## Combinations

```

start:
    entry { state noCooling }

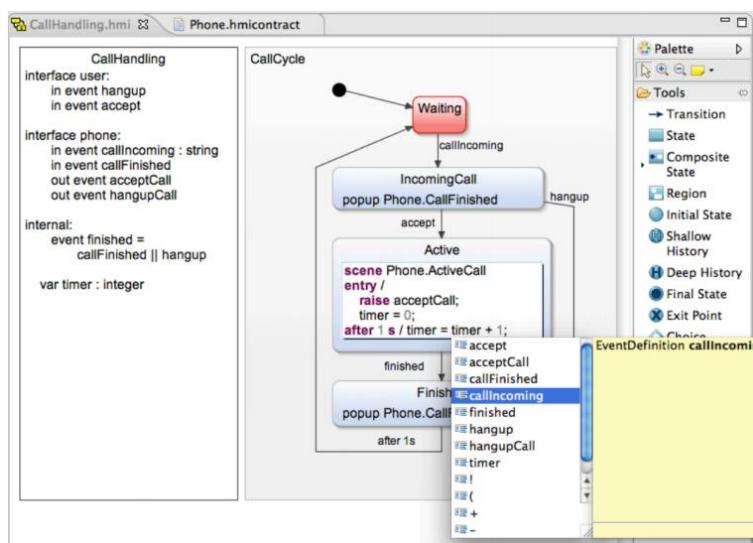
state noCooling:
    check ( (RC->needsCooling) && (cc.cl->stehzeit > 333) ) {
        state rccooling
    }
    on isDown ( RC.rcdoor->open ) {
        set RC.rcfan->active = true
        set RC.rclight->active = false
        perform rcfanabschalttask after 10 {
            set RC.rcfan->active = false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

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

# Behavior



# Behavior

## Combinations

purely structural languages often use expressions to specify constraints

```
c/s interface IDriver {  
    int setDriverValue(int addr, int value)  
        pre value > 0  
}
```

Example

Extended C

# Language Modularity

expressivity  
coverage  
semantics  
separation of  
concerns

completeness  
paradigms  
modularity  
concrete  
syntax

process

**Language Modularity,  
Composition and Reuse (LMR&C)**

increase efficiency  
of DSL development

## Language Modularity, Composition and Reuse (LMR&C)

increase efficiency  
of DSL development

### 4 ways of composition:

Referencing

Reuse

Extension

Reuse

## Language Modularity, Composition and Reuse (LMR&C)

increase efficiency  
of DSL development

### 4 ways of composition:

distinguished regarding  
**dependencies** and **fragment**  
**structure**

**Dependencies:**

do we have to know about the reuse when designing the languages?

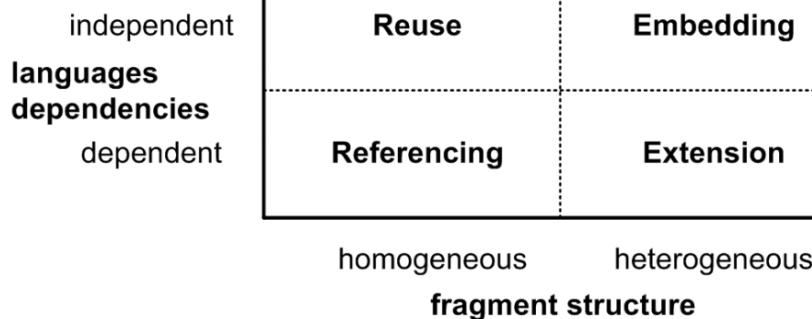
**Dependencies:**

do we have to know about the reuse when designing the languages?

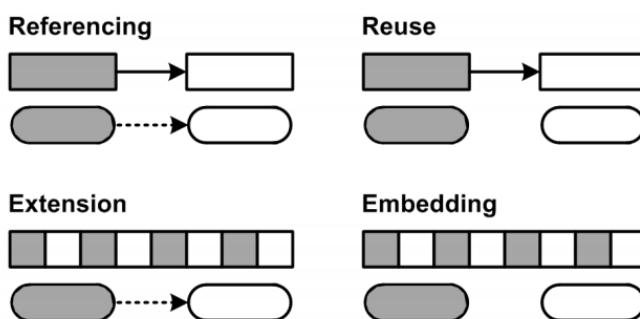
**Fragment Structure:**

homogeneous vs. heterogeneous  
(„mixing languages“)

## Dependencies & Fragment Structure:



## Dependencies & Fragment Structure:



# Referencing

independent languages dependencies dependent	Reuse	Embedding
	Referencing	Extension
	homogeneous      heterogeneous fragment structure	

```

graph LR
    I2["I2  
B1"] -- dashed --> I1["I1  
A1  
A2  
A3"]
    A2 -- solid --> A3
  
```

# Referencing

independent languages dependencies dependent	Reuse	Embedding
	Referencing	Extension
	homogeneous      heterogeneous fragment structure	

**Dependent**

```

graph LR
    I2["I2  
B1"] -- green dashed --> I1["I1  
A1  
A2  
A3"]
    I2 -- red solid --> A2
    A2 -- red solid --> A3
  
```

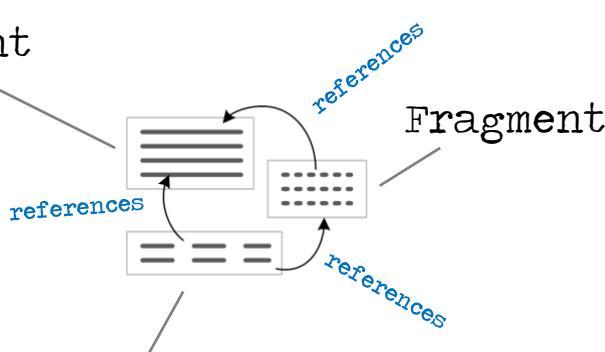
**No containment**

# Referencing

Used in  
Viewpoints

# Referencing

Fragment



Fragment

# 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->steht)
         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

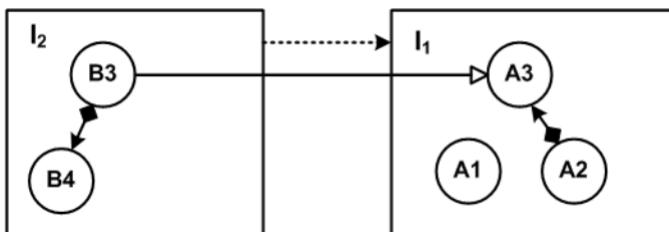
```

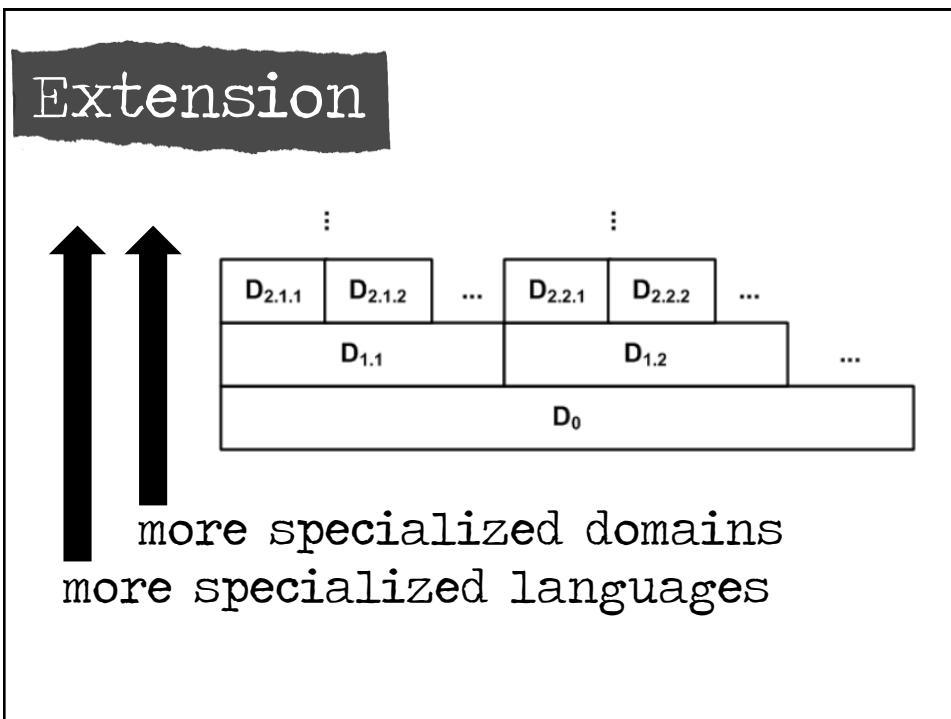
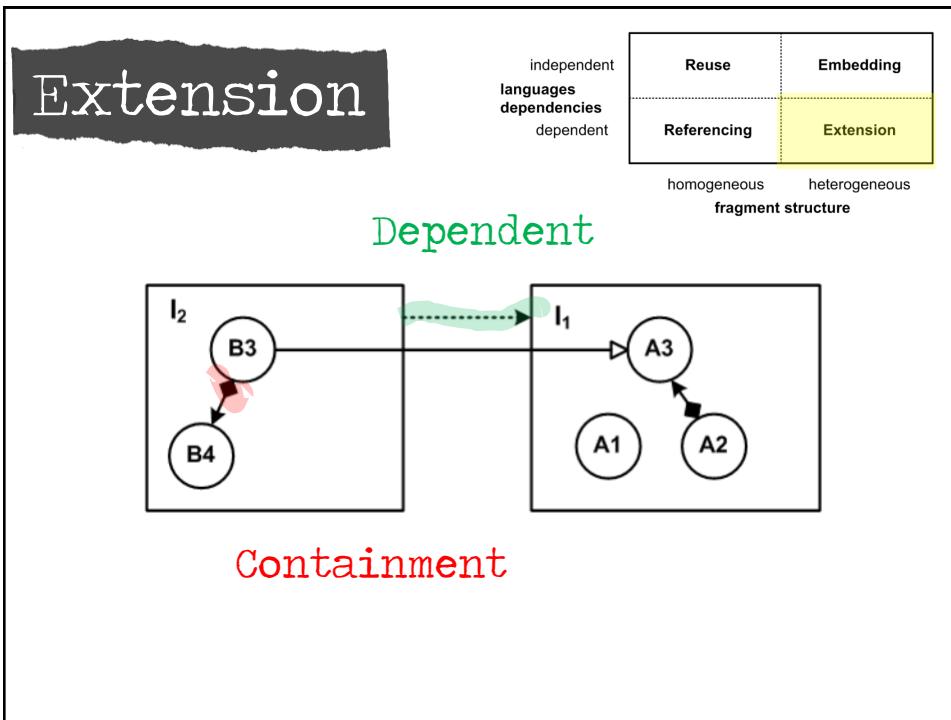
Example  
Refrige  
rators

# Extension

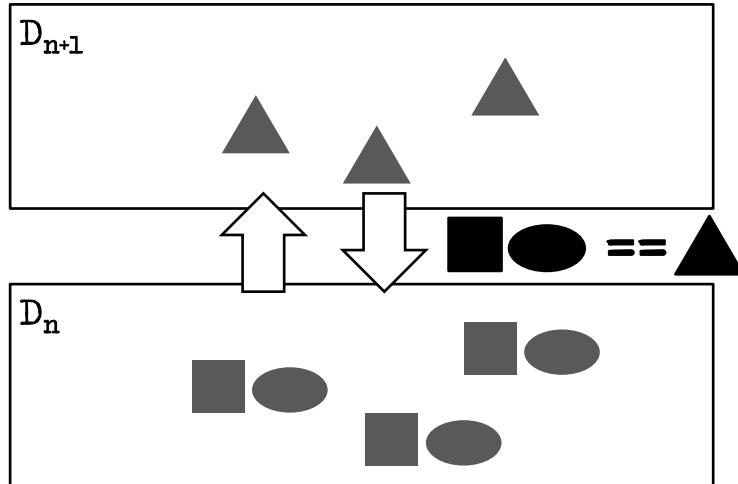
	Reuse	Embedding
independent languages dependencies	Referencing	Extension
dependent		
homogeneous		heterogeneous

fragment structure

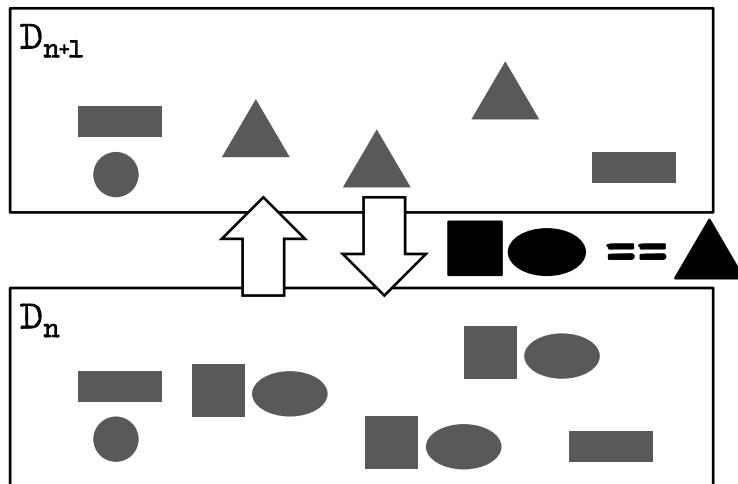




## Extension

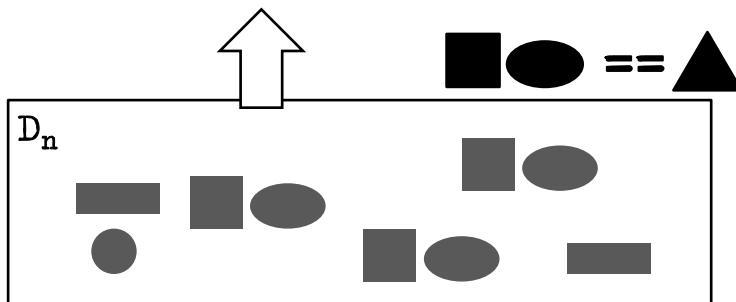


## Extension



## Extension

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



## Extension

### Drawbacks

tightly bound to base  
potentially hard to analyze  
the combined program

# Extension

```

module main imports OsekKernel, EcAPI, BitLevelUtilities {

    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>;
}

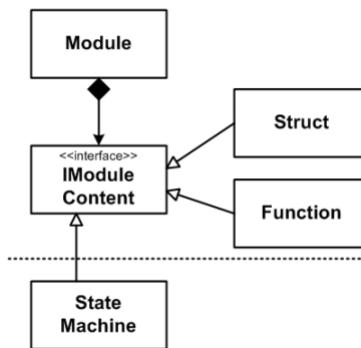
void updateMotorSettings( int left, int right ) {
    nxt_motor_set_speed(MOTOR_PORT_T::NXT_MOTOR_S1, left);
    nxt_motor_set_speed(MOTOR_PORT_T::NXT_MOTOR_S2, right);
}

```

Example

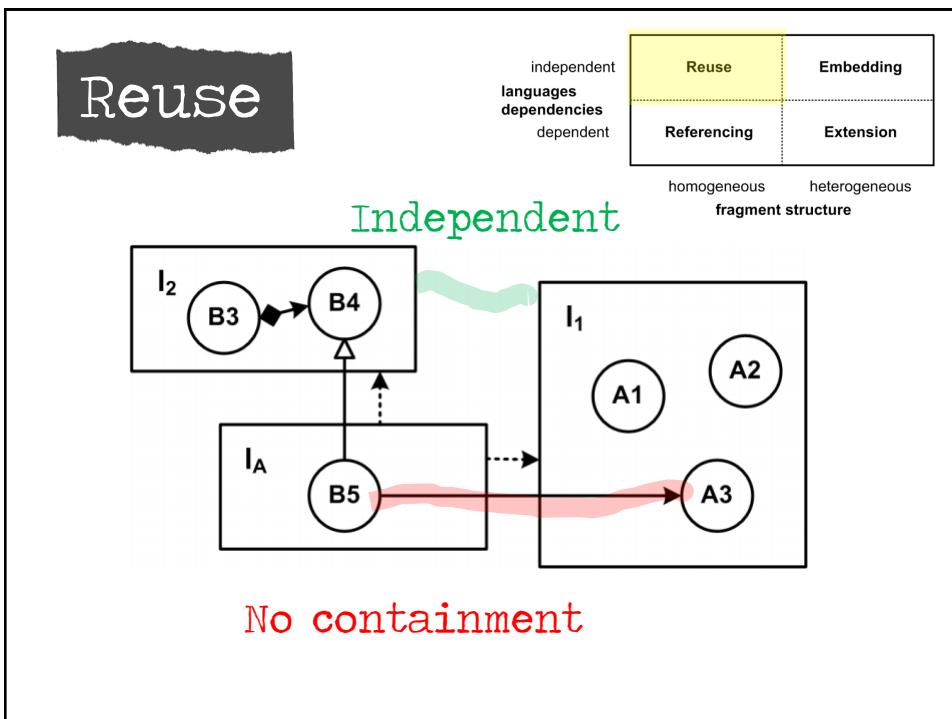
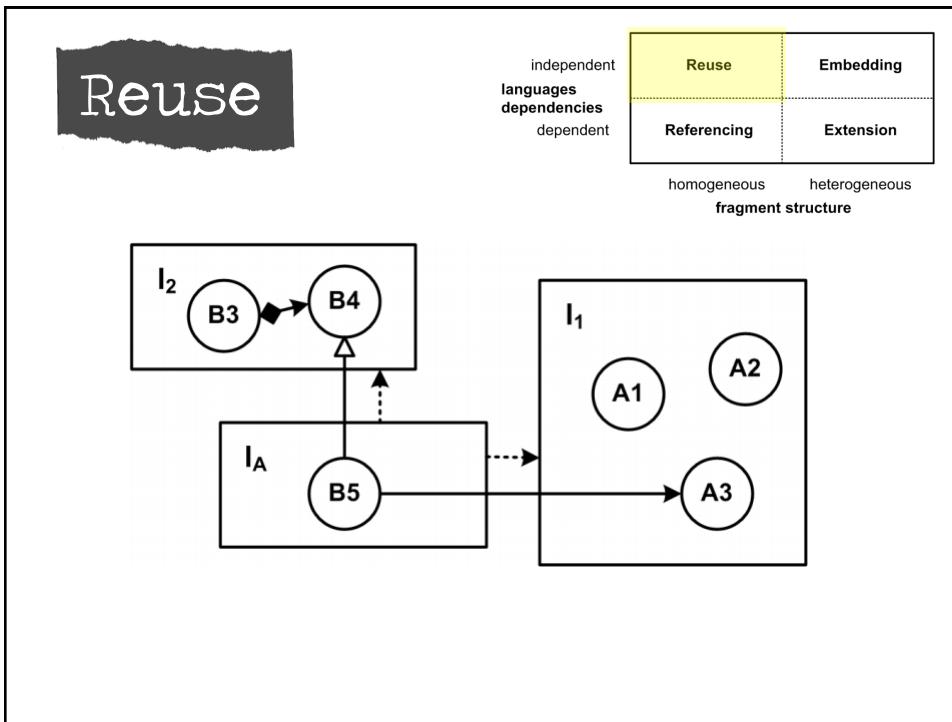
Extended C

# Extension



Example

Extended C



## Reuse

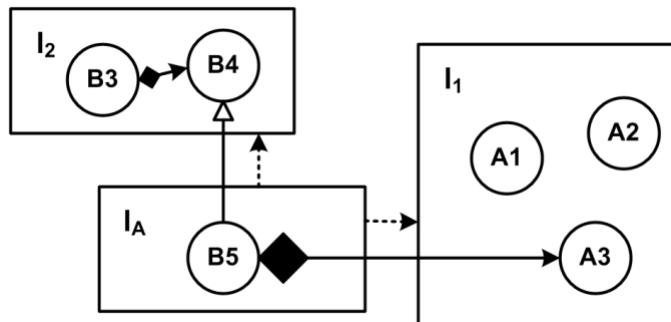
Often the referenced language is built expecting it will be reused.

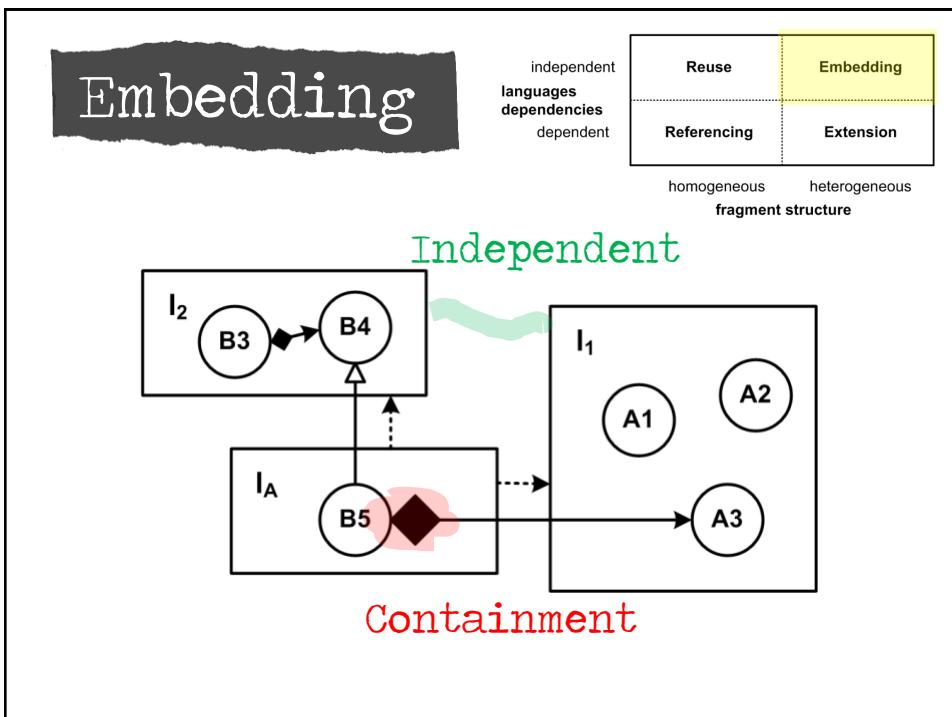
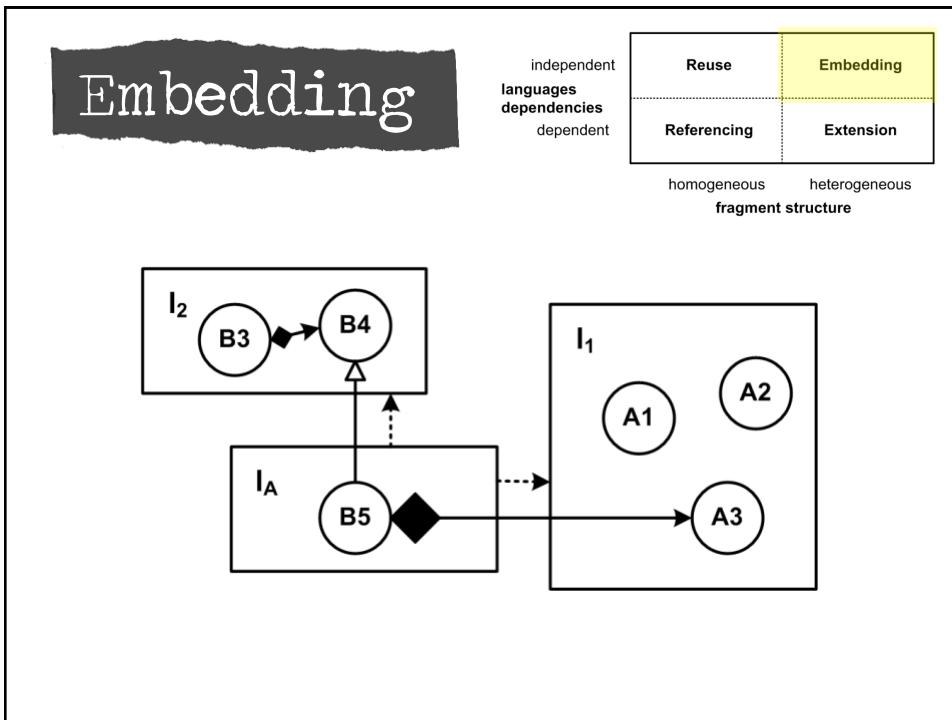
Hooks may be added.

## Embedding

independent languages	Reuse	Embedding
dependencies	Referencing	Extension
dependent		

homogeneous      heterogeneous  
fragment structure





# Embedding

The screenshot shows the Capgemini Pension Workbench application window. The menu bar includes File, Edit, Projection, Navigation, Search, Format, Tools, Dev, Generate, Pension, Team, and NN. A tab labeled 'NLCPA-14w2-21112008' is active. The left sidebar has a 'Table of Contents' section with categories like Library, Documentation, Tag definitions, Shared, and Rules. The main pane displays the 'Elements...' and 'Rules' sections. Under 'Rules', there is a detailed view of a rule named 'Rule Bereken Mutatieperiode'. The rule's algorithm is defined as follows:

```

if maximum('Mutates per datum') == 1 then daysof(duration/valid('Mutates per datum'))) else 0
  
```

The rule has three test cases:

Name	Valid time	Transaction time	Fixture	Product	Element	Expected value	Actual value
Gelijke datums	03/01/2008		Mutatieperiode - Mutatedatum = Mutatedatum Vorig			3	0
Periode < 30	03/01/2008		Mutatieperiode - Mutatedatum > Mutatedatum Vorig (binnen 1 maand)			15	15
Periode > 30	03/01/2008		Mutatieperiode - Mutatedatum > Mutatedatum Vorig (meer maanden)			60	

Below the table, there are links for 'Bereken Mutatieperiode • Test cases • Unit test: Gelijke datums' and buttons for 'Doc | Splitter | Pension | PensionDecorated | AM'.

Example  
Pension  
Plans

# Embedding

Embedding often uses Extension to extend the embedded language to adapt it to its new context.

## Challenges - Syntax

Extension and Embedding  
requires modular concrete  
syntax

Many tools/formalisms cannot  
do that

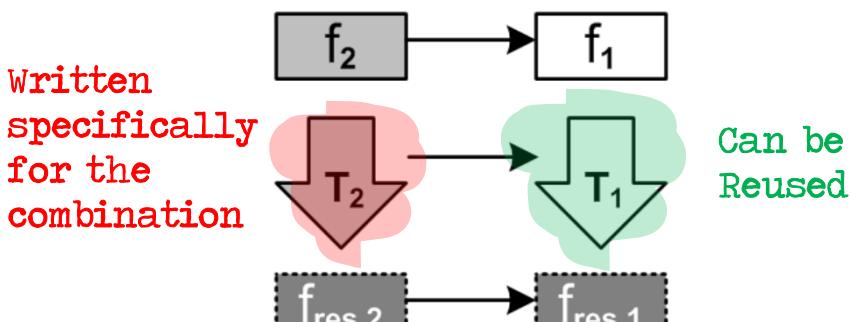
## Challenges - Type Systems

Extension: the type system of  
the base language must be  
designed to be extensible/  
overridable

## Challenges - Type Systems

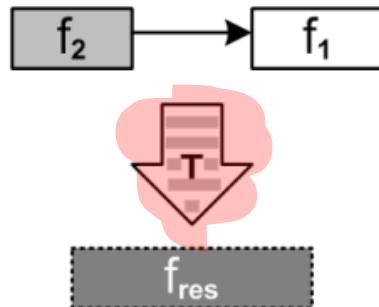
Reuse and Embedding: Rules that affect the interplay can reside in the adapter language.

## Challenges - Trafo & Gen Referencing (I)



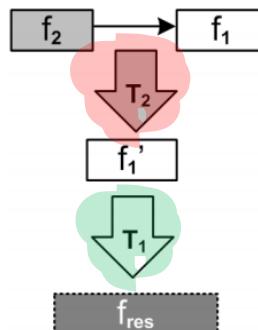
Two separate, dependent single-source transformations

## Challenges - Trafo & Gen Referencing (II)



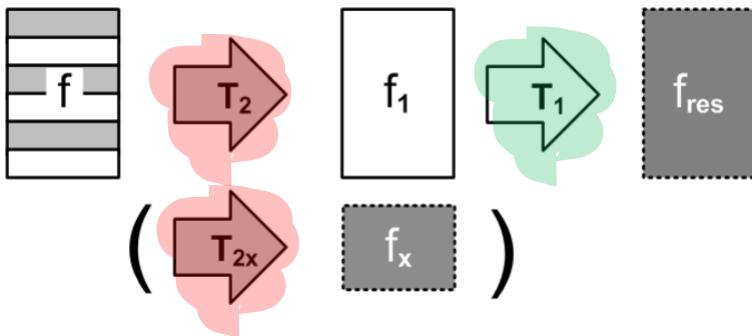
A single multi-sourced transformation

## Challenges - Trafo & Gen Referencing (III)



A preprocessing trafo that changes the referenced frag in a way specified by the referencing frag

# Challenges - Trafo & Gen Extension

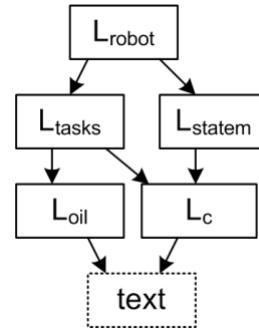


Transformation by assimilation, i.e.  
generating code in the host lang  
from code expr in the extension lang.

# Challenges - Trafo & Gen Extension

```
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
    }
}
```

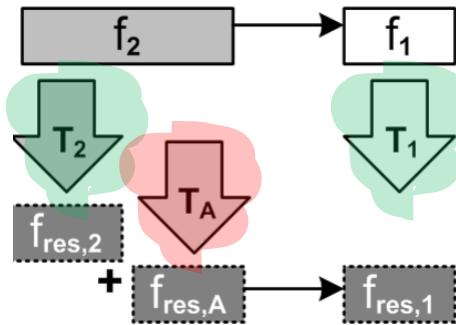


Example

Exten  
ded C

## Challenges - Trafo & Gen

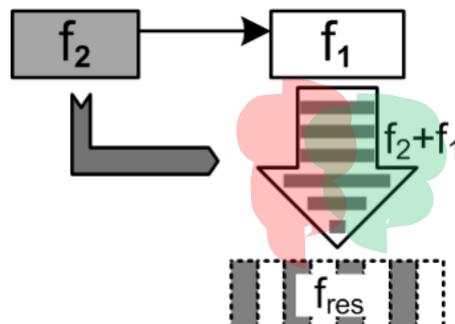
### Reuse (I)



Reuse of existing  
transformations for both fragments  
plus generation of adapter code

## Challenges - Trafo & Gen

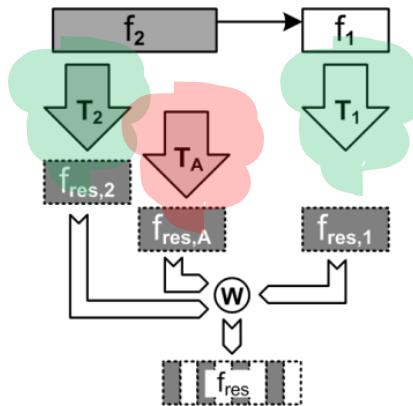
### Reuse (II)



composing transformations

## Challenges - Trafo & Gen

### Reuse (III)

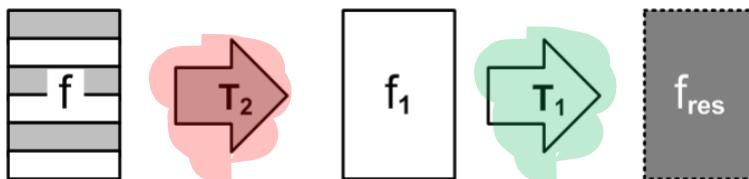


generating separate artifacts  
plus a weaving specification

## Challenges - Trafo & Gen

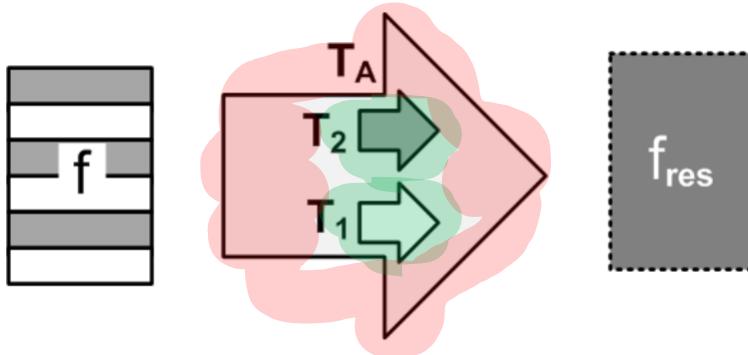
### Embedding (I)

a purely embeddable language  
may not come with a generator.

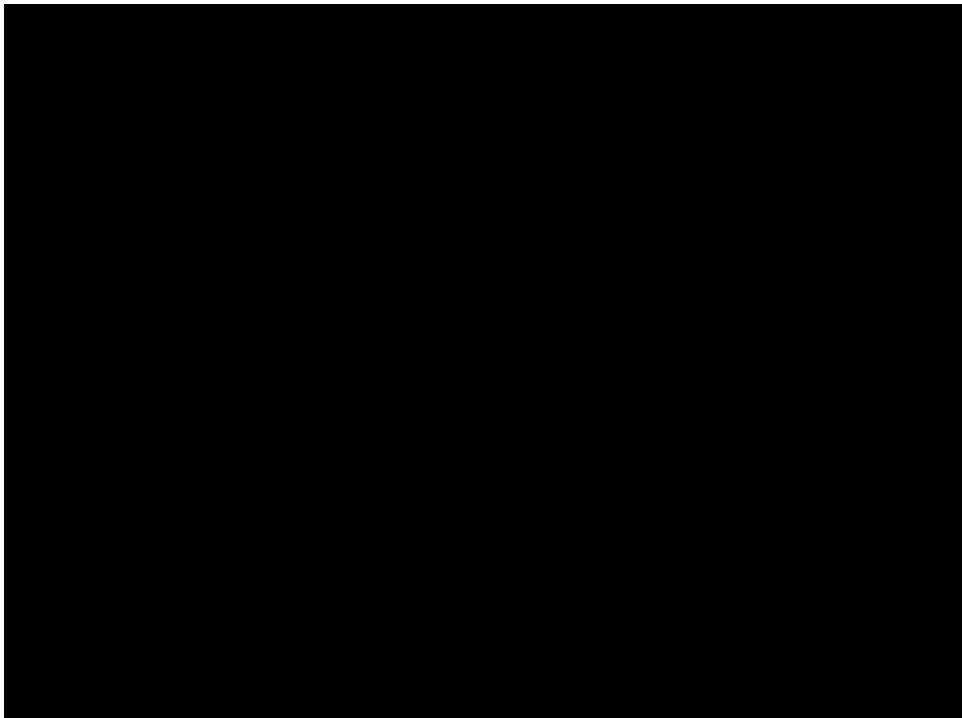


Assimilation (as with Extension)

## Challenges - Trafo & Gen Embedding (II)



Adapter language can coordinate the transformations for the host and for the embedded languages.



# Concrete Syntax

expressivity  
coverage  
semantics  
separation of  
concerns

completeness  
paradigms  
modularity  
concrete  
syntax

process

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.

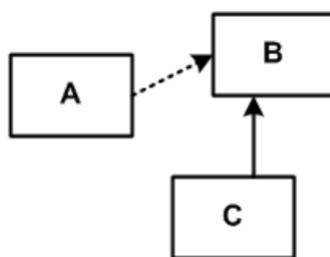
## Default: Text

Editors simple to build  
Productive  
Easy to integrate w/ tools  
Easy to evolve programs  
  
... then add other forms,  
if really necessary

## Default: Text

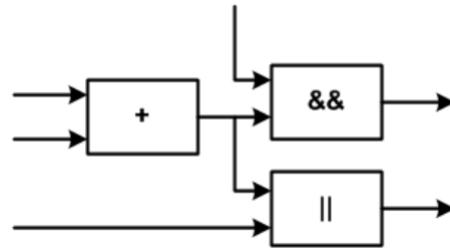
Editors simple to build  
Productive  
Easy to integrate w/ tools  
Easy to evolve programs

## Graphical in case...



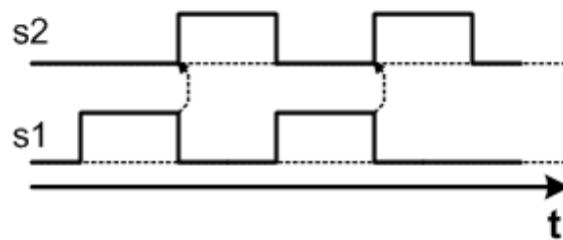
Relationships

Graphical in case...



Flow and  
Dependency

Graphical in case...



Causality  
and Timing

# Symbolic

Either  
Mathematical,  
or often  
highly  
inspired by  
domain

☒ **3.3 Commutatiegetallen op 1 leven¶**

$$D_x = v^x * \frac{l}{100} \quad \approx 6 \text{ Dec (3)}$$

Implemented in [VB401¶](#)

¶

$$N_x = \sum_{t=0}^{l-x} D_{x+t} \quad \approx 7 \text{ Dec (3)}$$

¶

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

$$D_{n,x} = \frac{x+n}{D_x} \quad \approx 19 \text{ Dec (4)}$$

$$a_x = \bar{a}_x - 1 \quad \approx 21 \text{ Dec (3)}$$

$$\bar{a}_x = \bar{a}_x - 0,5 \quad \approx 22 \text{ Dec (3)}$$

¶

$$\bar{a}_{xn} = \frac{N_x - N_{x+n}}{D_x} \quad \approx 23 \text{ Dec (3)}$$

$$\bar{a}_{xn} = \bar{a}_{xn} - 0,5 + 0,5 * E_n \quad \approx 25 \text{ Dec (3)}$$

¶

# Tables

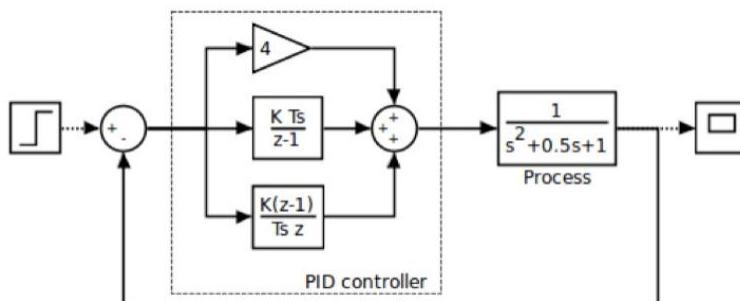
Name	Documentation	Tags	Valid time	Transaction time	Fixture	Product	Element	Expected value	Actual value
Accrued right at retirement		⌚	2006-12-31	2007-9-24	Jan De Jong	Old Age Pension	Accrued right	761.0402	761.0402
Accrued Right last final pay premium last year		⌚	2004-1-1	2007-9-24	Jan De Jong	Old 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 retirement (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

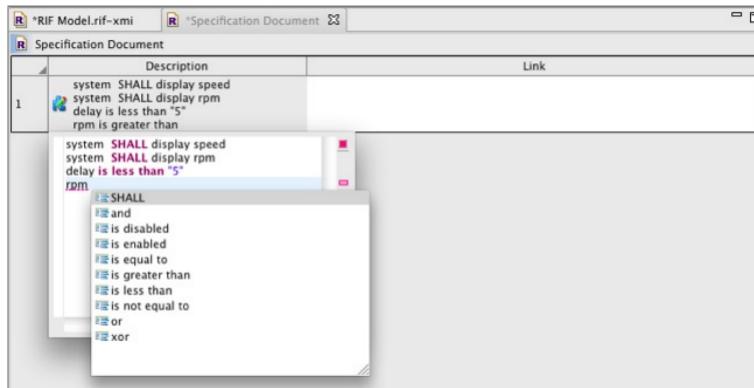
```
c/s interface Decider {
    int decide(int x, int y) pre
}

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 |
        }
}
```

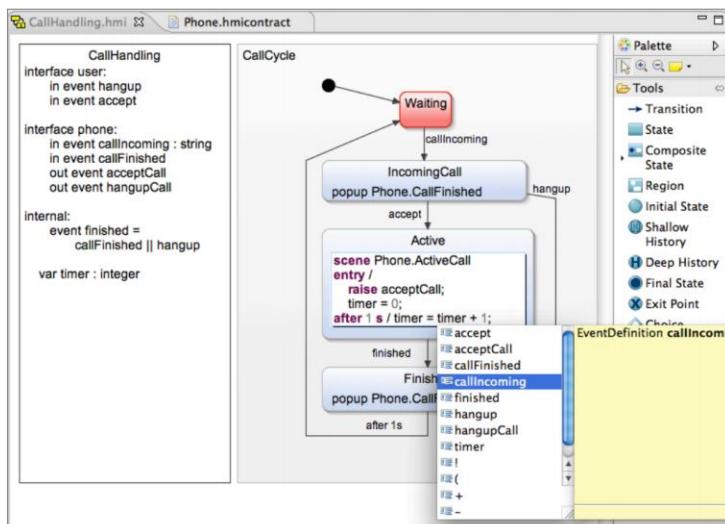
# Combinations



# Combinations



# Combinations



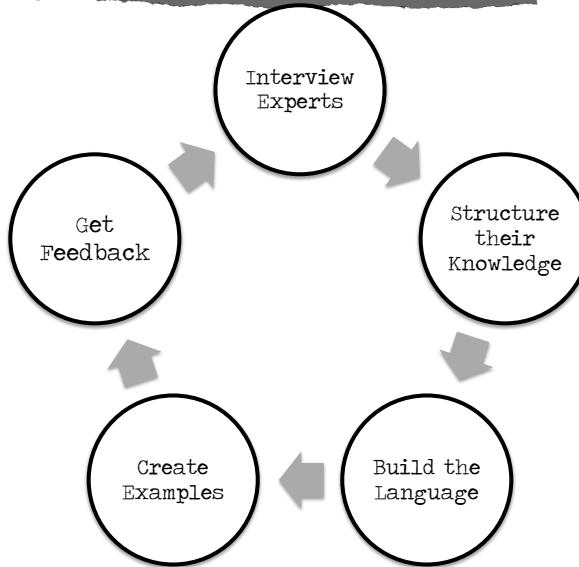
# Process

expressivity  
coverage  
semantics  
separation of  
concerns

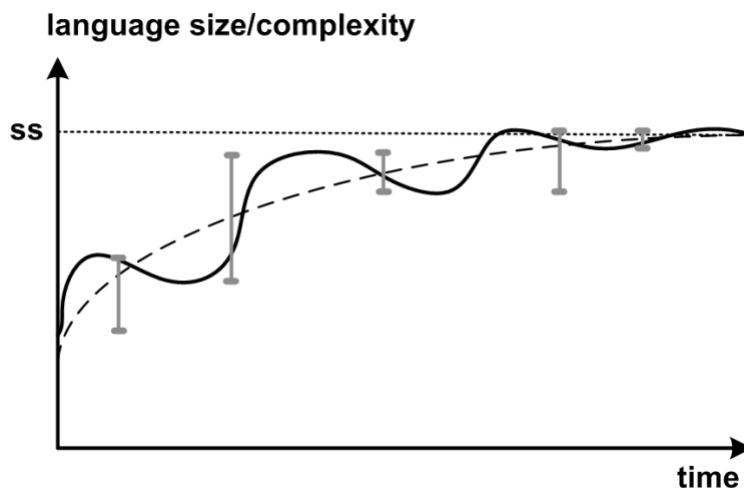
completeness  
paradigms  
modularity  
concrete  
syntax

process

## Domain Analysis



## Iterate to goal



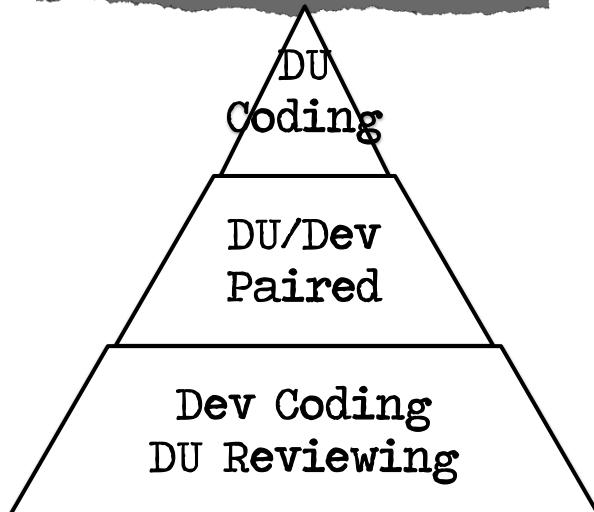
## Documentation

Create example-based tutorials!

## Domain Folks Programming?

Precision vs.  
Algorithmics!

## Domain Folks Programming?

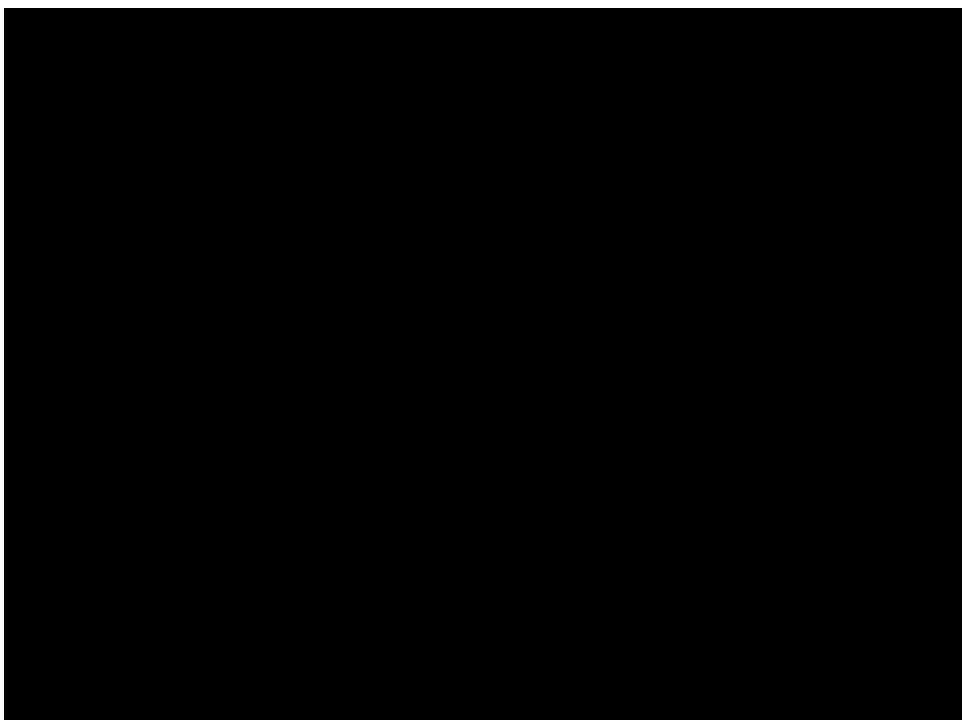


## DSL as a Product

Release Plan  
Bug Tracker  
Testing!  
Support  
Documentation

## Reviews

Reviews become  
easier --- less  
code, more  
domain-specific



# The End.

This material is part of  
my upcoming (early 2013)  
book DSL Engineering.  
Stay in touch, it may  
become a free eBook ☺

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