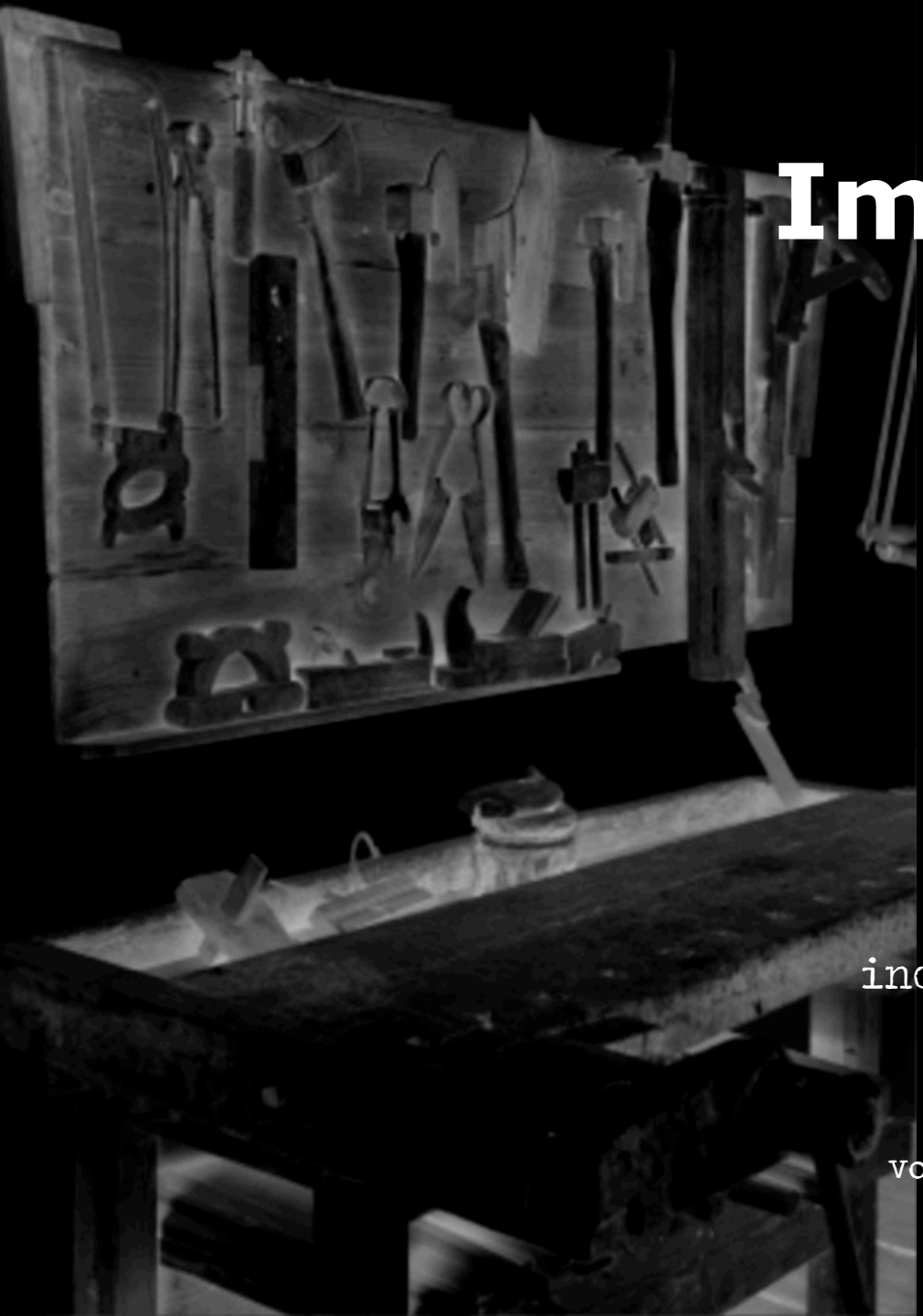


DSL Implementation

… with language Workbenches.

v1.1 - Jan 16, 2013

Markus Voelter
independent/itemis
voelter@acm.org

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

Last Year's Talk	This Year's Talk
Design No Tools The „why"	Implementation 3 Tools The „how"

Last Year's Talk

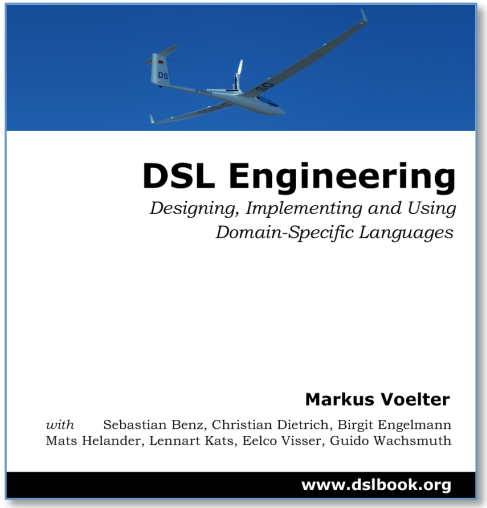
This Year's Talk

Design
No Tools
The „why"

Implementation
3 Tools
The „how"

Both taken from my new book:
http://dslbook.org

available Feb 2013



Intro-duction

Language Workbench

(Martin Fowler)

Language Workbench

(Martin Fowler)

Freely
define
languages and
integrate
them

Language Workbench

(Martin Fowler)

use

persistent

abstract

representation

?

Language Workbench

(Martin Fowler)

language ::=

schema

+ editors

+ generators

Language Workbench

(Martin Fowler)

projectional editing?

Language Workbench

(Martin Fowler)

persist

incomplete

or

contradictory

information

Language Workbench

(Martin Fowler)

powerful

editing

testing

refactoring

debugging

groupware

+

language definition

implies

IDE definition

Language Workbench

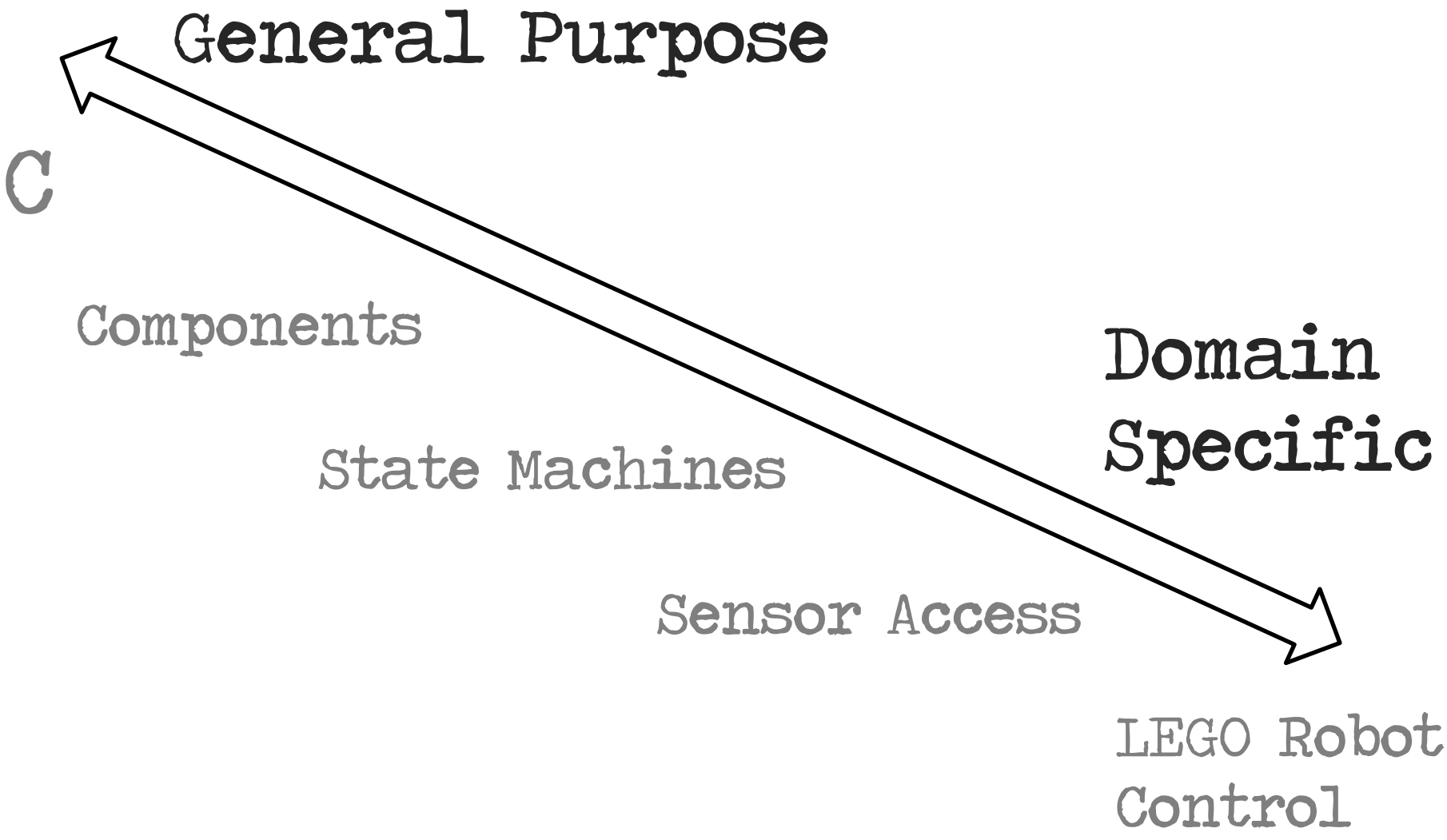
(Martin Fowler)

support for „classical“ programming

+

„classical“ and modeling

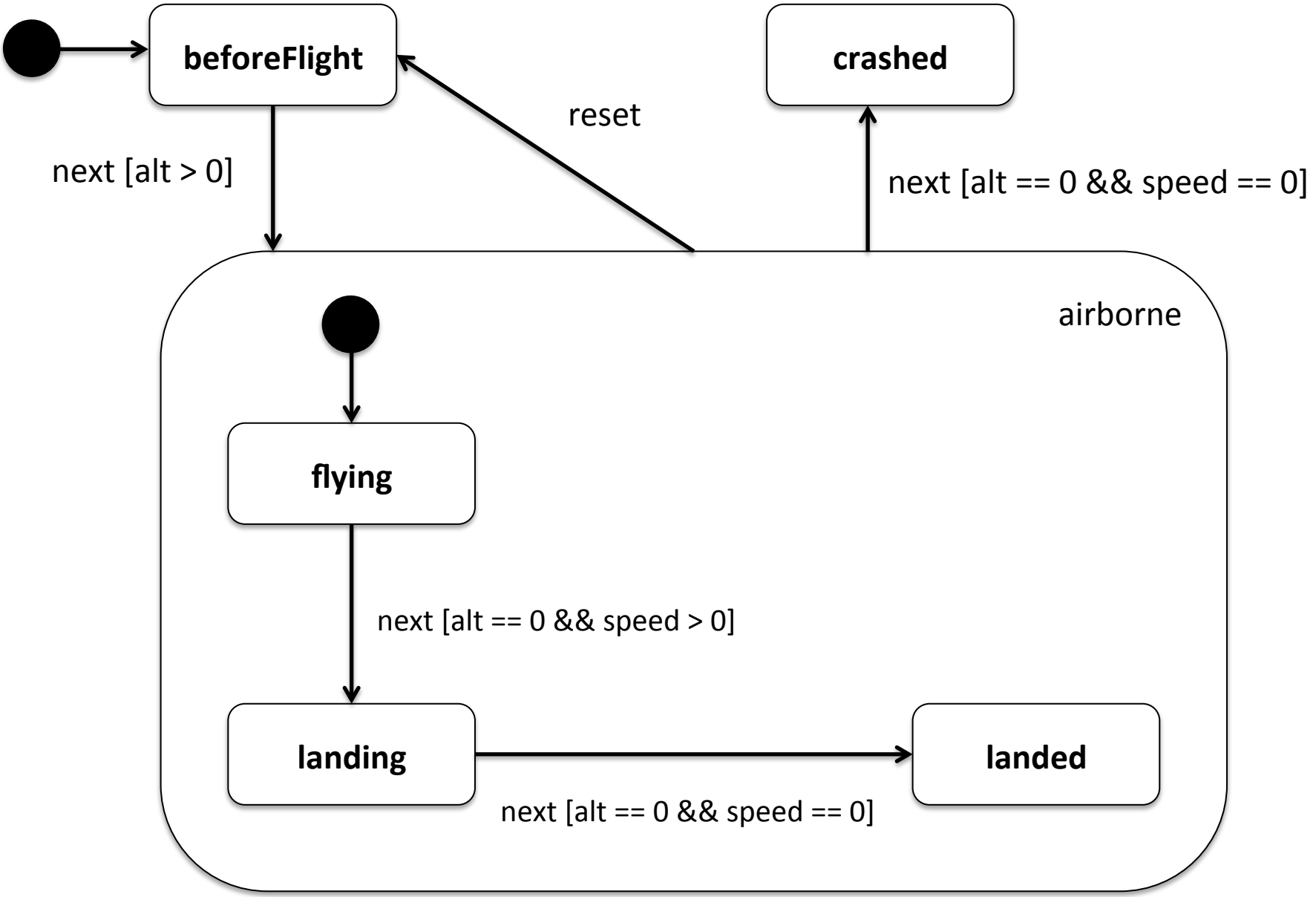
	more in GPLs	more in DSL
Domain Size	large and complex	smaller and well-defined
Designed by	guru or committee	a few engineers and domain experts
Language Size	large	small
Turing-completeness	almost always	often not
User Community	large, anonymous and widespread	small, accessible and local
In-language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


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

Model or Code?

```
Trackpoint* makeTP(uint16 alt, int16 speed) {
  static int8 trackpointCounter = 0;
  trackpointCounter++;
  Trackpoint* tp = ((Trackpoint*) malloc(sizeof Trackpoint));
  tp->id = trackpointCounter;
  tp->timestamp = trackpointCounter;
  tp->alt = alt
  tp->speed = speed
  return tp;
}
```

Model or Code?



Model or Code?

```
statemachine HierarchicalFlightAnalyzer initial = beforeFlight {
  in next()
  in reset()
  out crashNotification() -> raiseAlarm
  state beforeFlight {
    on next [tp->alt > 0 m] -> airborne
  }
  composite state airborne initial = flying {
    on reset [ ] -> beforeFlight
    on next [tp->alt == 0 m && tp->speed == 0 mps] -> crashed
    state flying {
      on next [tp->alt == 0 m && tp->speed > 0 mps] -> landing
      on next [tp->speed > 200 mps] -> airborne
      on next [tp->speed > 100 mps] -> airborne
    }
    state landing {
      on next [tp->speed == 0 mps] -> landed
      on next [ ] -> landing
    }
    state landed {
    }
  }
  state crashed {
  }
}
```

Model or Code?

```
statemachine HierarchicalFlightAnalyzer initial = beforeFlight {
  in next(Trackpoint* tp)
  in reset()
  out crashNotification() -> raiseAlarm
  readable var int16 points = 0
  state beforeFlight {
    on next [tp->alt > 0 m] -> airborne
    exit { points += TAKEOFF; }
  }
  composite state airborne initial = flying {
    on reset [ ] -> beforeFlight { points = 0; }
    on next [tp->alt == 0 m && tp->speed == 0 mps] -> crashed
    state flying {
      on next [tp->alt == 0 m && tp->speed > 0 mps] -> landing
      on next [tp->speed > 200 mps] -> airborne { points += VERY_HIGH_SPEED; }
      on next [tp->speed > 100 mps] -> airborne { points += HIGH_SPEED; }
    }
    state landing {
      on next [tp->speed == 0 mps] -> landed
      on next [ ] -> landing { points--; }
    }
    state landed {
      entry { points += LANDING; }
    }
  }
  state crashed {
    entry { send crashNotification(); }
  }
}
```

Model or Code?

Does it really matter?

What is the difference?

Who cares?

We don‘t want to
model,
we want to
program!

We don‘t want to model, we want to program!

… at different levels of abstraction

… from different viewpoints

… integrated!

We don‘t want to model, we want to program!

… with different degrees of domain-specificity

… with suitable notations

… with suitable expressiveness

We don‘t want to

model,

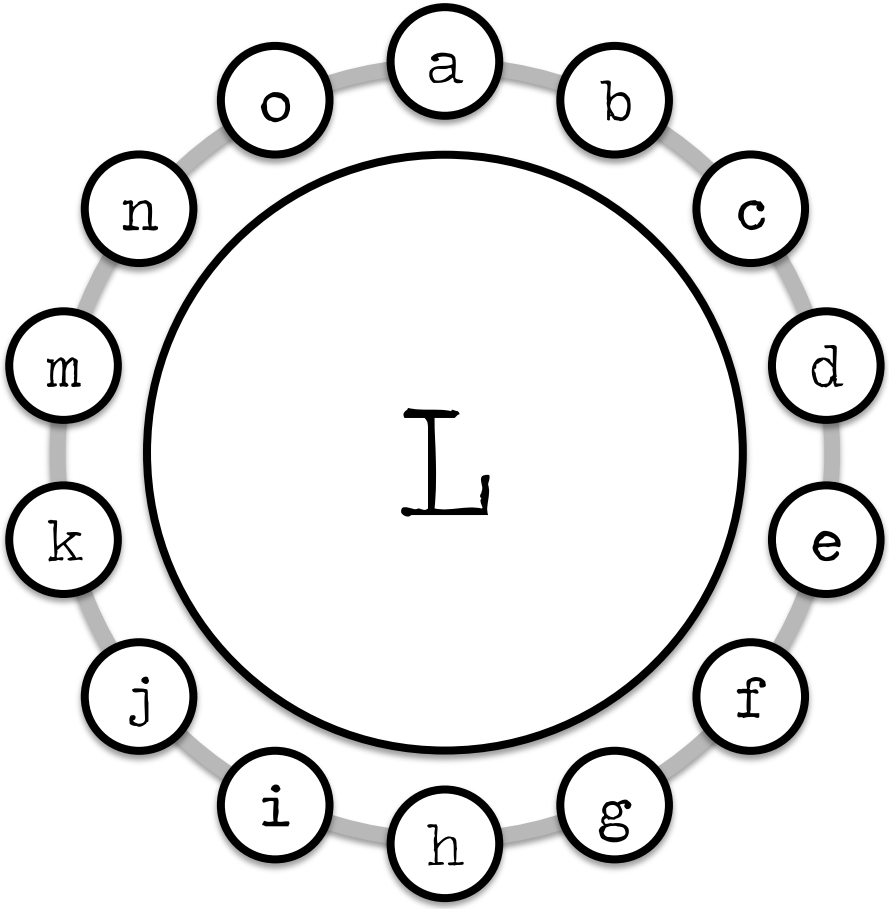
we want to

program!

and always:

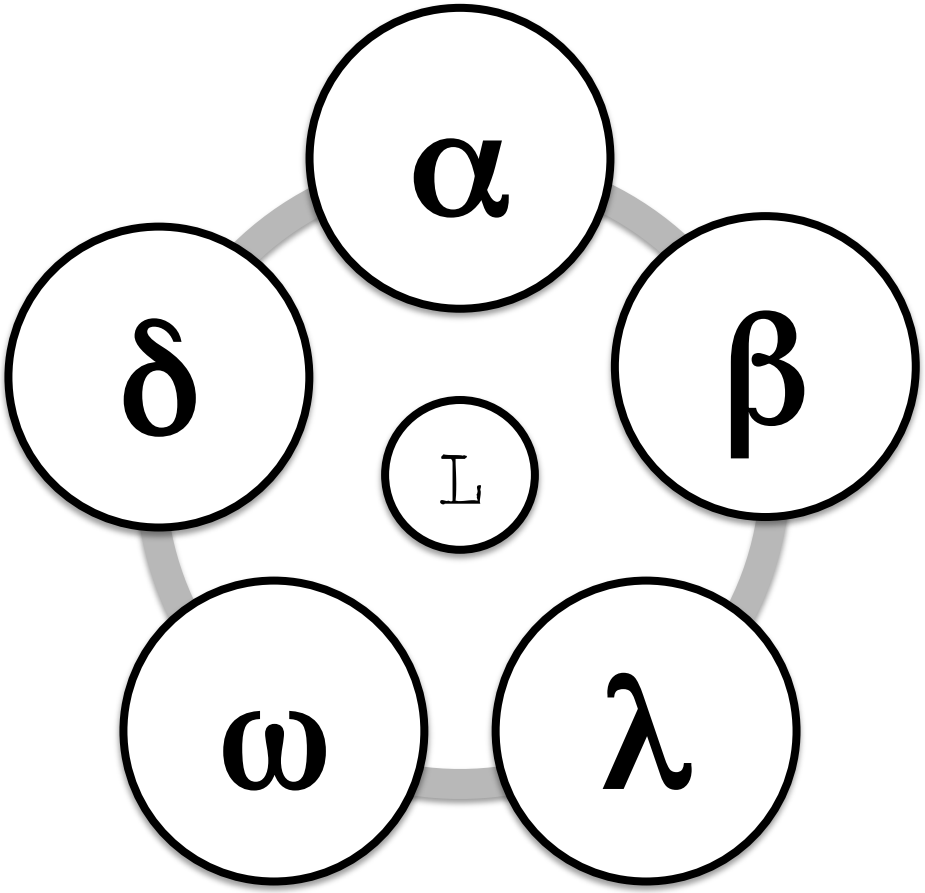
precise and tool processable

Big Language



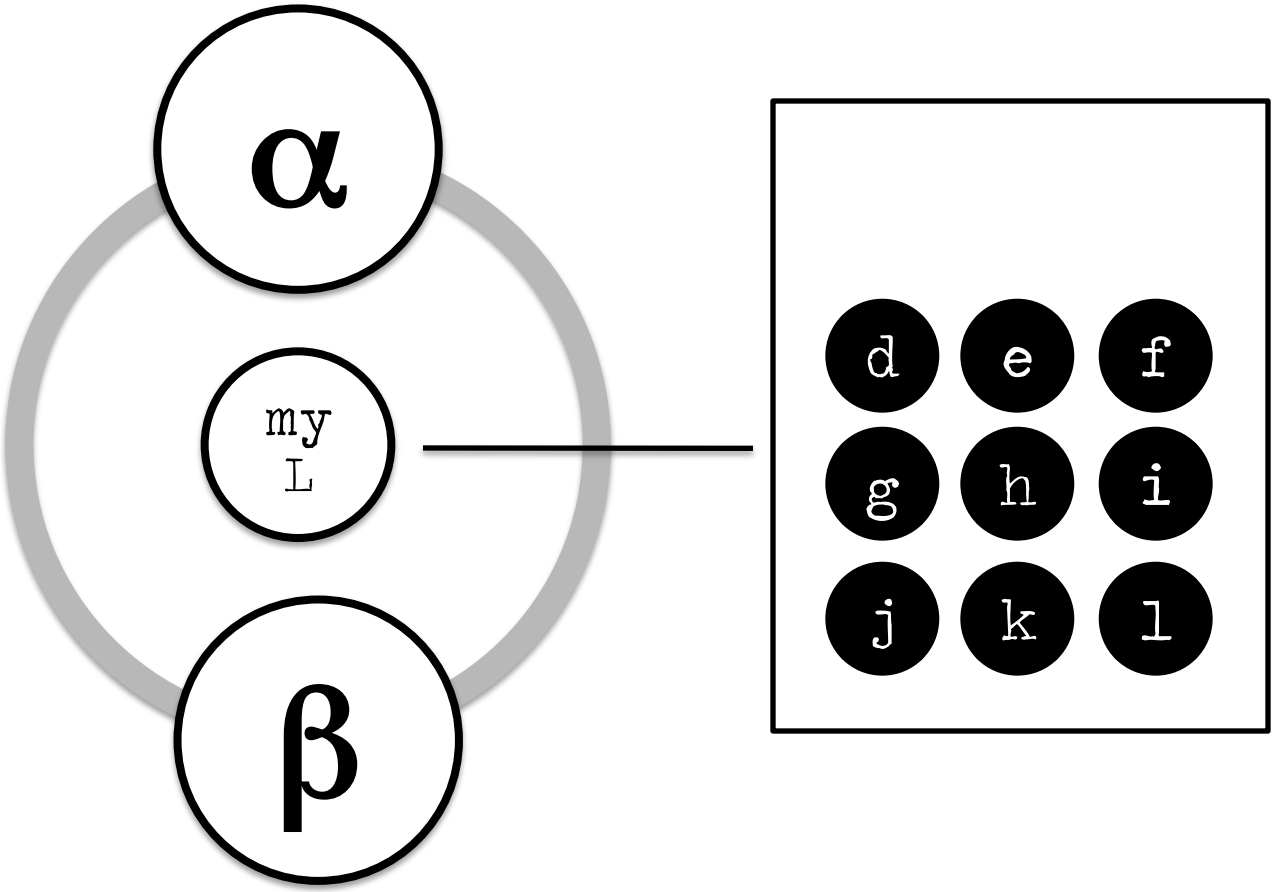
with **many** first class concepts!

Small Language



with a few, orthogonal
and poweful concepts

Modular Language

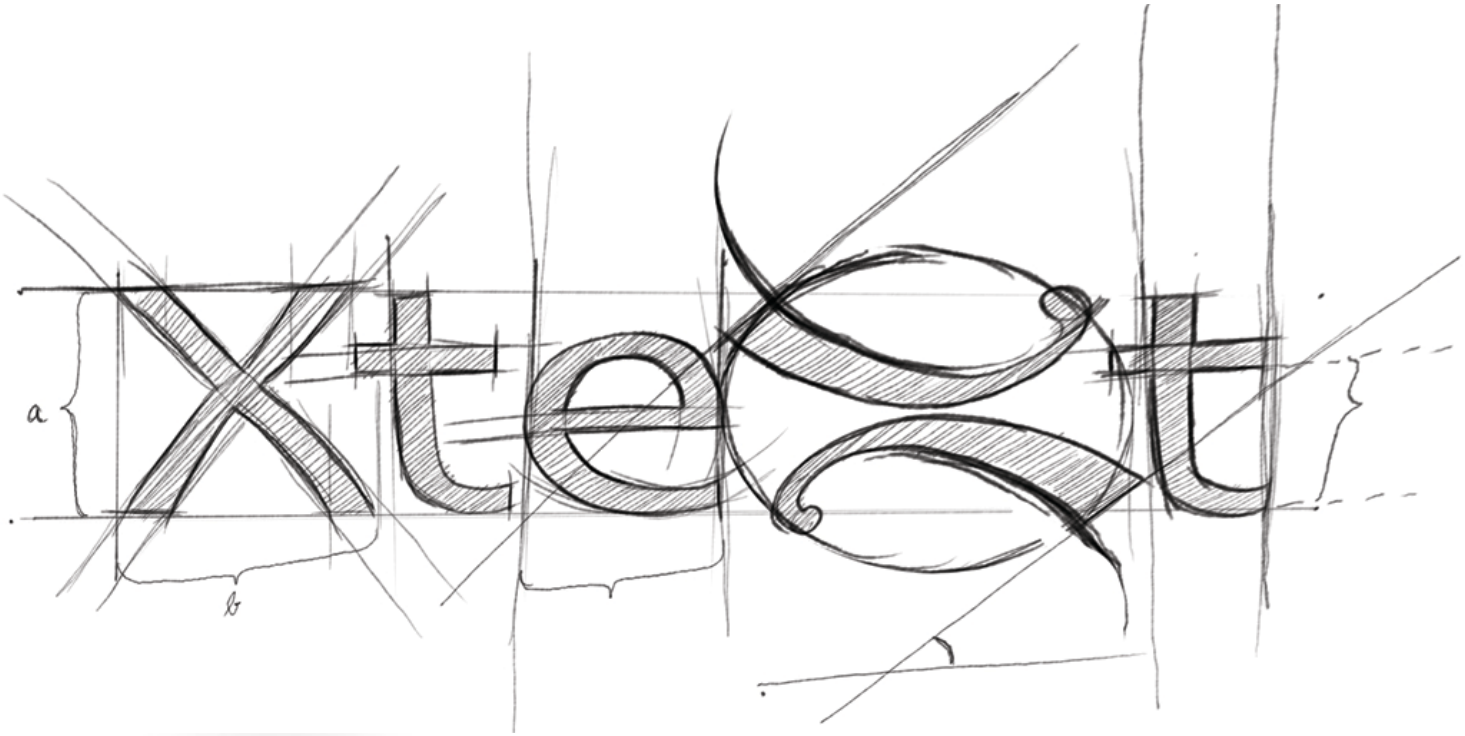


with many optional,
composable modules

Example Tools

Example Tools

Parser-based LL(k)



eclipse

itemis

Example Tools

Projectional

MPS

JetBRAINS

Example Tools

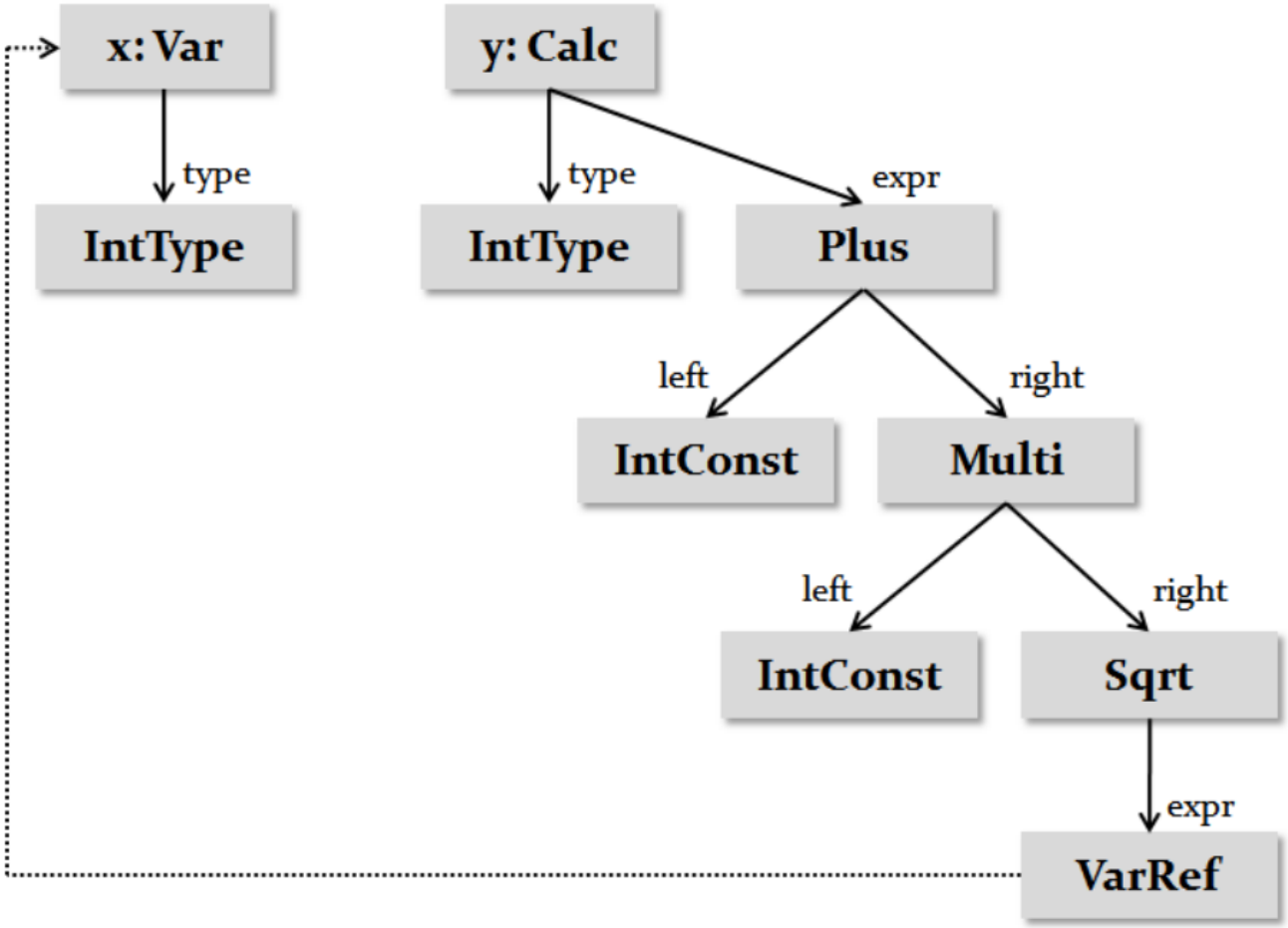
Parser-based GLR

Spoofax

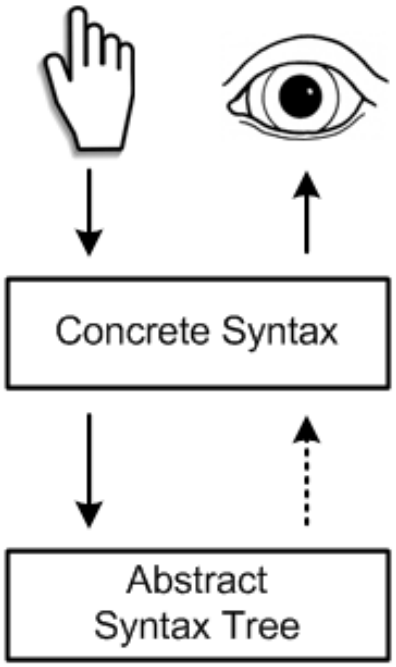
Syntax

Concrete and Abstract

```
var x: int;
calc y: int = 1 + 2 * sqrt(x)
```

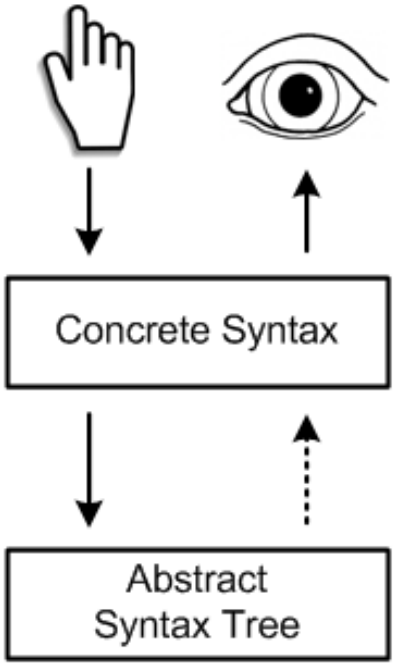


Parsing

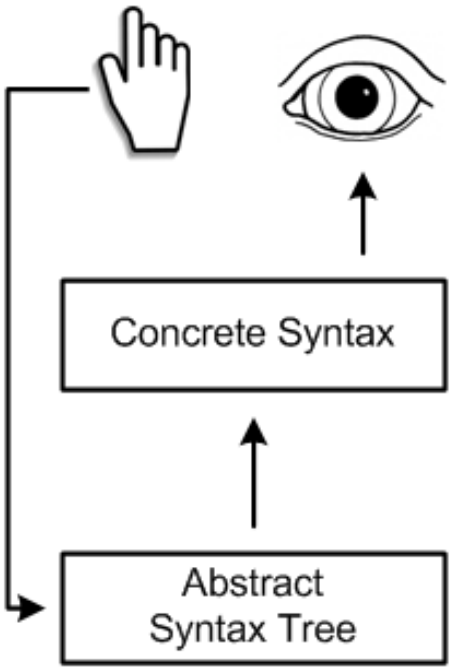


Parsing

Parsing vs. Projection



Parsing



Projection

Parsing

BNF, EBNF Grammars

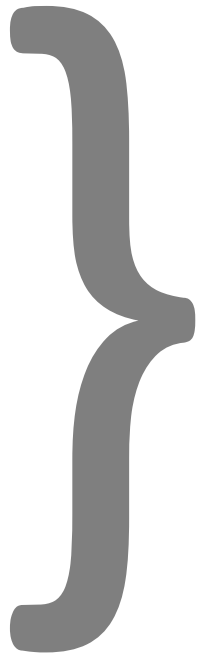
$$S ::= P_1 \ldots P_n$$

```
Exp ::= NUM
      | Exp "+" Exp
      | Exp "*" Exp
```

Parsing

Grammar Classes

LL
LL(k)
LL(*)
LR
…



Limitations wrt. all Context Free Grammars

Parsing

Grammar Class Limitations

```
Exp ::= ID
      | STRING
      | Exp "." ID
```

```
"Tim".length
```

Left Recursion!

Not supported by LL Parsers

Parsing

Grammar Class Limitations

```
Exp ::= ID
      | STRING
      | Exp "." ID
```

```
"Tim".length
```

Left Recursion!

Not supported by LL Parsers -> Left Factoring

```
Exp ::= ID
      | STRING
      | Exp ("." ID)+
```

```
Exp ::= ID
      | STRING
      | FieldPart ("." ID)+

FieldPart ::= ID
            | STRING
```

Parsing

Grammar Class Limitations

```
Exp ::= ID
      | STRING
      | FieldPart ("." ID)+

FieldPart ::= ID
            | STRING
```

```
Exp       ::= FieldPart ("." ID)*

FieldPart ::= ID
            | STRING
```

Alternative: → LL(1)

GLR or Earley parsers

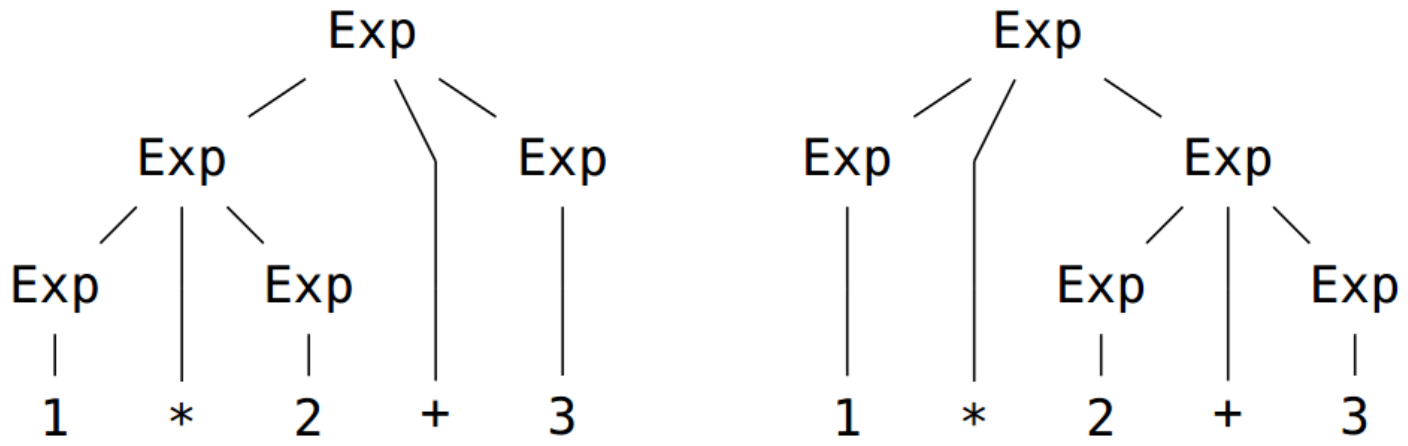
→ parses all CFGs

→ O(n), $O(n^3)$ worst case

Parsing

Ambiguities & Grammar Classes

```
Exp ::= NUM
      | Exp "+" Exp
      | Exp "*" Exp
```



Parsing

Ambiguities & Grammar Classes

```
Exp ::= NUM
      | Exp "+" Exp
      | Exp "*" Exp
```

```
Expr ::= Expr "+" Mult
       | Mult
Mult ::= Mult "*" NUM
       | NUM
```

→ LR

```
Expr ::= Mult ("+" Mult)*
Mult ::= NUM ("*" NUM)*
```

→ LL

Problem: Precedence encoded in grammar structure!

Ambiguities & GLR Parsers

```
Exp ::= NUM
      | Exp "+" Exp
      | Exp "*" Exp
```

```
Exp ::= NUM
      | Exp "+" Exp {left}
      | Exp "*" Exp {left}
```

```
Exp ::= Exp "*" Exp {left}
>
Exp ::= Exp "+" Exp {left}
```

Better: Declarative, Extensible

Projection

MPS

Very Flexible Syntax

```
exported component Judge extends nothing {
  provides FlightJudger judger
  int16 points = 0;
  void judger_reset() ← op judger.reset {
    points = 0;
  } runnable judger_reset
  void judger_addTrackpoint(Trackpoint* tp) ← op judger.addTrackpoint {
    points += 0
  } runnable judger_addTrackpoint
  int16 judger_getResult() ← op judger.getResult {
    return points;
  } runnable judger_getResult
}
```

	tp->alt <= 2000 m	tp->alt >= 2000 m
tp->speed < 150 mps	0	10
tp->speed >= 150 mps	5	20

Projection

MPS

Very Flexible Syntax

```
void vectorDemo() {
  vector<int16, 3> aVector = [1; 2; 3] * 512;

  vector<int16, 3> resultOfCrossProduct = aVector x aVector;

  matrix<int16, 2x3> aMatrix = [1 + 2   2 * 7   42;
                                3       51      24];

  matrix<int16, 3x2> transposedMatrix = aMatrix';
} vectorDemo (function)
```

Projection

MPS

Very Flexible Syntax

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

Challenges: Editor Usabililty, Infrastructure Integration

Projection

MPS

Editor Usability

Traditionally a challenge.
Mostly solved in MPS.

Projection

MPS

Editor Usability

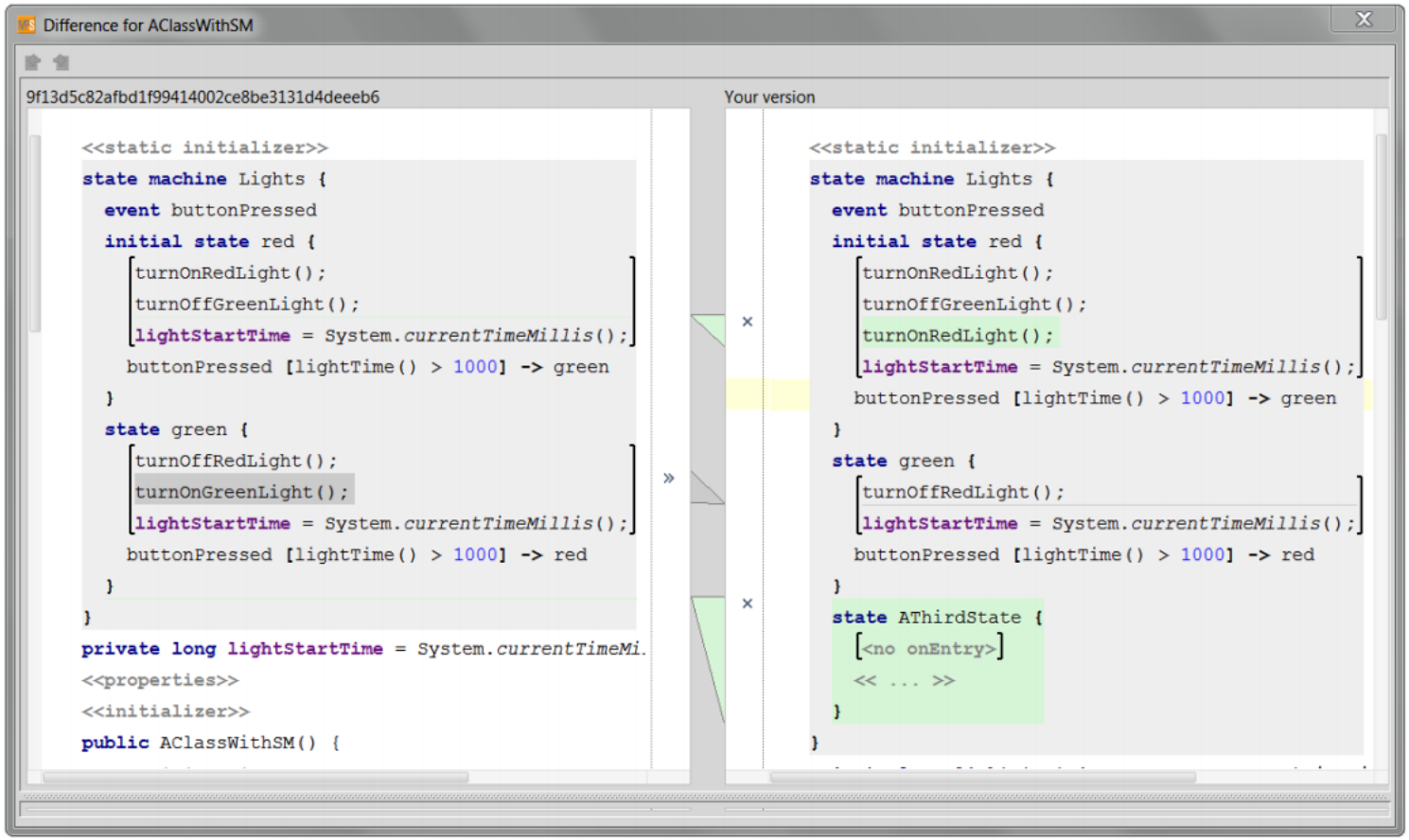
Traditionally a challenge.
Mostly solved in MPS.

Aliases
Side Transformations
Smart References
Smart Delimiters
Delete Actions
Wrappers
Create Ref Target

Projection

MPS

Infrastructure Integration



AST Formalisms

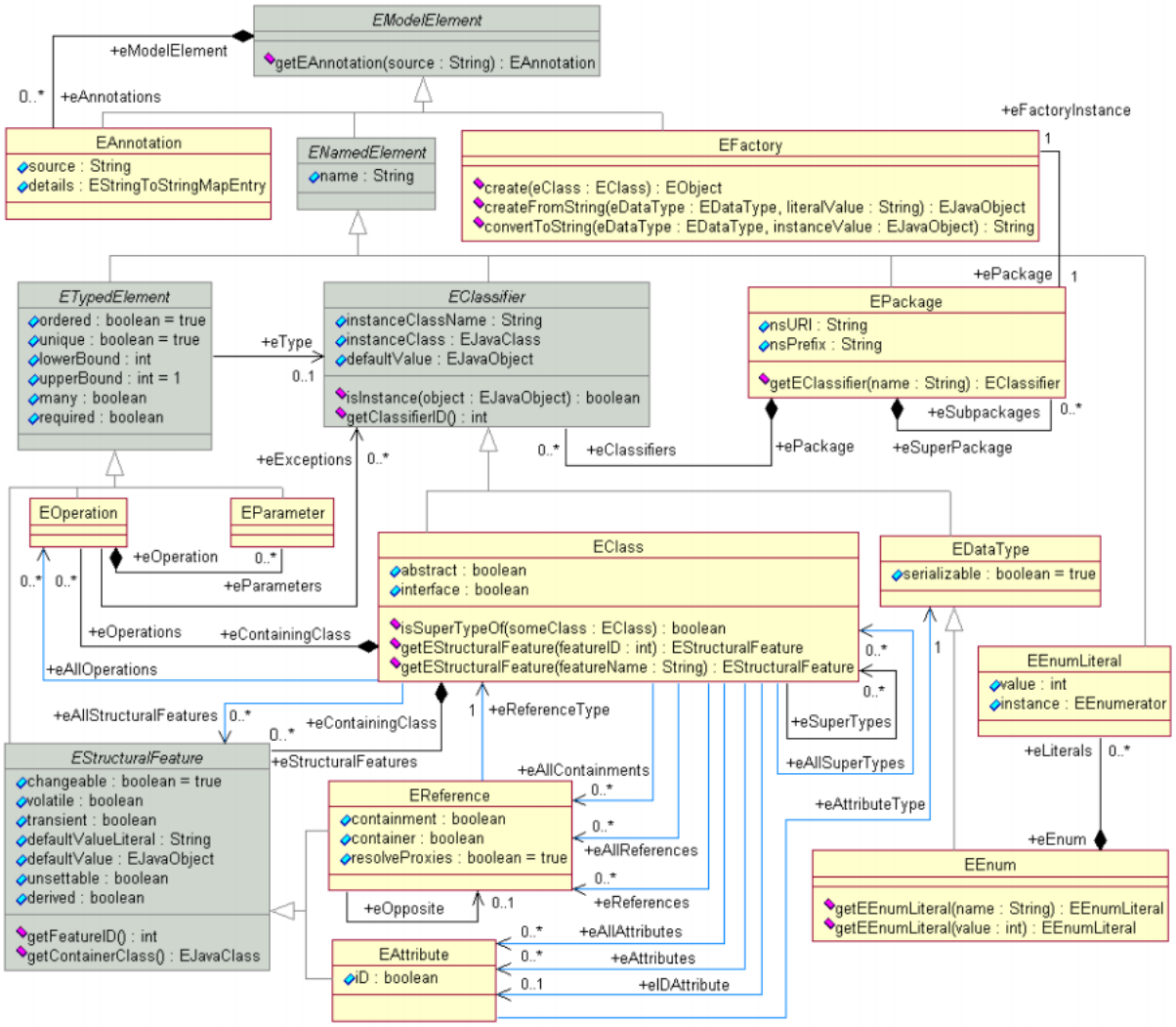
The Essence

Things
- with properties
- child Things
- references to other Things
- extending other Things

AST Formalisms

Xtext

Ecore



MPS Structure

```
concept ConceptDeclaration extends AbstractConceptDeclaration
                           implements INamedConcept

  instance can be root: false

  properties:
    helpURL : string
    rootable : boolean

  children:
    InterfaceConceptReference  implementsInterfaces       0..n
    LinkDeclaration            linkDeclaration            0..n
    PropertyDeclaration        propertyDeclaration        0..n
    ConceptProperty            conceptProperty            0..n
    ConceptLink                conceptLink                0..n
    ConceptPropertyDeclaration conceptPropertyDeclaration 0..n
    ConceptLinkDeclaration     conceptLinkDeclaration     0..n

  references:
    ConceptDeclaration         extendsConcept           0..1
```

Spoofax Aterms

```
Order([ItemNumber(5), Quantity(15), Customer("Mr.\ White")])
```

```
Order(5, 15, "Mr. White")
```

Scoping

Typical Approach

Today's Approach (Xtext or MPS)

Scope = Function that returns the possible target elements for a referenece

Typical Approach

Today's Approach (Xtext or MPS)

Scope = Function that returns the possible target elements for a referenece

Implemented as procedural code.

Typical Approach

Today's Approach (Xtext or MPS)

Scope = Function that returns the possible target elements for a referenece

Implemented as procedural code.

Associated with context directly, via naming convention or procedurally.

Typical Approach

Today's Approach (Xtext or MPS)

```
// <X>, <R>: scoping the <R> reference of the <X> concept
public IScope scope_<X>_<R>(<X> ctx, EReference ref );

// <X>: the language concept we are looking for as a reference target
// <Y>: the concept from under which we try to look for the reference
public IScope scope_<X>(<Y> ctx, EReference ref);
```

Xtext

Typical Approach

MPS

Today's Approach (Xtext or MPS)

```
link {target}
  referent set handler:
    <none>
  search scope:
    (referenceNode, linkTarget, enclosingNode, ...)
              ->join(ISearchScope | sequence<node<State>>) {
      enclosingNode.ancestor<Statemachine>.states;
    }
  validator:
    <default>
  presentation :
    <none>
```

MPS

New Idea

Using DSLs for defining DSLs

Use dedicated DSLs to define aspects of DSLs

New Idea

Using DSLs for defining DSLs

Use dedicated DSLs to define aspects of DSLs

Old idea for grammar/syntax.

New Idea

Using DSLs for defining DSLs

Use dedicated DSLs to define aspects of DSLs

Old idea for grammar/syntax.

Now increasingly used for other aspects such as scoping (we'll see type system, debugging later)

Defining Namespaces

```
entity Customer {
  name : String // Customer.name
}

entity Product {
  name : String // Product.name
}
```

Define Namespaces:

```
Module(m, _):   defines Module m
Entity(e, _):   defines Entity e
Function(f, _): defines Function f
```

Qualified Names

```
Module(m, _):   defines Module m scopes Entity
Entity(e, _):   defines Entity e scopes Property, Function
Function(f, _): defines Function f scopes Variable
```

Defining Namespaces

Name Qualification implies Uniqueness

```
Block(_): scopes Variable
```

Limitations on Visibility

```
Declare(v, _): defines Variable v in subsequent scope
```

```
For(v, t, init, cond, update, body): defines Variable v in cond, update,
      body
```

Defining References

```
entity Customer {
  name : String
}

entity Order {
  customer : Customer
  function getCustomerName(): String {
    return customer.name;
  }
}
```

Defining References

```
entity Customer {
  name : String
}

entity Order {
  customer : Customer
  function getCustomerName(): String {
    return customer.name;
  }
}
```

```
PropAccess(exp, p): refers to Property p in Entity e
```

customer.name

Not clear which Entity!

Defining References

```
entity Customer {
  name : String
}

entity Order {
  customer : Customer
  function getCustomerName(): String {
    return customer.name;
  }
}
```

```
PropAccess(exp, p):
  refers to Property p in Entity e
  where exp has type EntityType(e)
```

customer.name

Integration with type system required.

Con-straints

Standard Constraints

Boolean Expressions

```
@Check(CheckType.NORMAL)
public void checkOrphanEndState( CustomState ctx ) {
    CoolingProgram coopro = Utils.ancestor(ctx, CoolingProgram.class);
    TreeIterator<EObject> all = coopro.eAllContents();
    while ( all.hasNext() ) {
        EObject s = all.next();
        if ( s instanceof ChangeStateStatement ) {
            ChangeStateStatement css = (ChangeStateStatement) s;
            if ( css.getTargetState() == ctx ) return;
        }
    }
    error("no transition ever leads into this state",
      CoolingLanguagePackage.eINSTANCE.getState_Name());
}
```

Xtext

CheckType.NORMAL
CheckType.FAST
CheckType.EXPENSIVE

Standard Constraints

MPS

Boolean Expressions

```
checking rule stateUnreachable {
  applicable for concept = State as state
  do {
    if (!state.initial &&
        state.ancestor<concept = Statemachine>.
            descendants<concept = Transition>.
            where({~it => it.target == state; }).isEmpty) {
        error "orphan state - can never be reached" -> state;
    }
  }
}
```

MPS

Evaluated only on demand via ReadListeners

Standard Constraints

Boolean Expressions

```
constraint-warning:
  Entity(theName, _) -> (theName,
     $[Entity [theName] does not have a capitalized name])
  where
      not(<string-starts-with-capital> theName)
```

```
constraint-error:
  Entity(name, _) -> (name, $[Duplicate definition])
  where
    defs := <index-lookup-all> name;
    <gt> (<length> defs, 1)
```

Spoofax

Based on Pattern Matching → very concise

Dataflow Analysis

The problem

Is code unreachable?

Is a variable read before it is written?

Can a value be null when it is read?

Can we know statically that a variable always has a constant value (and hence can be optimized)?

→ Dataflow Analysis

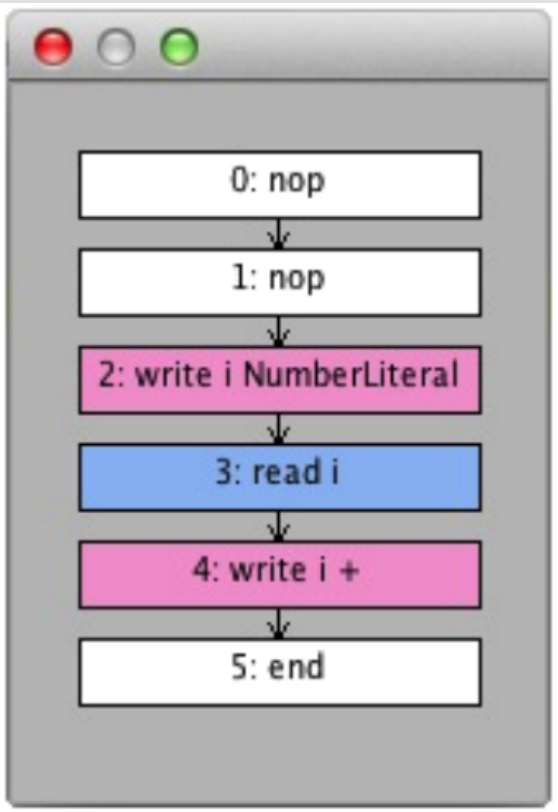
The Dataflow Graph

```
void trivialFunction() {
  int8 i = 10;
  i = i + 1;
}
```

Dataflow Analysis

The Dataflow Graph

```
void trivialFunction() {
  int8 i = 10;
  i = i + 1;
}
```



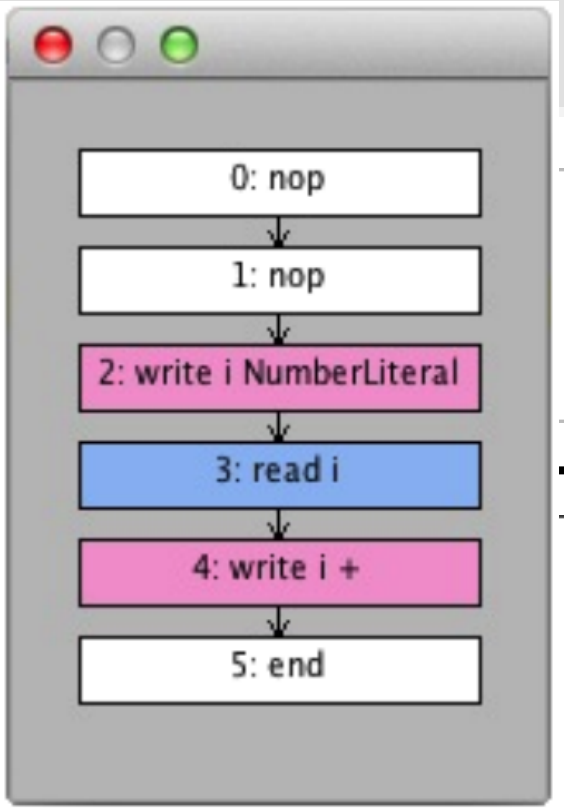

```
data flow builder for LocalVariableDeclaration {
  (node)->void {
    if (node.init != null) {
      code for node.init
      write node = node.init
    } else {
      nop
    }
  }
}
```

Dataflow Analysis

The Dataflow Graph

```
void trivialFunction() {
  int8 i = 10;
  i = i + 1;
}
```

```
data flow builder for AssigmentStatement {
  (node)->void {
    code for node.rvalue
    write node.lvalue = node.rvalue
  }
}
```



Dataflow Analysis

MPS

… for the IfStatement

```
nop
code for node.condition
ifjump after elseIfBlock // elseIfBlock is a label defined later
code for node.thenPart
{ jump after node }
```

Dataflow Analysis

MPS

… for the IfStatement

```
nop
code for node.condition
ifjump after elseIfBlock // elseIfBlock is a label defined later
code for node.thenPart
{ jump after node }
label elseIfBlock
foreach elseIf in node.elseIfs {
  code for elseIf
}
if (node.elsePart != null) {
  code for node.elsePart
}
```

Dataflow Analysis

MPS

… for the IfStatement

```
nop
code for node.condition
ifjump after elseIfBlock // elseIfBlock is a label defined later
code for node.thenPart
{ jump after node }
label elseIfBlock
foreach elseIf in node.elseIfs {
  code for elseIf
}
if (node.elsePart != null) {
  code for node.elsePart
}
```

```
code for node.condition
ifjump after node
code for node.body
{ jump after node.ancestor<concept = IfStatement> }
```

Dataflow Analysis

MPS

The actual Anylses

predefined

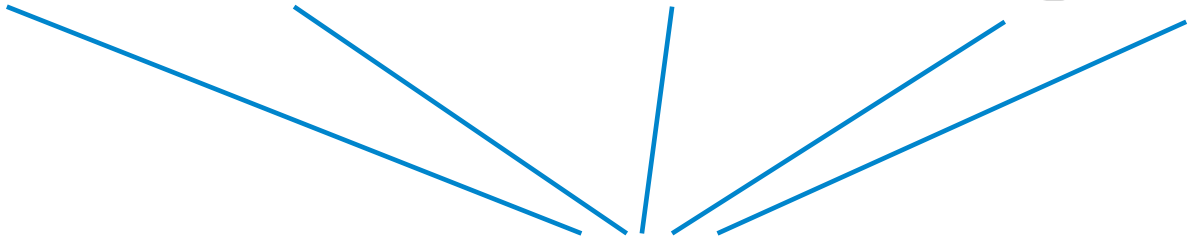
you can also build your own ones

```
public class DataflowUtil {

  private Program prog;

  public DataflowUtil(node<> root) {
    prog = DataFlow.buildProgram(root); // build a program object and store
       it
  }

  public void checkForUnreachableNodes() {
    // grab all instructions that are unreachable (predefined functionality
     )
    sequence<Instruction> allUnreachableInstructions =
             ((sequence<Instruction>) prog.getUnreachableInstructions());
    // remove those that may legally be unreachable
    sequence<Instruction> allWithoutMayBeUnreachable =
            allUnreachableInstructions.where({~instruction =>
                !(Boolean.TRUE.equals(instruction.
                        getUserObject("mayBeUnreachable"))); });

    // get the program nodes that correspond to the unreachable
     instructions
    sequence<node<>> unreachableNodes = allWithoutMayBeUnreachable.
            select({~instruction => ((node<>) instruction.getSource()); });

    // output errors for each of those unreachable nodes
    foreach unreachableNode in unreachableNodes {
      error "unreachable code" -> unreachableNode;
    }
  }
}
```

Type System

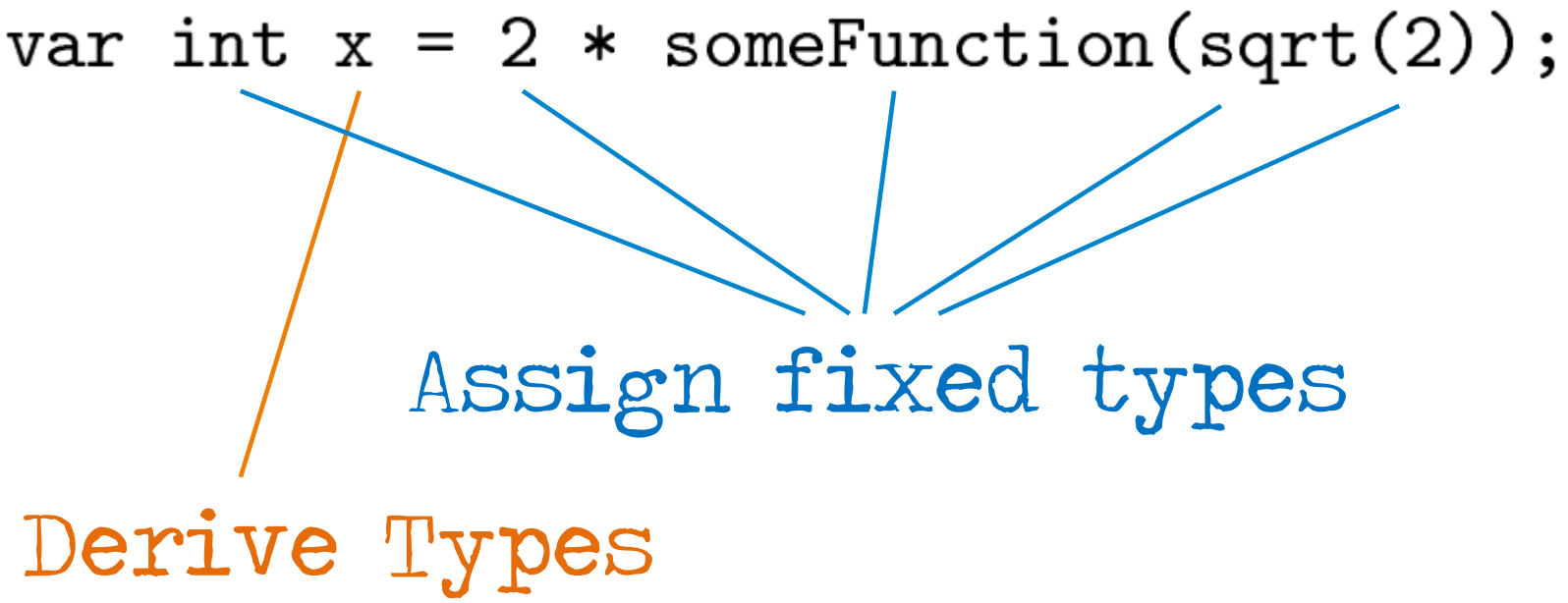
Type Systems?

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

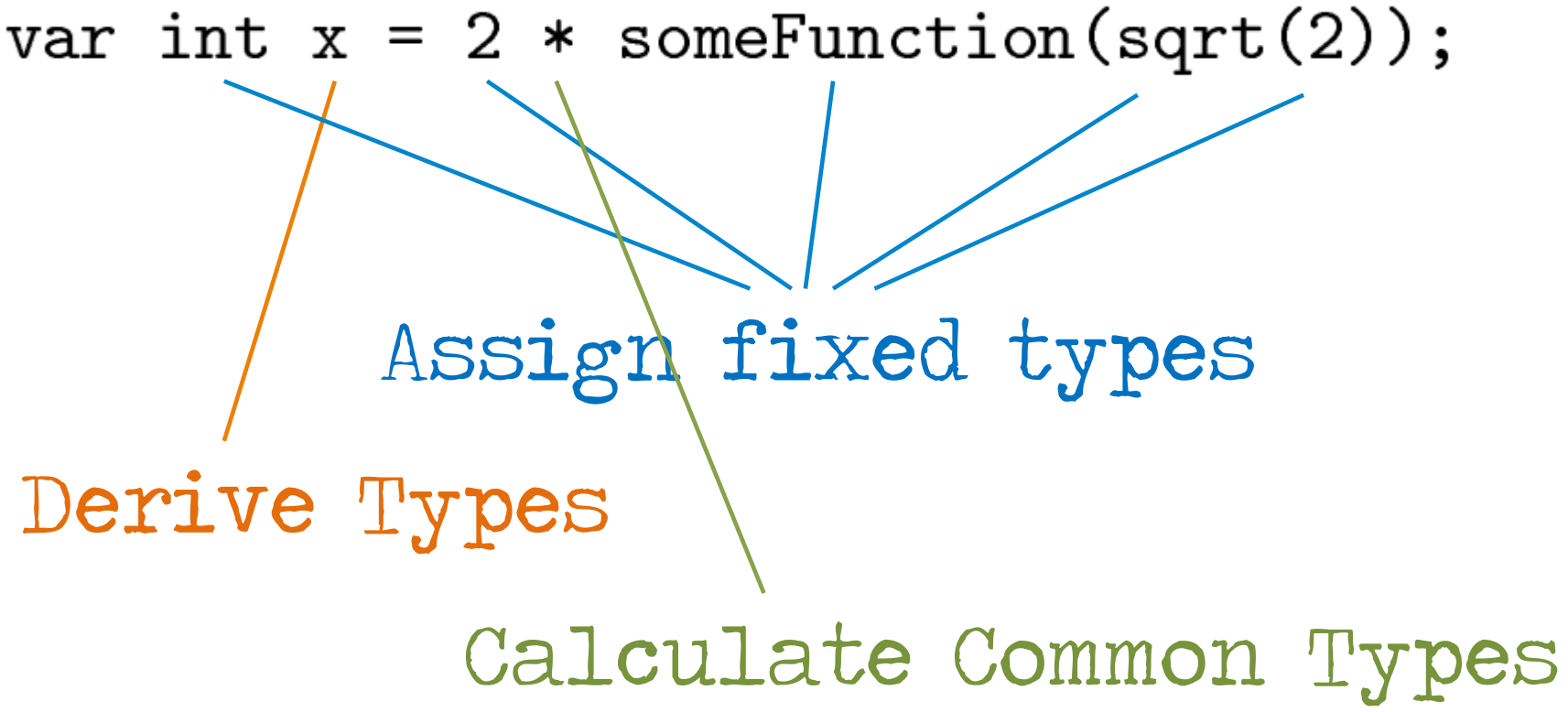


Assign fixed types

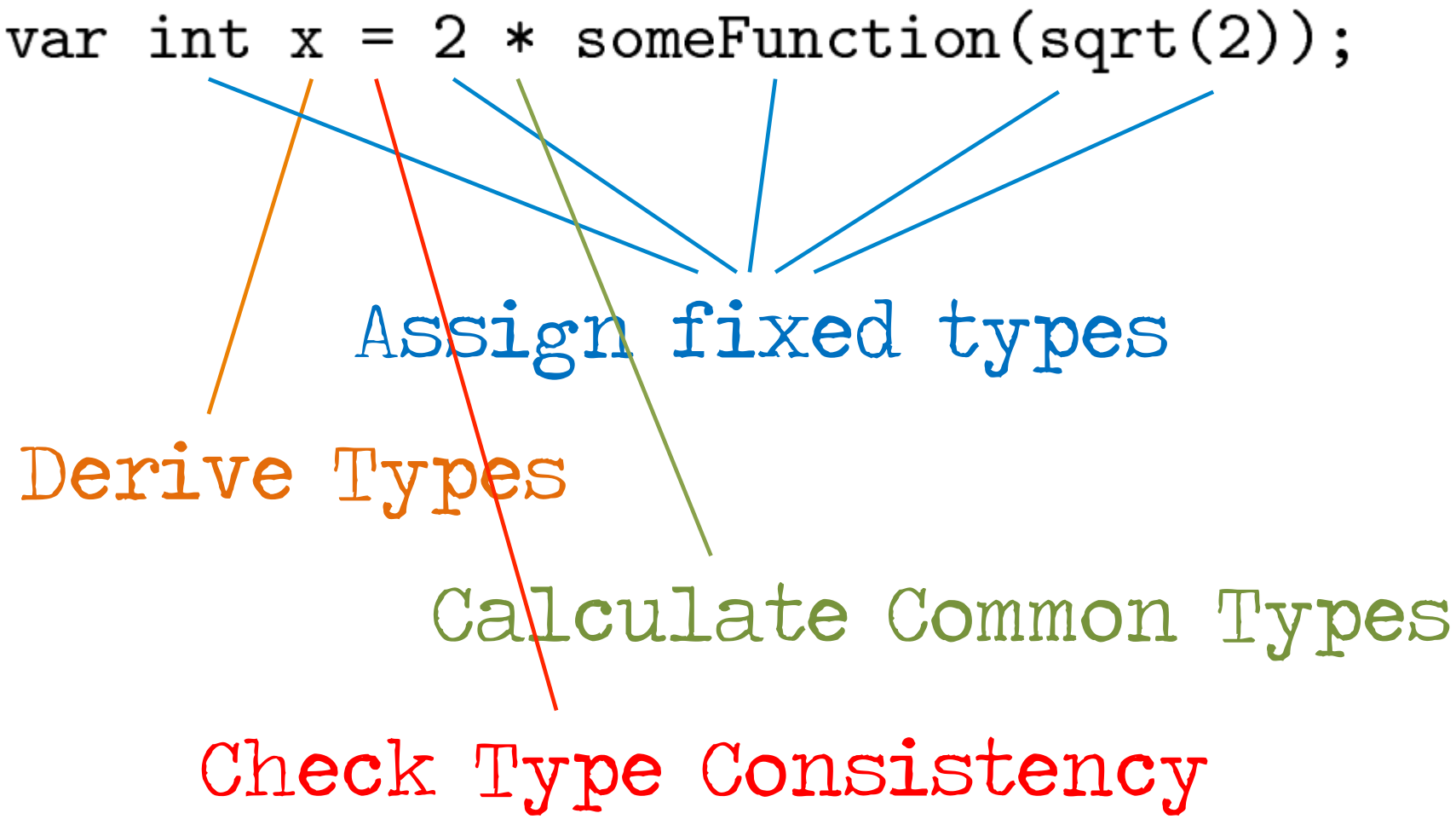
Type Systems?



Type Systems?



Type Systems?



Classic Approach

Xtext

… based on Functions/Recursion

Example Code:

```
var i: int          // 1
var i: int = 42     // 2
var i: int = 33.33  // 3
var i = 42          // 4
```

Classic Approach

Xtext

… based on Functions/Recursion

Example Code:

```
var i: int          // 1
var i: int = 42     // 2
var i: int = 33.33  // 3
var i = 42          // 4
```

Type System Implementation:

```
typeof( LocalVarDecl lvd ) {
    return typeof( lvd.type )
}

typeof( IntType it ) { return it }
typeof( DoubleType dt ) { return dt }
```

Classic Approach

Xtext

… based on Functions/Recursion

Better Type System Implementation:

```
typeof( LocalVarDecl lvd ) {
    if !isSpecified( lvd.type ) && !isSpecified( lvd.init )
        raise error

    if isSpecified( lvd.type ) && !isSpecified( lvd.init )
      return typeof( lvd.type )

    if !isSpecified( lvd.type ) && isSpecified( lvd.init )
      return typeof( lvd.init )

    // otherwise...
    assert typeof( lvd.init ) isSameOrSubtypeOf typeof( lvd.type )
    return typeof( lvd.type )
}
```

Declarative Type Sys.

MPS

Unification/Solver (MPS)

Example Code:

```
var i: int          // 1
var i: int = 42     // 2
var i: int = 33.33  // 3
var i = 42          // 4
```

Type System Implementation:

```
typeof( LocalVarDecl.type ) :>=: typeof( LocalVarDecl.init )
typeof( LocalVarDecl )      :==: typeof( LocalVarDecl.type )
```

Declarative Type Sys.

MPS

Unification/Solver (MPS)

Example Code:

```
var i: int          // 1
var i: int = 42     // 2
var i: int = 33.33  // 3
var i = 42          // 4
```

Type System Implementation:

```
typeof( LocalVarDecl.type ) :>=: typeof( LocalVarDecl.init )
typeof( LocalVarDecl )      :==: typeof( LocalVarDecl.type )
```

Example Scenario 1

```
// var i: int
typeof( int ) :>=: typeof( int )      // ignore
typeof( T )   :==: typeof( int )      // T := int
```

Declarative Type Sys.

MPS

Unification/Solver (MPS)

Example Code:

```
var i: int          // 1
var i: int = 42     // 2
var i: int = 33.33  // 3
var i = 42          // 4
```

Type System Implementation:

```
typeof( LocalVarDecl.type ) :>=: typeof( LocalVarDecl.init )
typeof( LocalVarDecl )      :==: typeof( LocalVarDecl.type )
```

Example Scenario 2

```
// var i: int = 42
typeof( int ) :>=: typeof( int )    // true
typeof( T )   :==: typeof( int )    // T := int
```

Declarative Type Sys.

MPS

Unification/Solver (MPS)

Example Code:

```
var i: int          // 1
var i: int = 42     // 2
var i: int = 33.33  // 3
var i = 42          // 4
```

Type System Implementation:

```
typeof( LocalVarDecl.type ) :>=: typeof( LocalVarDecl.init )
typeof( LocalVarDecl )      :==: typeof( LocalVarDecl.type )
```

Example Scenario 3

```
// var i: int = 33.33
typeof( int ) :>=: typeof( double )   // error!
typeof( T )   :==: typeof( int )      // T := int
```

Declarative Type Sys.

MPS

Unification/Solver (MPS)

Example Code:

```
var i: int          // 1
var i: int = 42     // 2
var i: int = 33.33  // 3
var i = 42          // 4
```

Type System Implementation:

```
typeof( LocalVarDecl.type ) :>=: typeof( LocalVarDecl.init )
typeof( LocalVarDecl )      :==: typeof( LocalVarDecl.type )
```

Example Scenario 4

```
// var i = 42
typeof( U ) :>=: typeof( int )  // U := int
typeof( T ) :==: typeof( U )    // T := int
```

Declarative Type Sys.

„Automatic" Type Inference

Example Code:

```
var i: int[]
var i: int[] = {1, 2, 3}
var i = {1, 2, 3}
```

Declarative Type Sys.

MPS

„Automatic" Type Inference

Example Code:

```
var i: int[]
var i: int[] = {1, 2, 3}
var i = {1, 2, 3}
```

Type System Implementation:

```
typevar T
foreach ( e: init.elements )
    typeof(e) :<=: T

typeof( LocalVarDecl.type ) :>=: new ArrayType(T)
typeof( LocalVarDecl )      :==: typeof( LocalVarDecl.type )
```

That is all!

Declarative Type Sys.

Spoofax

Pattern Matching

```
type-of: Int(value) -> NumType()
```

Declarative Type Sys.

Spoofax

Pattern Matching

```
type-of: Int(value) -> NumType()
```

```
type-of:
  Add(exp1, exp2) -> NumType()
  where
    <type-of> exp1 => NumType();
    <type-of> exp2 => NumType()
```

Declarative Type Sys.

Spoofax

Pattern Matching

```
type-of: Int(value) -> NumType()
```

```
type-of:
  Add(exp1, exp2) -> NumType()
  where
    <type-of> exp1 => NumType();
    <type-of> exp2 => NumType()
```

```
type-of:
  Add(exp1, exp2) -> StringType()
  where
    <type-of> exp1 => StringType();
    <type-of> exp2 => StringType()
```

Declarative Type Sys.

Spoofax

Pattern Matching

```
type-of: Int(value) -> NumType()
```

```
type-of:
  Add(exp1, exp2) -> NumType()
  where
    <type-of> exp1 => NumType();
    <type-of> exp2 => NumType()
```

```
type-of:
  Add(exp1, exp2) -> StringType()
  where
    <type-of> exp1 => StringType();
    <type-of> exp2 => StringType()
```

```
Property(p, t): defines Property p of type t
Param(p, t):    defines Variable p of type t
```

Transformation & Generation

Text Template Engine

Xtext's Xtend (M2T, M2M)

Not „just" a template language like StringTemplate or Xpand.

Actually a very nice and full-blown GPL with special support for text templating and model navigation, query and construction.

Text Template Engine

Xtext

Xtext's Xtend (M2T, M2M)

Java Like

Many improvemnts over Java

Modularization via DI

Template Expressions

Smart Whitespace Handling

Functional Abstractions

Dispatch on Arguments

Extension Methods

Builders

Text Template Engine

Xtext

Xtext's Xtend

*CoolingLanguageGenerator.xtend

```
class CoolingLanguageGenerator implements IGenerator {

    @Inject extension StatementExtensions se
    @Inject extension ExpressionExtensions ee
    @Inject extension DataTypeHelper dh

    override void doGenerate(Resource resource, IFileSystemAccess fsa) {
        for( p: (resource.contents.first as CoolingResource).coolingPrograms){
            fsa.generateFile(p.name+"/"+p.name+".c", p.compile)
        }
    }

    def compile(CoolingProgram p){
        '''
        /*
         ============================================================================================
         Name        : «p.name».c
         Version     :
         Copyright   : Copyright (c) 2011 itemis AG (http://www.itemis.eu). All rights reserved.
         ============================================================================================
         */

        #include "framework.h"

        void init (void){
            new_state = «p.startState.name»;
            «IF p.initBlock!=null»
                «FOR variable: p.variables.filter(v | v.init != null)»
                    «variable.name» = «variable.init.compileExpr»;
                «ENDFOR»

                «FOR s: p.initBlock.statements»
                    «s.compileStatement»
                «ENDFOR»
            «ENDIF»
        }
```

Dispatch Methods

```
def dispatch String compileExpr (Equals e){
    e.left.compileExpr + " == " + e.right.compileExpr
}

def dispatch String compileExpr (Greater e){
    e.left.compileExpr + " > " + e.right.compileExpr
}

def dispatch String compileExpr (Plus e){
    e.left.compileExpr + " + " + e.right.compileExpr
}

def dispatch String compileExpr (NotExpression e){
    "!(" + e.expr.compileExpr + ")"
}

def dispatch String compileExpr (TrueExpr e){
    "TRUE"
}

def dispatch String compileExpr (ParenExpr pe){
    "(" + pe.expr.compileExpr + ")"
}

def dispatch compileExpr (NumberLiteral nl){
    nl.value
}
```

Builder Syntax (1)

```
class Transformation {

  @Inject extension CoolingBuilder
  CoolingLanguageFactory factory = CoolingLanguageFactory::eINSTANCE

  def CoolingProgram transform(CoolingProgram p ) {
    p.states += emergencyState
    p.events += emergencyEvent
    for ( s: p.states.filter(typeof(CustomState)).filter(s|s !=
      emergencyState) ) {
      s.events += s.eventHandler [
          symbolRef [
              emergencyEvent()
          ]
          changeStateStatement(emergencyState())
        ]
    }
    return p;
  }

  def create result: factory.createCustomState emergencyState() {
    result.name = "EMERGENCY_STOP"
  }

  def create result: factory.createCustomEvent emergencyEvent() {
    result.name = "emergency_stop_button_pressed"
  }
}
```

Builder Syntax (2)

```
class CoolingBuilder {

    CoolingLanguageFactory factory = CoolingLanguageFactory::eINSTANCE

    def eventHandler( CustomState it, (EventHandler)=>void handler ) {
        val res = factory.createEventHandler
        res
    }

    def symbolRef( EventHandler it, (SymbolRef)=>void symref ) {
        val res = factory.createSymbolRef
        it.events += res
    }

    def symbol( SymbolRef it, CustomEvent event ) {
        it.symbol = event
    }

    def changeStateStatement( EventHandler it, CustomState target ) {
        val res = factory.createChangeStateStatement
        it.statements += res
        res.targetState = target
    }
}
```

Xtext, Xbase and Java

Instead of generating Java source, M2M to Java AST (for structurally relevant parts)

Xtext, Xbase and Java

Instead of generating Java source, M2M to Java AST (for structurally relevant parts)

You get code gen implicitly.

Xtext, Xbase and Java

Instead of generating Java source, M2M to Java AST (for structurally relevant parts)

You get code gen implicitly, but also:

- scoping
- type system
- debugger

Model Inferrence

Xtext, Xbase and Java

```
class EntityWithExprDslJvmModelInferrer extends AbstractModelInferrer {

    @Inject extension IQualifiedNameProvider
    @Inject extension JvmTypesBuilder

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

Model Inferrence

Xtext, Xbase and Java

```
class EntityWithExprDslJvmModelInferrer extends AbstractModelInferrer {

    @Inject extension IQualifiedNameProvider
    @Inject extension JvmTypesBuilder

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

```
acceptor.accept(
    entity.toClass( entity.fullyQualifiedName ) [
        documentation = entity.documentation ...
    ]
)
```

Model Inferrence

Xtext, Xbase and Java

```
acceptor.accept(
    entity.toClass( entity.fullyQualifiedName ) [
        documentation = entity.documentation ...
    ]
)
```

```
for ( attr : entity.attributes ) {
    members += attr.toField(attr.name, attr.type)
    members += attr.toGetter(attr.name, attr.type)
    members += attr.toSetter(attr.name, attr.type)
}
```

Xtext, Xbase and Java

```
public JvmOperation toSetter(EObject sourceElement, final String name,
                             JvmTypeReference typeRef) {
  JvmOperation result = TypesFactory.eINSTANCE.createJvmOperation();
  result.setVisibility(JvmVisibility.PUBLIC);
  result.setSimpleName("set" +  nullSaveName(Strings.toFirstUpper(name)));
  result.getParameters().add(toParameter(sourceElement, nullSaveName(name),
                             cloneWithProxies(typeRef)));
  if (name != null) {
    setBody(result, new Functions.Function1<ImportManager, CharSequence>()
      {
        public CharSequence apply(ImportManager p) {
          return "this." + name + " = " + name + ";";
        }
    });
  }
  return associate(sourceElement, result);
}
```

Model Inferrence

Xtext, Xbase and Java

```
public JvmOperation toSetter(EObject sourceElement, final String name,
                             JvmTypeReference typeRef) {
  JvmOperation result = TypesFactory.eINSTANCE.createJvmOperation();
  result.setVisibility(JvmVisibility.PUBLIC);
  result.setSimpleName("set" +  nullSaveName(Strings.toFirstUpper(name)));
  result.getParameters().add(toParameter(sourceElement, nullSaveName(name),
                             cloneWithProxies(typeRef)));
  if (name != null) {
    setBody(result, new Functions.Function1<ImportManager, CharSequence>()
      {
        public CharSequence apply(ImportManager p) {
          return "this." + name + " = " + name + ";";
        }
    });
  }
  return associate
}
```

Java text for the body/ implementation.
(no IDE support here)

Multi-Stage Trafo

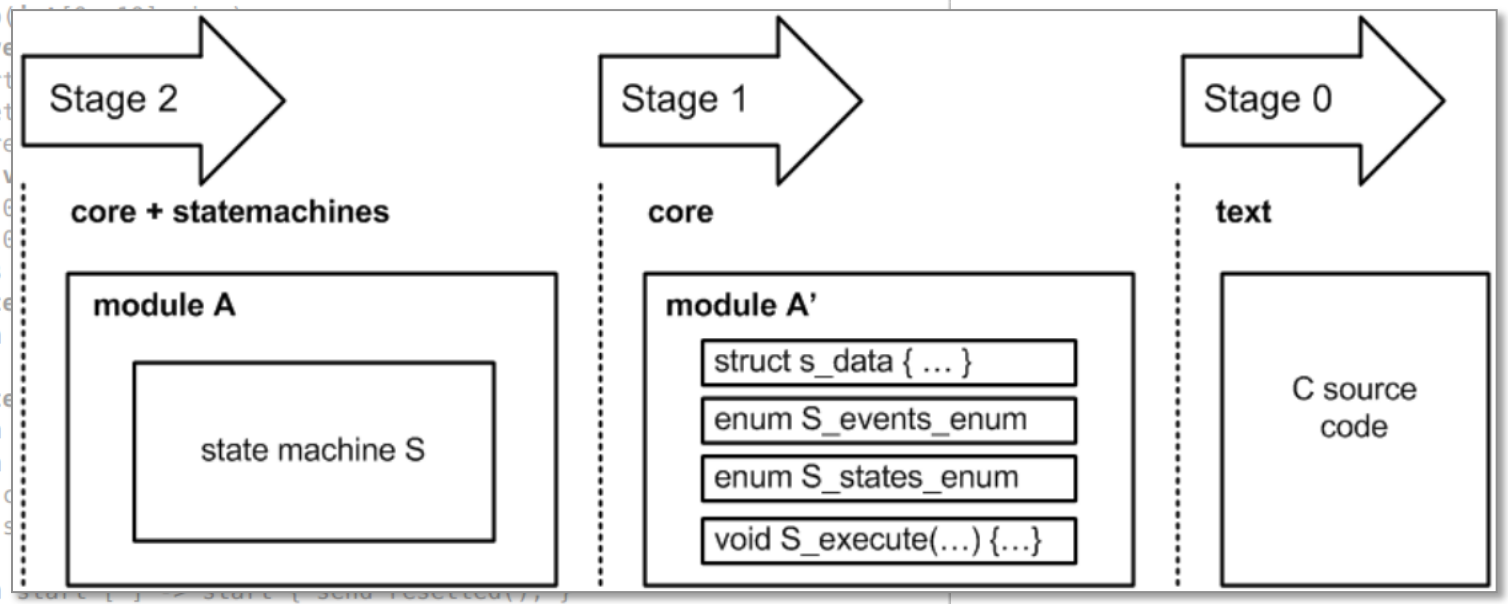
MPS' Transformations

```
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' Transformations

```
module Statemachine imports nothing {

  statemachine Counter {
    in events
      start()
      step(
    out eve
      start
      reset
      incre
    local v
      int[0
      int[0
    states
      state
        on
      }
      state
        on
        on
          C
          s
        }
        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 ) {
      $LOOP$ case ->$aState: {
               switch ( event ) {
                 $LOOP$ case ->$anEvent: {
                          $LOOP$ if ( $COPY_SRC$true ) {
                                   $COPY_SRCL$int8_t exitActions; 
                                   $COPY_SRCL$int8_t transActions; 
                                   instance->__currentState = ->$aState;
                                   $COPY_SRCL$int8_t entryActions; 
                                   return;
                                 } if
                          break;
                        }
               } switch
               break;
             }
      } switch
TF>

  } statemachineFunction (function)
}
```

Transformations

MPS

with concrete Syntax & IDE

```
template generateSwitchCase
input    Statemachine
content node:
module dummy imports nothing {
  enum events { anEvent; }
  enum states { aState; }
  struct stat
    states __
  };
  void statem
    <TF swit
      $LO
                                    return;
                                  } if
                                break;
                              }
                            } switch
                            break;
                          }
                        } switch

  } statemachineFunction (function)
}
```

```
concept     EventArgRef
inheritors false
condition   <always>
     --> content node:
         module dummy imports nothing {
           void dummy2(void*[ ] arguments) {
                 <TF [ (*(($COPY_SRC$[int8_t]*) (arguments[$[1]]))) ] TF>; }
         }
```

Transformations

Builders are supported as well

```
node<ImplementationModule> immo = build ImplementationModule
  name = #(aNamePassedInFromAUserDialog)
  contents += tc:TestCase
          name = "testCase1"
          type = VoidType
          contents += #(MainFunctionHelper.createMainFunction())
            body = StatementList statements += ReturnStatement
                    expression = ExecuteTestExpression
                                 tests += TestCaseRef
                                          testcase -> tc
```

Language Extension:

- No builder functions necessary
- IDE support
- But is specific for Models

Pattern Matching

For text generation

```
to-java:
  Entity(x, ps) ->
  $[ class [x] {
         [ps']
     }
  ]
  with
    ps' := <map(to-java)> ps

to-java:
    Property(x, t) ->
    $[ private [t'] [x];

       public [t'] get_[x] {
           return [x];
       }

       public void set_[x] ([t'] [x]) {
           this.[x] = [x];
       }
    ]
    with
      t' := <to-java> t
```

Pattern Matching

Spoofax

For transformation

```
to-java: NumType()    -> IntBaseType()
to-java: BoolType()   -> BooleanBaseType()
to-java: StringType() -> ClassType("java.lang.String")
to-java: EntType(t)   -> ClassType(t)

to-java:
  Entity(x, ps) -> Class([], x, ps')
     ps' := <mapconcat(to-java)> ps

to-java:
  Property(x, t) -> [field, getter, setter]
  with
    t'      := <to-java> t ;
    field   := Field([Private()], t', x) ;
    getter := Method([Public()], t', $[get_[x]], [], [Return(VarRef(x))]) ;
    setter := Method([Public()], Void(), $[set_[x]], [Param(t', x)], [
      assign]) ;
    assign := Assign(FieldRef(This(), x), VarRef(x))
```

Pattern Matching

Spoofax

M2M with concrete Syntax

```
  to-java:
    |[ entity |[x]| { |[ps]| } ]| ->
    |[ class |[x]| { |[<mapconcat(to-java)> ps]| } ]|

to-java:
    |[ |[x]| : |[t]| ]| ->
    |[ private |[t']| |[x]|;

       public |[t']| |[x]| { return |[x]|; }

       public void |[x]| (|[t']| |[x]|) { this.|[x]| = |[x]|; } ]|
    with
      t' := <to-java> t
```

Pattern Matching

Spoofax

M2M with concrete Syntax

```
to-java:
  |[ entity |[x]| { |[ps]| } ]| ->
  |[ class
```

```
to-java:
  |[ |[x]|
  |[ priva
```

```
    publi
```

```
    publi
  with
    t' :=
```

```
module Stratego-Mobl-Java
imports Mobl
imports Java     [ ID   => JavaId ]
imports Stratego [ Id   => StrategoId
                   Var  => StrategoVar
                   Term => StrategoTerm ]
```

```
exports context-free syntax

  "|[" Module "]|"          -> StrategoTerm {"ToTerm"}
  "|[" Import "]|"          -> StrategoTerm {"ToTerm"}
  "|[" Entity "]|"          -> StrategoTerm {"ToTerm"}
  "|[" EntityBodyDecl "]|" -> StrategoTerm {"ToTerm"}

  "|[" JClass  "]|"    -> StrategoTerm {"ToTerm"}
  "|[" JField "]|"     -> StrategoTerm {"ToTerm"}
  "|[" JMethod "]|"    -> StrategoTerm {"ToTerm"}
  "|[" JFeature* "]|" -> StrategoTerm {"ToTerm"}
```

Needs escaped composition grammar!

Inter-pretation

Interpreters

Xtend is very useful!

- Java Like
- General Purpose Language
- Many improvemnts over Java
- **Functional Abstractions**
- **Dispatch on Arguments**
- **Extension Methods**
- Builders
- Template Expressions
- Smart Whitespace Handling
- Trafo Modularization via DI

MPS interpreter extension

```
public static Double eval(node<Expression> ex, final node<FunctionUnitTest> test) {
  ErrorMarkers.remove(ex);
  return dispatch <Double> (ex)                                                    ;
          MulExpression          -> eval($.leftExpression, test) * eval($.rightExpression, test)
          DivExpressionFraction  -> eval($.numerator, test) / eval($.denominator, test)
          IntegerConstant        -> new Double($.value)
          FloatingPointConstant  -> Double.valueOf($.value)
          Exp                    -> Math.pow(eval($.base, test), eval($.exp, test))
          SymbolReference        -> getValue($, test)
          default: 0.0
}
```

makes writing dispatchers much simpler - modular extension!

Embedding Interpreters

```
simple product SimpleHomeInsurance

  attributes
    squareMeters : int16
    numberOfRooms : int16

  calculate
    int16, 0
```

	squareMeters < 150	squareMeters >= 150
numberOfRooms < 3	10	20
numberOfRooms == 3	30	40
numberOfRooms > 3	40	50

tests

squareMeters	numberOfRooms		
100	2	10	10.0
150	3	40	40.0
200	5	35	50.0

evaluate

Add Test Case

real-time feedback in the IDE!

IDE Services

Syntax Highlighting

MPS

Declarative Approach

```
editor for concept GlobalVariableDeclaration
  node cell layout:
    [/
    # preventNameManglingFlag #
    [- # externFlag # # exportedFlag # var % type % { name } ; -]
    /]

editor component exportedFlag
  applicable concept:
    IModuleContent
  component cell layout:
    ?^ exported

Style:
<no base style> {
  text-foreground-color : darkGreen
  font-style : bold
}
```

Syntax Highlighting

MPS

Declarative Approach

```
Style:
<no base style> {
  text-foreground-color :
      (node, editorContext)->Color {
       boolean hasIncoming = node.ancestor<concept = Statemachine
            descendants<concept = Transition>.any({~it =>
              it.targetState == node; });
       if (hasIncoming) {
         return Color.black;
       } else {
         return Color.gray;
       }
      }
}
```

Syntax Highlighting

Spoofax

Declarative Approach

```
module DSLbook-Colorer

imports DSLbook-Colorer.generated

colorer

  Type.NumType: darkgreen
```

Find References

MPS

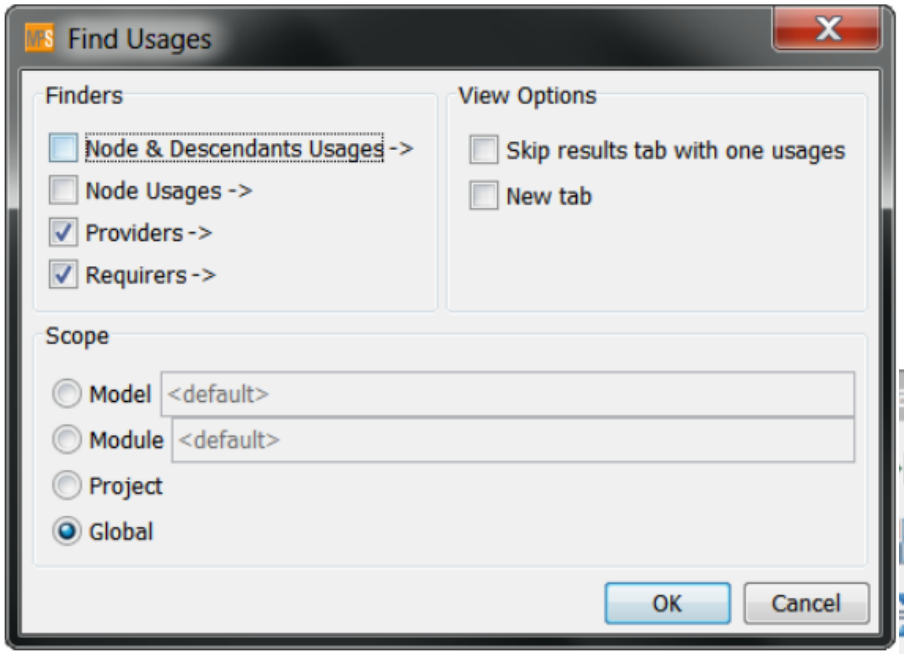
Custom Finders

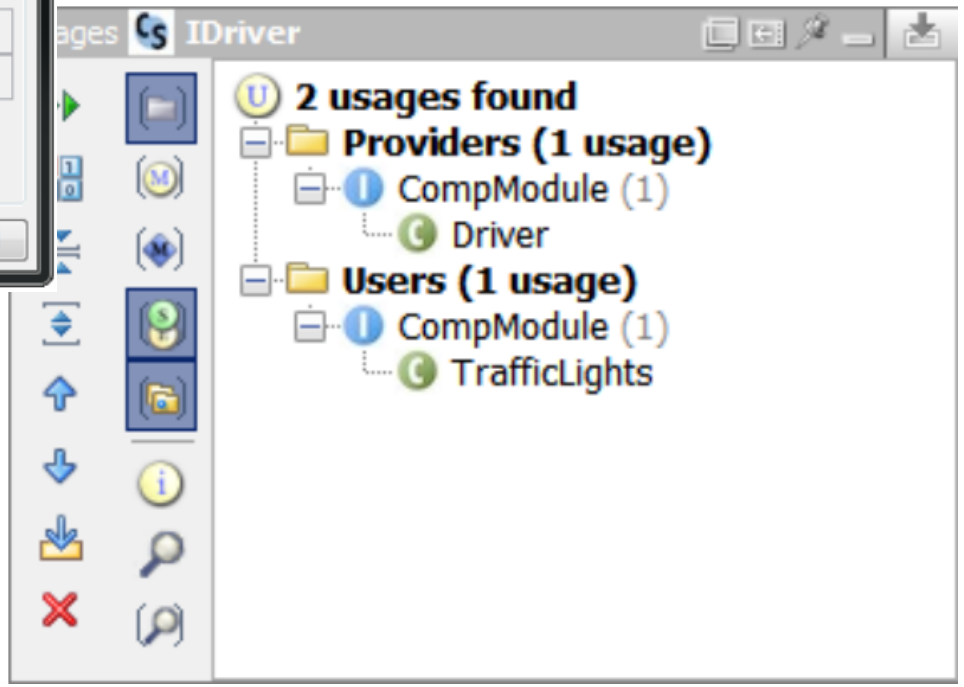
```
finder findProviders for concept Interface
  description: Providers

  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





Formatting

Spoofax

Declarative Box Language

```
[
   Module                    -- KW["module"] _1 _2,
   Module.2:iter-star        -- _1,
   Import                    -- KW["import"] _1,
   Entity                    -- KW["entity"] _1 KW["{"] _2 KW["}"],
   Entity.2:iter-star        -- _1,
   Property                  -- _1 KW[":"] _2,
   Function                  -- KW["function"] _1 KW["("] _2 KW[")"]
                                   KW[":"] _3 KW["{"] _4 KW["}"],
   Function.2:iter-star-sep  -- _1 KW[","],
   Function.4:iter-star      -- _1,
   Param                     -- _1 KW[":"] _2,
   EntType                   -- _1,
   NumType                   -- KW["int"],
   BoolType                  -- KW["boolean"],
   StringType                -- KW["string"],
   Declare                   -- KW["var"] _1 KW["="] _2 KW[";"],
   Assign                    -- _1 KW["="] _2 KW[";"],
   Return                    -- KW["return"] _1 KW[";"],
   Call                      -- _1 KW["."] _2 KW["("] _3 KW[")"],
   PropAccess                -- _1 KW["."] _2,
   Plus                      -- _1 KW["+"] _2,
   Mul                       -- _1 KW["*"] _2,
   Var                       -- _1,
   Int                       -- _1
]
```

Visualization

Jan Koehnlein's GraphView

```
diagram EClassHierarchy type EClass {
   node EClassNode for this {
      label Name for this
      edge SuperType for each this.getESuperTypes()
         => call EClassNode for this
      }
   }
}
```

```
stylesheet EClassHierarchy
   for EClassHierarchy

style EClassNode.SuperType {
   var arrow = new PolygonDecoration()
   arrow.setScale(10,10)
   arrow.backgroundColor = color(#ffffff)
   arrow.lineWidth = 2
   this.targetDecoration = arrow
}
```

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

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.

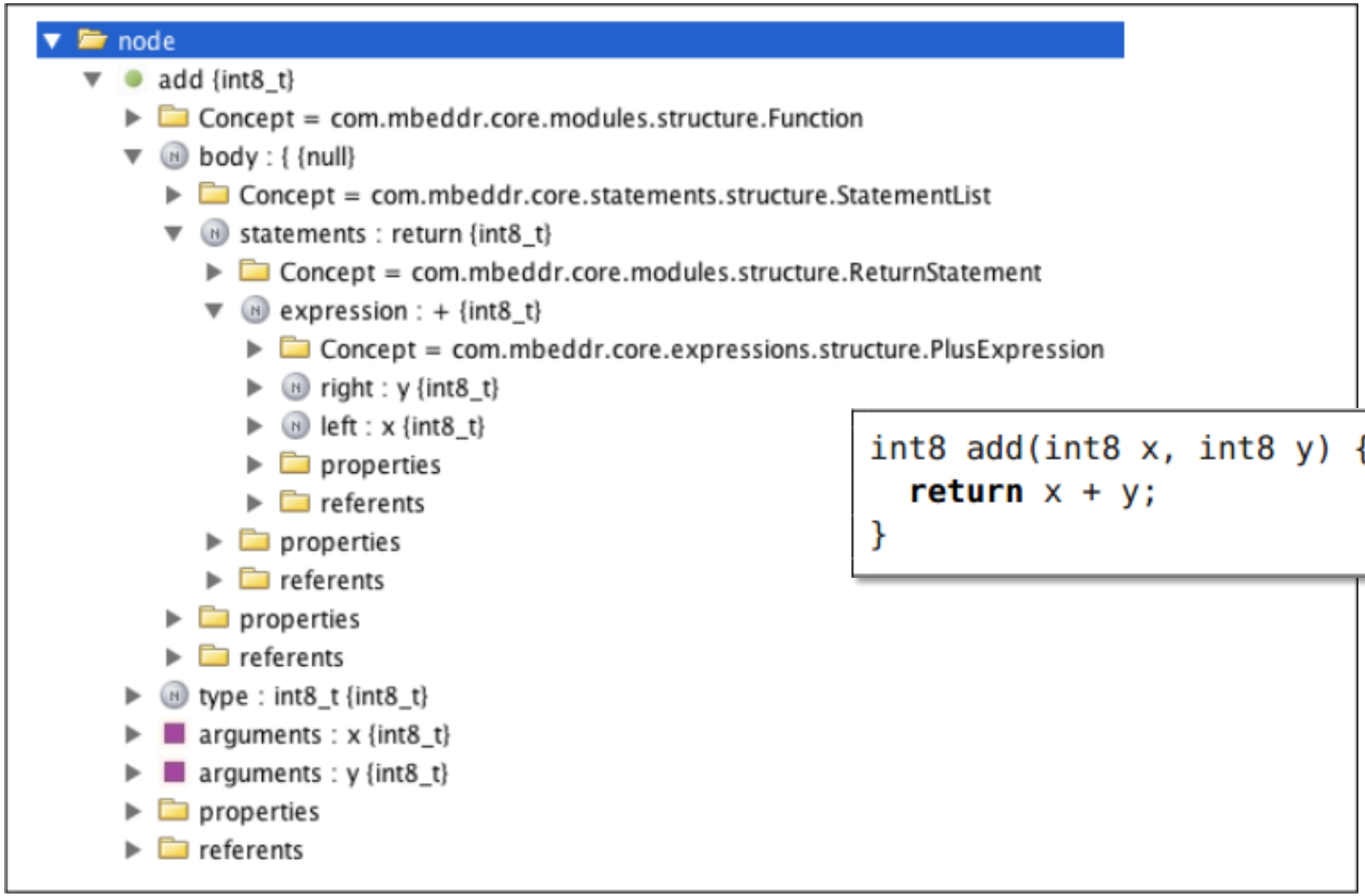
Can test various language aspects such as scopes, code completion, type system; also extensible.

Inspecting/ Debugging

Inspecting Programs

MPS

Parsing doesn't happen!

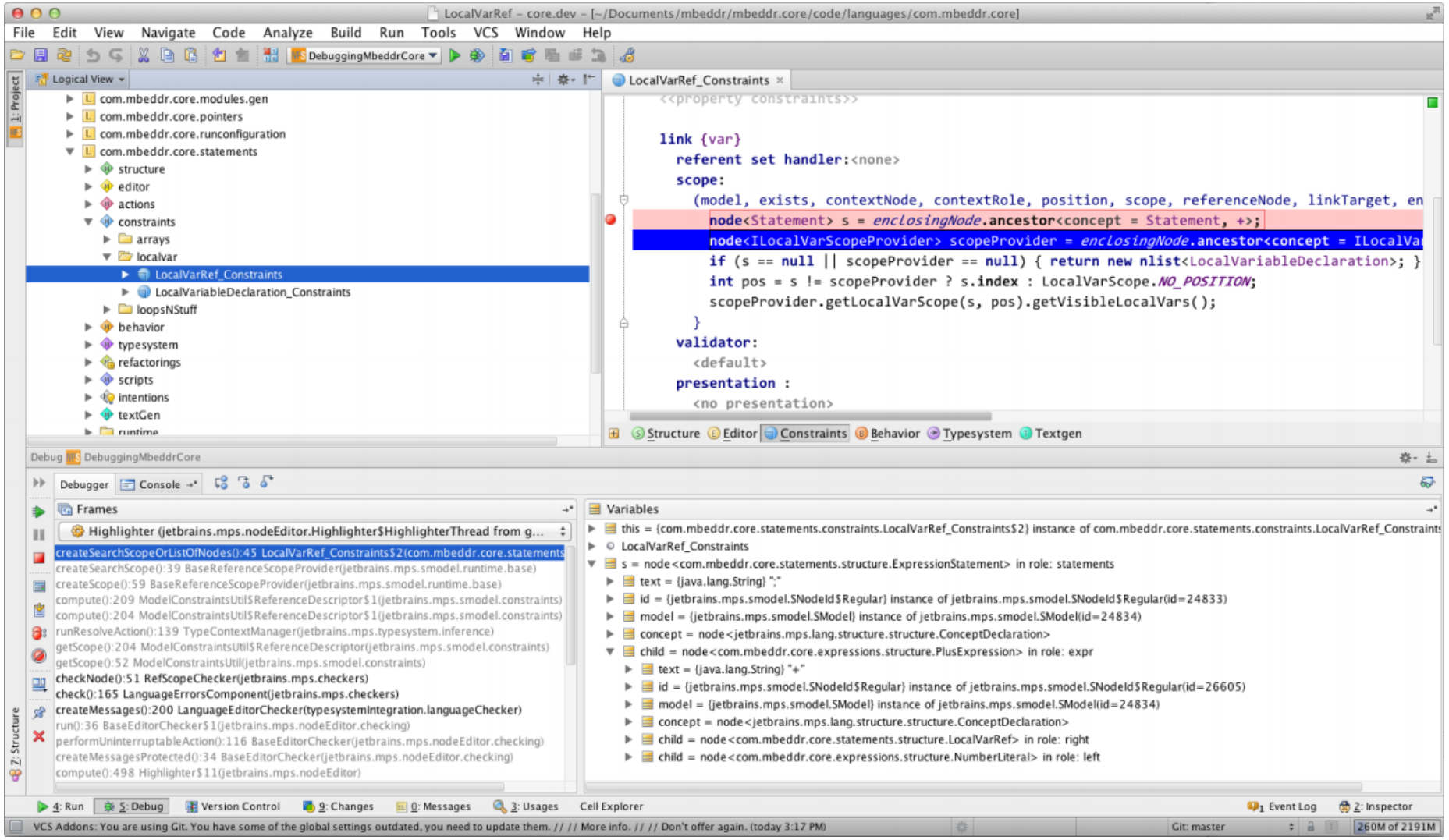



```
int8 add(int8 x, int8 y) {
  return x + y;
}
```

Debugging DSLs

MPS

Behavior

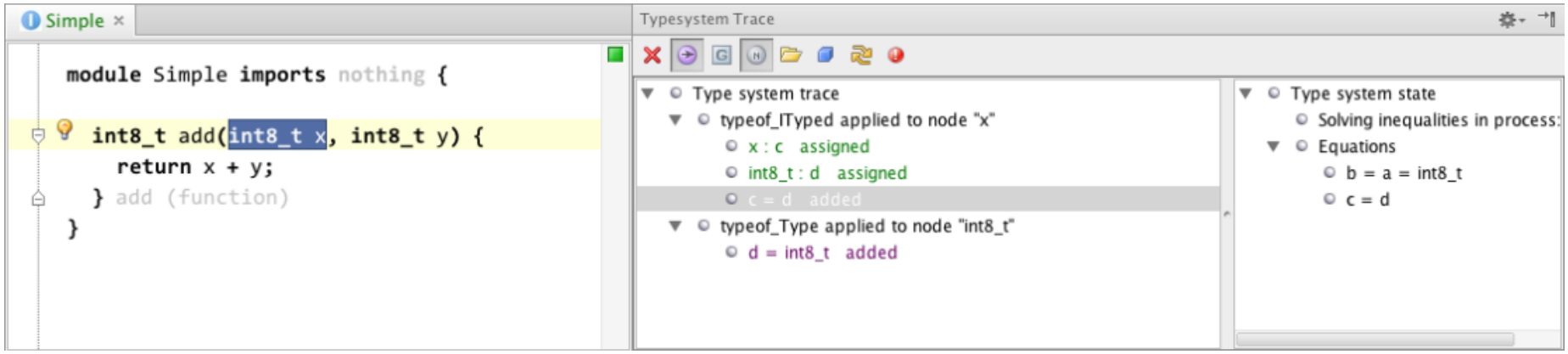


Debugging DSLs

MPS

Type System

Declarative Type Systems use Equations and a Solver.

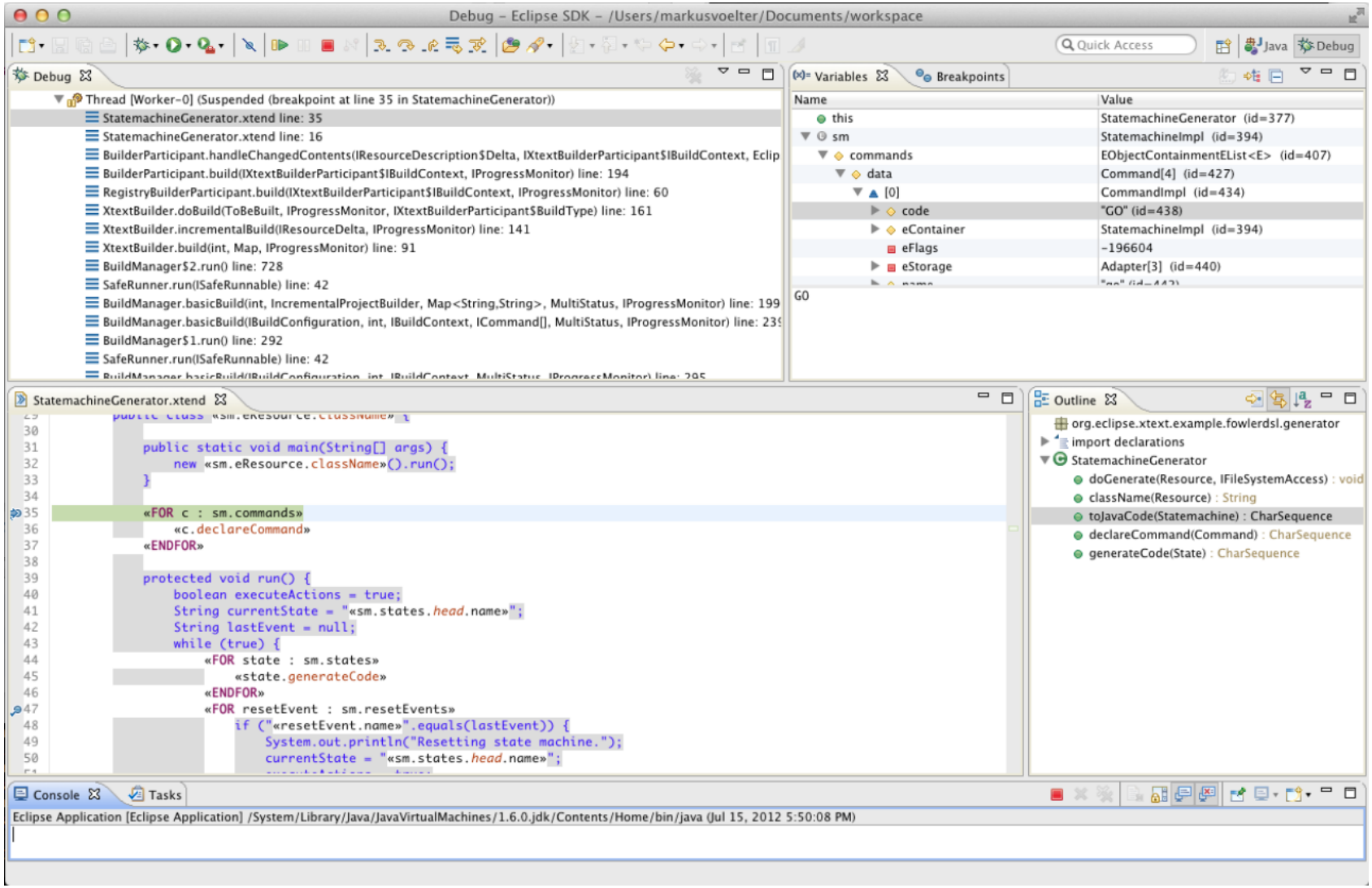


Equations + Solver State can be visualized (not always intuitive)

Debugging DSLs

Xtext

Transformations & Code Gens

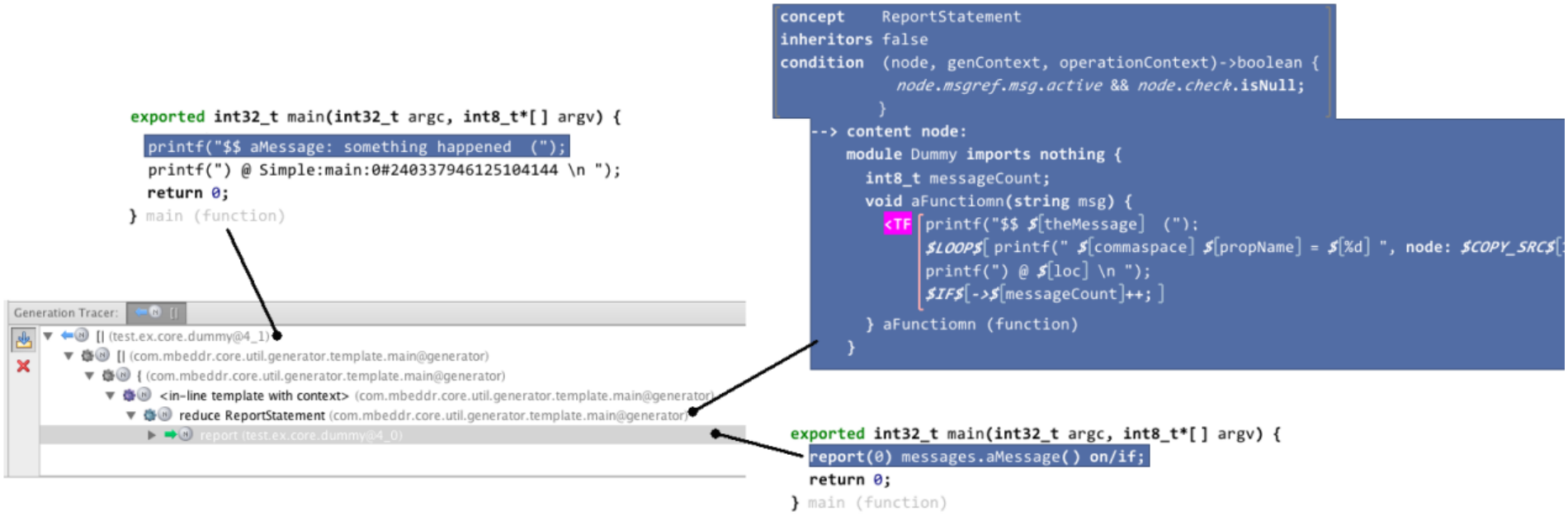


Debugging DSLs

MPS

Transformations

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



Xbase Automatic Debugging

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

Mostly. Sometimes you have to manually establish traces.

Debugging DSL Programs

MPS

Extensible Debuggers

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

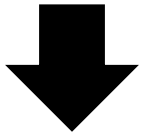
Debugging DSL Programs

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

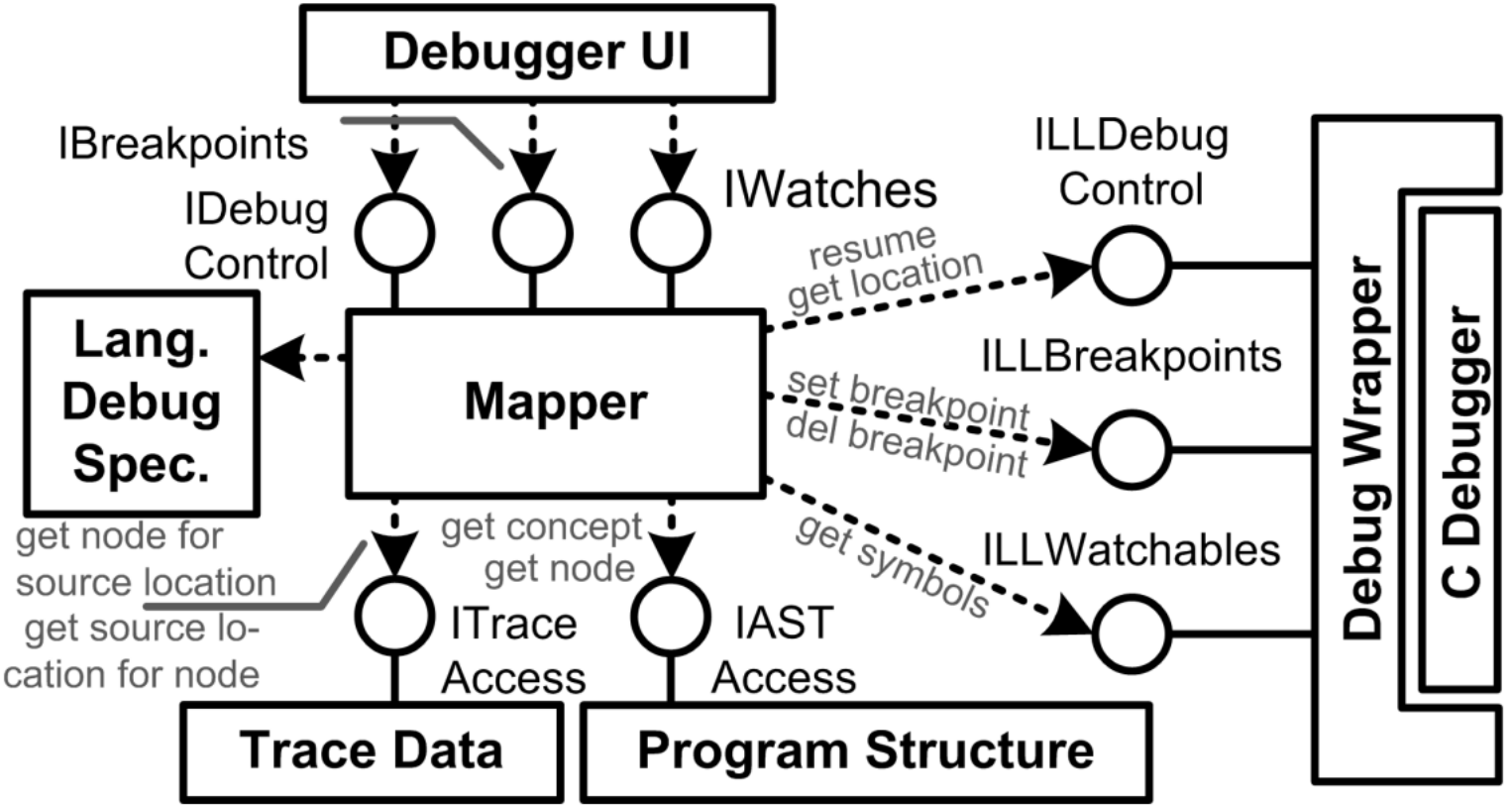
Extensible Debuggers

```
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
}
```

Debugger Architecture



Modulari-zation, Extension and Composition

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 ex-tensions!

Language Extension

„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

„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

„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.

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

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);
  }
}
```

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.

Meta

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;
}
```

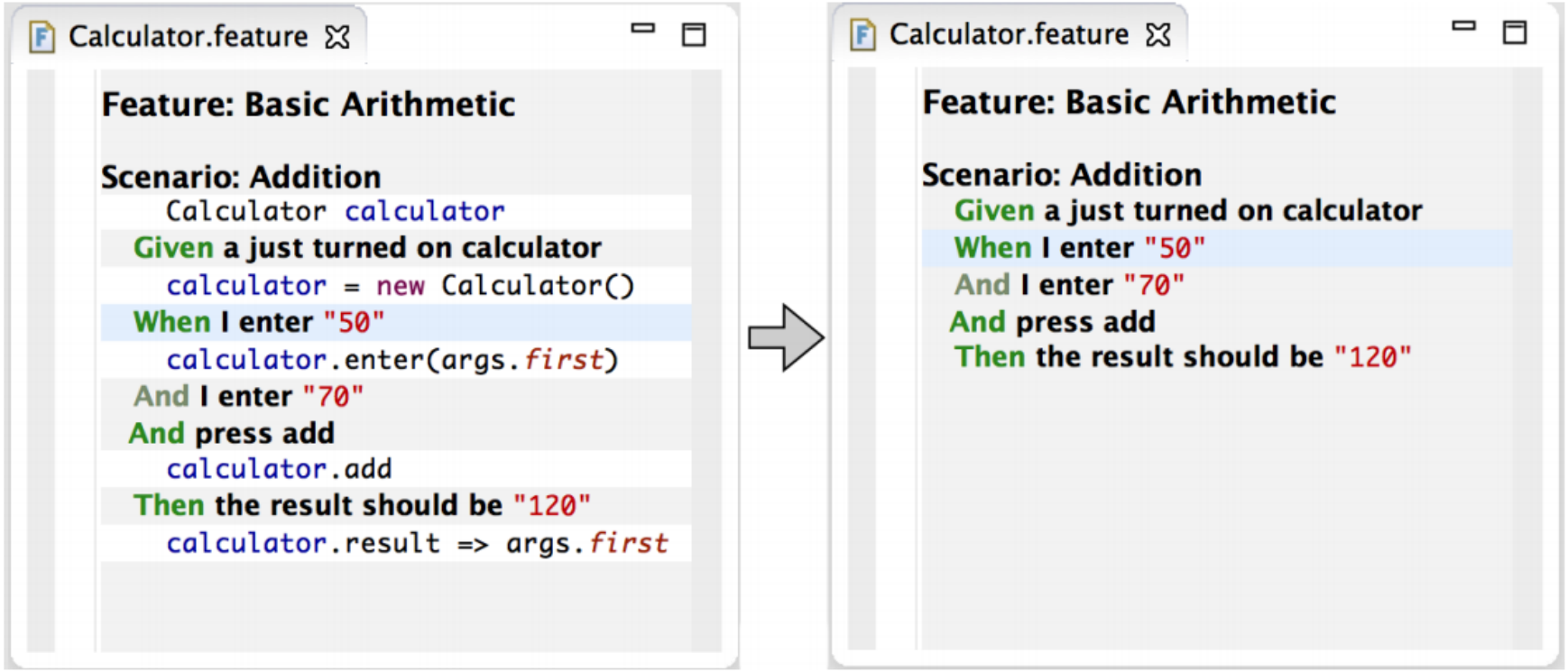
```
node<GlobalVariableDeclaration> gvd = build GlobalVariableDeclaration
  name = name
  type = #(type)
  preventNameMangling = true
  extern = isExtern
;
```

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

Extending the Tool

Xtext

… using Platform APIs



Jnario custom folding editor.

The Web

Web-based DSL Editing

Challenges

(Re-)Implementing LWB
in the Browser

Browser-Based Parsing

Tier-Splitting

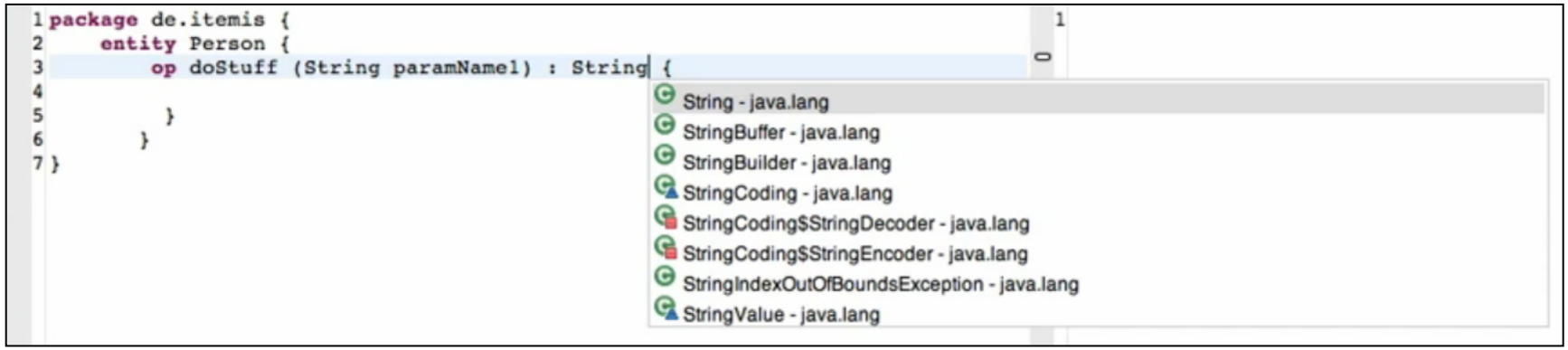
Multi-User (real-time?)

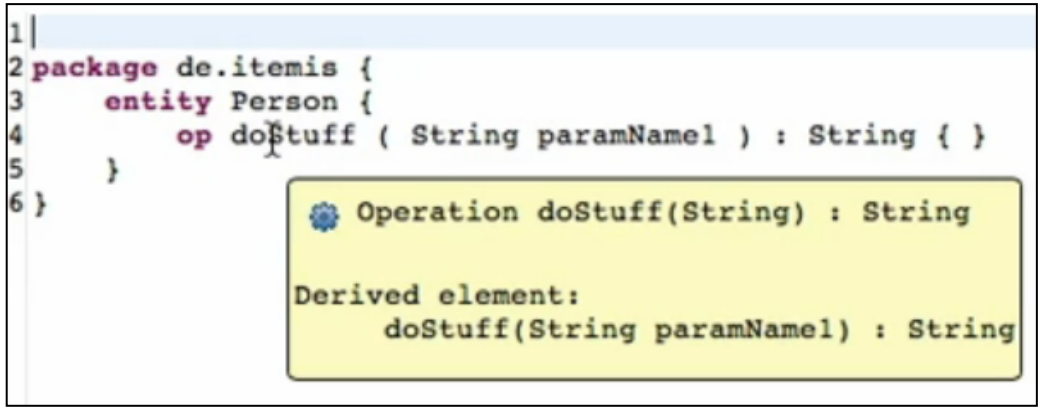
A Research Agenda:

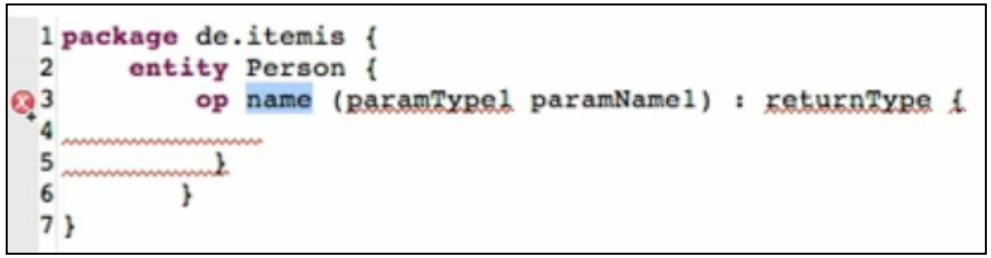
http://researchr.org/publication/KatsVKV2012

Web-based DSL Editing

Xtext







Prototype

Based on Orion

An Eclipse-Instance
On server per client

Developed in a
research project

Web-based DSL Editing

MPS*

JetBrains/MPS

Generic Projectional Editor being developed; driven by Youtrack Workflows

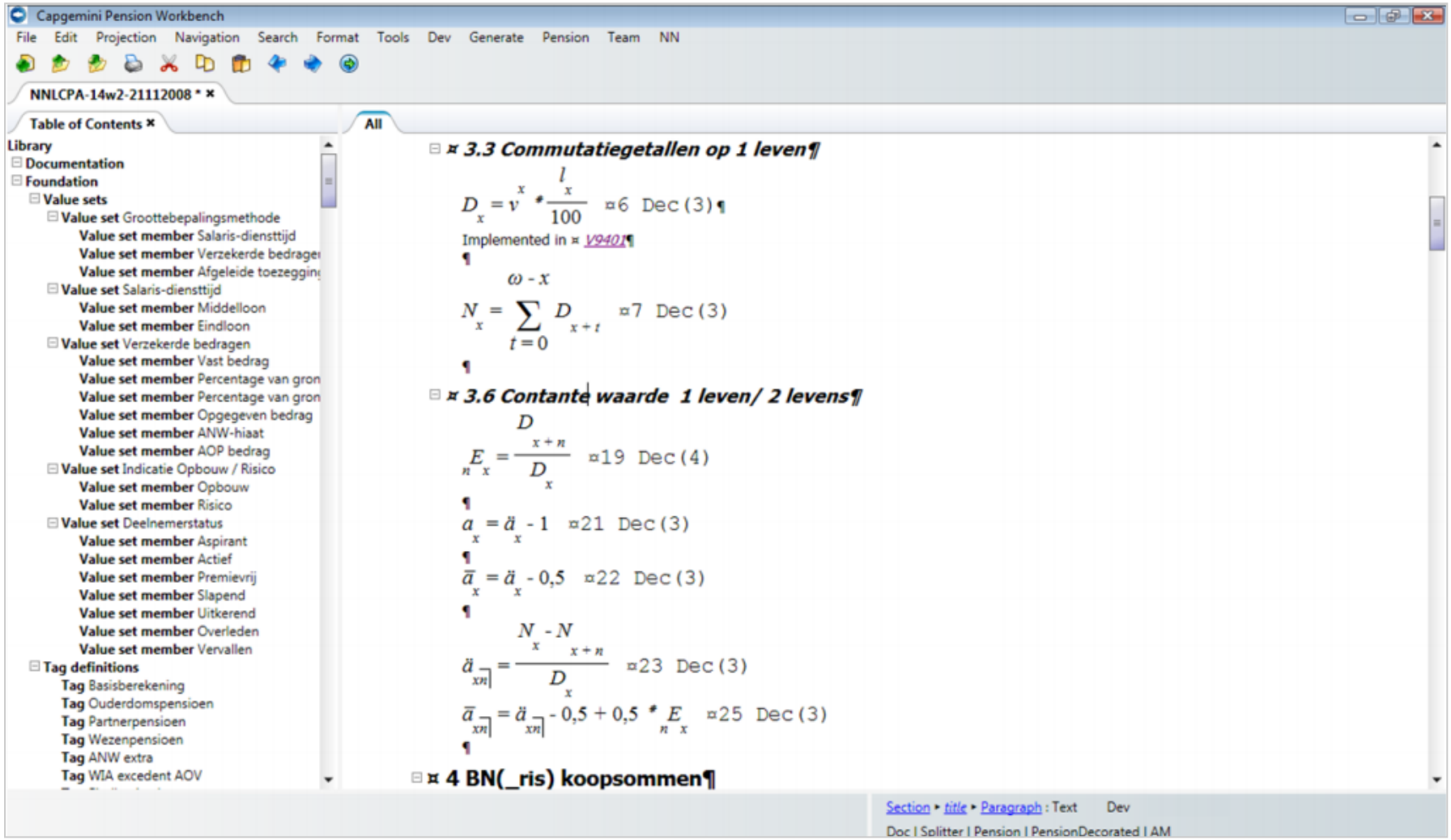
```
schedule rule Notify assignee about overdue issues

daily 10:00:00 [now > issue.dueDate] {
  if (!isResolved()) {
    var user;
    if (issue.assignee == null) {
      user = issue.project.leader;
    } else {
      user = issue.assignee;
    }
    user.notify("Issue is overdue", "Please look at the issue" + getId());
  }
}
```

Java-Scipt based (completely in Browser)

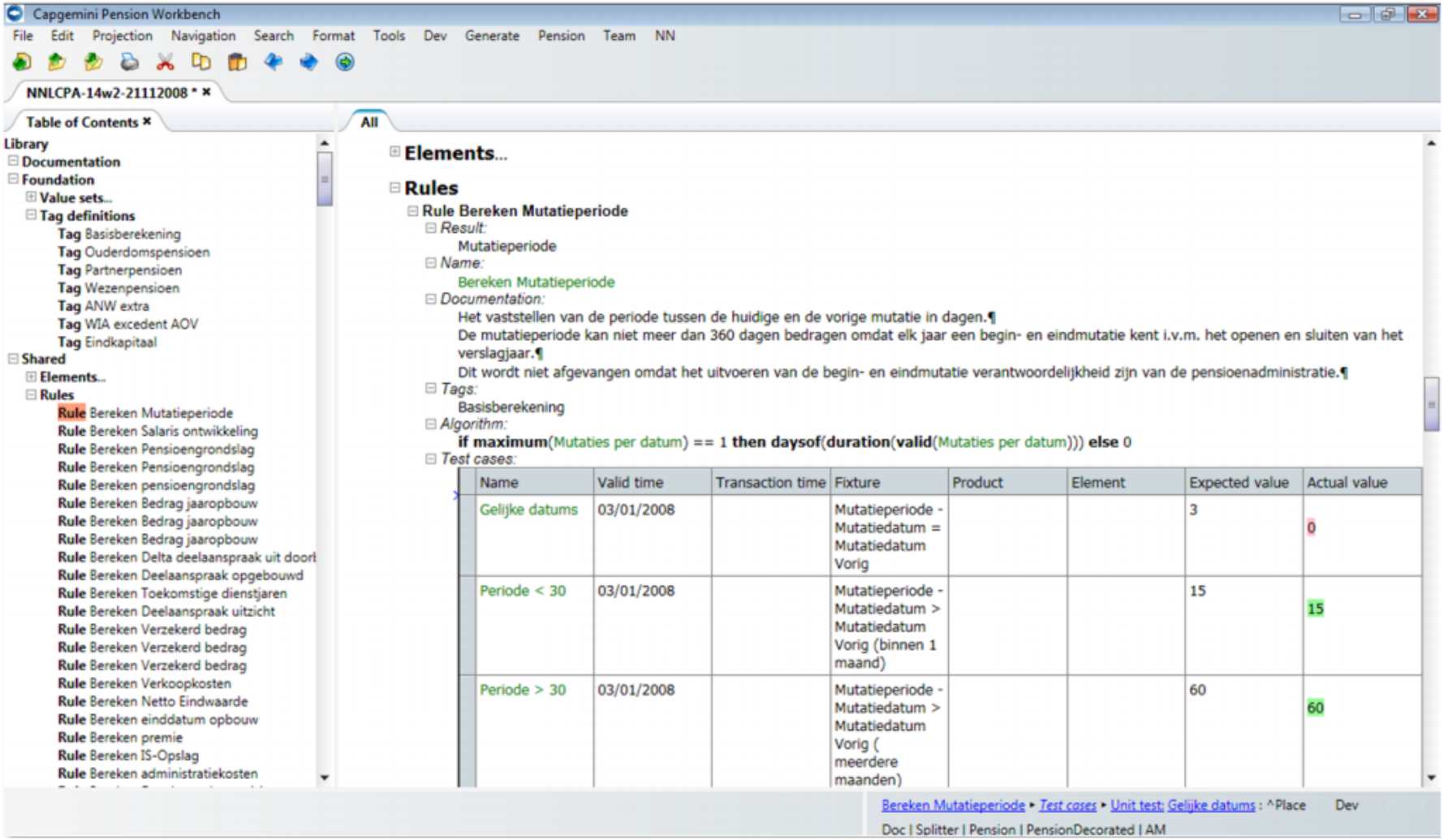
Will be made compatible with MPS language definitions

Other Tools

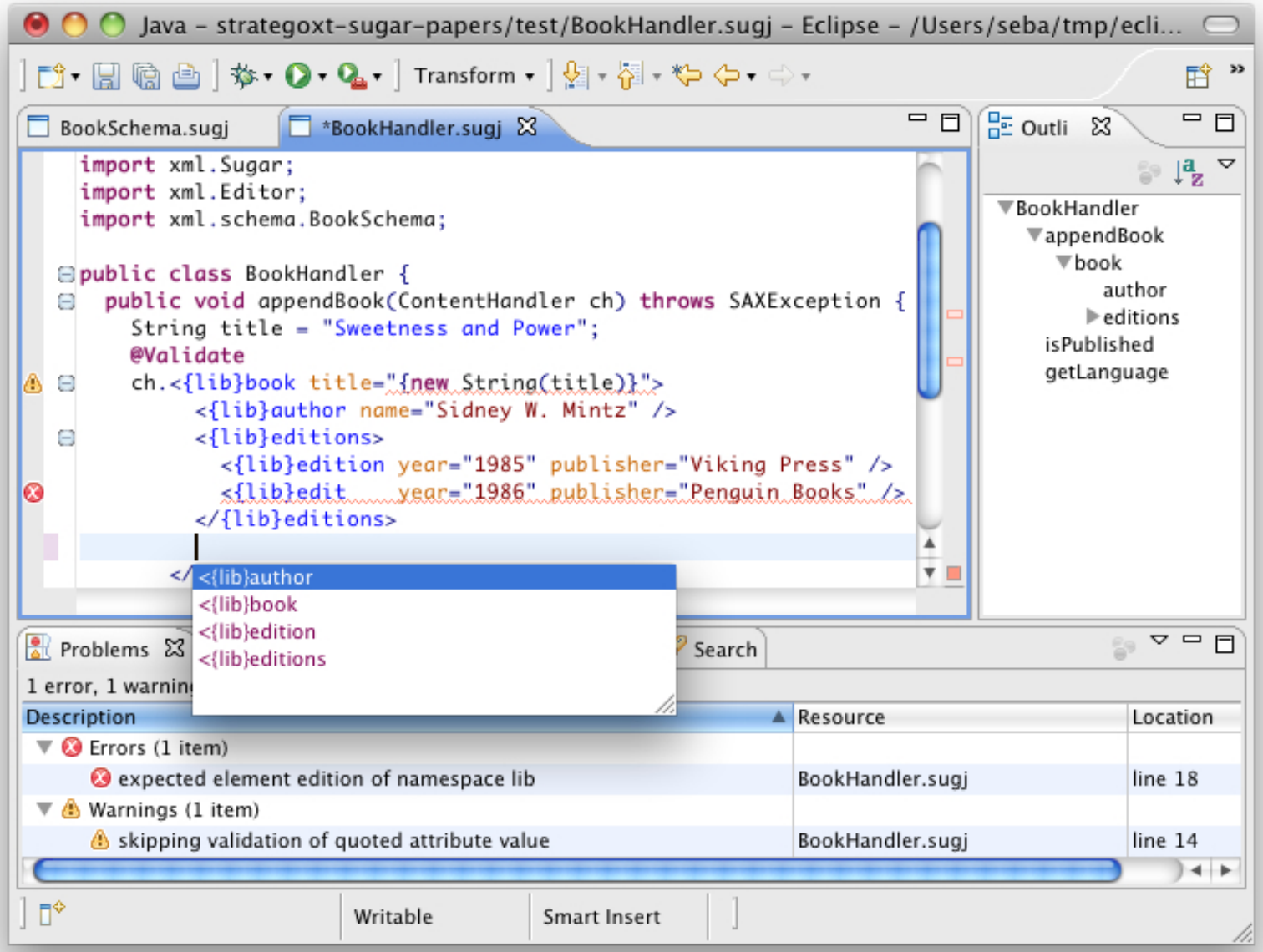
Intentional



Intentional



SugarJ (Uni Marburg)



Language Workbenches

Language Workbench
Comp {etition | arison}

Dozens of LWBs compared based on the same example languages

http://languageworkbenches.net

The End.

voelter.de | dslbook.org

@markusvoelter
+Markus Voelter