DSL Design
A conceptual framework for building good DSLs

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/
This material is based on this book:

http://dslbook.org

available Feb 2013
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
<table>
<thead>
<tr>
<th></th>
<th>more in GPLs</th>
<th>more in DSL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain Size</td>
<td>large and complex</td>
<td>smaller and well-defined</td>
</tr>
<tr>
<td>Designed by</td>
<td>guru or committee</td>
<td>a few engineers and domain experts</td>
</tr>
<tr>
<td>Language Size</td>
<td>large</td>
<td>small</td>
</tr>
<tr>
<td>Turing-completeness</td>
<td>almost always</td>
<td>often not</td>
</tr>
<tr>
<td>User Community</td>
<td>large, anonymous and widespread</td>
<td>small, accessible and local</td>
</tr>
<tr>
<td>In-language abstraction</td>
<td>sophisticated</td>
<td>limited</td>
</tr>
<tr>
<td>Lifespan</td>
<td>years to decades</td>
<td>months to years (driven by context)</td>
</tr>
<tr>
<td>Evolution</td>
<td>slow, often standardized</td>
<td>fast-paced</td>
</tr>
<tr>
<td>Incompatible Changes</td>
<td>almost impossible</td>
<td>feasible</td>
</tr>
</tbody>
</table>
Big Language

with many first class concepts!
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 ...
    }
}
namespace com.mycompany.test {
    system testSystem {
        instance dc: DelayCalculator
        instance screen1: InfoScreen
        instance screen2: InfoScreen
        connect dc.screens to
            (screen1.default, screen2.default)
    }
}
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
  }

  }
}
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
      }
   }  


```plaintext
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 rccooping
  }
  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 rccooping:
  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 abtau
assert-value cc.c1->active is false

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

Example

Refrigerators
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 code

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);
  }
}
3.3 Commutatiegetallen op 1 leven

\[ D_x = \frac{x}{v} \cdot \frac{l_x}{100} \approx 6 \text{ Dec}(3) \]

Implemented in \[ \omega(x) \]

\[ N = \sum_{t=0}^{\omega-x} D_x \approx 7 \text{ Dec}(3) \]

3.6 Contante waarde 1 leven/2 levens

\[ E_x = \frac{x}{D_x} \approx 19 \text{ Dec}(4) \]

\[ a_x = \frac{\tilde{a}_x}{x} - 1 \approx 21 \text{ Dec}(3) \]

\[ \tilde{a}_x = \tilde{a}_x - 0.5 \approx 22 \text{ Dec}(3) \]

\[ \frac{N - N_x}{D_x} = \frac{x}{\sum_{x}^{x} D_x} \approx 23 \text{ Dec}(3) \]

\[ \frac{\tilde{a}_x}{x} = \frac{\tilde{a}_x - 0.5 + 0.5 E_x}{x} \approx 25 \text{ Dec}(3) \]

4 BN(-ris) koopsommen

Example

Pension Plans
Web DSL Example

```java
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 }
}
```
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
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
Model

an abstraction or simplification of reality
Model

an abstraction or simplification of reality

which ones?

what should we leave out?

cosurvey.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
Model Purpose

... code generation
... analysis and checking
... platform independence
... stakeholder integration
Model Purpose

... code generation
... analysis and checking
... platform independence
... stakeholder integration
Model Purpose

... code generation
... analysis and checking
... platform independence
... stakeholder integration
Programs
Languages
Domains
Domain

body of knowledge in the real world

deductive top down

existing software (family)

inductive bottom up
Domain

deductive
top down

body of
knowledge
in the real
world

Example
Penion Plans

Example
Refrigerators
existing software
(family)

inductive bottom up

Domain

Example

Extended Components

Example

Extended C
A DSL $L_D$ for D is a language that is specialized to encoding $P_D$ programs. More efficient and smaller.
Programs are trees

```
  M
 / \  
A   B
   /   
  C     D
   / 
  E   F
   / 
  G
```

Element
Fragments are subtrees w/ root

fragment root

fragment

fragment
Parent-Child Relation

\[ f : M \xrightarrow{\text{parent}} Cdnf \xrightarrow{\text{child}} \]

Diagram:
- M is the parent
- Cdnf is the child

Nodes:
- A
- B
- C
- D
- E
- F
- G
Programs and Fragments

\[ f \Rightarrow \text{element} \Rightarrow \text{fragment} \]
Programs are graphs, really.
Programs are graphs, really.
Languages are sets of concepts

\[ L \subseteq \{C_1, C_2, C_3, \ldots, C_n\} \]
Languages are sets of concepts

\[ L \rightarrow \text{lo} \rightarrow \text{concept} \rightarrow \text{language} \]
Programs and languages

\[ C_1 \quad C_2 \quad C_3 \quad C_n \]

\[ \text{LocalVariableDeclaration} \]

\[ co \Rightarrow \text{element} \Rightarrow \text{concept} \]

\[ x = 3; \]
Language:
concept inheritance

\[ L_1 \rightarrow C_1 \rightarrow C_2 \rightarrow L_2 \]

\[ L_1 \rightarrow C_1 \rightarrow C_3 \rightarrow L_2 \]

\[ C_1 \rightarrow \text{Statement} \]

\[ C_1 \rightarrow \text{LocalVariableDeclaration} \]

\[ \text{super} \]

\[ \text{Inh}_1 \]

\[ \text{sub} \]
Language does not depend on any other language

\[ \forall r \in \text{Refs}_1 \mid \text{lo}(r.\text{to}) = \text{lo}(r.\text{from}) = l \]
\[ \forall s \in \text{Inh}_1 \mid \text{lo}(s.\text{super}) = \text{lo}(s.\text{sub}) = l \]
\[ \forall c \in \text{Cdn}_1 \mid \text{lo}(c.\text{parent}) = \text{lo}(c.\text{child}) = l \]

Independence

Fragment does not depend on any other fragment

\[ \forall r \in \text{Refs}_f \mid \text{fo}(r.\text{to}) = \text{fo}(r.\text{from}) = f \]
\[ \forall e \in \text{E}_f \mid \text{lo}(\text{co}(e)) = l \]
Independence

Hardware:

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

Cooling Algorithm

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

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)
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>
    reseted() <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 reseted(); }
      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)
Domain Hierarchy
Domain Hierarchy

all programs

embedded software

automotive

avionics

Example Extended C
Design Dimensions

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

process
Expressivity

expressivity  completeness
coverage  paradigms
semantics  modularity
separation of  concrete
concerns  syntax

process
Shorter Programs

More Accessible Semantics
For a limited Domain!

Domain Knowledge encapsulated in language
Smaller Domain

More Specialized Language

Shorter Programs
The
do-what-I-want
language
Ψ
Single Program vs. Class/Domain
No Variability!
Domain Hierarchy

more specialized domains
more specialized languages
Reification

$D_n$
Reification

\[ D_{n+1} \]

\[ \Rightarrow \]

\[ D_n \]
Reification

Transformation/Generation

Language
Definition

\[ \square \circ = \triangle \]
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!

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

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

Declarative! Directly represents Semantics.
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!
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

In-Language Abstraction
User-Definable
Simpler Language

Special Treatment!
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
}
```
Language Evolution Support
Precision vs. Algorithmics
Coverage

expressivity  |  completeness
coverage     |  paradigms
semantics    |  modularity
separation of concerns | concrete
process  |  syntax
NOTICE

THIS PART OF THE SLIDE DECK HAS BEEN SKIPPED.

PLEASE REFER TO THE DSL ENGINEERING BOOK.
Static Semantics

Execution Semantics
Static Semantics

Constraints

Type Systems
Unique State Names
Unreachable States
Dead End States

Example
Extended C
Unique State Names
Unreachable States
Dead End States
Easier to do on a declarative Level!
Unique State Names
Unreachable States
Dead End States

Easier to do on a declarative Level!

Thinking of all constraints is a coverage problem!
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?
Assign fixed types
Derive Types
Calculate Common Types

What does a type system do?
Assign fixed types
Derive 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
Refrigerators

Example

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

incompatible type BoolType|COMPARABLE and StringType (on a AssignmentStatement)
What does it all mean?

Execution Semantics
Def: Semantics

... via mapping to lower level

$$\text{semantics}(p_{LD}) := q_{LD-1}$$

where $$\text{OB}(p_{LD}) == \text{OB}(q_{LD-1})$$

OB: Observable Behaviour (Test Cases)
Def: Semantics

... via mapping to lower level

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

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

$D_{n+1}$

$D_n$
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 

}
Transformation

\[ L_D \]

\[ L_{D-1} \]  Known Semantics!
Transformation

Correct!?

$L_D$

$L_{D-1}$

Transformation
Run tests on both levels; all pass.
Coverage Problem!
parameter t_abtaustart: int
parameter t_abtaudauer: int
parameter T_abtauEnde: int

var tuenNachlaufSchwelle: 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 tuenNachlaufSchwelle = currStep + 30
  }
  exit {
    perform rcfanabschalttask after max(5, tuenNachlaufSchwelle-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 abtau
assert-value cc.c1->active is false
mock: set RC->accumulatedRuntime = 0
step 5
assert-currentstate-is abtau
assert-value cc.c1->active is false
step 15
assert-currentstate-is noCooling
## Transformation

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

### Example Pension Plans
Example Refrigerators
Multi-Stage

\[ L_3 \]
\[ \downarrow \]
\[ L_2 \]
\[ \downarrow \]
\[ L_1 \]
\[ \downarrow \]
\[ L_0 \]

Modularization
Multi-Stage: Reuse

Reusing Later Stages Optimizations!
Multi-Stage: Reuse

Robot Control
State Machine

Components

C (MPS tree)

C Text
Multi-Stage: Reuse

Robot Control
State Machine

Components

Consistency
Model Checking

Efficient Mappings

C Type System

C (MPS tree)

Syntactic
Correctness,
Headers

C Text

Extended C

Example
Multi-Stage: Reuse

Reusing Early Stages

Portability
Multi-Stage: Reuse

Extended Example

Java

C#
Multi-Stage: Preprocess

Adding an optional, modular emergency stop feature
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 $\rightarrow$ step through
- functional $\rightarrow$ eval recursively
- declarative $\rightarrow$ ? solver ?
<table>
<thead>
<tr>
<th>Name</th>
<th>Documentation</th>
<th>Tags</th>
<th>Valid time</th>
<th>Transaction time</th>
<th>Fixture</th>
<th>Product</th>
<th>Element</th>
<th>Expected value</th>
<th>Actual value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accrued right at retireme</td>
<td></td>
<td></td>
<td>2006-12-31</td>
<td>2007-9-24</td>
<td>Jan De Jong</td>
<td>Old Age Pension</td>
<td>Accrued right</td>
<td>761.0402</td>
<td>761.0402</td>
</tr>
<tr>
<td>Accrued Right last final pay</td>
<td></td>
<td></td>
<td>2004-1-1</td>
<td>2007-9-24</td>
<td>Jan De Jong</td>
<td>Old Age Pension</td>
<td>Accrued right</td>
<td>705.0589</td>
<td>705.0589</td>
</tr>
<tr>
<td>premium last year</td>
<td></td>
<td></td>
<td>2006-1-1</td>
<td>2007-9-24</td>
<td>Jan De Jong</td>
<td>Old Age Pension</td>
<td>Premium old age pension</td>
<td>329.0625</td>
<td>329.0625</td>
</tr>
<tr>
<td>Accrued right at retireme 2)</td>
<td></td>
<td></td>
<td>2006-12-31</td>
<td>2007-9-24</td>
<td>Piet Van Dijk</td>
<td>Old Age Pension</td>
<td>Accrued right</td>
<td>740.94</td>
<td>724.7658</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1985-12-31</td>
<td>2007-9-24</td>
<td>Jan De Jong</td>
<td>Old Age Pension</td>
<td>Accrued Right in service period</td>
<td>73.661</td>
<td>73.661</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1985-12-31</td>
<td>2007-9-24</td>
<td>Jan De Jong</td>
<td>Old Age Pension</td>
<td>Years of service in service period</td>
<td>3.7534</td>
<td>3.7534</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1987-12-31</td>
<td>2007-9-24</td>
<td>Jan De Jong</td>
<td>Old Age Pension</td>
<td>Pension base average FP</td>
<td>7750</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1998-12-31</td>
<td>2007-9-24</td>
<td>Jan De Jong</td>
<td>Old Age Pension</td>
<td>Accrued Right in service period</td>
<td>387.7449</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1998-12-31</td>
<td>2007-9-24</td>
<td>Jan De Jong</td>
<td>Old Age Pension</td>
<td>Years of service in service period</td>
<td>10.8082</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1998-12-31</td>
<td>2007-9-24</td>
<td>Jan De Jong</td>
<td>Old Age Pension</td>
<td>Pension base average FP</td>
<td>8250</td>
<td></td>
</tr>
</tbody>
</table>
Refrigerators

```prolog
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
    }
  }
```

Example

Refrigerators
## Behavior

### Refrigerators

**Example**

### Refrigerators

**Simulation View**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RC.accumulatedRun</td>
<td>80</td>
</tr>
<tr>
<td>RC.needsCooling</td>
<td>false</td>
</tr>
<tr>
<td>c1.active</td>
<td>false</td>
</tr>
<tr>
<td>ccfan.active</td>
<td>false</td>
</tr>
<tr>
<td>rcdoor.open</td>
<td>false</td>
</tr>
<tr>
<td>rceva.evaTemp</td>
<td>20</td>
</tr>
<tr>
<td>rcfan.active</td>
<td>false</td>
</tr>
</tbody>
</table>

**Queue**

<table>
<thead>
<tr>
<th>Event</th>
<th>Data</th>
</tr>
</thead>
</table>

**Commands**

- St... Command

**Variable Values**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tuerNachlaufSchwel...</td>
<td>0</td>
</tr>
</tbody>
</table>

**Running Tasks**

<table>
<thead>
<tr>
<th>Task</th>
<th>Sinc...</th>
</tr>
</thead>
</table>

**Status**

- Current Test: KIRAbtau
- Current State: -
- Current Step: -

**Control**

- Autorun
- Single Step
- Enable Breakpoints

**Actions**

- Assert Current State
- Show Log
- Copy Test to Clipboard

**Buttons**

- Assert Selected Property
- Assert Selected Variable
Interpretation

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

An interpreter :-)

Text Editor/IDE → Textual Source Code → Parser → AST (Model as Objects) → Graphical Editor → Interpreter

Generator → GPL Source Code

Interpreter

Compiler → Byte/Machine Code
Transformation

+ Code Inspection

Interpretation
Transformation

+ Code Inspection

OSGi Generators

Interpretation
Transformation

+ Code Inspection
+ Debugging

Interpretation
Transformation

+ Code Inspection

+ Debugging

Platform Interactions

Interpretation
Transformation

+ Code Inspection
+ Debugging
+ Performance & Optimization

Interpretation
Transformation

+ Code Inspection
+ Debugging
+ Performance & Optimization

Efficiency for Real-Time S/w

Memory Use

Interpretation

Example

Embedded C
<table>
<thead>
<tr>
<th>Transformation</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ Code Inspection</td>
<td></td>
</tr>
<tr>
<td>+ Debugging</td>
<td></td>
</tr>
<tr>
<td>+ Performance &amp; Optimization</td>
<td></td>
</tr>
<tr>
<td>+ Platform Conformance</td>
<td></td>
</tr>
</tbody>
</table>
Transformation

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

Web Frameworks and Standards

Example

Web DSL
Transformation

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

Interpretation

+ Turnaround Time
Transformation

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

Interpretation

+ Turnaround Time
  Testing

Example

Pension Plans

Example

Refrigerators
Transformation

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

Interpretation

+ Turnaround Time
+ Runtime Change
Transformation

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

Interpretation

+ Turnaround Time
+ Runtime Change Business Rules without Redeployment

Example Pension Plans
Def: Semantics

... via mapping to lower level

\[ L_D \downarrow \]

Transformation

\[ L_{D-1} \]
Multiple Mappings
... at the same time

$\mathbb{L}_D$

$\mathbb{L}_x \quad \mathbb{L}_y \quad \mathbb{L}_z$

Similar Semantics?
Multiple Mappings

... at the same time

$L_D \begin{array}{c} \circ \text{T} \\ \hline \end{array}$

$L_x L_y L_z$

$\begin{array}{c} \circ \text{T} \\ \circ \text{T} \\ \circ \text{T} \end{array}$

Similar Semantics?

all green!
Multiple Mappings

... at the same time

$\mathbf{L}_D \circ \mathbf{T}$

$\mathbf{L}_x \circ \mathbf{T}$
$\mathbf{L}_y \circ \mathbf{T}$
$\mathbf{L}_z \circ \mathbf{T}$

Similar Semantics?

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

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 rccoooling
  }
  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 rccoooling:
  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 rcfanabschalltask 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 abtau
assert-value cc.c1->active is false
mock: set RC->accumulatedRuntime = 0
step 5
assert-currentstate-is abtau
assert-value cc.c1->active is false
step 15
assert-currentstate-is noCooling
Multiple Mappings

... alternatively, selectably

\[ L_D \]

Extend \( L_D \) to include explicit data that determines transformation

\[ L_{x} \quad L_{y} \quad L_{z} \]
Multiple Mappings

... alternatively, selectably

\[ L_D \]

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.11
    contents:
    field int8_t count = 0
```

Restricted Port leads to reduced overhead C
Multiple Mappings

... alternatively, selectably

\[ L_D \]

External Data:
- Switches
- Annotation Model

\[ L_x \]
\[ L_y \]
\[ L_z \]
Multiple Mappings

... alternatively, selectably

External Data:
- Switches
- Annotation Model

Switch Control
Java vs. C
Multiple Mappings

... alternatively, selectably

Heuristics: Analyze model to try to decide

\[ L_D \]

\[ L_X \quad L_Y \quad L_Z \]
Multiple Mappings

... alternatively, selectably

TESTING!
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!
requirements modules: bls

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) => handleOut

    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;
        }
      }
    }
  }
}

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
            $\begin{array}{ccc}
            x == 0 & x > 0 \\
            y == 0 & 0 & 1 \\
            y > 0 & 1 & 2 \\
            \end{array}$
        }
    }
}
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

\[ \forall r \in \text{Refs}_f \mid \text{fo}(r.\text{to}) = \text{fo}(r.\text{from}) = f \]

\[ \forall e \in E_f \mid \text{lo}(\text{co}(e)) = l \]

independent

dependent
Viewpoints

Hardware

Behaviour

Tests
Viewpoints: Why?

Sufficiency
Different Stakeholders
Different Steps in Process - VCS unit!
Viewpoints

Hardware
Product Management

Tests

Behaviour
Thermodynamics Experts

Example Refrigerators
Viewpoints

independent

sufficient?
contains all the data for running a meaningful transformation
Viewpoints

sufficient hardware structure documentation

sufficient implementation code
Viewpoints: Why?

1:n Relationships
Viewpoints

Hardware

Behaviour

Tests

1:n

n:1
Viewpoints

Well-defined Dependencies

No Cycles!

Avoid Synchronization!
(unless you use a projectional editor)
Viewpoints: Why?

L_D  \rightarrow L_{\text{Annotation}}

L_{D-1}
Progressive Refinement
Views on Programs

Achmea demo plan

T-OP65-TOTAAL-2006
Het totale ouderdomspensioen, opgebouwd in de oude of de nieuwe regeling

1999-01-01
[Diagram of connections between T-OP-TOTAAL and T-OP65-TOTAAL-2006]

2006-01-01

Example Pension Plans
Completeness

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

process
NOTICE

THIS PART OF THE SLIDE DECK HAS BEEN SKIPPED.

PLEASE REFER TO THE DSL ENGINEERING BOOK.
Fundamental Paradigms

expressivity | completeness
coverage | paradigms
semantics | modularity
separation of concerns | concrete
process | syntax
NOTICE

THIS PART OF THE SLIDE DECK HAS BEEN SKIPPED.

PLEASE REFER TO THE DSL ENGINEERING BOOK.
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:

- Independent languages and dependencies
- Reuse and embedding
- Referencing and extension
- Homogeneous and heterogeneous fragment structure
Dependencies & Fragment Structure:

Referencing

Reuse

Extension

Embedding
Referencing

Dependent

No containment
Referencing

Used in Viewpoints
Referencing

Fragment

Fragment

Fragment
Referencing

Example Refrigerators

```plaintext
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
```
Extension

languages
dependencies
dependent

Reuse  Embedding

Referencing  Extension

homogeneous  heterogeneous

fragment structure

\begin{figure}
\centering
\includegraphics[width=\textwidth]{diagram}
\caption{Diagram of Extension}
\end{figure}
Extension

Containment

Dependent

<table>
<thead>
<tr>
<th>Reuse</th>
<th>Embedding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Referencing</td>
<td>Extension</td>
</tr>
</tbody>
</table>

- independent
- languages
- dependencies
- dependent

- homogeneous
- heterogeneous
- fragment structure
Extension

more specialized domains
more specialized languages
Extension

\[ D_{n+1} \]

\[ D_n \]
Extension

\[ D_{n+1} \]

\[ D_n \]
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
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_S1);
    nxt_motor_set_speed(MOTOR_PORT_T::NXT_PORT_S1);
}
Extension

![Diagram of Extension Concept]

**Example Extended C**
Reuse

Independent

No containment
Reuse

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

Hooks may be added.
Embedding

<table>
<thead>
<tr>
<th>Reuse</th>
<th>Embedding</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Referencing

<table>
<thead>
<tr>
<th>homogeneous</th>
<th>heterogeneous</th>
</tr>
</thead>
<tbody>
<tr>
<td>fragment structure</td>
<td></td>
</tr>
</tbody>
</table>

independent
languages
dependencies
dependent

I₂

B₃ → B₄

I₁

A₁ → B₅ → A₃

Iₐ

B₅

A₂
Embedding

Containment

Independent
Embedding

Example Pension Plans
Embedding often uses Extension to extend the embedded language to adapt it to its new context.
NOTICE

THIS PART OF THE SLIDE DECK HAS BEEN SKIPPED.

PLEASE REFER TO THE DSL ENGINEERING BOOK.
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
<table>
<thead>
<tr>
<th>Name</th>
<th>Documentation</th>
<th>Tags</th>
<th>Valid time</th>
<th>Transaction time</th>
<th>Fixture</th>
<th>Product</th>
<th>Element</th>
<th>Expected value</th>
<th>Actual value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accrued right at retireme</td>
<td></td>
<td></td>
<td>2006-12-31</td>
<td>2007-9-24</td>
<td>Jan De Jong</td>
<td>Old Age Pension</td>
<td>Accrued right</td>
<td>761.0402</td>
<td>761.0402</td>
</tr>
<tr>
<td>Accrued Right last final pay</td>
<td></td>
<td></td>
<td>2004-1-1</td>
<td>2007-9-24</td>
<td>Jan De Jong</td>
<td>Old Age Pension</td>
<td>Accrued right</td>
<td>705.0589</td>
<td>705.0589</td>
</tr>
<tr>
<td>premium last year</td>
<td></td>
<td></td>
<td>2006-1-1</td>
<td>2007-9-24</td>
<td>Jan De Jong</td>
<td>Old Age Pension</td>
<td>Premium old age pension</td>
<td>329.0625</td>
<td>329.0625</td>
</tr>
<tr>
<td>Accrued right at retireme 2</td>
<td></td>
<td></td>
<td>2006-12-31</td>
<td>2007-9-24</td>
<td>Piet Van Dijk</td>
<td>Old Age Pension</td>
<td>Accrued right</td>
<td>740.94</td>
<td>724.7658</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1985-12-31</td>
<td>2007-9-24</td>
<td>Jan De Jong</td>
<td>Old Age Pension</td>
<td>Accrued Right in service period</td>
<td>73.661</td>
<td>73.661</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1985-12-31</td>
<td>2007-9-24</td>
<td>Jan De Jong</td>
<td>Old Age Pension</td>
<td>Years of service in service period</td>
<td>3.7534</td>
<td>3.7534</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1987-12-31</td>
<td>2007-9-24</td>
<td>Jan De Jong</td>
<td>Old Age Pension</td>
<td>Pension base average FP</td>
<td>7750</td>
<td>7750</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1998-12-31</td>
<td>2007-9-24</td>
<td>Jan De Jong</td>
<td>Old Age Pension</td>
<td>Accrued Right in service period</td>
<td>387.7449</td>
<td>387.7449</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1998-12-31</td>
<td>2007-9-24</td>
<td>Jan De Jong</td>
<td>Old Age Pension</td>
<td>Years of service in service period</td>
<td>10.8082</td>
<td>10.8082</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1998-12-31</td>
<td>2007-9-24</td>
<td>Jan De Jong</td>
<td>Old Age Pension</td>
<td>Pension base average FP</td>
<td>8250</td>
<td>8250</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Description</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>system SHALL display speed</td>
<td></td>
</tr>
<tr>
<td>system SHALL display rpm</td>
<td></td>
</tr>
<tr>
<td>delay is less than &quot;5&quot; rpm</td>
<td></td>
</tr>
<tr>
<td>rpm is greater than</td>
<td></td>
</tr>
</tbody>
</table>

- and
- is disabled
- is enabled
- is equal to
- is greater than
- is less than
- is not equal to
- or
- xor
Combinations
Process

expressivity  completeness
coverage  paradigms
semantics  modularity
separation of concrete
concerns  syntax

process
Domain Analysis

1. Interview Experts
2. Structure their Knowledge
3. Create Examples
4. Build the Language
5. Get Feedback
Iterate to goal
Create example-based tutorials!
Domain Folks Programming?

Precision vs. Algorithmics!
Domain Folks Programming?

- DU Reviewing
- Dev Coding
- DU/Dev Paired
- Coding
- DU
DSL as a Product

Release Plan
Bug Tracker
Testing!
Support
Documentation
Reviews become easier --- less code, more domain-specific
The End.

www.voelter.de
voelterblog.blogspot.de
@markusvoelter
+Markus Voelter
This material is based on this book:

http://dslbook.org

available Feb 2013