DSL Design

A conceptual framework for building good DSLs

Code Generation 2012

Markus Voelter
independent/itemis
voelter@acm.org

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

based on material from a paper written with Eelco Visser

E.Visser@tudelft.nl
http://eelcovisser.org/

DSL Design

Part of a conceptual framework for building good DSLs

Code Generation 2012

Markus Voelter
independent/itemis
voelter@acm.org

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

based on material from a paper written with Eelco Visser

E.Visser@tudelft.nl
http://eelcovisser.org/

8+1 Design Dimensions

expressivity	completeness
coverage	paradigms
semantics	modularity
separation of concerns	concrete syntax

process

Which abstractions go into the language, and why

expressivity	completeness
coverage	paradigms
semantics	modularity
separation of concerns	concrete syntax

process

Which portions of the domain is covered by the DSL?

expressivity
coverage
semantics
separation of concerns

completeness
paradigms
modularity
concrete syntax

process

What does it all mean?

expressivity
coverage
semantics
separation of concerns

completeness
paradigms
modularity
concrete syntax

process

Should the various concerns in the domain be separated into different viewpoints?

expressivity
coverage
semantics
separation of concerns

completeness
paradigms
modularity
concrete syntax

process

Is the DSL ableto express all% of the system, or do you have to „manually" write code?

expressivity
coverage
semantics
separation of concerns

completeness
paradigms
modularity
concrete syntax

process

Which language paradigms exist, and when and how can they be used in DSLs?

expressivity
coverage
semantics
separation of concerns

completeness
paradigms
modularity
concrete syntax

process

How do you modularize (and then compose and combine) languages (GPLs and DSLs)?

expressivity
coverage
semantics
separation of concerns

completeness
paradigms
modularity
concrete syntax

process

Which syntactic forms are most suitable, and how do you combine them?

expressivity	completeness
coverage	paradigms
semantics	modularity
separation of concerns	**concrete syntax**

process

What do you have to consider in terms of the development process?

expressivity	completeness
coverage	paradigms
semantics	modularity
separation of concerns	concrete syntax

process

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

Compo
nents

```
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

Refrigerators

```
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

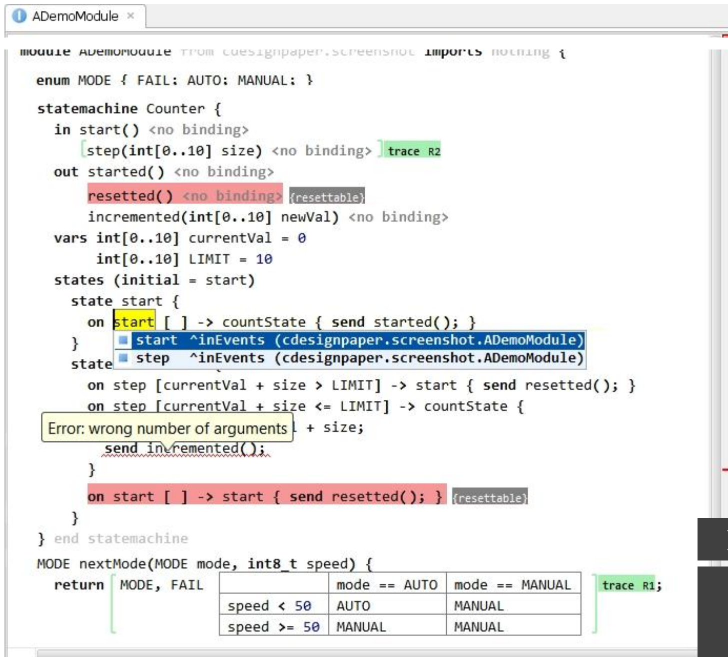
Refrigerators

```
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
```

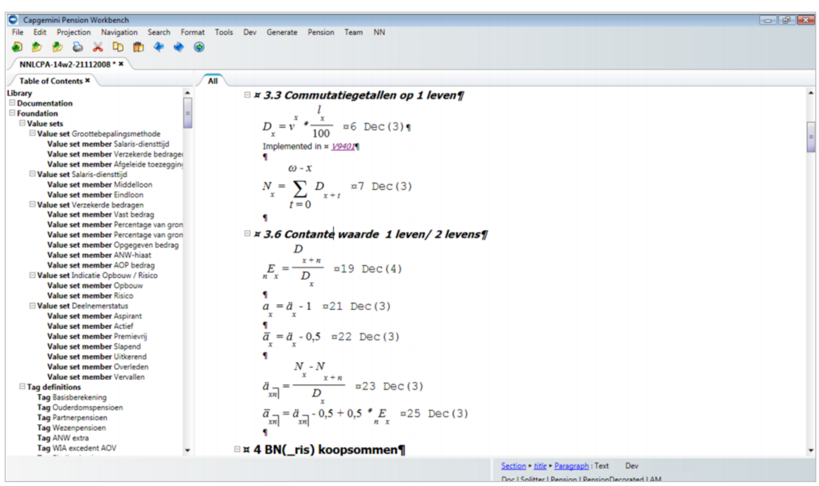
Example

Refrigerators



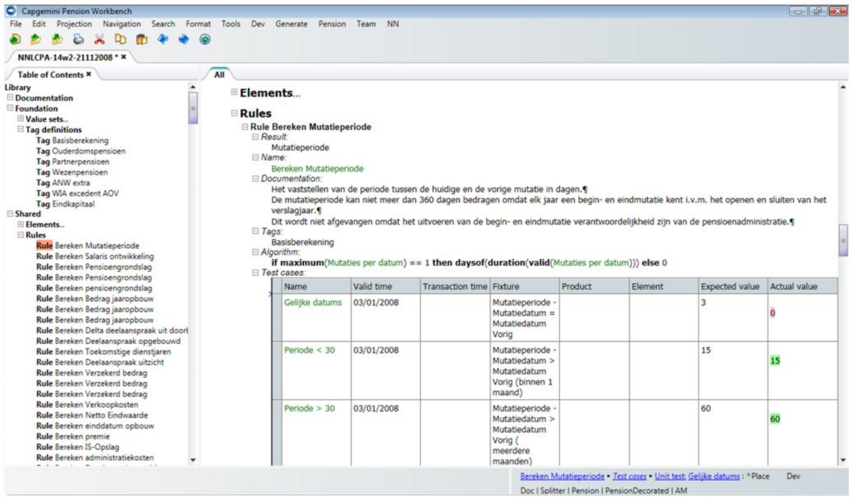
Example

Extended C



Example

Pension Plans



Example

Pension Plans

In this talk:

mostly | **a little bit**

mostly	a little bit
expressivity	completeness
coverage	paradigms
semantics	modularity
separation of concerns	concrete syntax

process

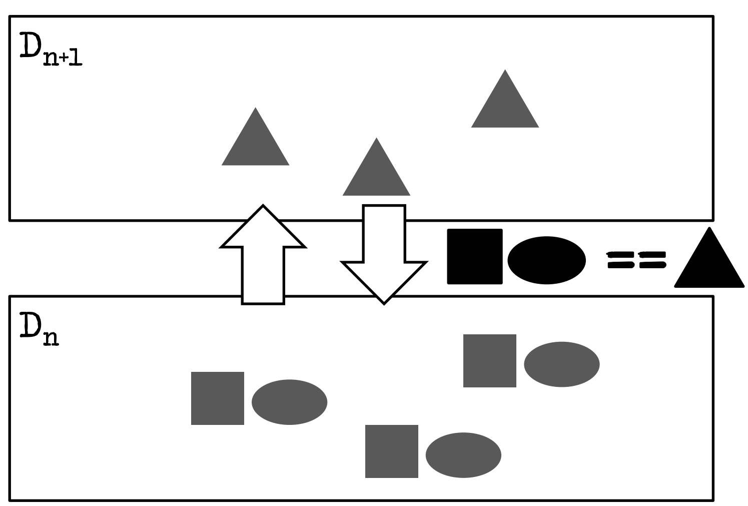
Expressivity

expressivity
coverage
semantics
separation of concerns

completeness
paradigms
modularity
concrete syntax

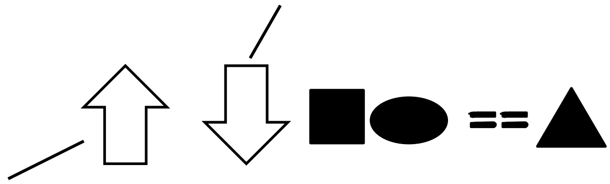
process

Reification



Reification

Transformation/
Generation



Language
Definition

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.

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

Refrigerators

Semantics & Execution

expressivity
coverage
semantics
separation of concerns

completeness
paradigms
modularity
concrete syntax

process

Static Semantics

Execution Semantics

Static Semantics

Constraints
Type Systems

What does it all mean?

Execution Semantics

Def: Semantics

… via mapping to lower level

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

OB: Observable Behaviour (Test Cases)

Def: Semantics

… via mapping to lower level

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

L_D

$\Downarrow$ Transformation
Interpretation

L_{D-1}

Mapping

L_D

$\Downarrow$ Transformation
Interpretation

L_{D-1} Known Semantics!

Transformation

L_D

Correct!? $\Downarrow$ Transformation
Interpretation

L_{D-1}

Transformation

Tests (D) L_D

Transformation
Interpretation

Tests (D-1) L_{D-1}

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.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

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

Example

Pension Plans

Def: Semantics

… via mapping to lower level

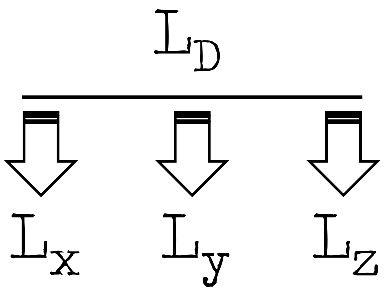
L_D

$\Downarrow$ Transformation Interpretation

L_{D-1}

Multiple Mappings

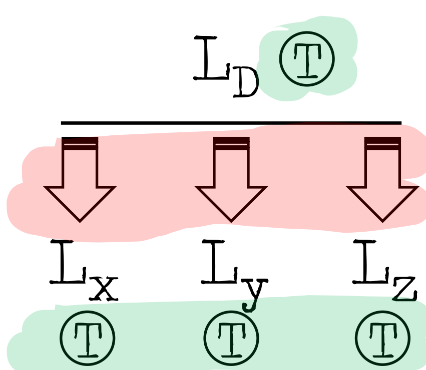
… at the same time



Similar Semantics?

Multiple Mappings

… at the same time



Similar Semantics?

all green!

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

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
        }
    }
```

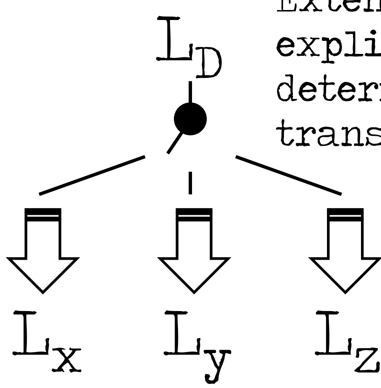
```
prolog {
    set RC->accumulatedRuntime = 80
}

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

Refrigerators

Multiple Mappings

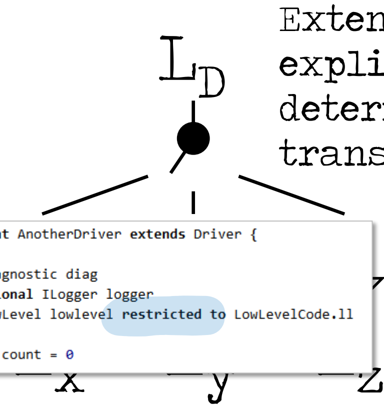
… alternatively, selectably



Extend L_D to include explicit data that determines transformation

Multiple Mappings

… alternatively, selectably



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
```

Restricted Port leads to reduced overhead C

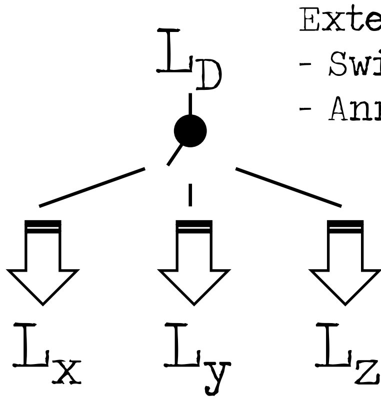
Example

Extended C

Multiple Mappings

… alternatively, selectably

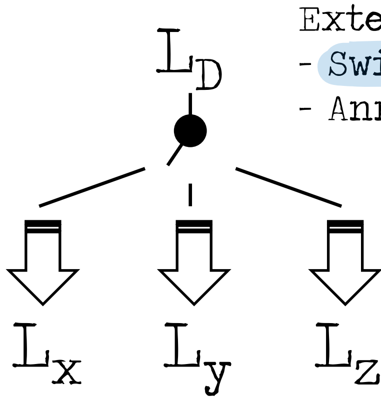
External Data:
- Switches
- Annotation Model



Multiple Mappings

… alternatively, selectably

External Data:
- Switches
- Annotation Model

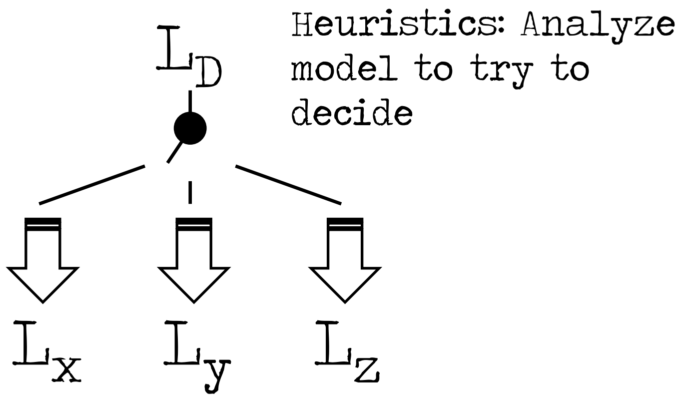


Switch Control
Java vs. C

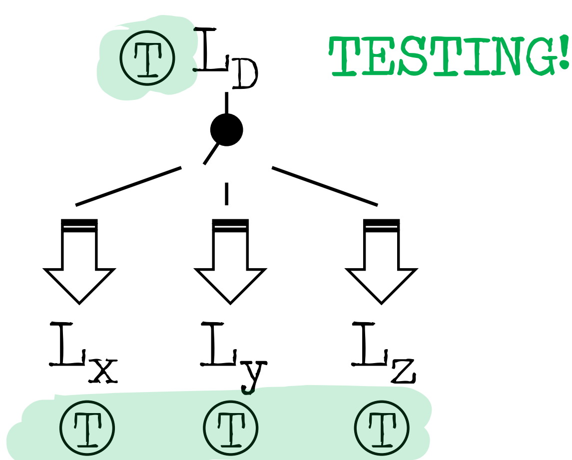
Example

Pension Plans

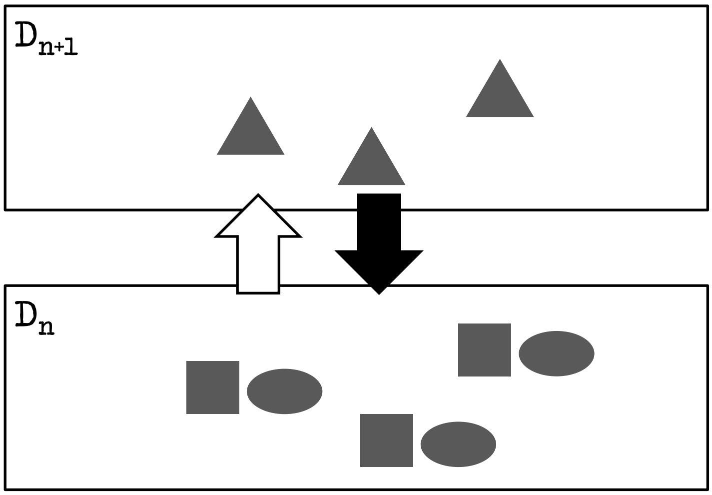
Multiple Mappings
... alternatively, selectably



Multiple Mappings
... alternatively, selectably

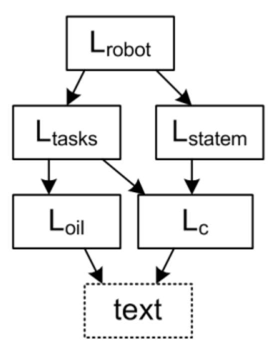


Transformation



Transformation

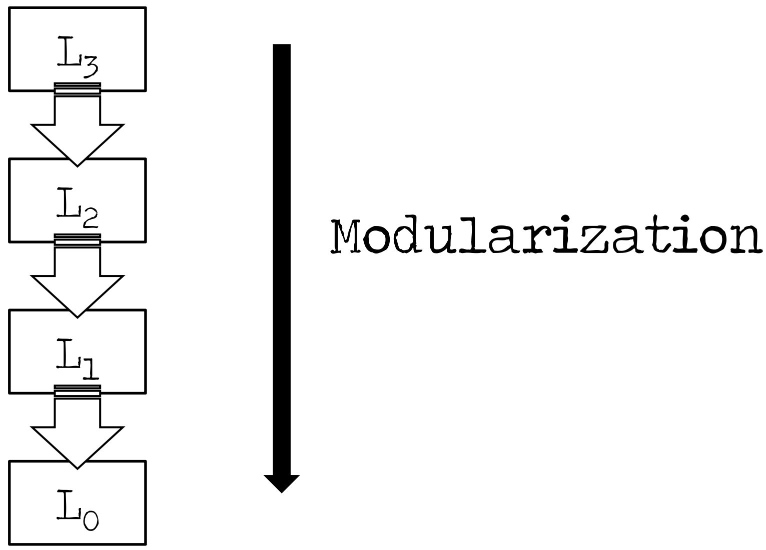
```
module impl imports <<imports>> {

  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

}
```



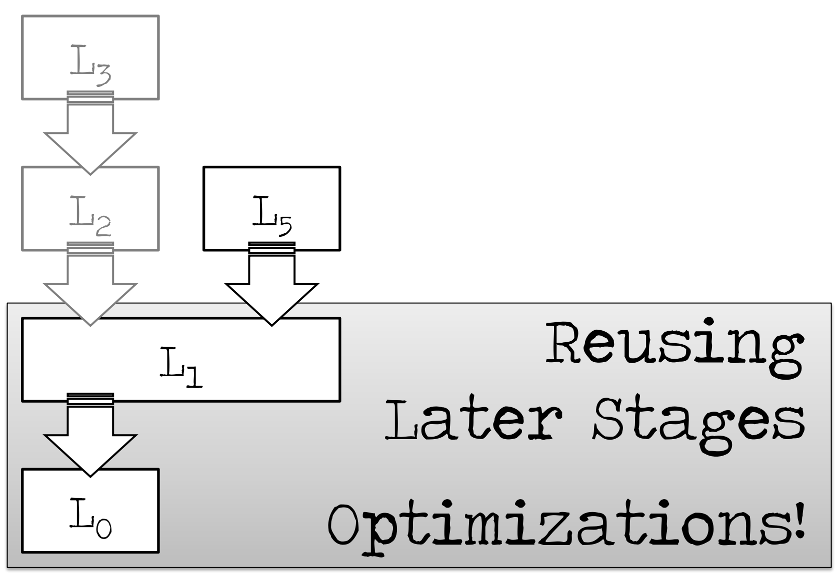
Example

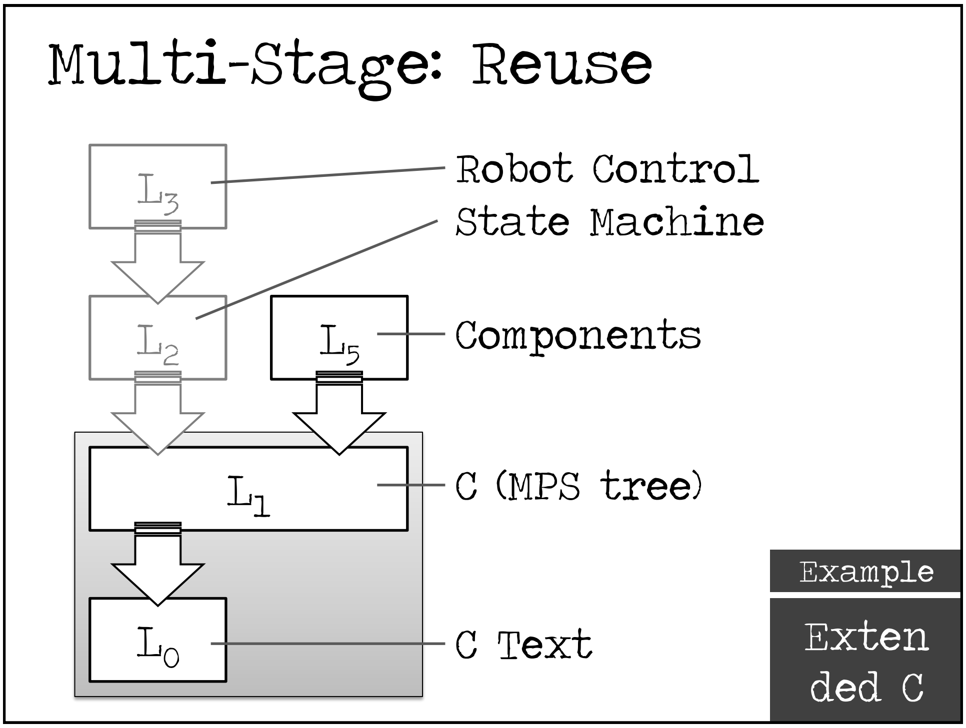
Extended C

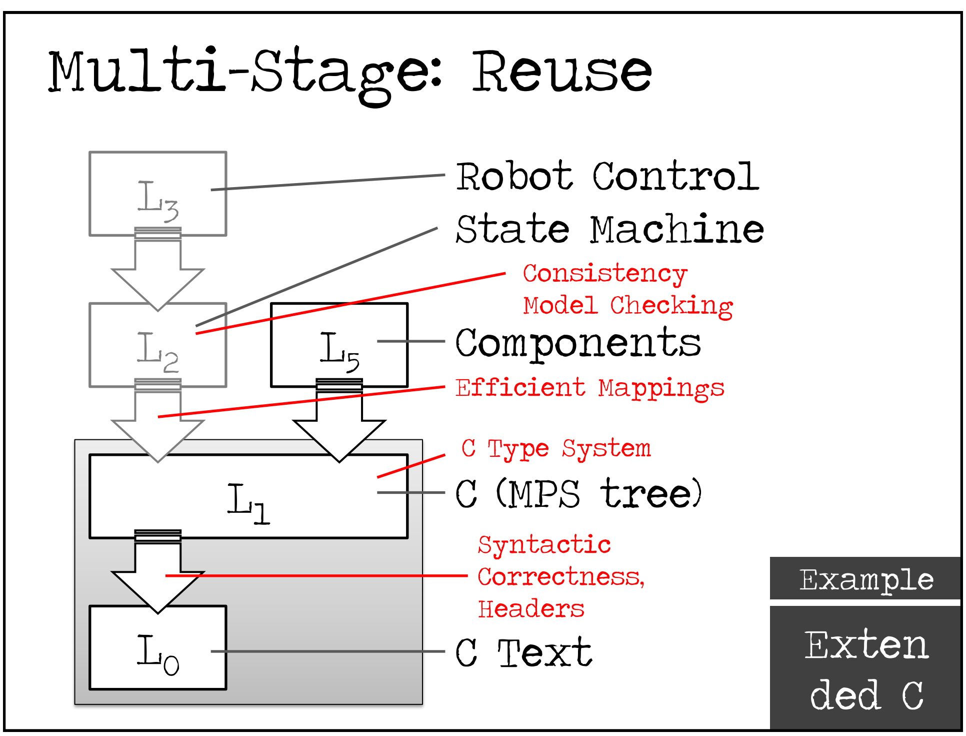
Multi-Stage



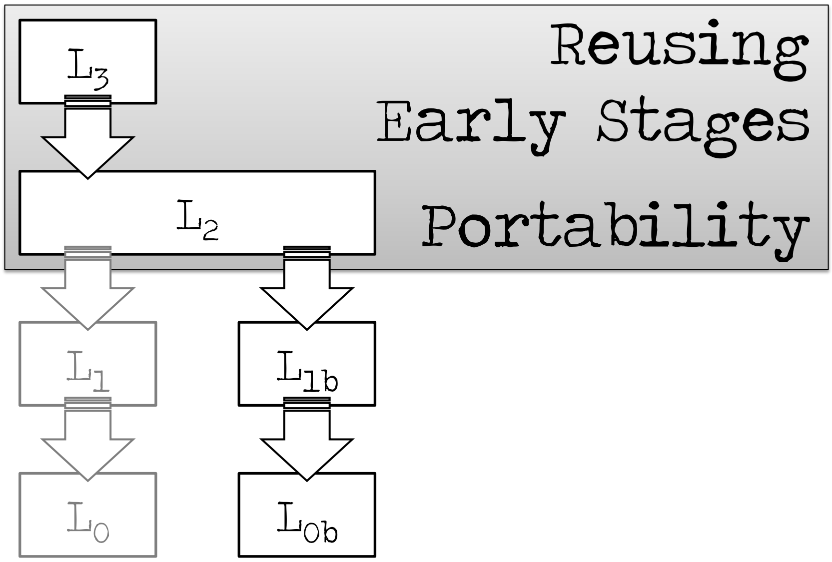
Multi-Stage: Reuse



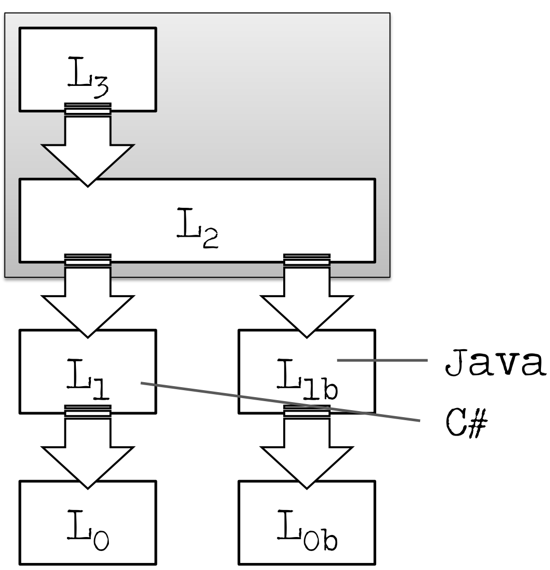

Multi-Stage: Reuse
L_3
L_2
L_5
L_1
L_0
Robot Control
State Machine
Components
C (MPS tree)
C Text
Example
Extended C


Multi-Stage: Reuse
L_3
L_2
L_5
L_1
L_0
Robot Control
State Machine
Consistency
Model Checking
Components
Efficient Mappings
C Type System
C (MPS tree)
Syntactic
Correctness,
Headers
C Text
Example
Extended C

Multi-Stage: Reuse



Multi-Stage: Reuse



Example

Pension Plans

Reduced Expressivity

bad? maybe.

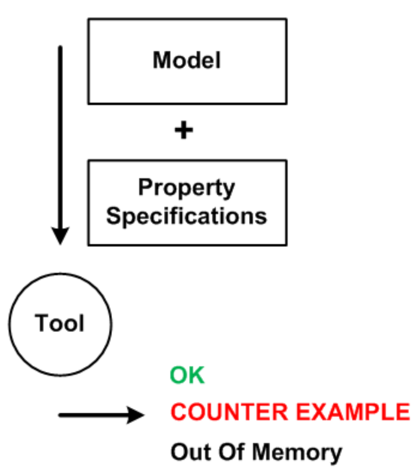
good? maybe!

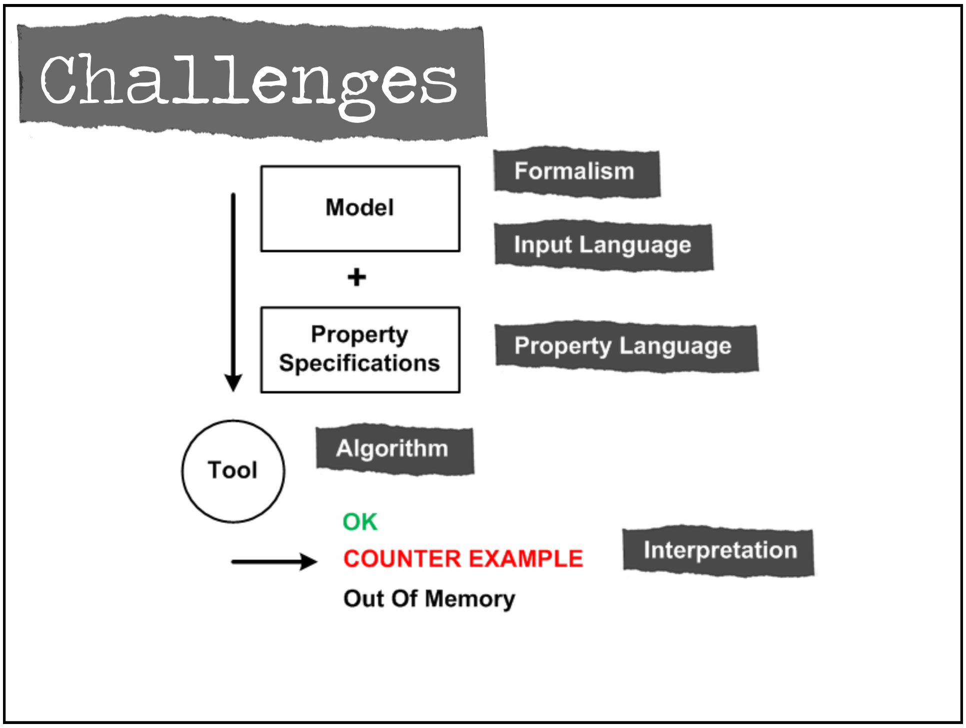
Model Checking

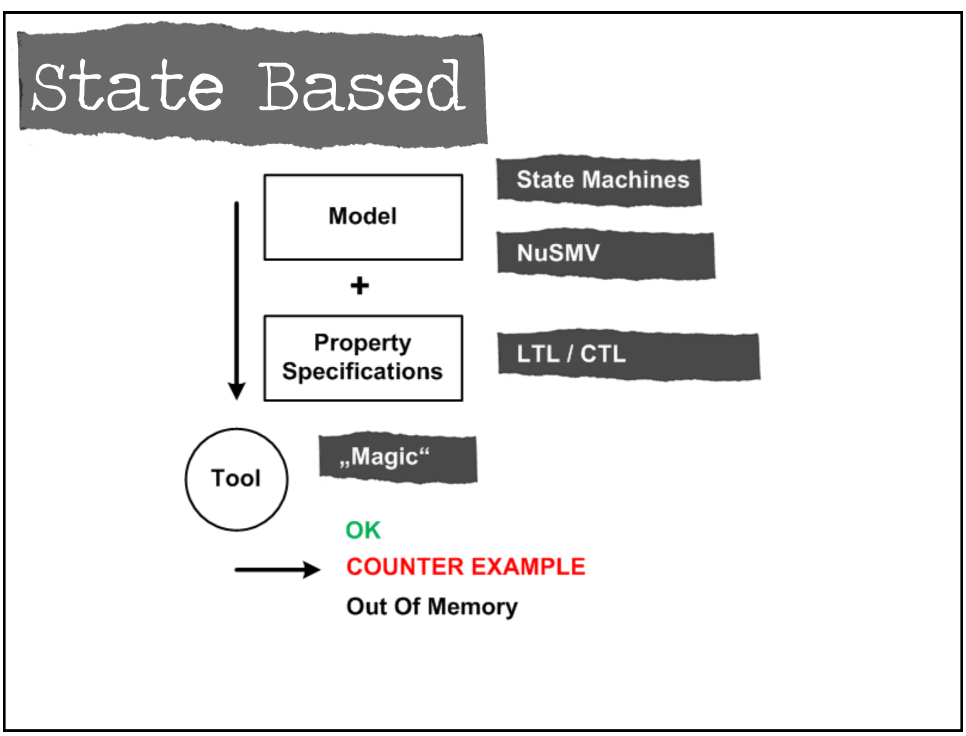
SAT Solving

Exhaustive Search, Proof!

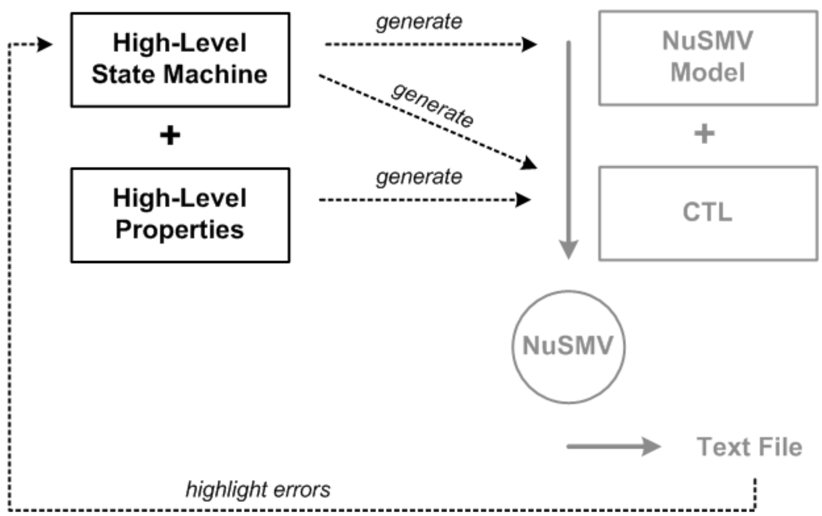
Challenges




Challenges
Model
Formalism
Input Language
+
Property Specifications
Property Language
Tool
Algorithm
OK
COUNTER EXAMPLE
Out Of Memory
Interpretation


State Based
Model
State Machines
NuSMV
+
Property Specifications
LTL / CTL
Tool
„Magic“
OK
COUNTER EXAMPLE
Out Of Memory

mbeddr Approach

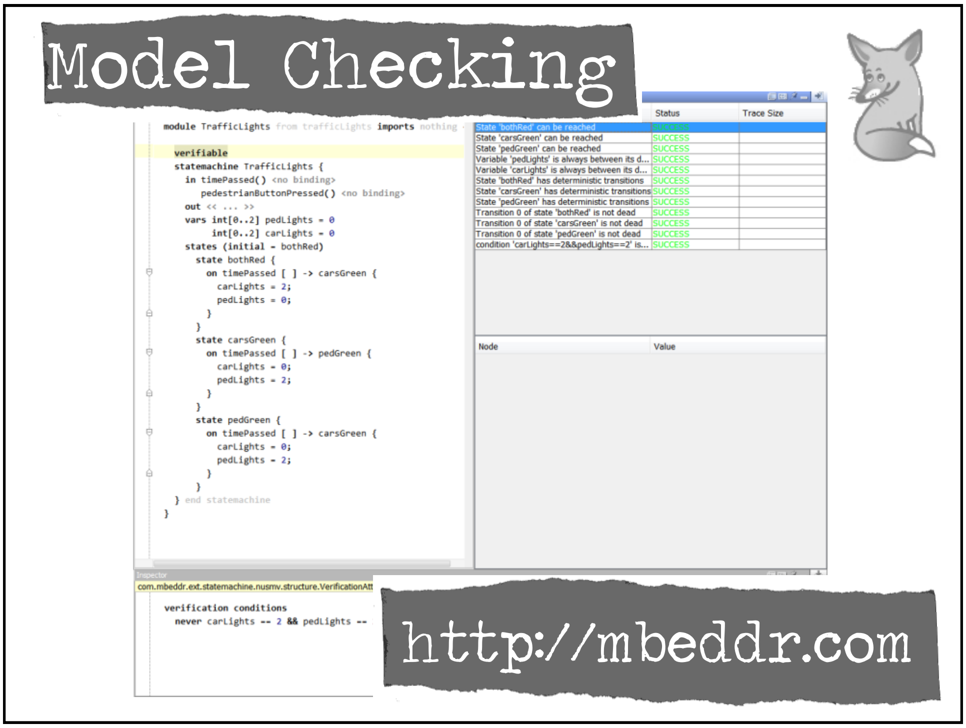


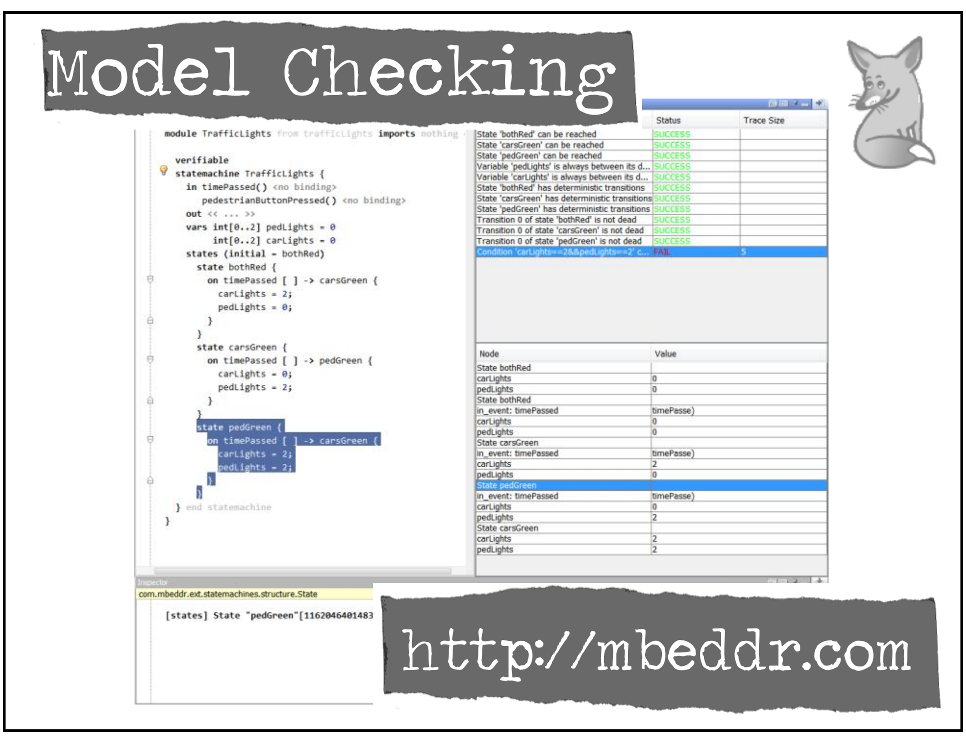
mbeddr Approach

easier to use

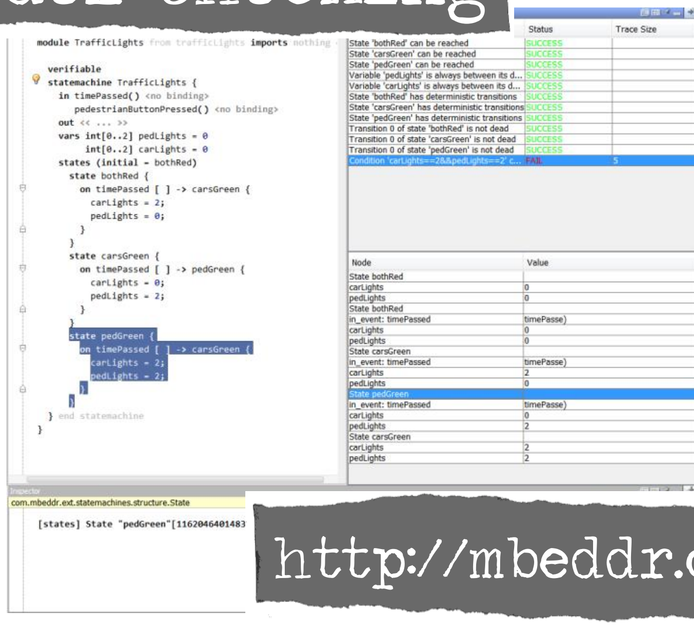
hopefully used more

full power: write CTL/LTL if you want to


Model Checking
module TrafficLights
verifiable
statemachine TrafficLights {
verification conditions
never carLights == 2 && pedLights ==
http://mbeddr.com


Model Checking
module TrafficLights
verifiable
statemachine TrafficLights {
[states] State "pedGreen"[1162046401483
http://mbeddr.com

Model Checking



http://mbeddr.com

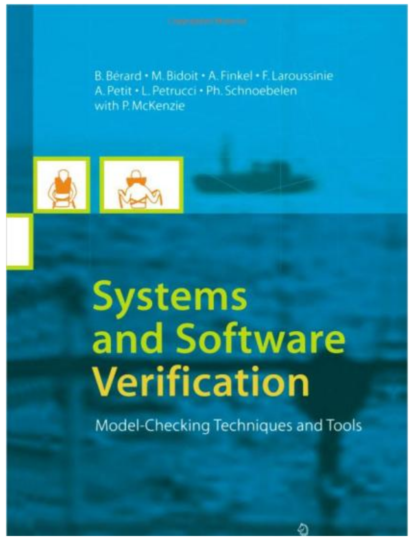
Model Checking

Finds problems in state machines.

... even ones you didn't think of!

Much more complete than manual testing.

Model Checking

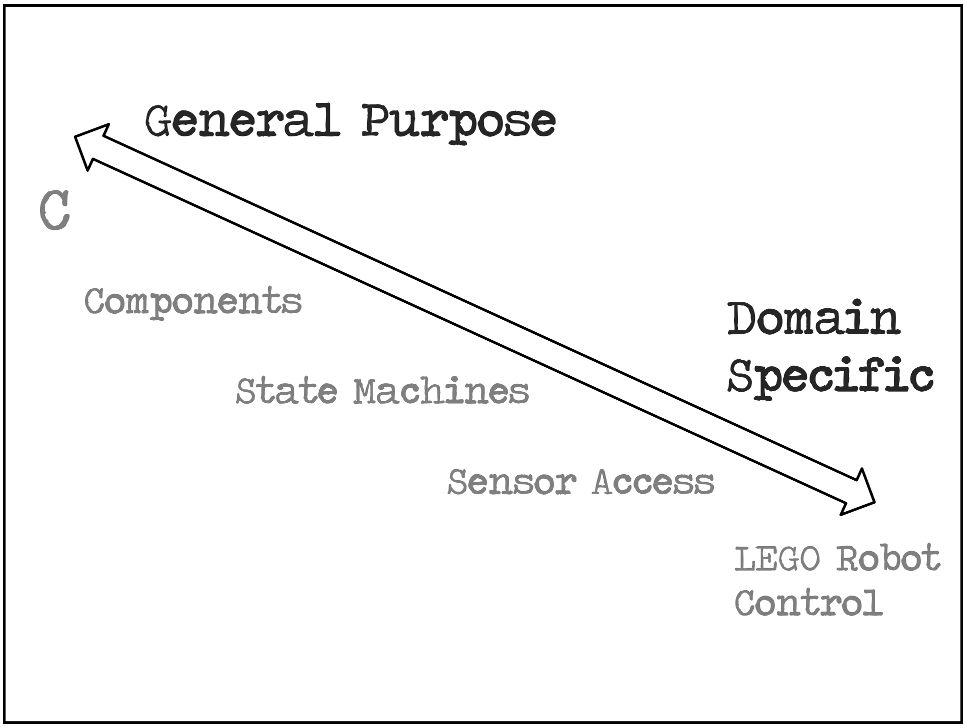


Language Modularity

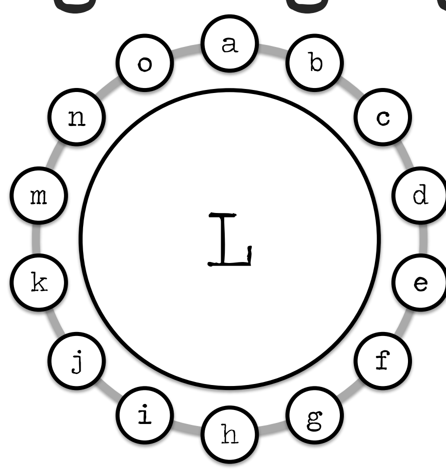
expressivity
coverage
semantics
separation of concerns

completeness
paradigms
modularity
concrete syntax

process

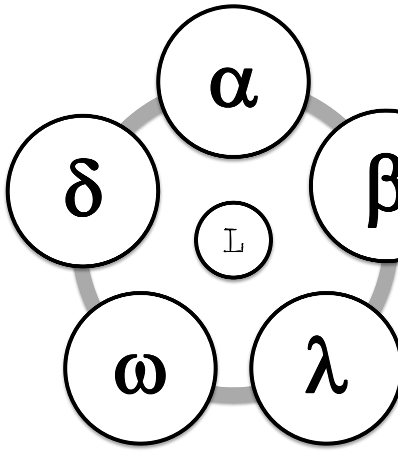


Big Language



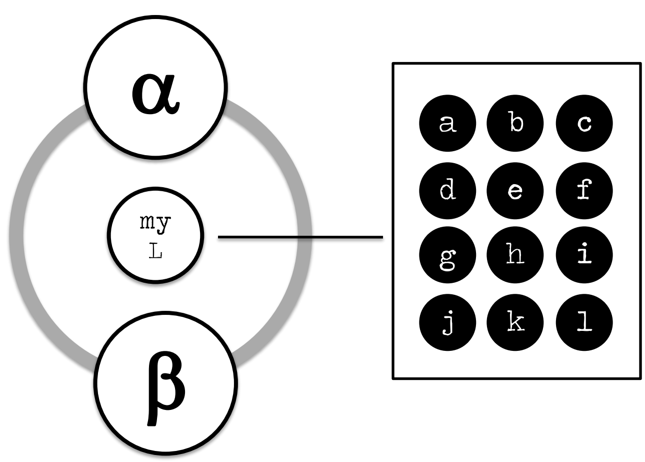
with **many** first class concepts!

Small Language



with a **few, orthogonal** and **poweful** concepts

Modular Language



with many **optional, composable** modules

Language
does not depend on
any other language

Independence

Fragment
does not depend on
any other fragment

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

Refrigerators

Homogeneous

Fragment

everything expressed
with one language

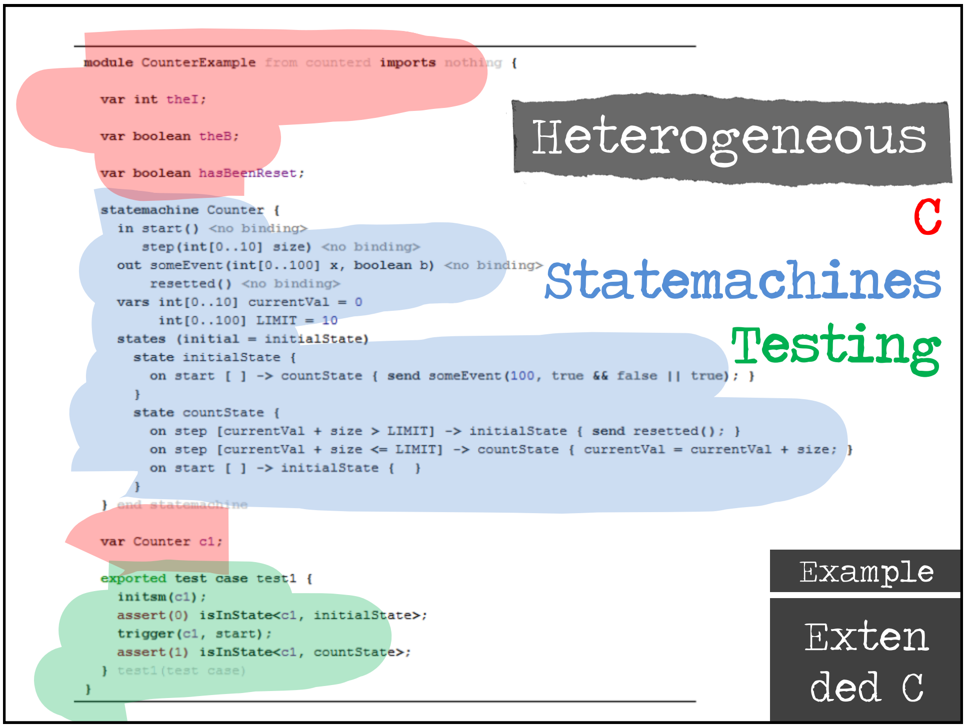
$$\forall e \in E_f \mid lo(e) = l$$
$$\forall c \in Cdn_f \mid lo(c.parent) = lo(c.child) = l$$

```
module CounterExample from counterd imports nothing {

  var int theI;

  var boolean theB;

  var boolean hasBeenReset;

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

Heterogeneous

Example

Extended C



Language Modularity, Composition and Reuse

increase efficiency of DSL development

4 ways of composition:

Referencing
Extension
Reuse
Embedding

Language Modularity, Composition and Reuse

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:

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

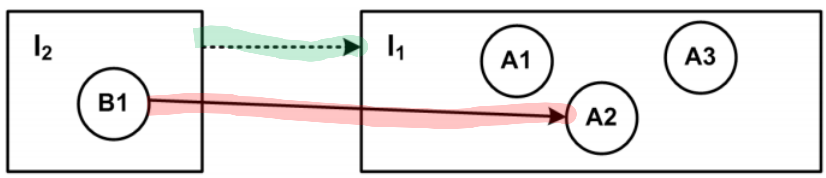
fragment structure

Referencing

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

fragment structure

Dependent

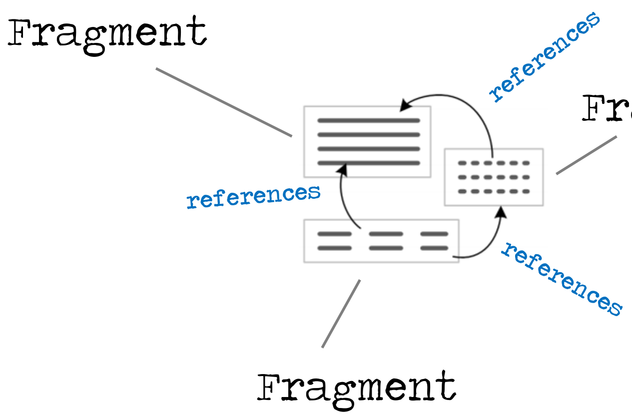


No containment

Referencing

Used in Viewpoints

Referencing



Referencing

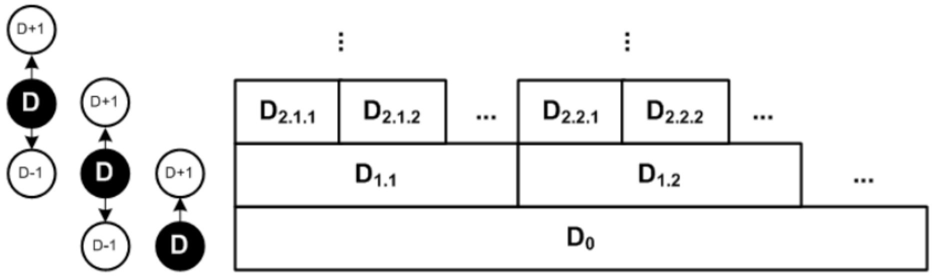
```
parameter t_abtaustart: int
parameter t_abtaudauer: int
parameter T_abtauEnde: int

var tuerNachlaufSchwelle: int = 0

start:
    entry { state noCooling }

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


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

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

step 10
assert-currentstate-is noCooling

mock: set RC->accumulatedRuntime = 110
step


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

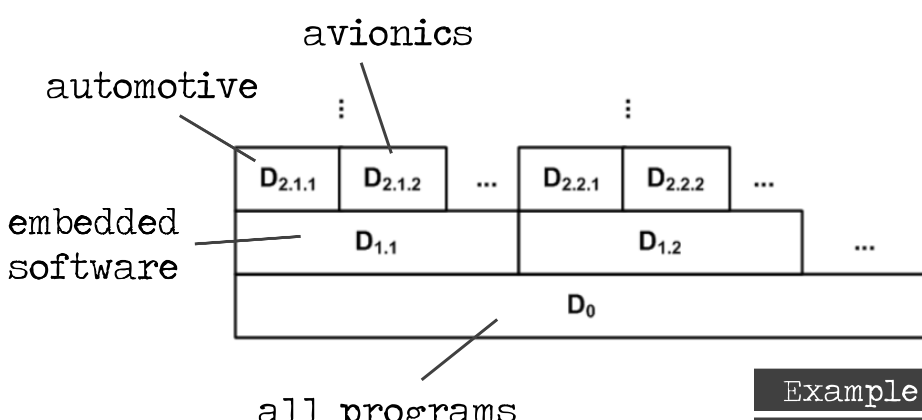
Example

Refrige
rators

Domain Hierarchy



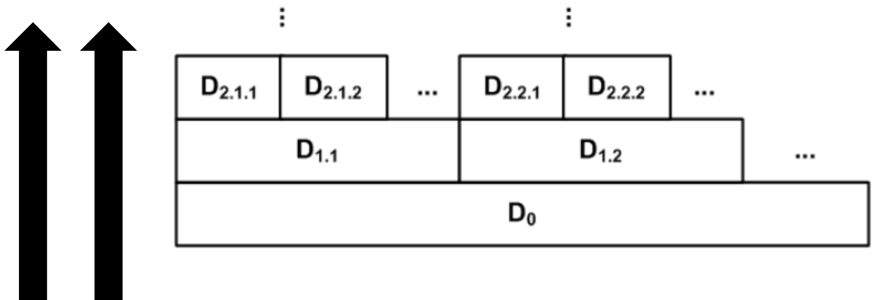
Domain Hierarchy



Example

Extended C

Extension



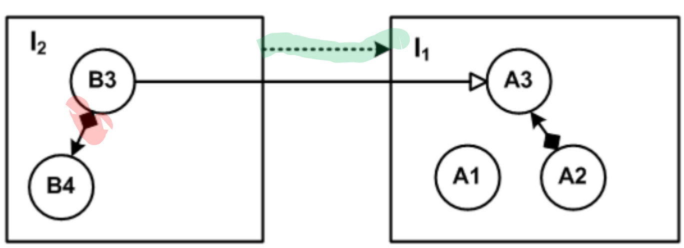
more specialized domains
more specialized languages

Extension

languages dependencies	homogeneous	heterogeneous
independent	Reuse	Embedding
dependent	Referencing	Extension

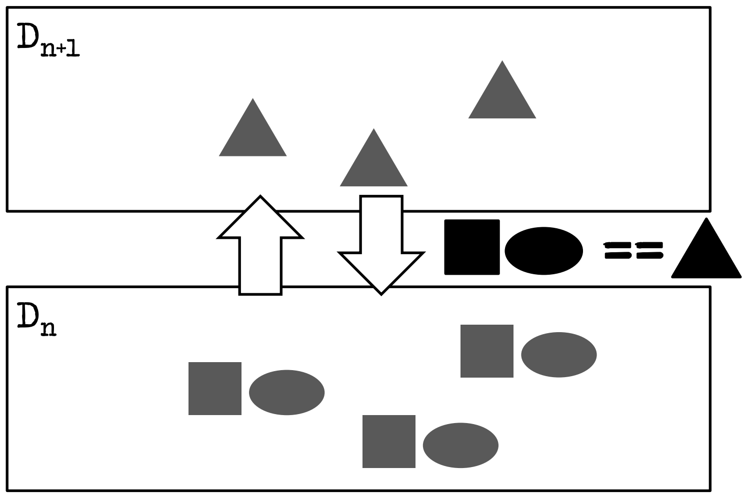
fragment structure

Dependent

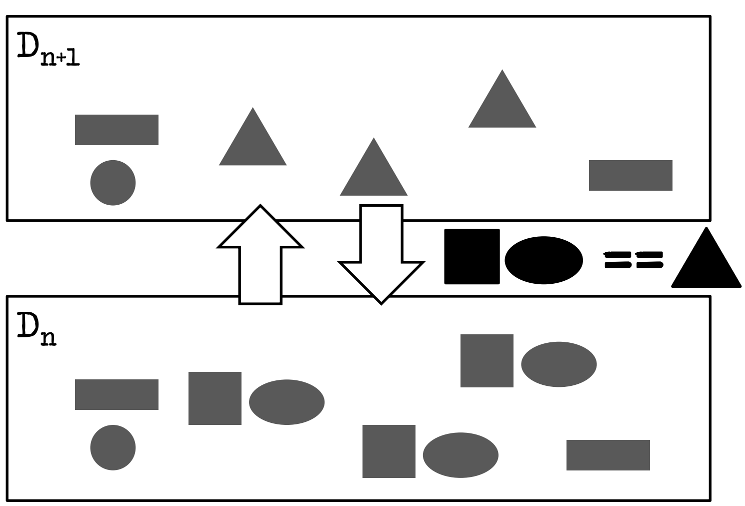


Containment

Extension

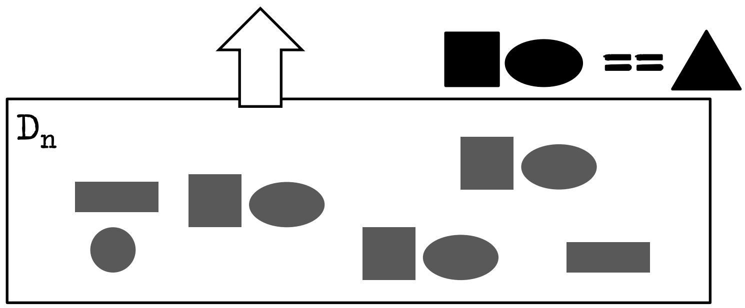


Extension



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, BitLevelUtilies {

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

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

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

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

```
task run cyclic prio = 1 every = 2 {
  stateswitch linefollower
    state running
      int32 light = 0;
      light = ecrobot_get_light_sensor
                    (SENSOR_PORT_T::NXT_PORT_S1);
      if ( light < ( WHITE + BLACK ) / 2 ) {
        updateMotorSettings(SLOW, FAST);
      } else {
        updateMotorSettings(FAST, SLOW);
      }
    default
      <noop>;
}

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

Example

Extended C

```
module CounterExample from counterd imports nothing {

  var int theI;

  var boolean theB;

  var boolean hasBeenReset;

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

Extension

C

Statemachines

Testing

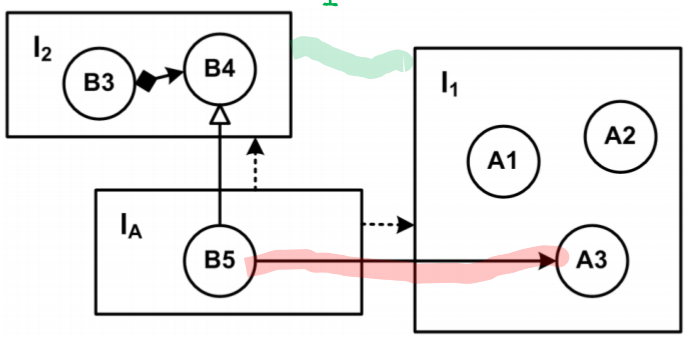
Example

Extended C

Reuse

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

Independent



No containment

Reuse

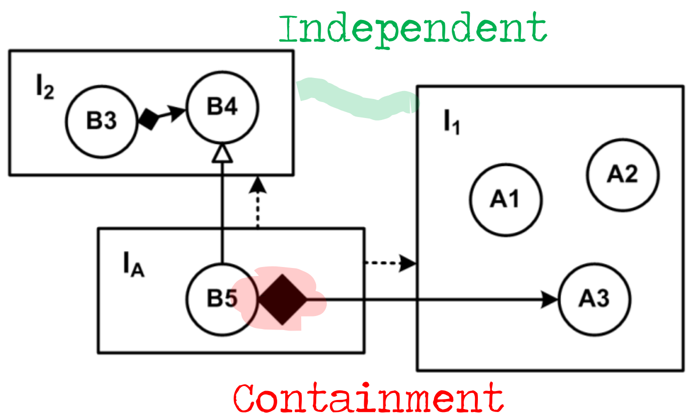
Often the referenced language is built expecting it will be reused.

Hooks may be added.

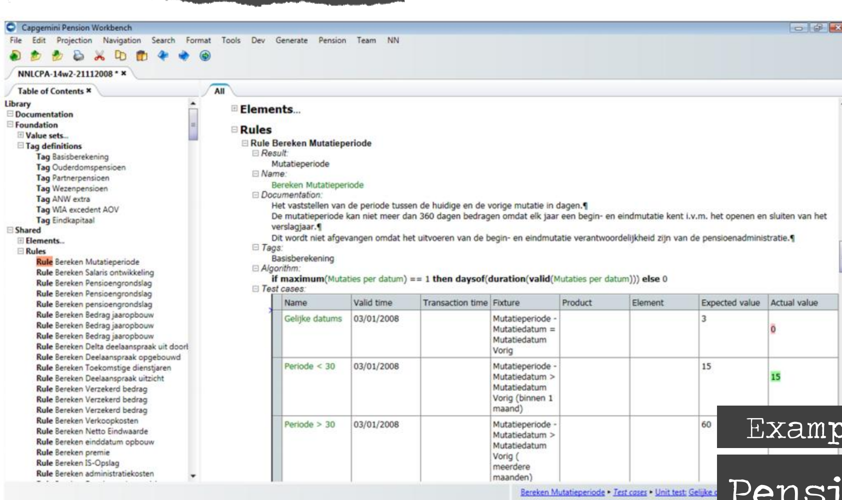
Embedding

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

languages dependencies (rows) / **fragment structure** (columns)



Embedding



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

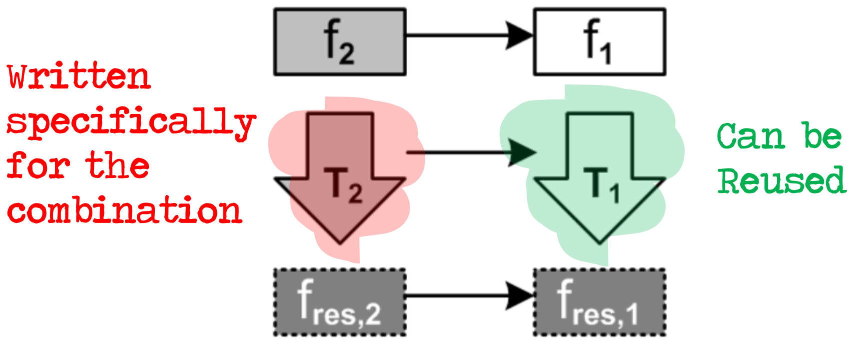
Extension: the type system of the base language must be designed to be extensible/ overridable

Challenges - Type Systems

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

Challenges - Trafo & Gen

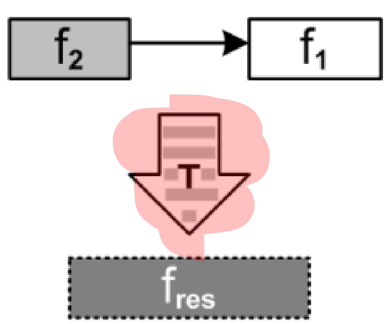
Referencing (I)



Two separate, dependent single-source transformations

Challenges - Trafo & Gen

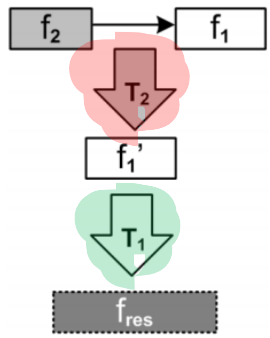
Referencing (II)



A single multi-sourced transformation

Challenges - Trafo & Gen

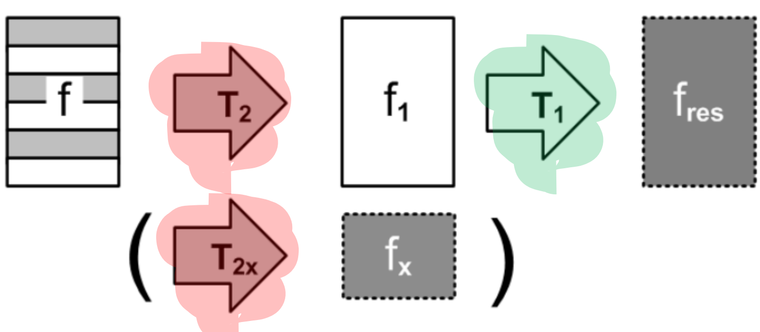
Referencing (III)



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

Challenges - Trafo & Gen

Extension

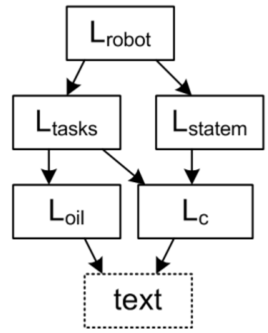


Transformation by assimiliation, i.e. generating code in the host lang from code expr in the extension lang.

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

}
```

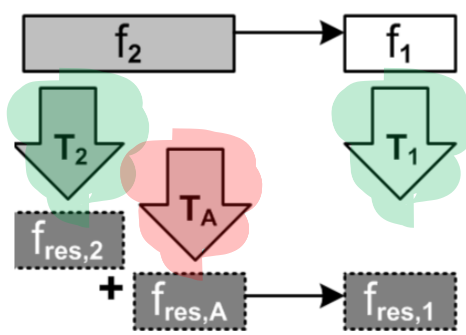


Example

Extended C

Challenges - Trafo & Gen

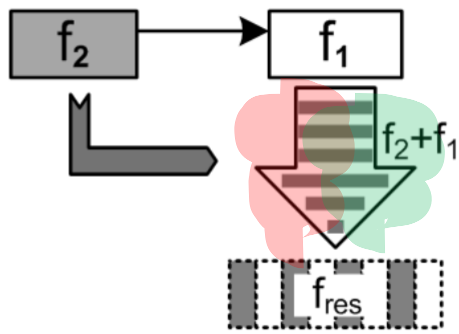
Reuse (I)



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

Challenges - Trafo & Gen

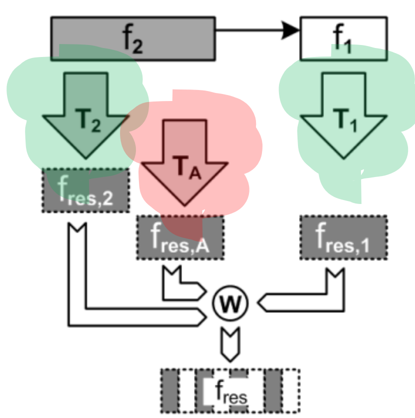
Reuse (II)



composing transformations

Challenges - Trafo & Gen

Reuse (III)

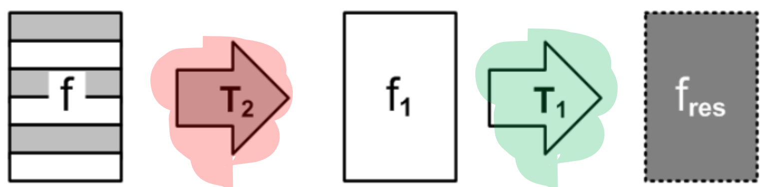


generating separate artifacts
plus a weaving specification

Challenges - Trafo & Gen

Embedding (I)

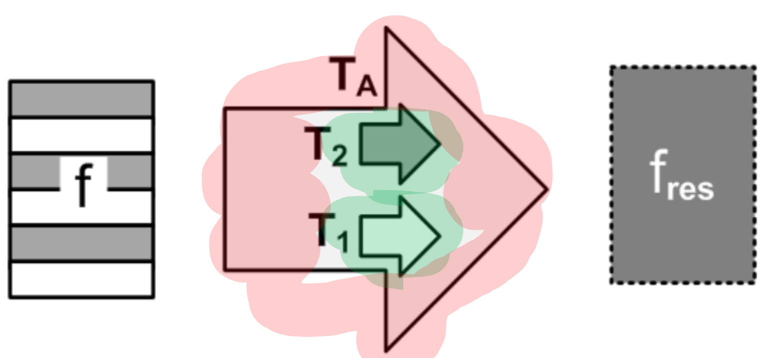
a purely embeddable language
may not come with a generator.



Assimilation (as with Extension)

Challenges - Trafo & Gen

Embedding (II)



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

Concrete Syntax

expressivity
coverage
semantics
separation of concerns

completeness
paradigms
modularity
concrete syntax

process

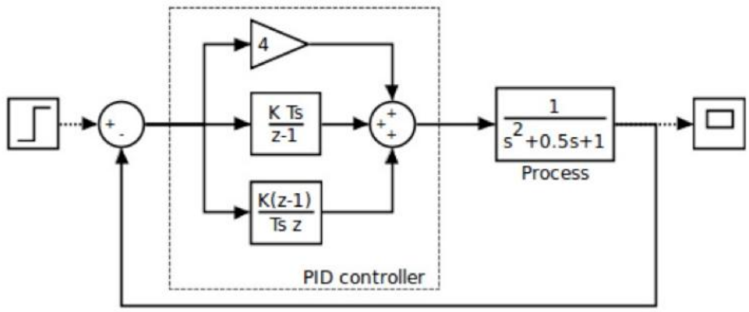
Combinations

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

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

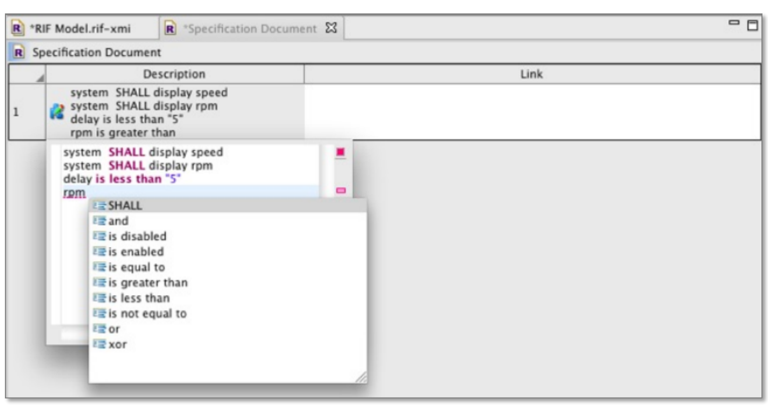
	x == 0	x > 0
y == 0	0	1
y > 0	1	2

```
      ;
    }
}
```

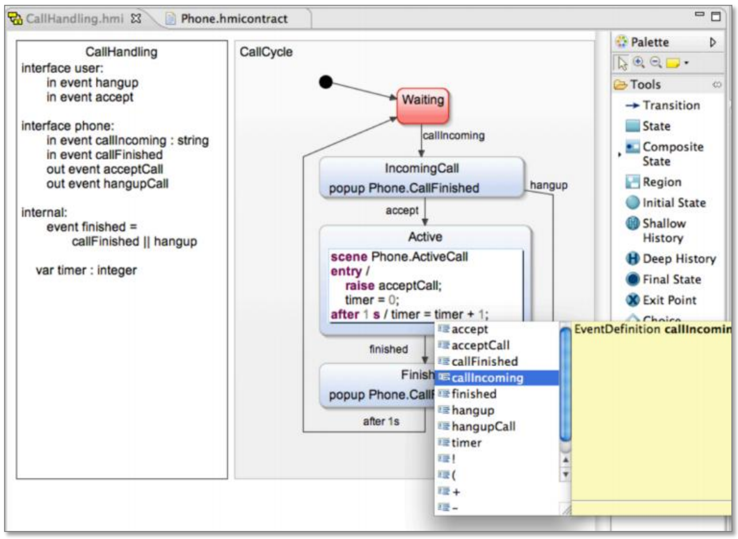
Combinations



Combinations



Combinations



The End.

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

http://voelter.de/dslbook

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