Working with multiple languages

Why and How

Markus Voelter
independent/itemis
voelter@acm.org

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

based on the DSL Engineering book and the mbeddr tool

I think there is
a world market
for maybe five
computers.

I think there is
a world market
for maybe five
computers.

Thomas J. Watson

640K is more memory than anyone will ever need.

640K is more memory than anyone will ever need.

Bill Gates

The Macintosh uses an experimental pointing device called a "mouse". There is no evidence that people want to use these things.

The Macintosh uses an experimental pointing device called a "mouse". There is no evidence that people want to use these things.

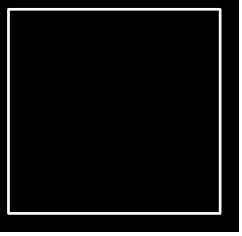
John C. Dvorak

Why would you want support for (modular) language composition?

You can always "cobble together" a few separate DSLs.

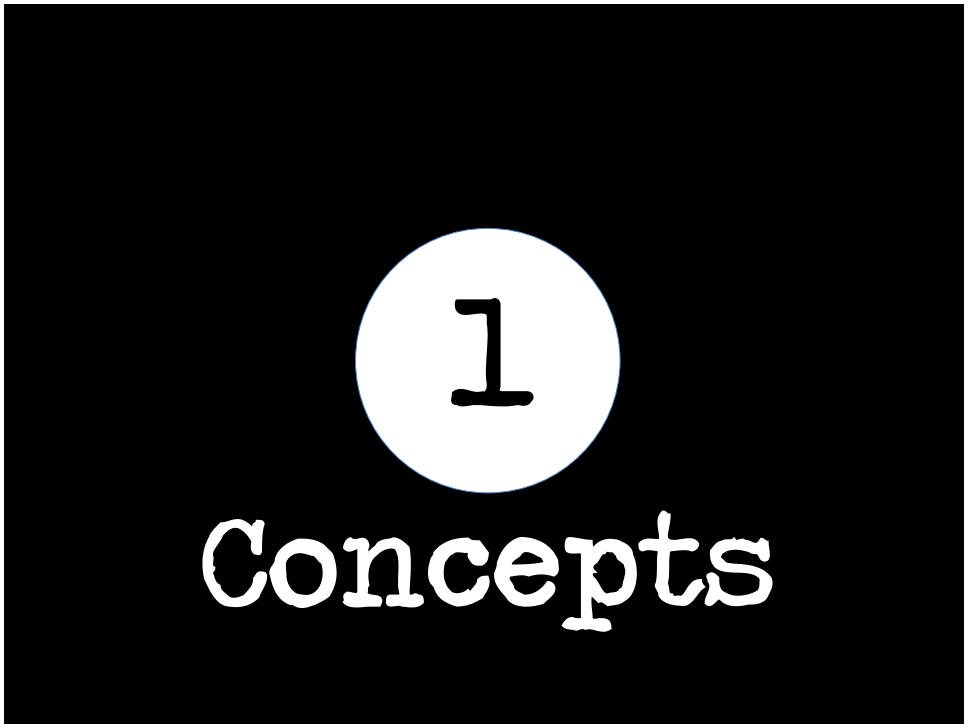
Why would you want support for (modular) language composition?

You can always "cobble together" a few separate DSLs.



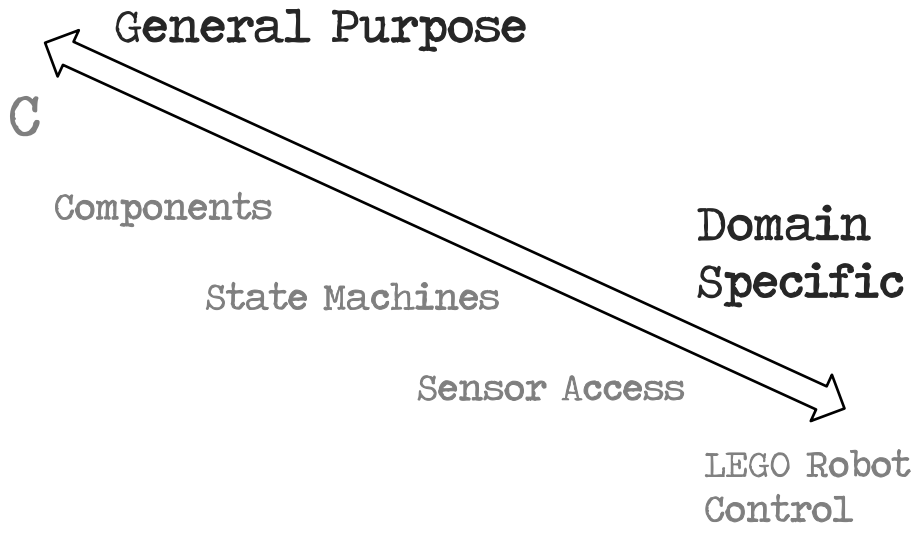
Anonymous

(1) Concepts
(2) Tools
(3) Demo

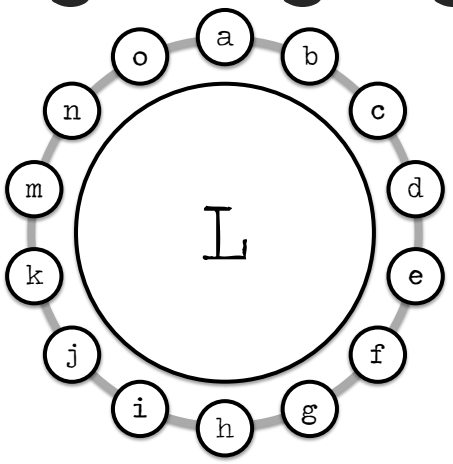


Introduction

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

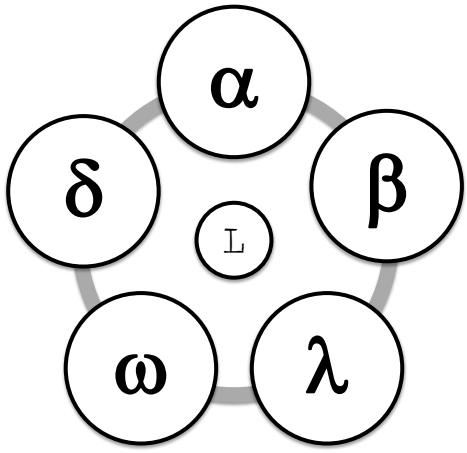


Big Language



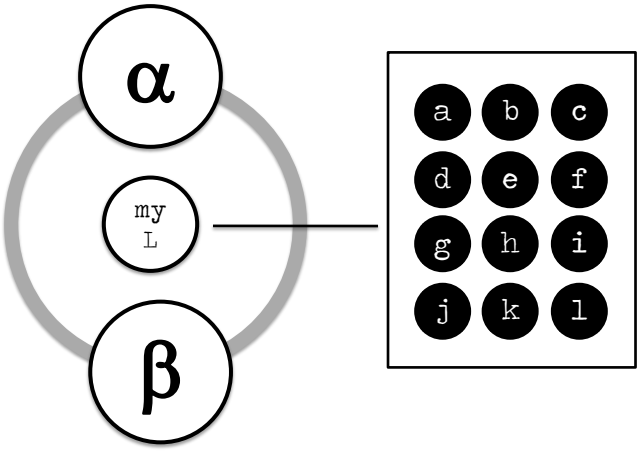
with **many** first class concepts!

Small Language



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

Modular Language



with many optional,
composable modules

Shorter
Programs

More
Accessible
Semantics

For a limited
Domain!

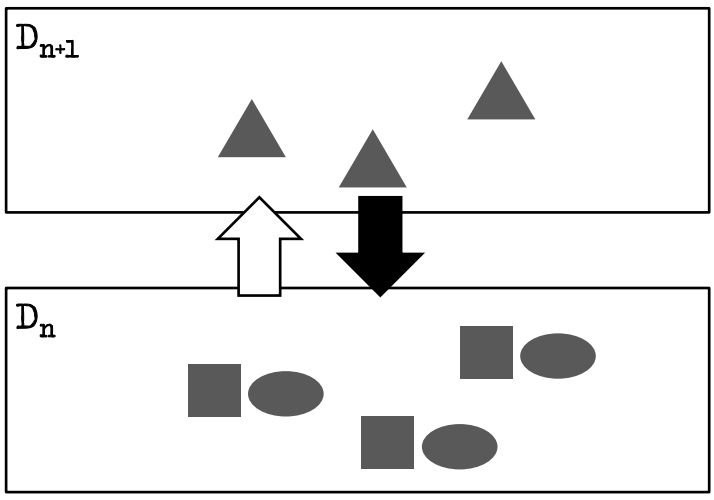
Domain Knowledge
encapsulated in
language

Multi-Stage Transformation

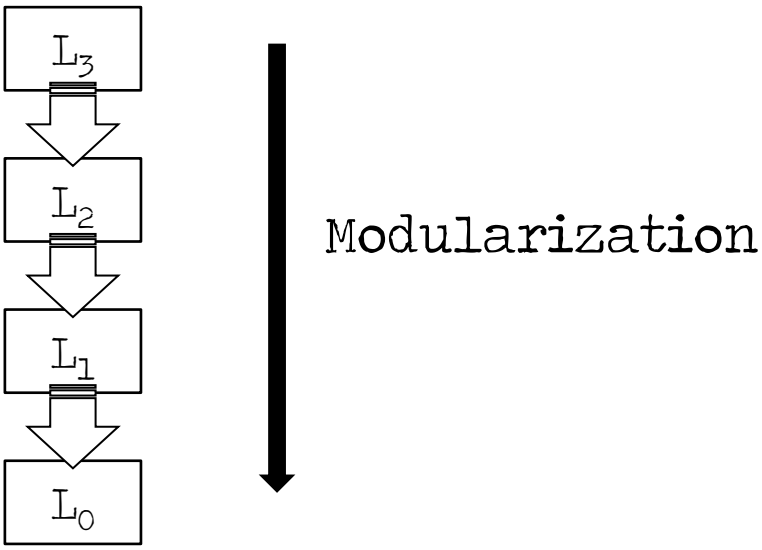
What does it all mean?

Execution Semantics

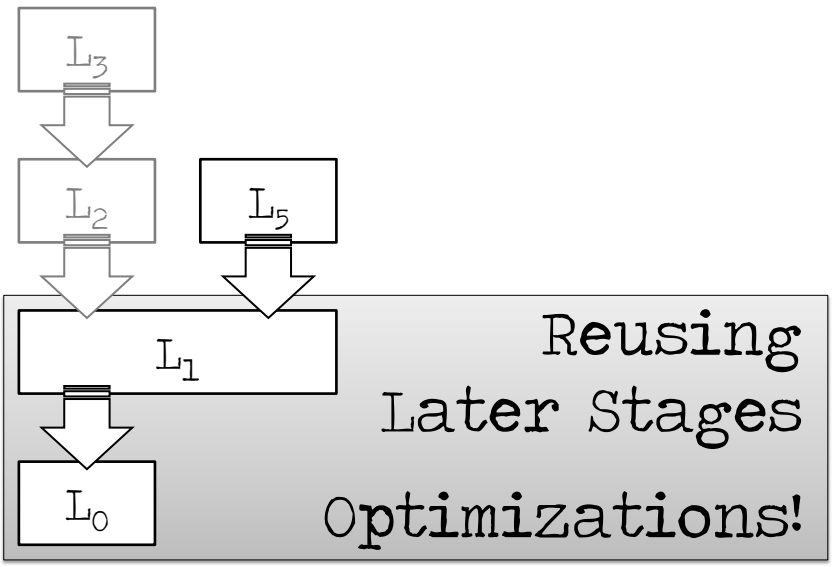
Transformation



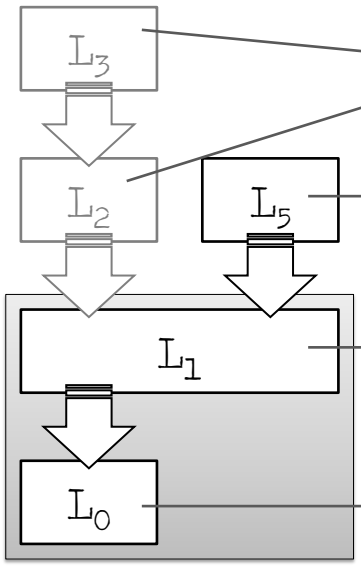
Multi-Stage



Multi-Stage: Reuse



Multi-Stage: Reuse

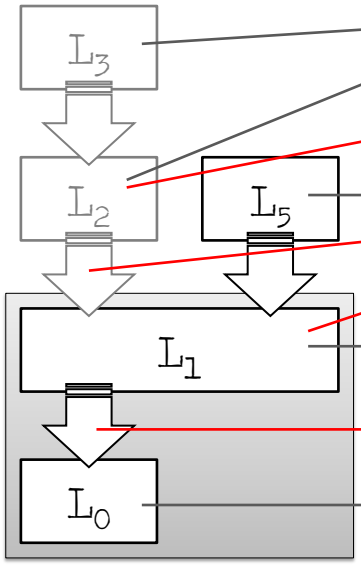


Robot Control
State Machine
Components
C (MPS tree)
C Text

Example

Exten
ded C

Multi-Stage: Reuse

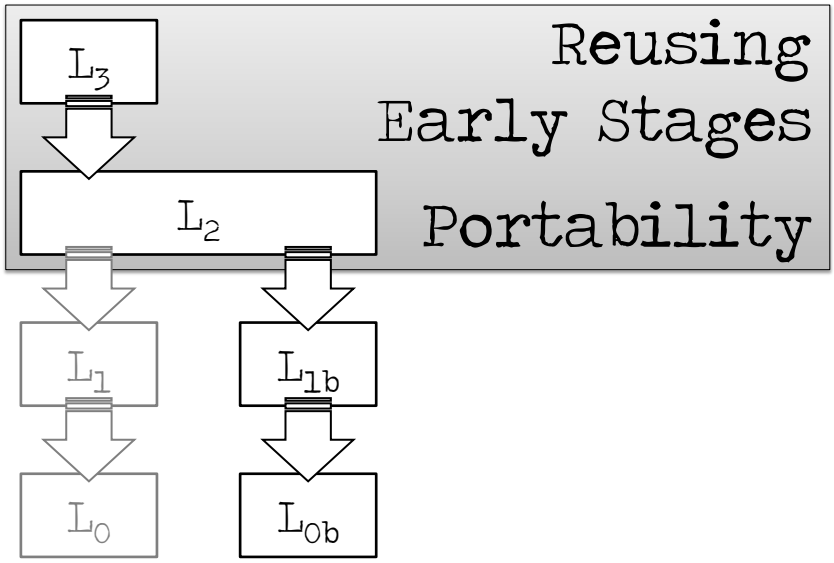


Robot Control
State Machine
Consistency
Model Checking
Components
Efficient Mappings
C Type System
C (MPS tree)
Syntactic
Correctness,
Headers
C Text

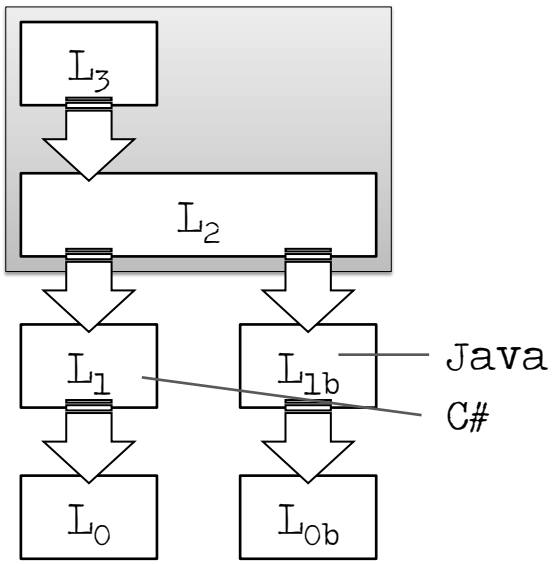
Example

Exten
ded C

Multi-Stage: Reuse



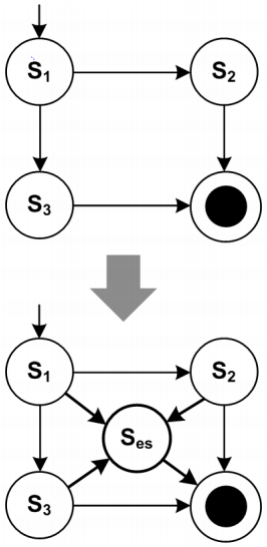
Multi-Stage: Reuse



Example

Exten
ded C

Multi-Stage: Preprocess

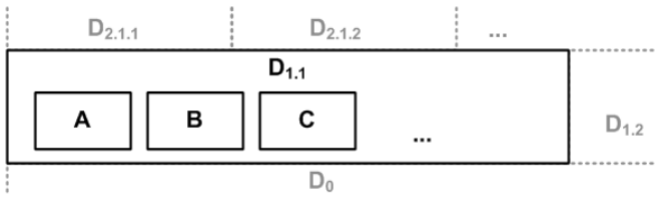


Adding an optional, modular emergency stop feature

Separation of Concerns - Referencing

Several Concerns

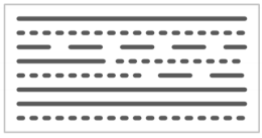
… in one domain



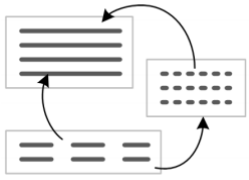
Several Concerns

… in one domain

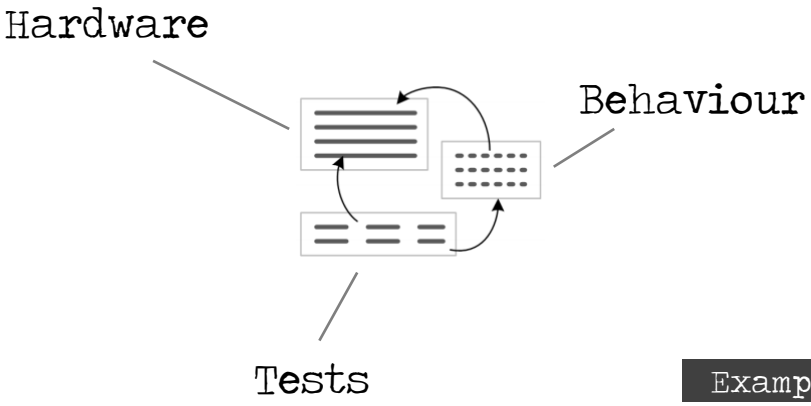
integrated into one fragment



separated into several fragments



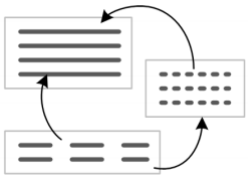
Viewpoints



Example

Refrige rators

Viewpoints: Why?

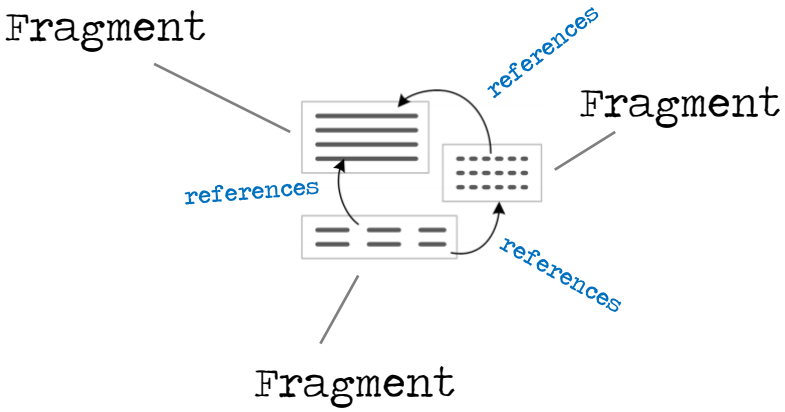


Sufficiency

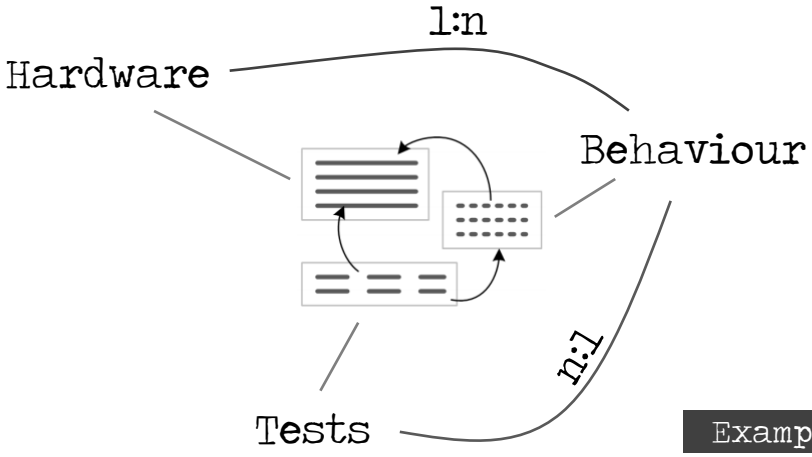
Different Stakeholders

Different Steps in Process - VCS unit!

Referencing



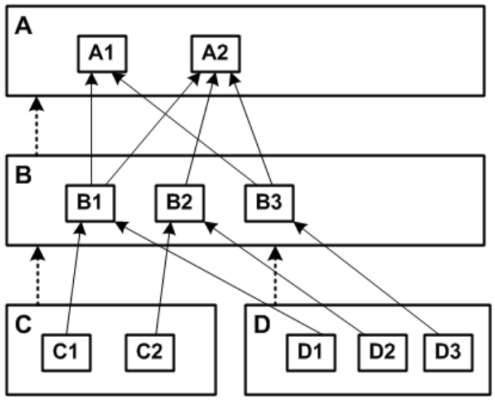
Viewpoints



Example

Refrige rators

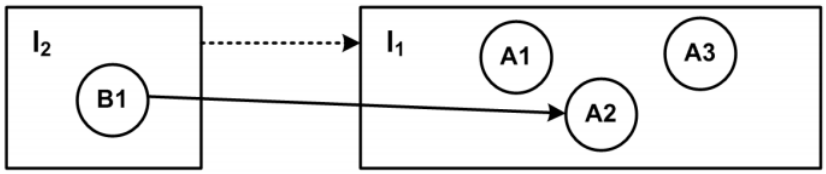
Progressive Refinement



Referencing

languages dependencies	homogeneous	heterogeneous
independent	Reuse	Embedding
dependent	Referencing	Extension

fragment structure

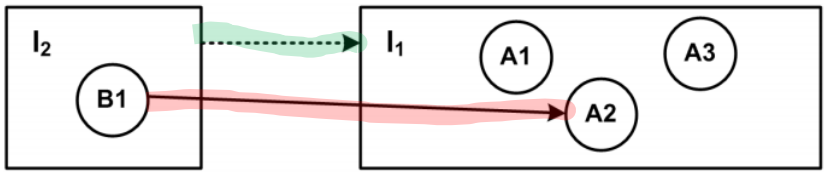


Referencing

languages dependencies	homogeneous	heterogeneous
independent	Reuse	Embedding
dependent	Referencing	Extension

fragment structure

Dependent



No containment

Referencing

```
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
        }
    }
```

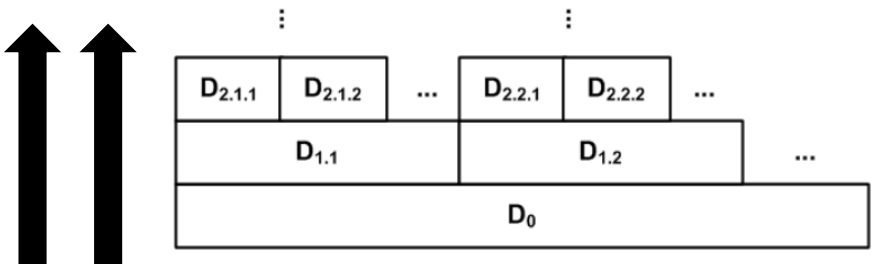
```
prolog {
    set RC->accumulatedRuntime = 80
}

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

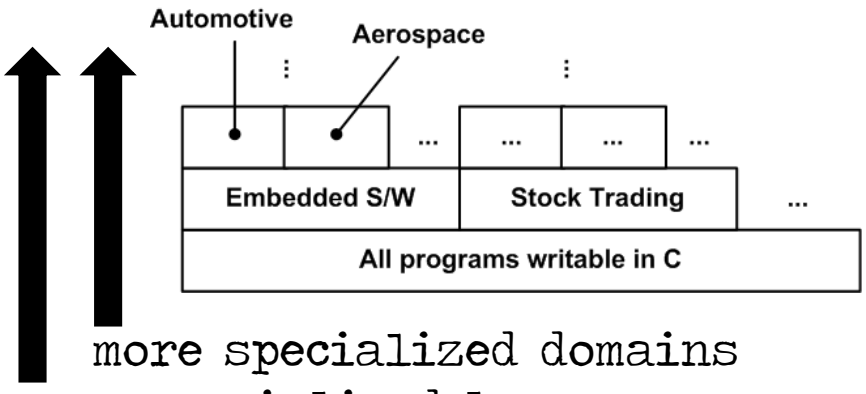
Growing a Language - Extension

Extension



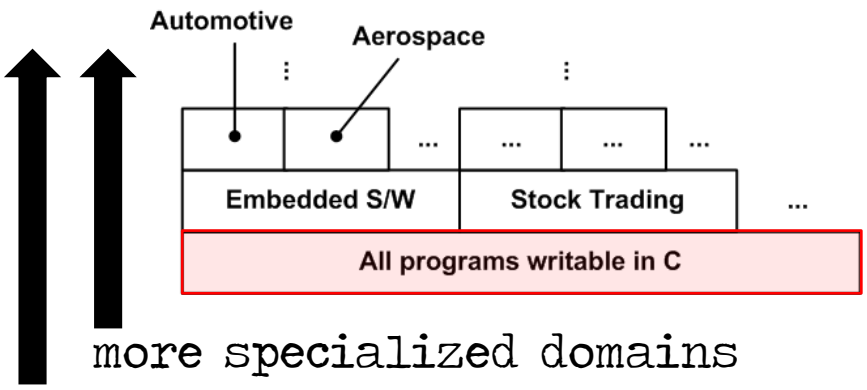
more specialized domains
more specialized languages

Extension



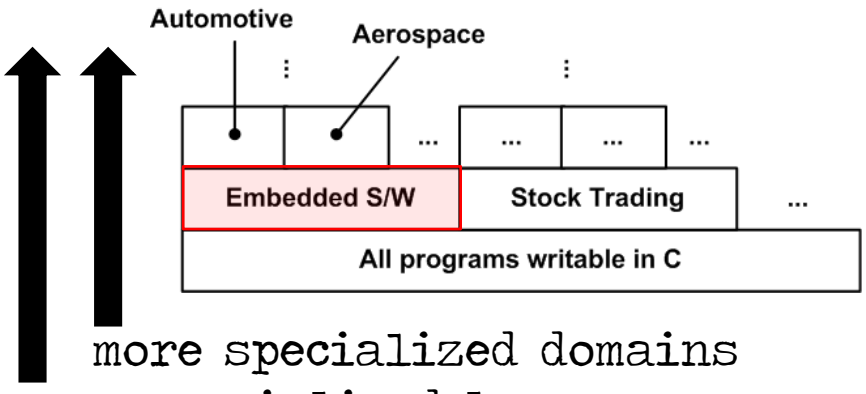
more specialized domains
more specialized languages

Extension



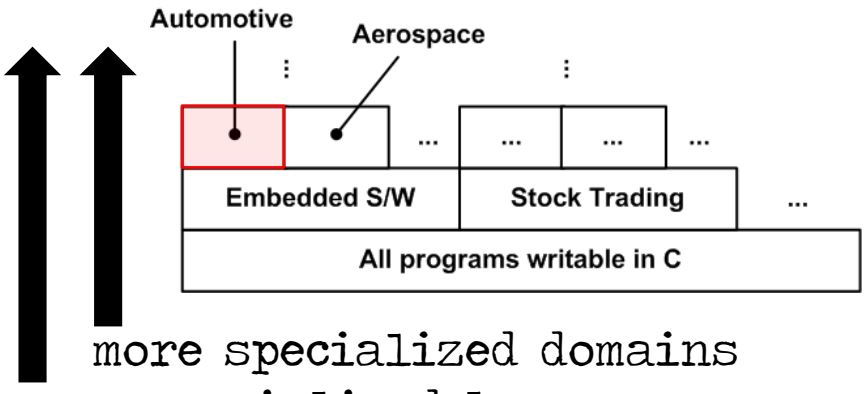
more specialized domains
more specialized languages

Extension



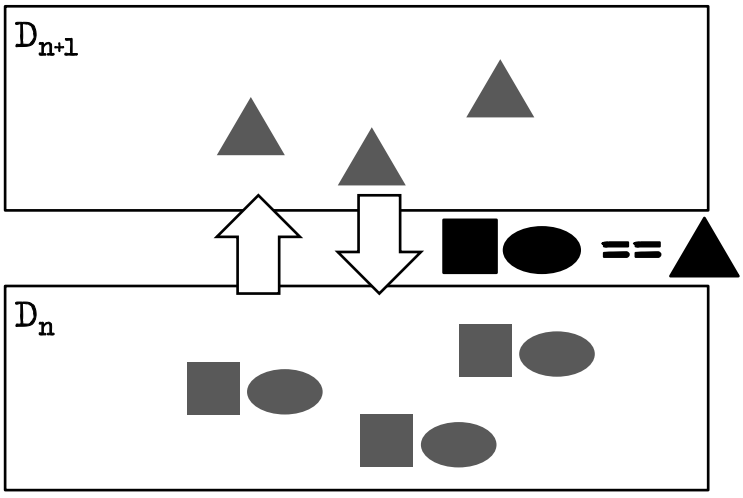
more specialized domains
more specialized languages

Extension

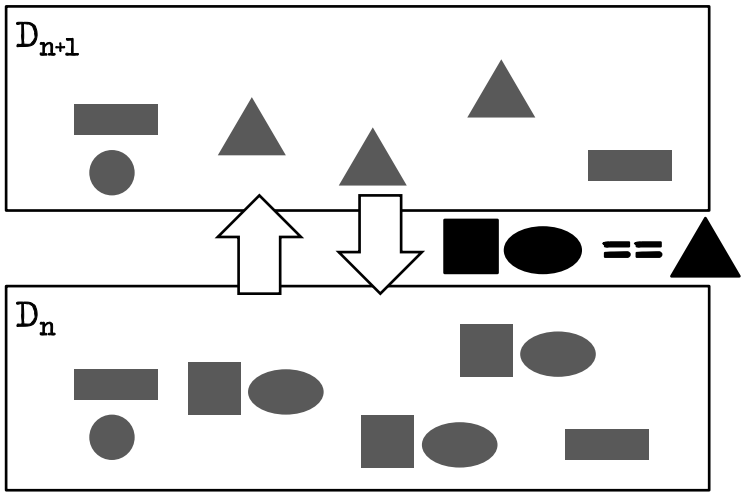


more specialized domains
more specialized languages

Extension

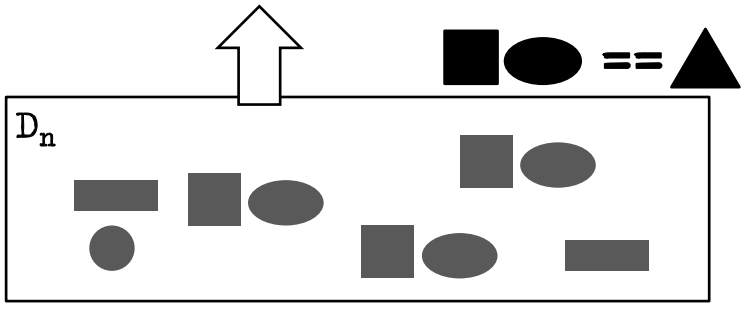


Extension



Extension

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



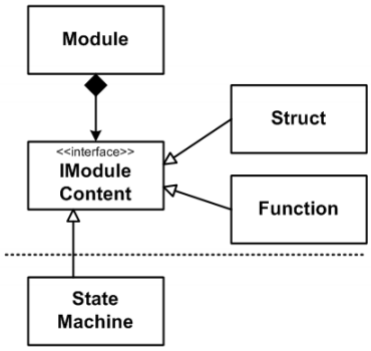
Extension

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

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

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

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

  terminate {
    ecrobot_set_light_sensor_inactive
              (SENSOR_PORT_T::NXT_PORT_S1);
  }

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

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

Example

Extended C

Extension



Example

Extended C

```
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!

Extension

Drawbacks

tightly bound to base
potentially hard to analyze
the combined program

Extension Composition

Extension combination refers to the ability to use a number of independent extensions of the same base language in a single program.

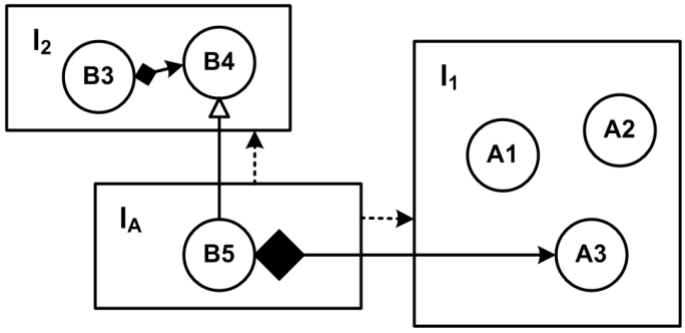
Critically Important!

Language Embedding

Embedding

languages dependencies	homogeneous	heterogeneous
independent	Reuse	Embedding
dependent	Referencing	Extension

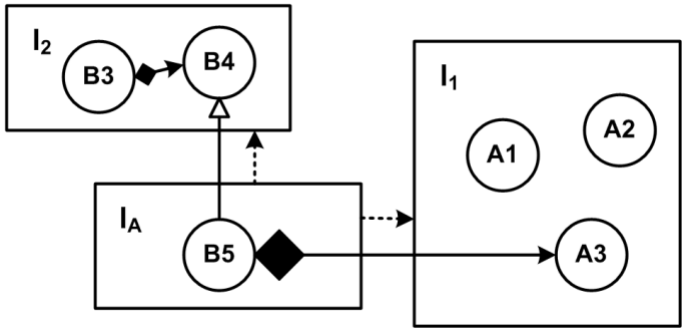
fragment structure



Embedding

languages dependencies	homogeneous	heterogeneous
independent	Reuse	Embedding
dependent	Referencing	Extension

fragment structure

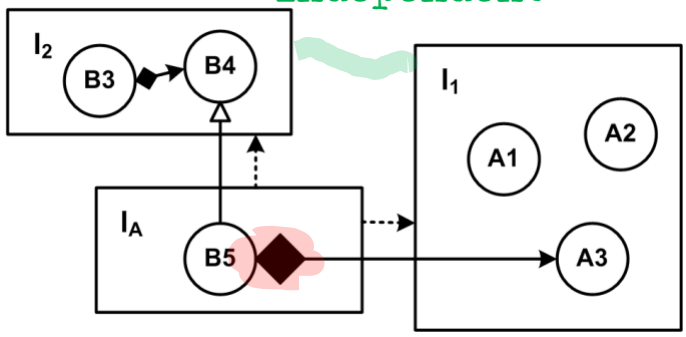


Embedding

languages dependencies	homogeneous	heterogeneous
independent	Reuse	Embedding
dependent	Referencing	Extension

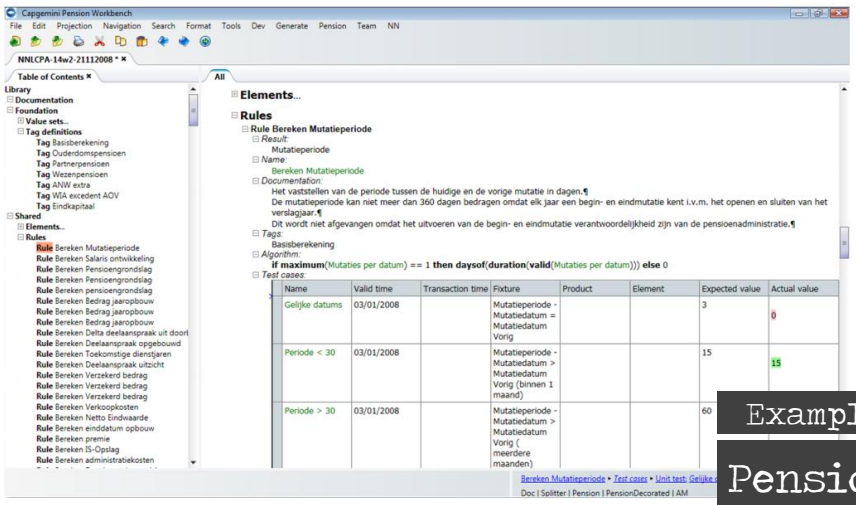
fragment structure

Independent



Containment

Embedding



Example

Pension Plans

Embedding

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

A Systematic Approach

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?

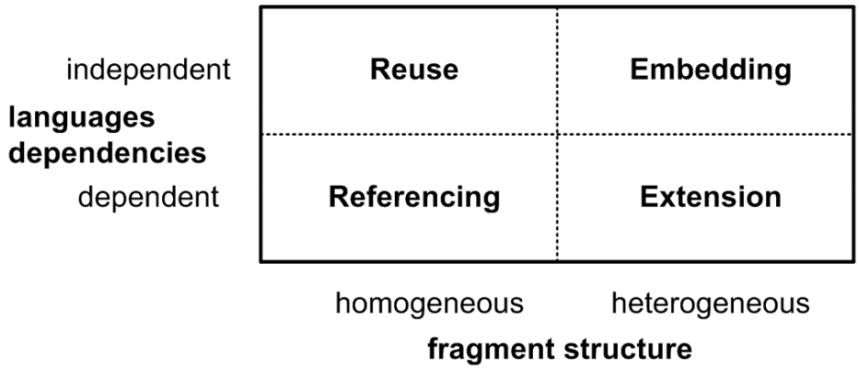
Dependencies:

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

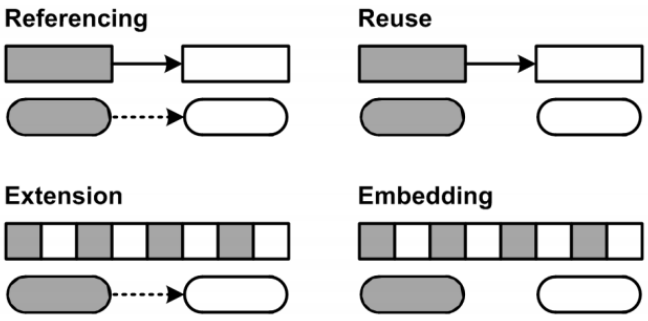
Fragment Structure:

homogeneous vs. heterogeneous („mixing languages")

Dependencies & Fragment Structure:



Dependencies & Fragment Structure:



Concrete Syntax

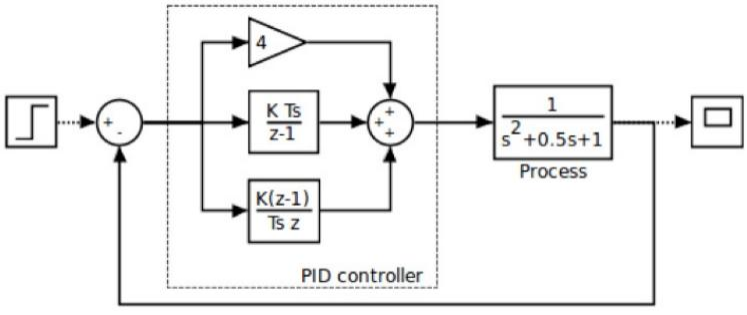
Combinations

```
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
```

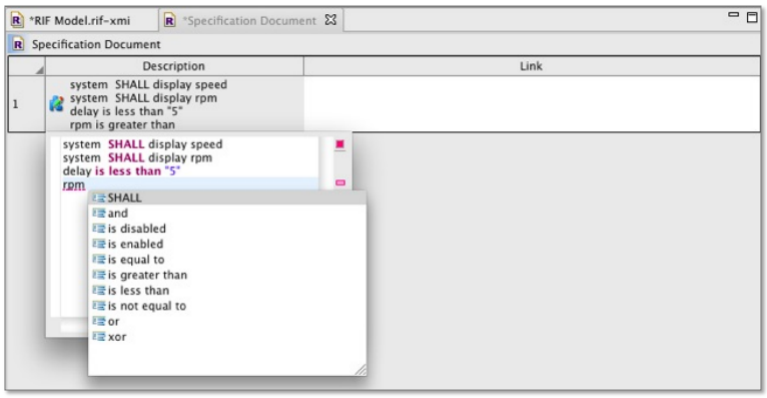
	x == 0	x > 0
y == 0	0	1
y > 0	1	2

```
    ;
    }
}
```

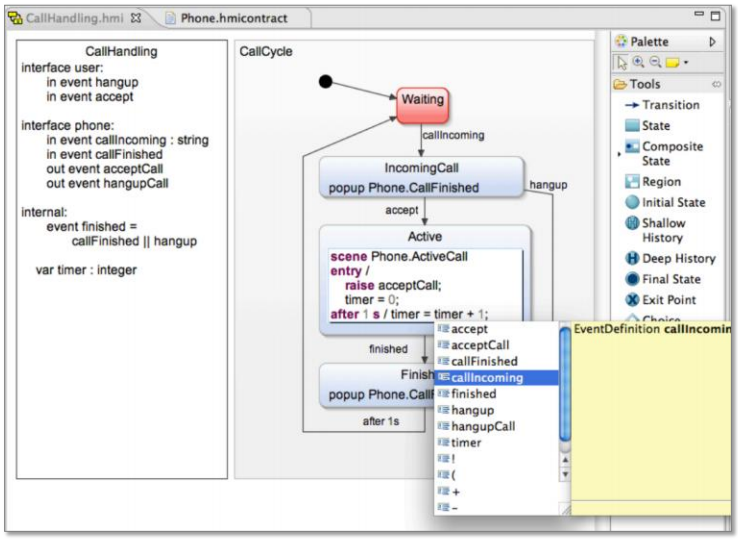
Combinations



Combinations



Combinations



2
Tools

Example Tools

Parser-based LL(k)

eclipse itemis

Example Tools

Projectional

MPS

JetBRAINS

Example Tools

Parser-based GLR

Spoofax

TU Delft

Language Extension

MPS

C extensibility in mbeddr

```
struct Position {
  uint16 x;
  uint16 y;
};

int8 add(int8 x, int8 y) {
  return x + y;
} add (function)

void normalizePosition(Position* p) {
  if (p->x > 100) {
    p->x = 100;
  } if
  if (p->y > 100) {
    p->y = 100;
  } if
} normalizePosition (function)
```

Plain C

Language Extension MPS

C extensibility in mbeddr

```
struct Position {
  uint16 x;
  uint16 y;
};

int8 add(int8 x, int8 y) {
  return x + y;
} add (function)

void normalizePosition(Position* p) {
  if (p->x > 100) {
    p->x = 100;
  } if
  if (p->y > 100) {
    p->y = 100;
  } if
} normalizePosition (function)
```

Plain C

```
exported struct Trackpoint {
  int8 id;
  int8/s/ timestamp;
  int8/m/ x;
  int8/m/ y;
  uint16/m/ alt;
  int16/mps/ speed;
};
```

C + Units

Language Extension MPS

C extensibility in mbeddr

```
requirements modules: FlightJudgementRules
module StateMachines imports DataStructures, stdlib_stub, stdio_stub {

  #define TAKEOFF = 100;
  #define HIGH_SPEED = 10;
  #define VERY_HIGH_SPEED = 20;
  #define LANDING = 100;

  statemachine HierarchicalFlightAnalyzer initial = beforeFlight {
    in next(Trackpoint* tp) <no binding>
    in reset() <no binding>
    out crashNotification() <no binding>
    readable var int16 points = 0
    state beforeFlight {
      on next [tp->alt > 0 m] -> airborne
      exit { points += TAKEOFF; }
    } state beforeFlight
    composite state airborne initial = landed {
      on reset [ ] -> beforeFlight { points = 0; }
      on next [tp->alt == 0 m && tp->speed == 0 mps] -> crashed
      state landed (airborne.landed) {
        entry { points += LANDING; }
      } state landed
    } state airborne
    state crashed {
      entry { send crashNotification(); }
    } state crashed
} }
```

C + State Machines

Language Extension — MPS

C extensibility in mbeddr

```
requirements modules: FlightJudgementRules
module StateMachines imports DataStructures, stdlib_stub, stdio_stub {

  #define TAKEOFF = 100;
  #define HIGH_SPEED = 10;
  #define VERY_HIGH_SPEED = 20;
  #define LANDING = 100;

  statemachine HierarchicalFlightAnalyzer initial = beforeFlight {
    in next(Trackpoint* tp) <no binding>
    in reset() <no binding>
    out crashNotification() <no binding>
    readable var int16 points = 0
    state beforeFlight {
      on next [tp->alt > 0 m] -> airborne
      exit { points += TAKEOFF; }
    } state beforeFlight
    composite state airborne initial = landed {
      on reset [ ] -> beforeFlight { points = 0; }
      on next [tp->alt == 0 m && tp->speed == 0 mps] -> crashed
      state landed (airborne.landed) {
        entry { points += LANDING; }
      } state landed
    } state airborne
    state crashed {
      entry { send crashNotification(); }
    } state crashed
} }
```

C + State Machines

… and Units!

-> combination of independent extensions!

Language Extension — Xtext

„Embedding" Xbase into DSL

```
entity Person {
    lastname : String
    firstname : String
    String fullName(String from) {
        return "Hello " + firstname + " "  + lastname + " from " + from
    }
}
```

Entities using Java types and Xtend code in the body of operations.

Language Extension — Xtext

„Embedding" Xbase into DSL

```
grammar org.xtext.example.lmrc.entityexpr.EntityWithExprDsl
        with org.eclipse.xtext.xbase.Xbase

generate entityWithExprDsl
           "http://www.xtext.org/example/lmrc/entityexpr/EntityWithExprDsl"

Module:
    "module" name=ID "{"
        entities+=Entity*
    "}";
```

```
Attribute:
    name=ID ":" type=JvmTypeReference;

Operation:
    type=JvmTypeReference name=ID "("  (parameters+=FullJvmFormalParameter
        (',' parameters+=FullJvmFormalParameter)*)? ")"
            body=XBlockExpression;
```

Language Extension — Xtext

„Embedding" Xbase into DSL

```
class EntityWithExprDslJvmModelInferrer extends AbstractModelInferrer {

    @Inject extension IQualifiedNameProvider
    @Inject extension JvmTypesBuilder

    def dispatch void infer(Entity entity,
                            IAcceptor<JvmDeclaredType> acceptor,
                            boolean isPrelinkingPhase) {
        ...
    }
}
```

See before … results in generator, scoping, typing, etc. with little additional work.

Language Extension — Xtext

Extending Xbase with Dates

Grammar

```
XDateLiteral:
    'date' ':' year=INT '-' month=INT '-' day=INT;
```

```
XLiteral returns xbase::XExpression:
    XClosure |
    XBooleanLiteral |
    XIntLiteral |
    XNullLiteral |
    XStringLiteral |
    XTypeLiteral |
    XDateLiteral;
```

Repeat complete Xliteral rule,
adding the XDateLiteral

Language Extension — Xtext

Extending Xbase with Dates

Type system

```
@Singleton
public class DomainModelTypeProvider extends XbaseTypeProvider {

    @Override
  protected JvmTypeReference type(XExpression expression,
                  JvmTypeReference rawExpectation, boolean rawType) {
    if (expression instanceof XDateLiteral) {
      return _type((XDateLiteral) expression, rawExpectation, rawType);
    }
    return super.type(expression, rawExpectation, rawType);
  }

  protected JvmTypeReference _type(XDateLiteral literal,
              JvmTypeReference rawExpectation, boolean rawType) {
    return getTypeReferences().getTypeForName(Date.class, literal);
  }
}
```

Language Extension

Xtext

Extending Xbase with Dates

Interpreter

```
public class DateExtensions {
    public static long operator_minus(Date a, Date b) {
        long resInMilliSeconds = a.getTime() - b.getTime();
        return millisecondsToDays( resInMilliSeconds );
    }
}
```

Compiler

```
public class DomainModelCompiler extends XbaseCompiler {
  protected void _toJavaExpression(XDateLiteral expr, IAppendable b) {
    b.append("new java.text.SimpleDateFormat(\"yyyy-MM-dd\").parse(\"" +
             expr.getYear() + "-" + expr.getMonth() + "-" +
             expr.getDay() + "\")");
  }
}
```

Embedding

MPS

Useful for Meta Data

```
requirements modules: FlightJudgementRules
module StateMachines imports DataStructures, stdlib_stub, stdio_stub {

  [#define TAKEOFF = 100;] -> implements PointsForTakeoff
  [#define HIGH_SPEED = 10;] -> implements FasterThan100
  [#define VERY_HIGH_SPEED = 20;] -> implements FasterThan200
  [#define LANDING = 100;] -> implements FullStop

  statemachine FlightAnalyzer initial = beforeFlight {
    in next(Trackpoint* tp) <no binding>
    in reset() <no binding>
    out crashNotification() ⊠ raiseAlarm
    readable var int16 points = 0
    state beforeFlight {
      entry { points = 0; } entry
      on next [tp->alt > 0 m] ⊠ airborne
      [exit { points += TAKEOFF; } exit] -> implements PointsForTakeoff
    } state beforeFlight
    state airborne {
      on next [tp->alt == 0 m && tp->speed == 0 mps] ⊠ crashed
      on next [tp->alt == 0 m && tp->speed > 0 mps] ⊠ landing
      [on next [tp->speed > 200 mps] ⊠ airborne { points += VERY_HIGH_SPEED; } on next] -> implements FasterThan200
      [on next [tp->speed > 100 mps] ⊠ airborne { points += HIGH_SPEED; } on next] -> implements FasterThan100
      on reset [ ] ⊠ beforeFlight
    } state airborne
    state landing {
      on next [tp->speed == 0 mps] ⊠ landed
      [on next [ ] ⊠ landing { points--; } on next] -> implements ShortLandingRoll
      on reset [ ] ⊠ beforeFlight
    } state landing
```

No Dependencies on anything.

Extending the Tool

MPS

… since it is bootstrapped!

```
public boolean isLinear(node<Expression> expr) {
  return dispatch<boolean> (expr) {
    NumberLiteral        ⇒ true
    EnumLiteralRef       ⇒ true
    LiteralWithUnit      ⇒ #(it.value)
    BinaryComparisonExpr ⇒ #(it.left) && #(it.right)
    BinaryLogicalExpress ⇒ #(it.left) && #(it.right)
    PlusExpression       ⇒ #(it.left) && #(it.right)
    LocalVarRef          ⇒ true
    ArgumentRef          ⇒ true
    MemberRef            ⇒ true
    SUArrowExpression    ⇒ #(it.member)
    SUDotExpression      ⇒ #(it.member)
    NotExpression        ⇒ #(it.expression)
    GlobalVarRef         ⇒ true
    MultiExpression      ⇒ it.left.isStaticallyEvaluatable() || it.right.isStaticallyEvaluatable()
    DivExpression        ⇒ it.left.isStaticallyEvaluatable() || it.right.isStaticallyEvaluatable()
    default false
  };
}
```

Extending the Tool

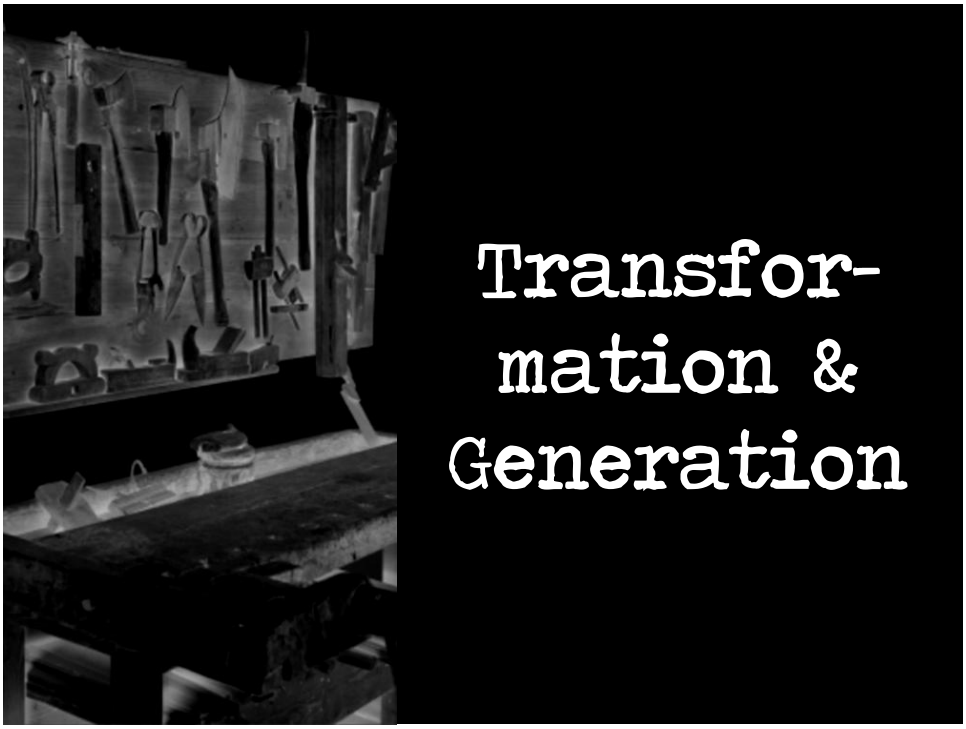
MPS

… since it is bootstrapped!

```
test TestGuard2 {
  node<PlusExpression> plus = <4 + 2 / 3>;
  node<Expression> cand = plus.rightExpression : DivExpression.rightExpression;
  boolean matched = false;
  when matched cand against [t /IntegerConstant #t.value == 3;
                             p /DivExpression;
                             a /PlusExpression #a.leftExpression.isInstanceOf(IntegerConstant) ==plus;] {
    matched = true;
  }
  assert true matched;
}
```

```
node<GlobalVariableDeclaration> gvd = build GlobalVariableDeclaration
  name = name
  type = #(type)
  preventNameMangling = true
  extern = isExtern
;
```

Extending MPS is (almost) like extending any other language!



Multi-Stage Trafo

MPS

MPS' Transformations

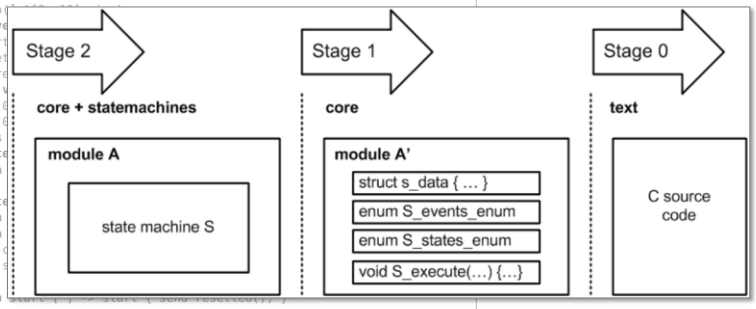
```
module Statemachine imports nothing {

  statemachine Counter {
    in events
      start()
      step(int[0..10] size)
    out events
      started()
      resetted()
      incremented(int[0..10] newVal)
    local variables
      int[0..10] currentVal = 0
      int[0..10] LIMIT = 10
    states ( initial = start )
      state start {
        on start [ ] -> countState { send started(); }
      }
      state countState {
        on step [currentVal + size > LIMIT] -> start { send resetted(); }
        on step [currentVal + size <= LIMIT] -> countState {
          currentVal = currentVal + size;
          send incremented(currentVal);
        }
        on start [ ] -> start { send resetted(); }
      }
  }

  Counter c1;
  Counter c2;

  void aFunction() {
    trigger(c1, start);
  }
}
```

Multi-Stage Trafo

MPS

MPS' Transformations

```
module Statemachine imports nothing {

  statemachine Counter {
    in events
      start()
      step(
    out eve
      start
      reset
      incr
    local
      int[
      int[
    states
      state
        on
      }
      state
        on
        on
      }
      on
    }
  }

  Counter c1;
  Counter c2;

  void aFunction() {
    trigger(c1, start);
  }
}
```



Transformations

MPS

with concrete Syntax & IDE

```
template weave_StatemachineTypesStuffIntoModule
input    Statemachine
content node:
module dummy imports nothing {
  <TF [ exported enum $[statemachineInEvents] { $LOOP$[$[anEvent]; ] } ] TF>
  <TF [ exported enum $[statemachineStates] { $LOOP$[$[aState]; ] } ] TF>
  <TF [ exported struct $[statemachineData] {                       ] TF>
        [   ->$[statemachineStates] __currentState;                 ]
        [   $LOOP$[$COPY_SRC$[int8_t] $[smLocalVar]; ]              ]
        [ };                                                        ]
}
```

IDE support for target language
Macros contain template code
M2M with M2T „feel"

Transformations MPS

with concrete Syntax & IDE

```
template weave_StatemachineTypesStuffIntoModule
input    Statemachine
content node:
module dummy imports nothing {
  <TF [ exported enum $[statemachineInEvents] { $LOOP$[$[anEvent]; ] } ] TF>
  <TF [ exported enum $[statemachineStates] { $LOOP$[$[aState]; ] } ] TF>
  <TF [ exported struct $[statemachineData] {                       ] TF>
        [   ->$[statemachineStates] __currentState;                 ]
        [  $LOOP$[$COPY_SRC$[int8_t] $[smLocalVar]; ]               ]
      [ };                                                          ]
}
```

```
comment       : <none>
mapping label : <no label>
mapped nodes  : (node, genContext, operationContext)->sequence<node<>> {
                  node.states;
                }
```

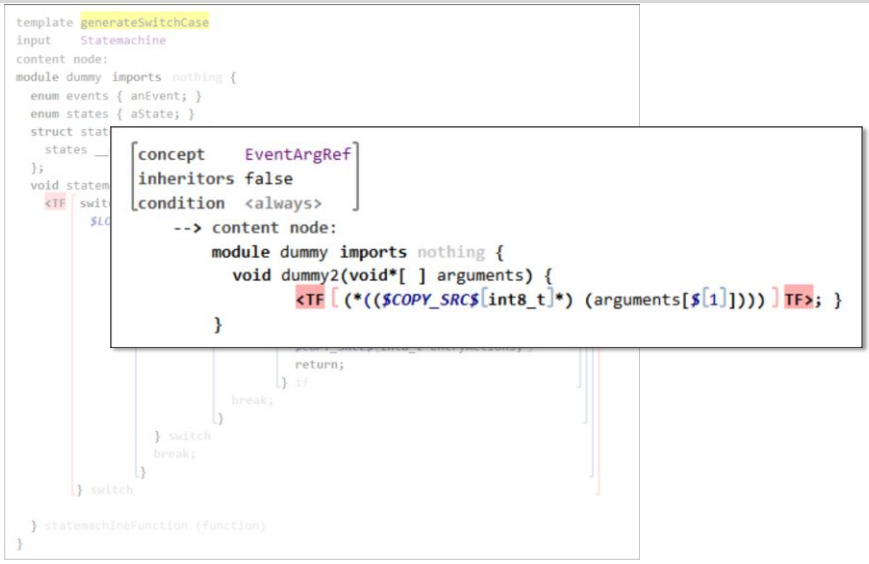
Transformations MPS

with concrete Syntax & IDE

```
template generateSwitchCase
input    Statemachine
content node:
module dummy imports nothing {
  enum events { anEvent; }
  enum states { aState; }
  struct statemachineData {
    states __currentState;
  };
  void statemachineFunction(statemachineData* instance, events event) {
    <TF [ switch ( instance->__currentState ) {                                  ] TF>
        [  $LOOP$[case ->$[aState]: {                                            ]
        [          switch ( event ) {                                            ]
        [            $LOOP$[case ->$[anEvent]: {                                 ]
        [                     $LOOP$[if ( $COPY_SRC$[true] ) {                   ]
        [                              $COPY_SRC$[int8_t exitActions;]           ]
        [                              $COPY_SRC$[int8_t transActions;]          ]
        [                              instance->__currentState = ->$[aState];   ]
        [                              $COPY_SRC$[int8_t entryActions;]          ]
        [                              return;                                   ]
        [                     } if                                               ]
        [                   break;                                               ]
        [            }                                                           ]
        [          } switch                                                      ]
        [          break;                                                        ]
        [  }                                                                     ]
      [ } switch                                                                 ]

  } statemachineFunction (function)
}
```

Transformations MPS

with concrete Syntax & IDE



Transformations MPS

With local dependencies

A transformation only specifies its relative priority to other transformations as necessary.

This leads to composability also of transformations.

IDE
Services

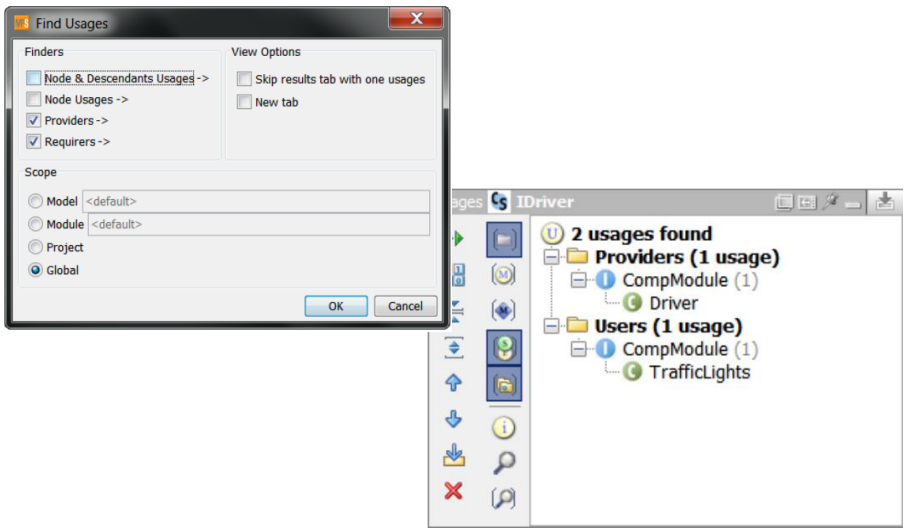
Find References — MPS

Custom Finders

```
finder findProviders for concept Interface
  description: Providers

  find(node, scope)->void {
    nlist<> refs = execute NodeUsages ( node , <same scope> );
    foreach r in refs.select(it|it.isInstanceOf(ProvidedPort)) {
      add result r.parent ;
    }
  }

  getCategory(node)->string {
    "Providers";
  }
```

Find References — MPS

Custom Finders



Testing

Parser/Syntax Testing — Spoofax

Declarative Approach

abstract syntax

```
test multiply and add [[1 + 2 * 3]] parse to Add(_, Mul(_, _))
test add and multiply [[1 * 2 + 3]] parse to Add(Mul(_, _), _)
test add and add [[1 + 2 + 3]] parse to Add(Add(_, _), _)
```

concrete syntax

```
test multiply and add [[1 + 2 * 3]] parse to [[1 + (2 * 3)]]
test add and multiply [[1 * 2 + 3]] parse to [[(1 * 2) + 3]]
test add and add [[1 + 2 + 3]] parse to [[(1 + 2) + 3]]
```

Constraints/Type Sys. — MPS

Declarative Approach

```
Test case testSubtyping
nodes
  ( [ <check types module TestSubtyping imports nothing { >
                  <dnode  double d = 10;>
                  double d2 = <dref  d>;
                  double d3 = <node d has type double>;
                  int8 i = <node d has error>;
                }                                          ] )

test methods
  test testReference {
    assert dref.var == dnode;
  }
```

Test annotations on the test subject - very concise.

Testing Semantics

MPS

Expressing tests in language

```
module UnitTestDemo {

  int32 main(int32 argc, int8*[ ] argv) {
    return test testMultiply;
  }

  test case testMultiply {
    assert (0) times2(21) == 42;
    assert (1) times2(0) == 0;
    assert (2) times2(-10) == -20;
  }

  int8 times2(int8 a) {
    return 2 * a;
  }
}
```

Generate/Interpret the tests along with the application code

Testing Editors

Xtext

Using Moritz Eysholdt's Xpect

```
// XPECT_TEST org.example.MyJUnitContentAssistTest END_TEST

// XPECT contentAssist at |Hel --> Hello
Hello Peter!

// XPECT contentAssist at |! --> !
Hello Heiko!
```

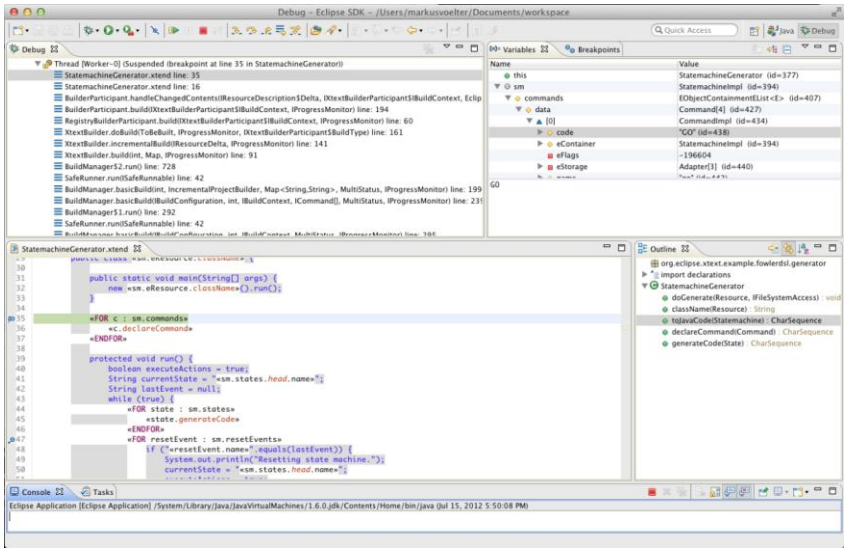
Test expectations in comments.

Can test various language aspects such as scopes, code completion, type system; also extensible.

Inspecting/ Debugging

Debugging DSLs — Xtext

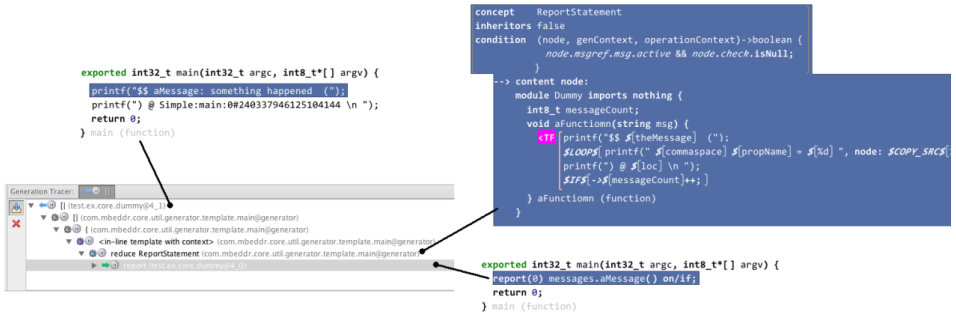
Transformations & Code Gens



Debugging DSLs — MPS

Transformations

Multi-Level, Declarative:
show intermediate models and the templates involved in trafo.



Debugging DSL Programs Xtext

Xbase Automatic Debugging

Make your DSL extend Xbase
Use a JVMModelInferrer → Java
you then get Debugging for free

Debugging DSL Programs MPS

Extensible Debuggers

```
int8 s = 0;
int8[] a = {1, 2, 3};
foreach (a sized 3) {
    s += it;
}
```

Debugging DSL Programs MPS

Extensible Debuggers

```
int8 s = 0;
int8[] a = {1, 2, 3};
foreach (a sized 3) {
    s += it;
}
```

Language Extension

```
int8 s = 0;
int8[] a = {1, 2, 3};
for (int __c = 0; __c < 3; __c++) {
    int8 __it = a[__c];
    s += __it;
}
```

Generated C Code

Debugging DSL Programs MPS

Extensible Debuggers

```
int8 s = 0;
int8[] a = {1, 2, 3};
foreach (a sized 3) {
    s += it;
}
```

Stepping

```
void contributeStepOverStrategies(list<IDebugStrategy> res) {
   ancestor
   statement list: this.body
}
```

```
void contributeStepIntoStrategies(list<IDebugStrategy> res) {
  subtree: this.array
  subtree: this.len
}
```

Debugging DSL Programs MPS

Extensible Debuggers

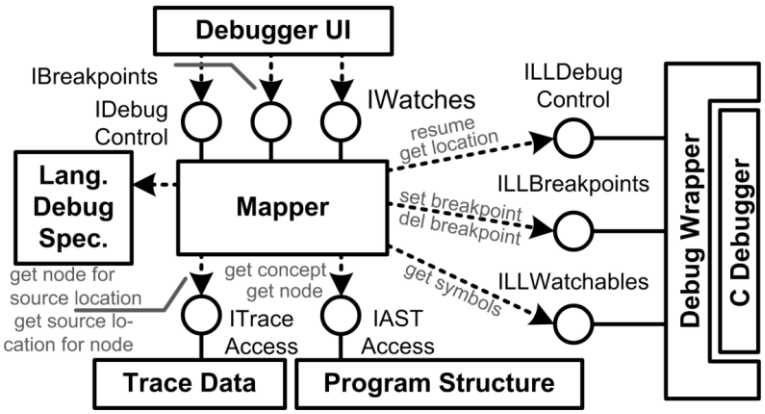
```
int8 s = 0;
int8[] a = {1, 2, 3};
foreach (a sized 3) {
    s += it;
}
```

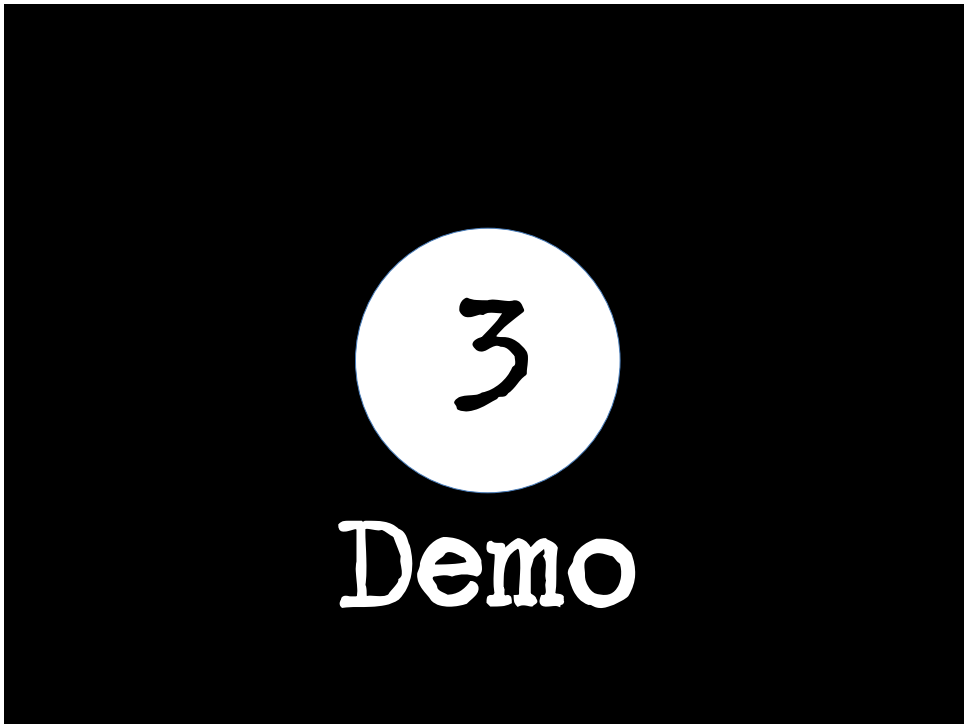
Watches

```
void contributeWatchables(list<UnmappedVariable> unmapped,
                          list<IWatchable> mapped) {
  hide "__c"
  map "__it" to "it"
    type: this.array.type : ArrayType.baseType
    category: WatchableCategories.LOCAL_VARIABLES
    context: this
}
```

Debugging DSL Programs MPS

Debugger Architecture




3
Demo

mbeddr
C the Difference. C the Future.

An extensible Version of C (and an IDE and a Debugger)

Bundesministerium für Bildung und Forschung

gefördert durch das BMBF
Förderkennzeichen 01IS11014

The End.

voelter.de | dslbook.org

@markusvoelter
+Markus Voelter