Lessons Learned about Language Engineering from the Development of mbeddr

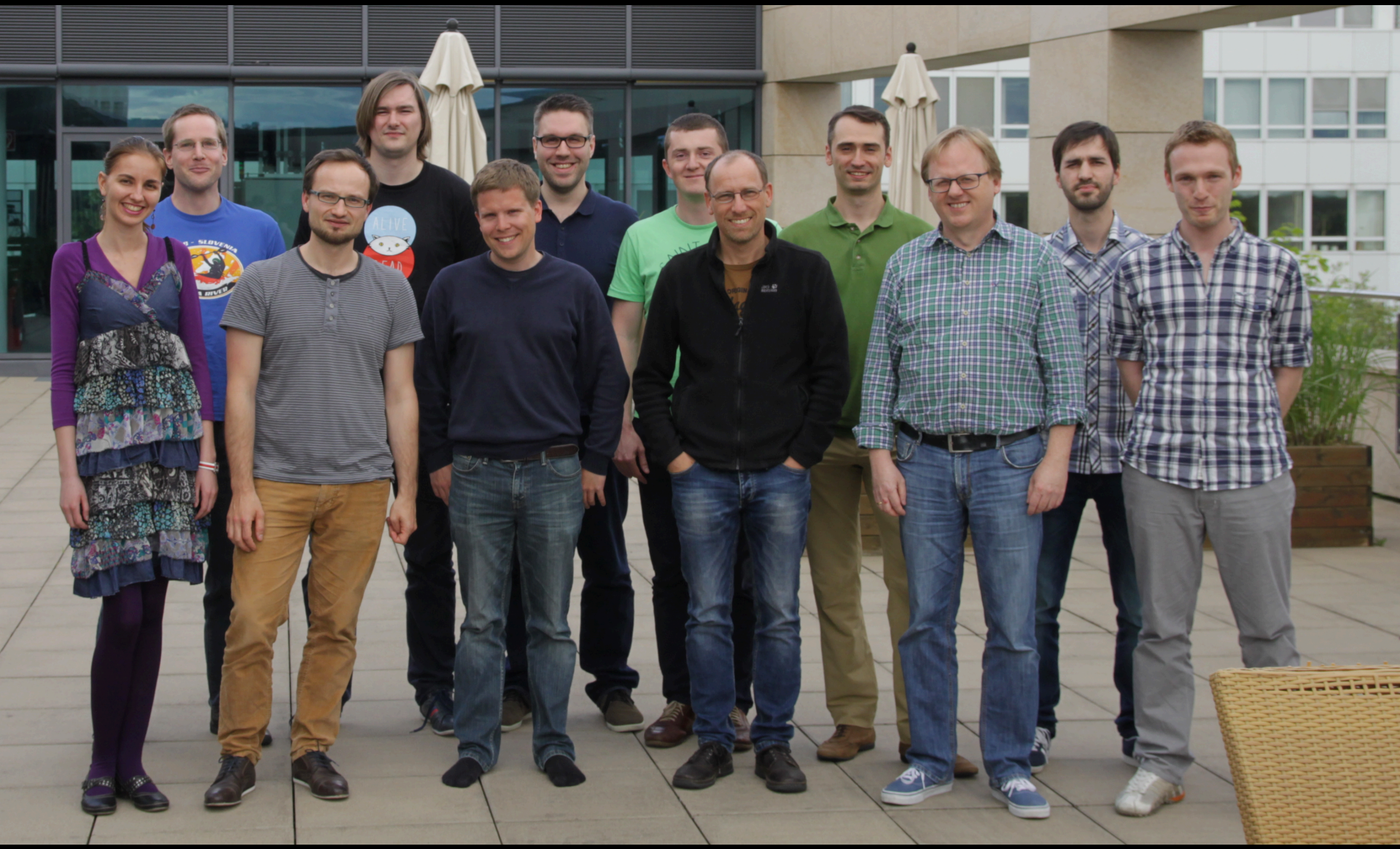
Markus Völter

voelter@acm.org
www.voelter.de
@markusvoelter

voelter { ingenieurbüro für softwaretechnologie // itemis

First:

It‘s not just me!



DOMAINS

- Health & Medical
- Automotive
- Aerospace
- Robotics
- Finance
- Embedded
- Science
- Government

Software Engineering

Requirements Engineering

Specification and Verification

Architecture

Safety and Security

Implementation and Test

Language Engineering

Mixed Notations and End User Programming

Informal -> Semiformal -> Formal

Languages + Verification

Optmizations, Performance and Concurrency

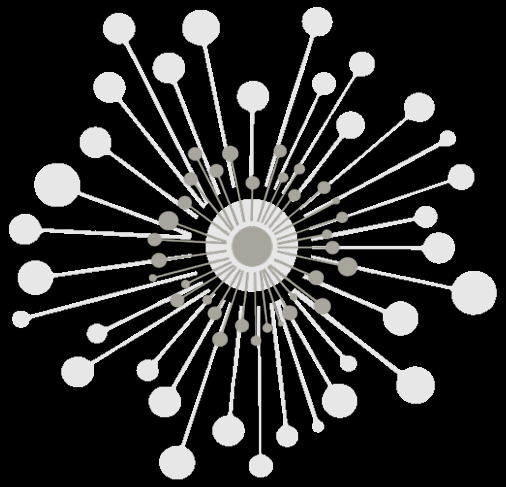
Fundamentals: Editors, Type Systems, Trafos

User-Friendly IDEs, Tools

Methodology

Lessons Learned about Language Engineering from the Development of mbeddr

1

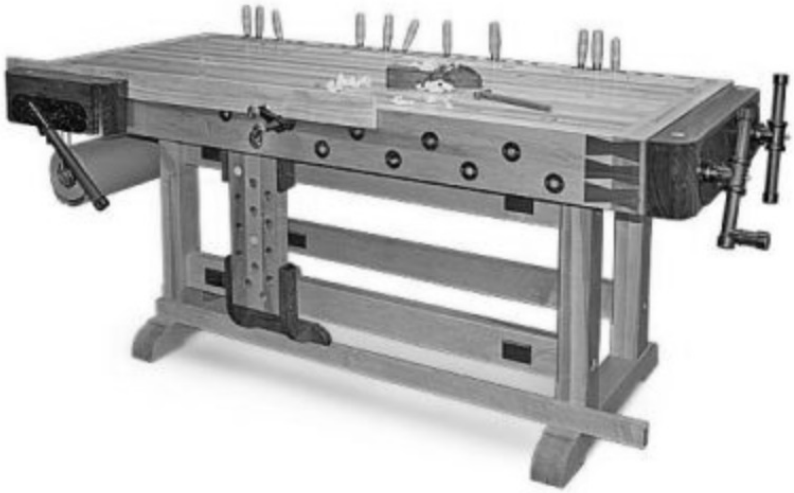


Background: MPS

Language Workbench

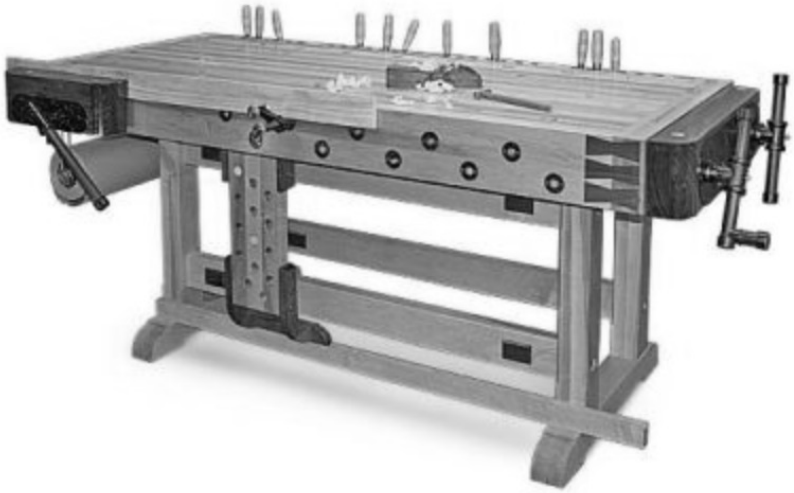
(Martin Fowler, 2004)

Freely
define
languages and
integrate
them



Language Workbench

(Martin Fowler, 2004)



powerful

editing

testing

refactoring

debugging

groupware

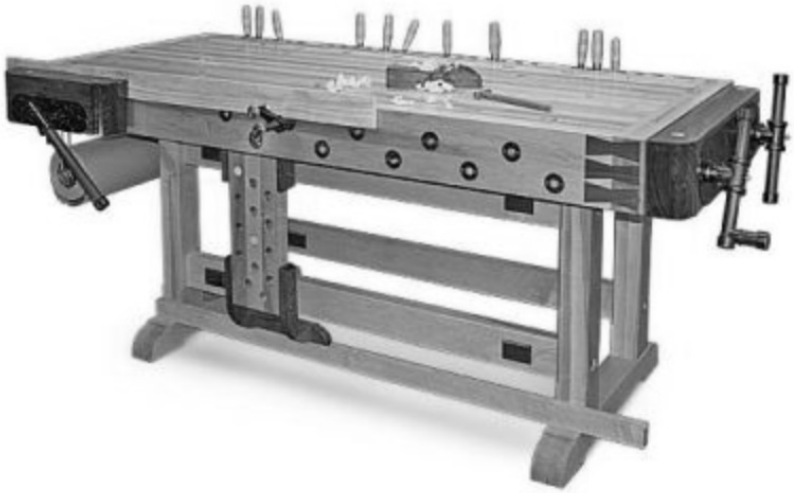
language definition

implies

IDE definition

Language Workbench

(Martin Fowler, 2004)



support for „classical“ programming + „classical“ and modeling

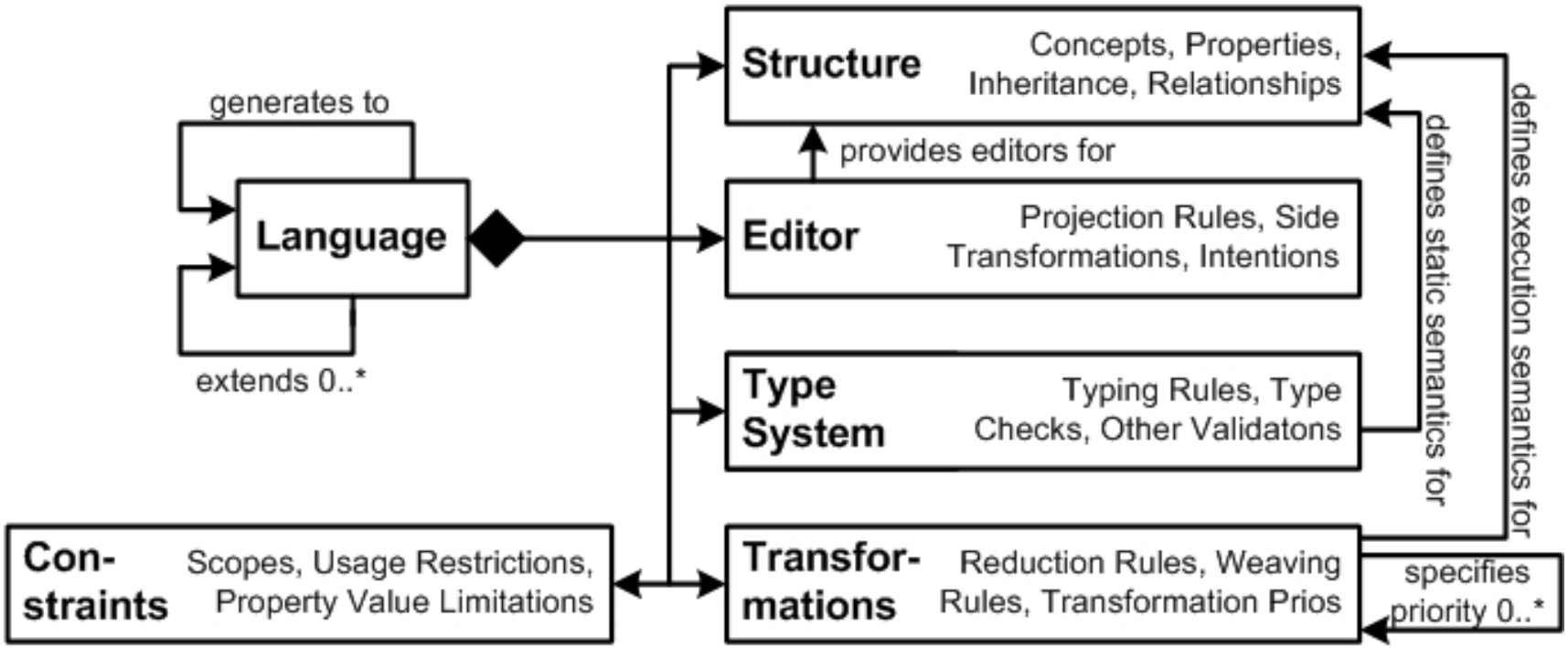
There‘s no difference!

A Language Workbench –

a tool for defining, composing and using ecosystems of languages.

[Language Workbench]

Comprehensive Support for many aspects of Language Definition.



+ Refactorings, Find Usages, Syntax Coloring, Debugging, ...

SIEMENS

Mercedes-Benz

fortiss

Continental

ZURICH

OHB

BOSCH

DYNAGEN
power controls you can trust

workday

HUAWEI

Fraunhofer
AISEC

Whirlpool
CORPORATION

UNIVERSITY OF
WATERLOO

McGill

VOLUNTIS
Connected Therapeutics

ERICSSON

SIOUX

Belastingdienst

i2S
INSURANCE KNOWLEDGE

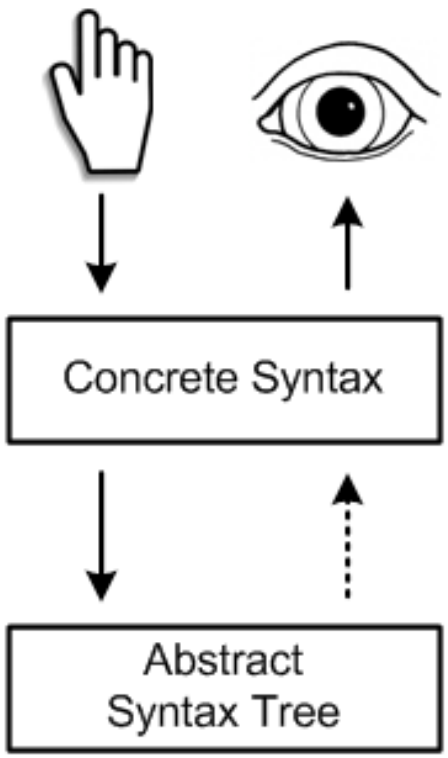
itemis

B/S/H/

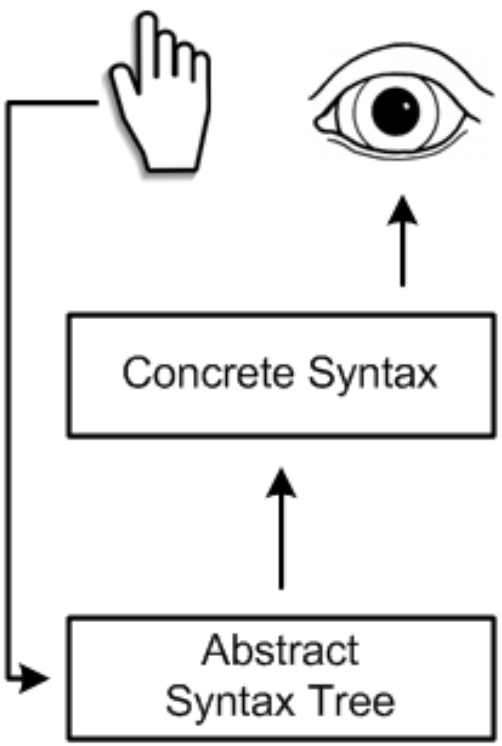
Audi

[Projectional Editing]

Parsing



Projectional Editing



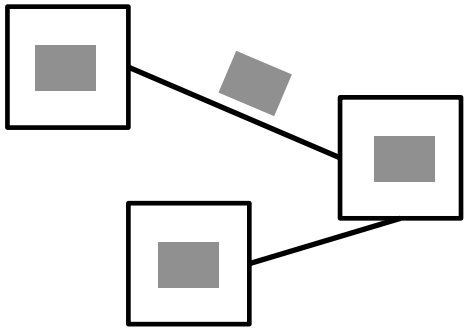
[Projectional Editing]
Syntactic Flexibility

Regular Code/Text

Mathematical

Tables

Graphical



[Projectional Editing]
Syntactic Flexibility

Regular Code/Text

```
//[ A documentation comment with references ]
  [ to @arg(data) and @arg(dataLen)        ]
void aSummingFunction(int8[] data, int8 dataLen) {
  int16 sum;
  for (int8 i = 0; i < dataLen; i++) {
    sum += data[i];
  } for
} aSummingFunction (function)
```

Mathematical

```
double midnight2(int32 a, int32 b, int32 c) {
```

$$\text{return}\ \frac{-b + \sqrt{b^2 - \sum_{i=1}^{4} a * c}}{2 * a};$$

```
} midnight2 (function)
```

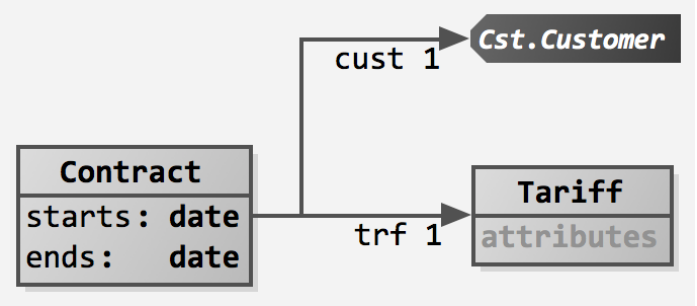
Tables

```
int16 decide(int8 spd, int8 alt) {
  return
```

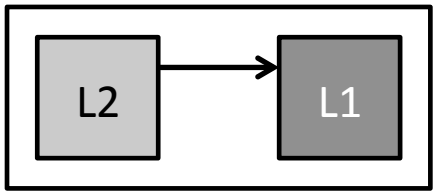
	spd > 0	spd > 100
alt < 0	1	1
alt == 0	10	20
alt > 0	30	40
alt > 100	50	60

```
otherwise 0;
} decide (function)
```

Graphical

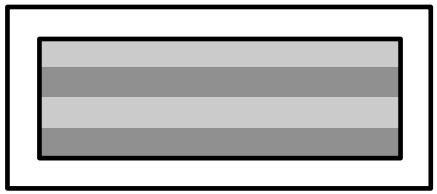


[Projectional Editing]
Language Composition



Separate Files

Type System
Transformation
Constraints



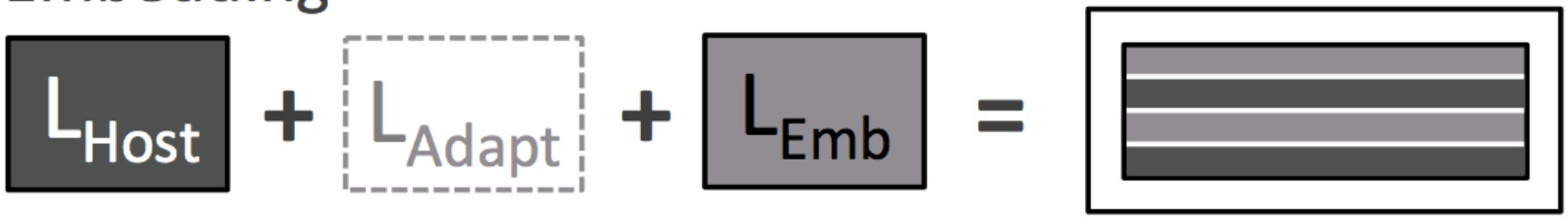
In One File

Type System
Transformation
Constraints
Syntax
IDE

[Projectional Editing]

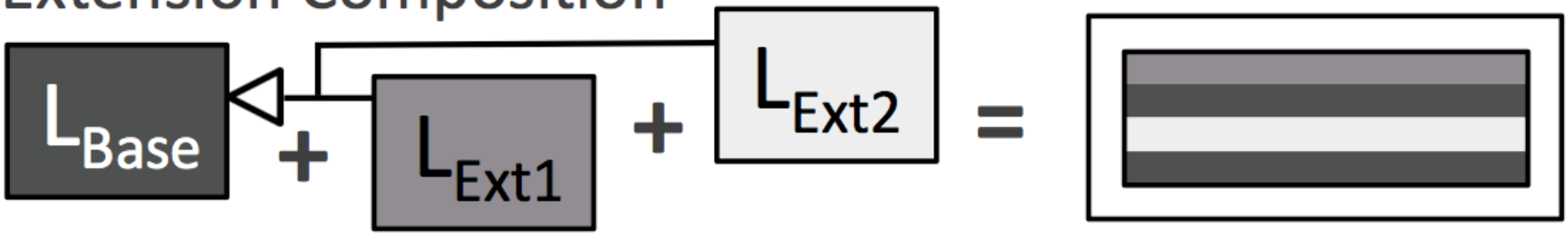
Language Composition

Embedding



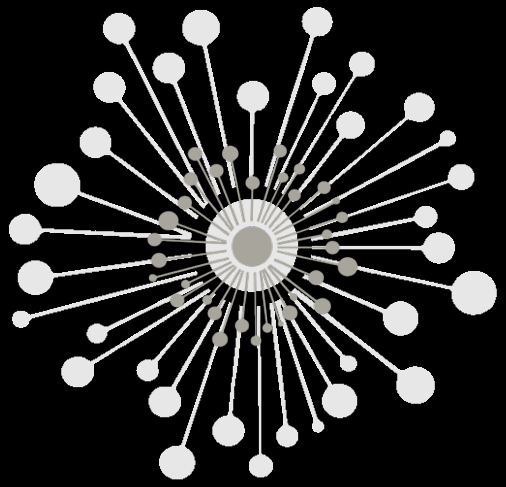
Extension

Extension Composition



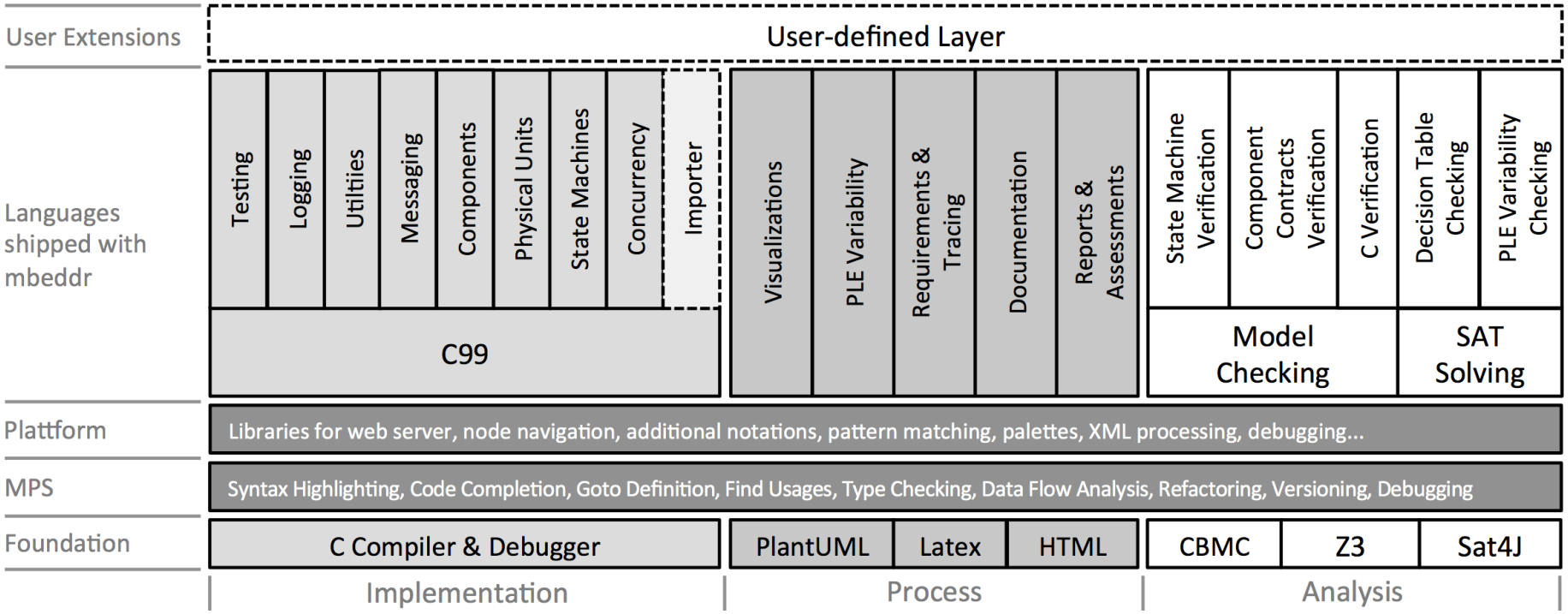
Lessons Learned about Language Engineering from the Development of mbeddr

2



Background: mbeddr

An extensible set of integrated languages for embedded software engineering.



Composable extensions, Diverse notations

DLT645ProtocolImpl

Events

```
dataReceived(uint8 data)
idle        [data == DLT645_FRONT_LEADING_BYTE] -> awaking
awaking     [data == DLT645_FRAME_STARTFLAG] -> recvFrame
              {
                send startFrame(idx, data);
                idx++;
              }
recvFrame   [idx < length - 1 && idx != DLT645_HEADER_DATALENGTH_OFFSET]
              idx++;
            }
            [idx == DLT645_HEADER_DATALENGTH_OFFSET] -> rec
              {
```

States

Error: type boolean is not a subtype of uint8

idx ^DLT645ProtocolImpl.DLT645ProtocolImpl.DLT645RxFrameParser.DLT645RxFrameParser.idx (StatemachineVar
rxQueueHeadIdx ^DLT645ProtocolImpl.DLT645ProtocolImpl.rxQue
rxQueueTailIdx ^DLT645ProtocolImpl.DLT645ProtocolImpl.rxQue

UnitDeclarations

```
section AngleUnits {
  exported unit ° := <no spec> f
  exported unit rad := <no spec>
  exported unit rotation := <no

  exported #constant FULL_ROTAT

  exported eager conversion ° ->
   val as <no type> -> val/°/ *
```

```
exported cs interface IGraphPlotter {
  handle newCurve(Color color, string const label)
  void addPoint(handle hCurve, double x, double y)
  void plot(string const title, handle hCurve)
  void plotAll(string const title)
  void clear(handle hCurve)
  void clearAll()
  void deleteCurve(handle hCurve)
  void dispose()
}
```

IGraphPlotter

DCFilterTestHarness

internal instances {

dcFilterTestRunner

dcFilterTestRunnerInst: DCFilterTestRunnerImpl

IMetrologyRawSignalSimulator

IMetrologyRawSignalData

metrologyRawSignalSimulatorIn MetrologyRawSignalSimulatorIm

IMetrologyRawSignalHandler

An Industrial Case Study

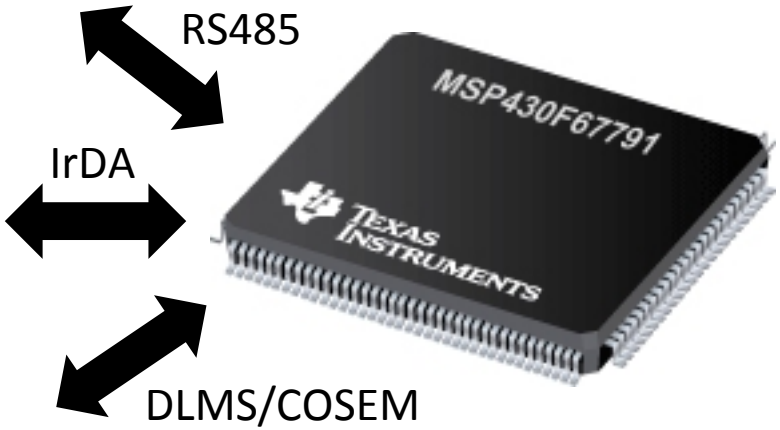
Smart Meter

Measures Voltage and Current
Computes Derived Values
Shows Data on LCD Display
Communicates through Networks

Precision is critical for Certification.
Evolvability is critical for it to be a viable business.

Developed with mbeddr, a set of domain-Specific extensions to C, plus an IDE.

Hardware Architecture



MQTT

UART

Application Logic

MSP430 F67791

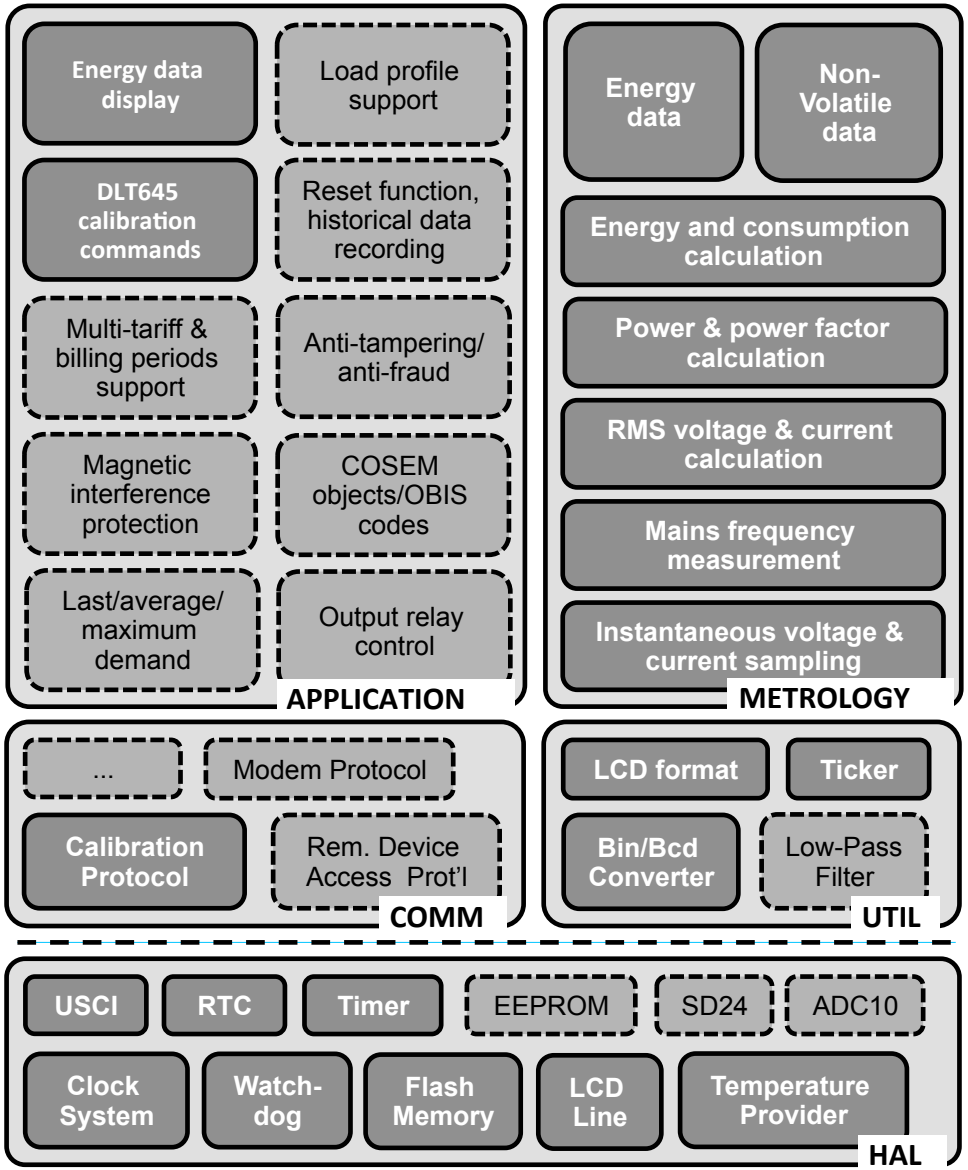
25 MHz
256K Flash ROM
32K RAM

Metrology

MSP430 F6736

25 MHz
128K Flash ROM
8K RAM

Software Architecture



No RTOS

Interrupt-Driven

One-Threaded Programming

Required Precision leads to 4096 Hz Sampling Rate

Interrupt-Triggered:

Measurement

Foreground Tasks:

App Logic, RTC

Size of the System

Criterion	Common	Metro	App	Total
# of Files	134	101	105	340
Total LOC	8,209	10,447	10,908	29,564
Code LOC	4,397	5,900	5,510	15,807
Comment LOC	950	2,402	2,620	5,972
Whitespace LOC	2,852	2,145	2,778	7,775

Common code runs on both processors, **Metro** runs on the metrology processor and **App** runs on the application / communication processor.

+ roughly the same amount again for **tests**.

ca. 30,000 SLOC

Use of Extensions

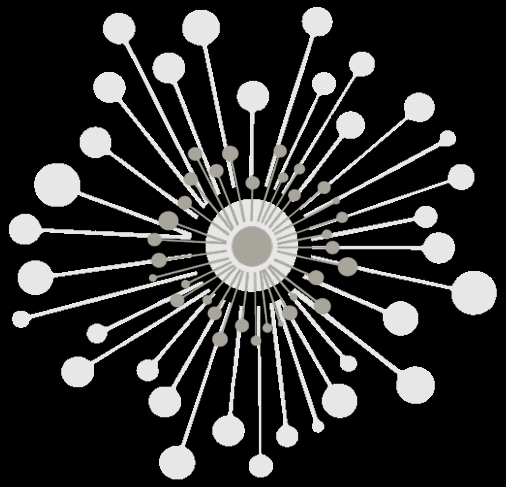
Category	Concept	Count
Chunks (≈ Files)	Implementation Modules	382
	Other (Req, Units, etc.)	46
C Constructs	Functions	310
	Structs / Members	144 / 270
	Enums / Literals	150 / 1,211
	Global Variables	334
	Constants	8,500
Components	Interfaces / Operations	80 / 197
	Atomic Components	140
	Ports / Runnables	630 / 640
	Parameters / Values	84 / 324
	Composite Components	27
	Component Config Code	1,222
State Machines	Machines	2
	States/Transitions/Actions	14 / 17 / 23
Physical Units	Unit Declarations	122
	Conversion Rules	181
	Types / Literals with Units	593 / 1,294

Category	Concept	Count
Product Line Variability	Feature Models / Features	4 / 18
	Configuration Models	10
	Presence Condition	117
Custom Extensions	Register Definition	387
	Interrupt Definitions	21
	Protocol Messages	42
Statements	Statements total	16,840
	Statements in components	6,812
	Statements in test cases	5,802
	Statements in functions	3,636
Testing	Test Cases / Suites	107 / 35
	Test-Specific Components	56
	Stub / Mock Components	9 / 8
	`assert` Statements	2,408

All mbeddr C extensions used a lot.
Some extensions built specifically for SM.

Lessons Learned about Language Engineering from the Development of mbeddr

3



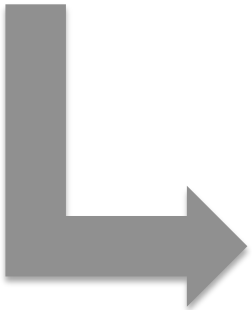
A 5-minute MPS tutorial

C Extension to dequeue a value

```
shared int64Q data;

cyclic task sumUp {
  dequeue if available from data -> val {
    sum += val;
  }
}
```

generated to



```
uint64 ___val = 0;
boolean ___taken = false;
atomic <data/readWrite> {
  if (!data.isEmpty) {
    ___val = data.take;
    ___taken = true;
  }
}
if (___taken) {
  sum += ___val;
}
```

Structure

```
concept ValExpr extends Expression
  alias val
```

```
concept DequeueStatement extends Statement
                         implements IAtomic
  alias: dequeue
  children:
    queue : GlobalVarRef  [1]
    body  : StatementList [1]
```

Editor + Actions

```
<default> editor for concept DequeueStatement
  node cell layout:
    [- dequeue from % queue % -> val % body % -]
```

Editor actions to implement editor behavior. Previously manually written, now generated from Grammar Cells. See SLE Paper :-)

Constraints

Tree Constraints:

```
can be child
  (childConcept, node, parentNode)->boolean {
    parentNode.ancestor<DequeueStatement> != null;
  }
```

Scopes:

```
concept GlobalVarRef extends Expression
  references:
    var : GlobalVarDecl[1]
```

```
link var scope:
  (enclosingNode, pos) -> nsequence<GlobalVarDecl> {
    enclosingNode.ancestor<Module>.contents.
          ofConcept<GlobalVarDecl>;
  }
```

Type System

Checking Rules:

```
checking rule for DequeueStatement {
  ensure node.queue.var.type.isInstanceOf(QueueType)
    else error "global variable not a queue"
         on dqs.queue;
  ensure node.queue.var.@shared != null
    else error "queue must be shared"
         on dqs.queue;
}
```

Typing Rules:

```
typing rule typeof_QueueValExpr for ValExpr {
  node<DequeueStatement> dqs =
            node.ancestor<DequeueStatement>;
  node<QueueType> qt = dqs.queue.type : QueueType;
  typeof(node) :==: typeof(qt.queue.elementType);
}
```

Behavior

```
concept behavior IAtomic {
  public abstract boolean getsReadLockFor(
         node<GlobalVarRef> r);
  public abstract boolean getsWriteLockFor(
         node<GlobalVarRef> r);
}
```

```
public boolean getsReadLockFor(node<GlobalVarRef> r) {
  return r.var == this.queue.var;
}
```

```
checking rule check_Lock for GlobalVarRef {
  node<IAtomic> a = node.ancestor<IAtomic>;
  if (a == null || !a.providesReadLockFor(node))
    error "global variables must be locked" -> gvr;
}
```

Generators/Transformations

```
concept     DequeueStatement
inheritors  false
condition   <always>
```

-->

```
content node:
@shared
intqueue q;
```

```
node.queue.var.type : QueueType.queue.elementType;
```

```
<TF {
  $COPY_SRC$ [int8 ] ___val = 0;
  boolean ___taken = false;
  atomic <->$ [q]/readWrite> {
    if (!$COPY_SRC$ [q].isEmpty) {
      ___val = $COPY_SRC$ [q].take;
      ___taken = true;
    }
  }
  if (___taken) {
TF>
```

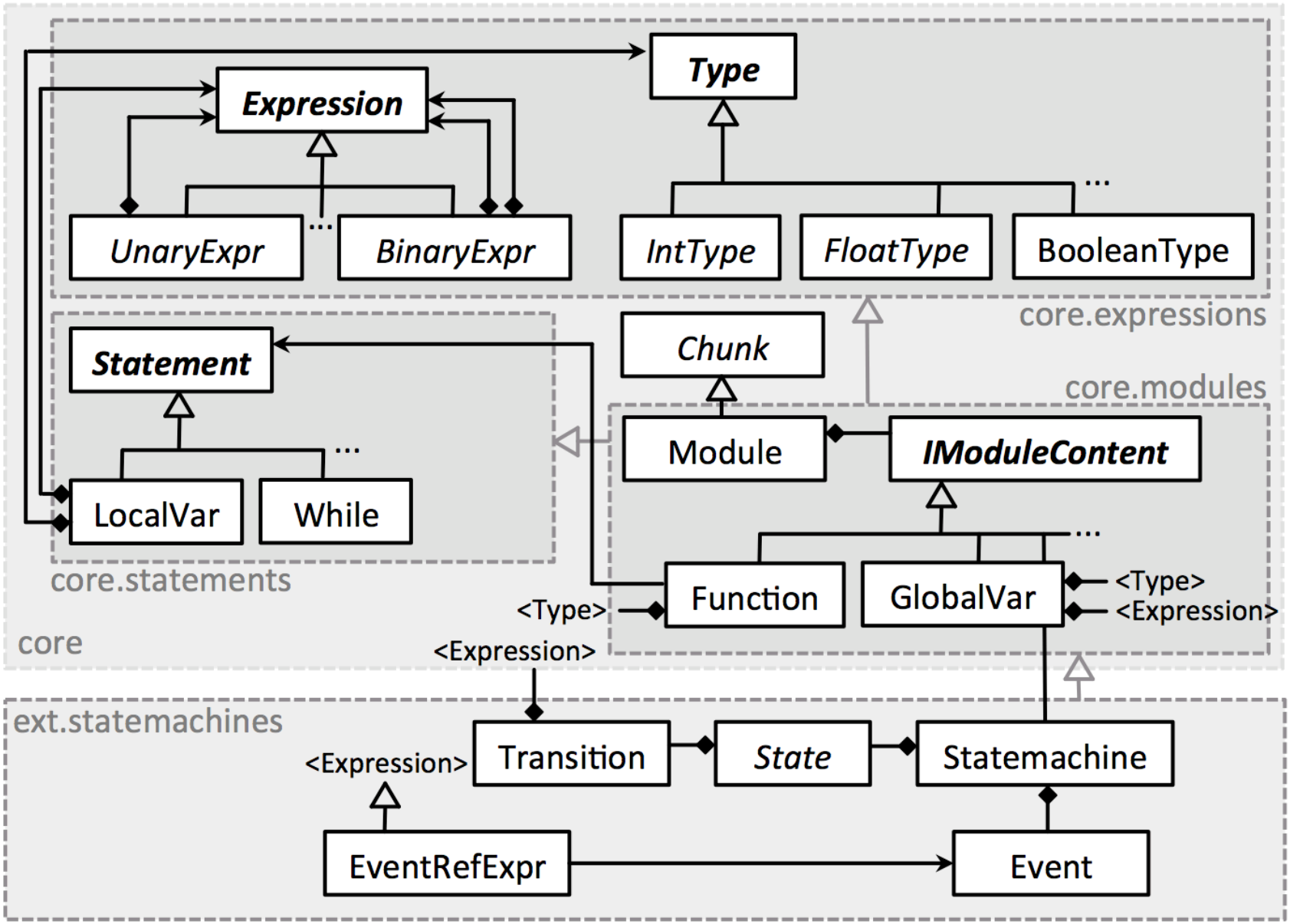
```
com.mbeddr.ext.concurrency.pthreads: main
                << com.mbeddr.core.utils: main
```

```
}
```

And many more...

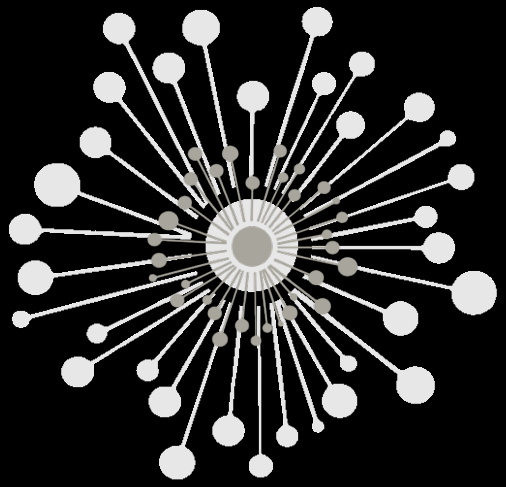
Intentions/Quick Fixes
Refactoring
Text Generators
Language Test Cases
Data Flow Graphs/Analyses
Debuggers
...

Modularity Fundamentally OO



Lessons Learned about Language Engineering from the Development of mbeddr

4



Project Stats

Total Effort

10

person years

Size of the Impl.

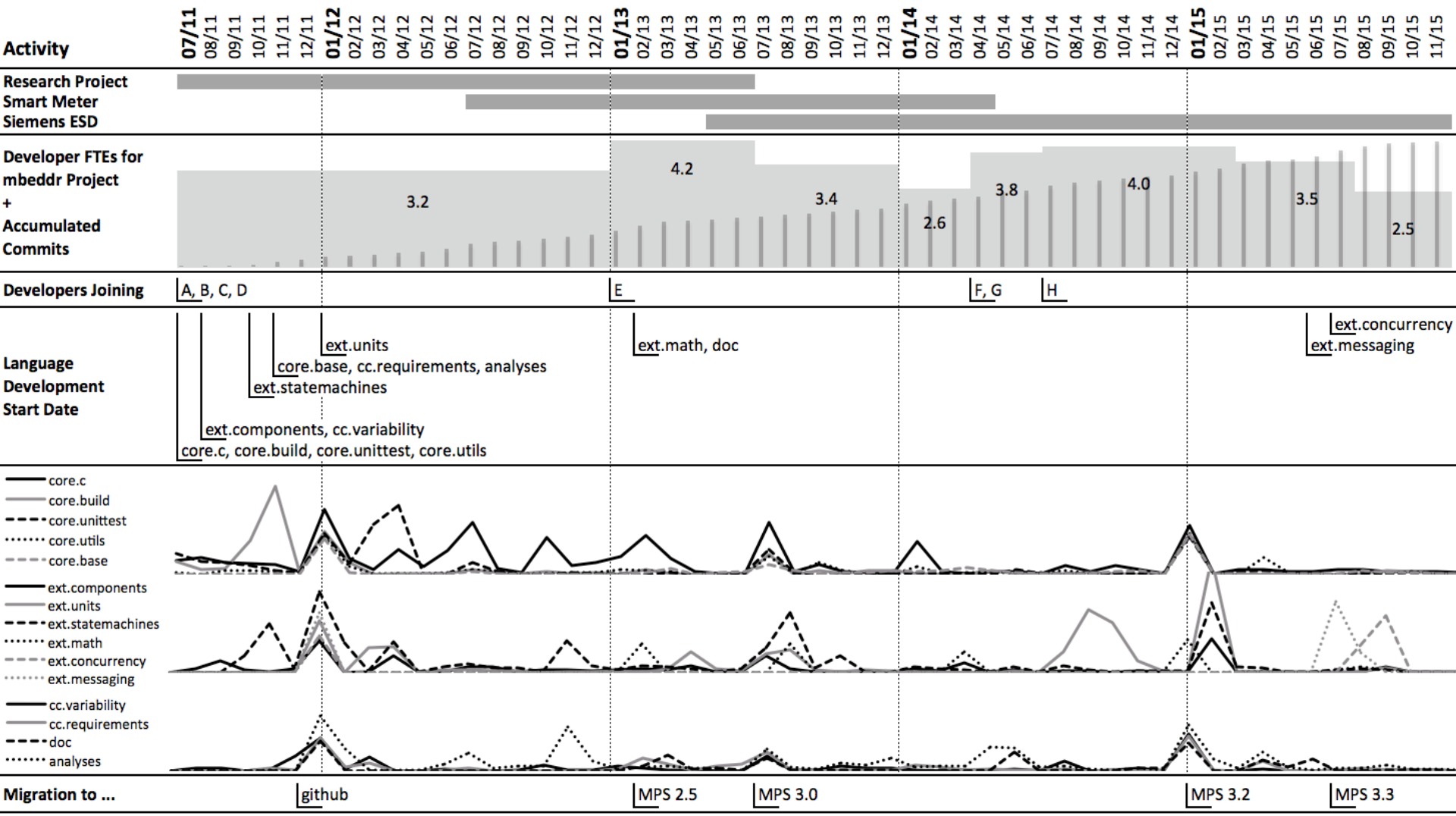
Part	Language	#L	#LC	#S	#C	#LOC
core	base	1	-	1	98	6,163
	C	8	-	7	354	20,114
	unittest	1	1	1	26	1,014
	utils	3	3	1	116	6,306
	build	2	-	1	48	2,080
ext	components	9	9	1	160	11,173
	state machines	1	1	1	48	3,194
	concurrency	3	3	0	65	3,078
	math	1	1	0	11	446
	messaging	2	2	0	60	2,151
	units	1	1	0	30	1,884
cc	variability	7	3	1	87	3,638
	reqmts/tracing	9	1	2	171	5,563
doc	doc	10	-	1	153	6,355
analyses	analysis	18	10	18	170	15,235
Total		81	34	38	1,597	88,394

ca. 90,000 SLOC

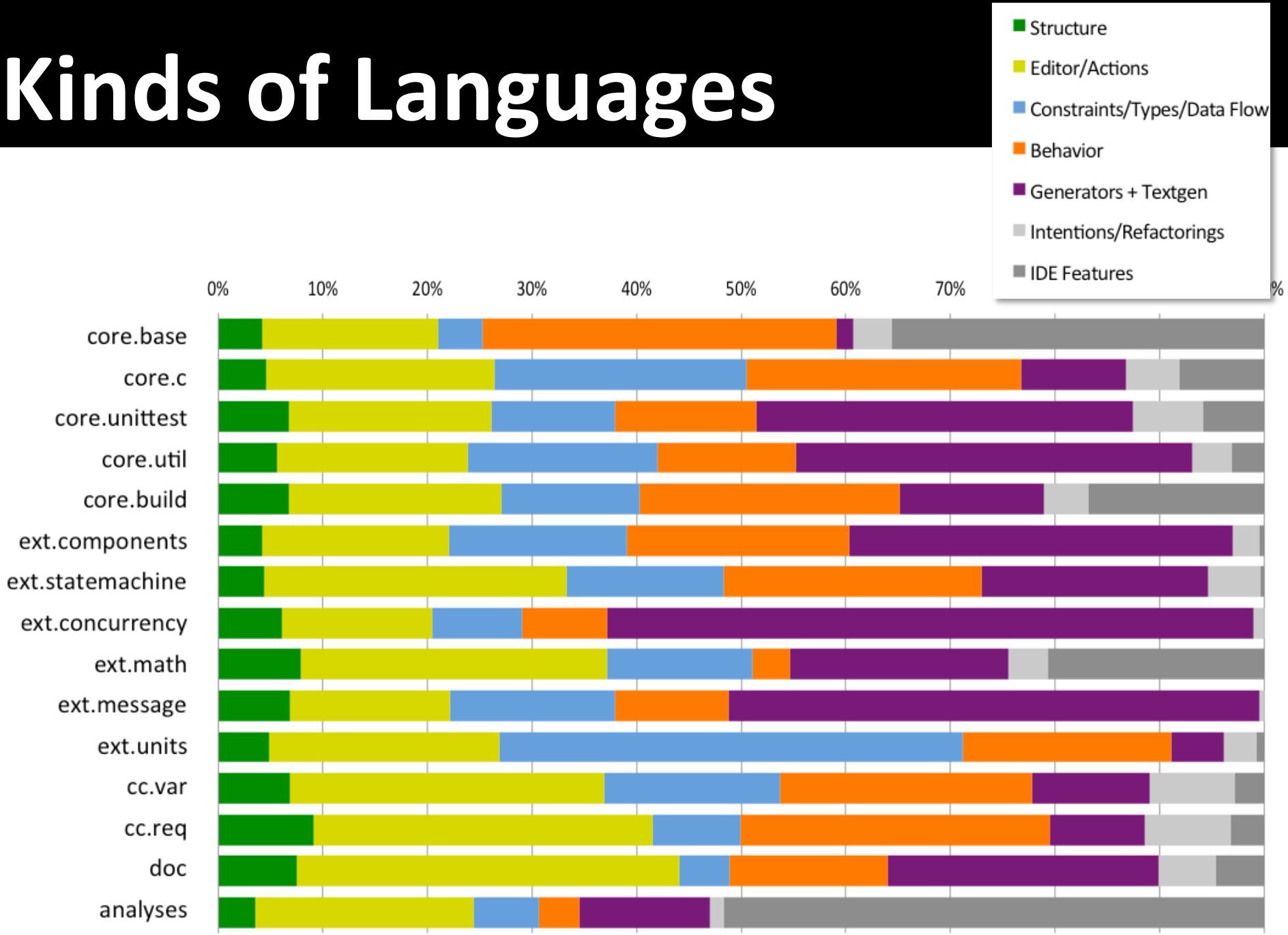
Tests vs. Implementation

	#Test Cases	#Assertions	#LOC
Type System Tests	324	1,381	8,243
Executable Tests	310	990	14,999
Other Tests	784	1,275	22,453
Total	1,418	3,646	45,695

Timeline of the Project

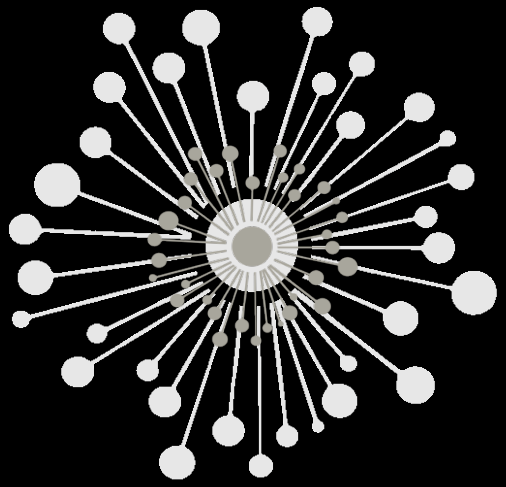


Kinds of Languages



Lessons Learned about Language Engineering from the Development of mbeddr

5



What can we learn?

RQ 1 Language Modualarity

RQ 2 Projectional Editing

RQ 3 Managing Complexity

RQ 4 Tool Scalability

RQ 5 Development Process

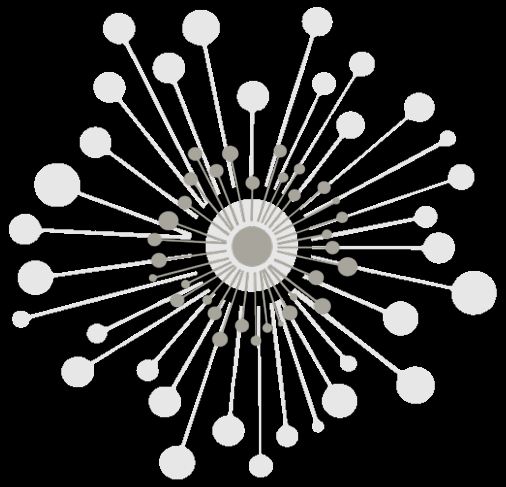
For a scientific treatment with all caveats and data and details, check out the paper:

http://bit.ly/2dvXnBF

Lessons Learned about Language Engineering from the Development of mbeddr

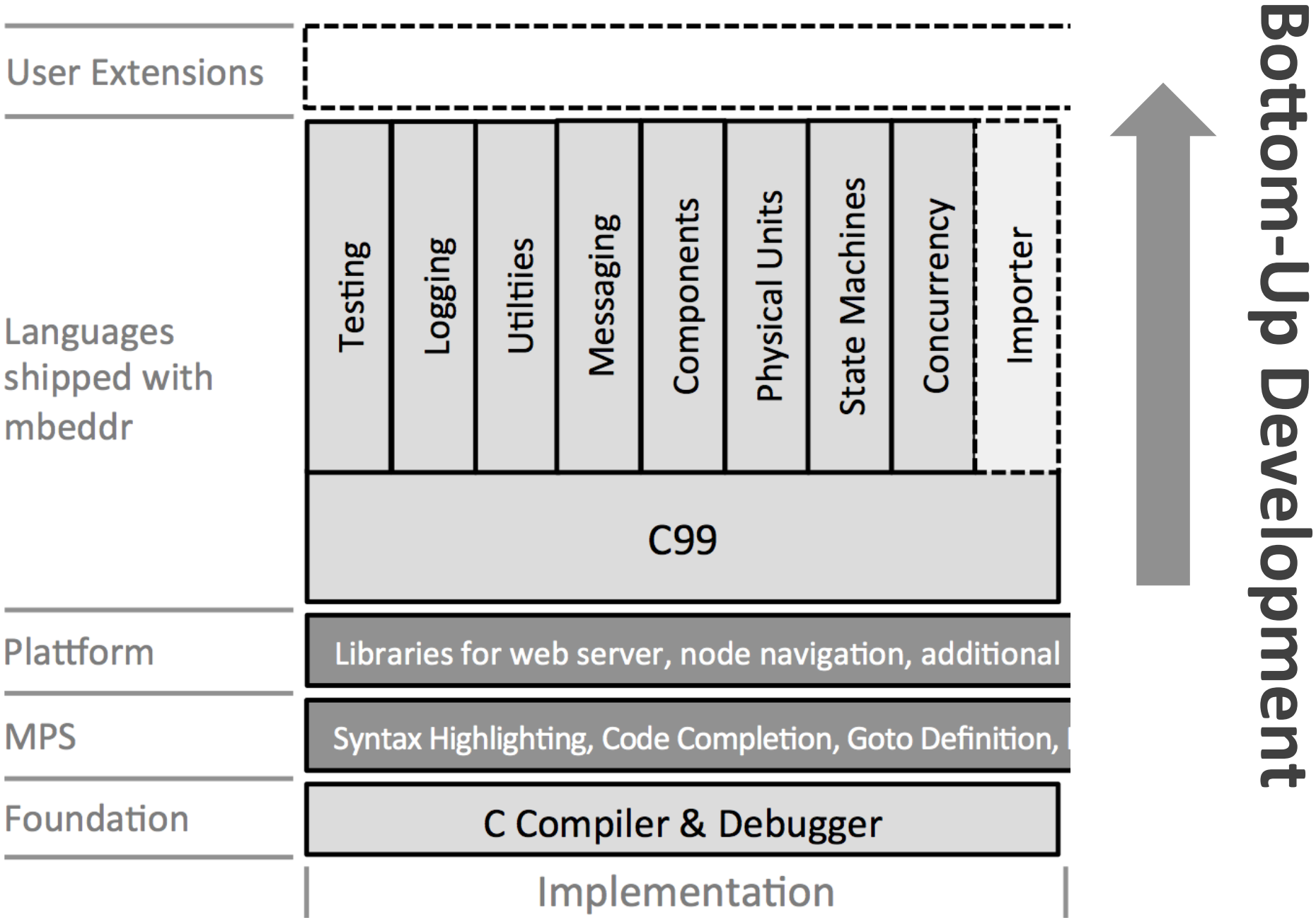
5

What can we learn?

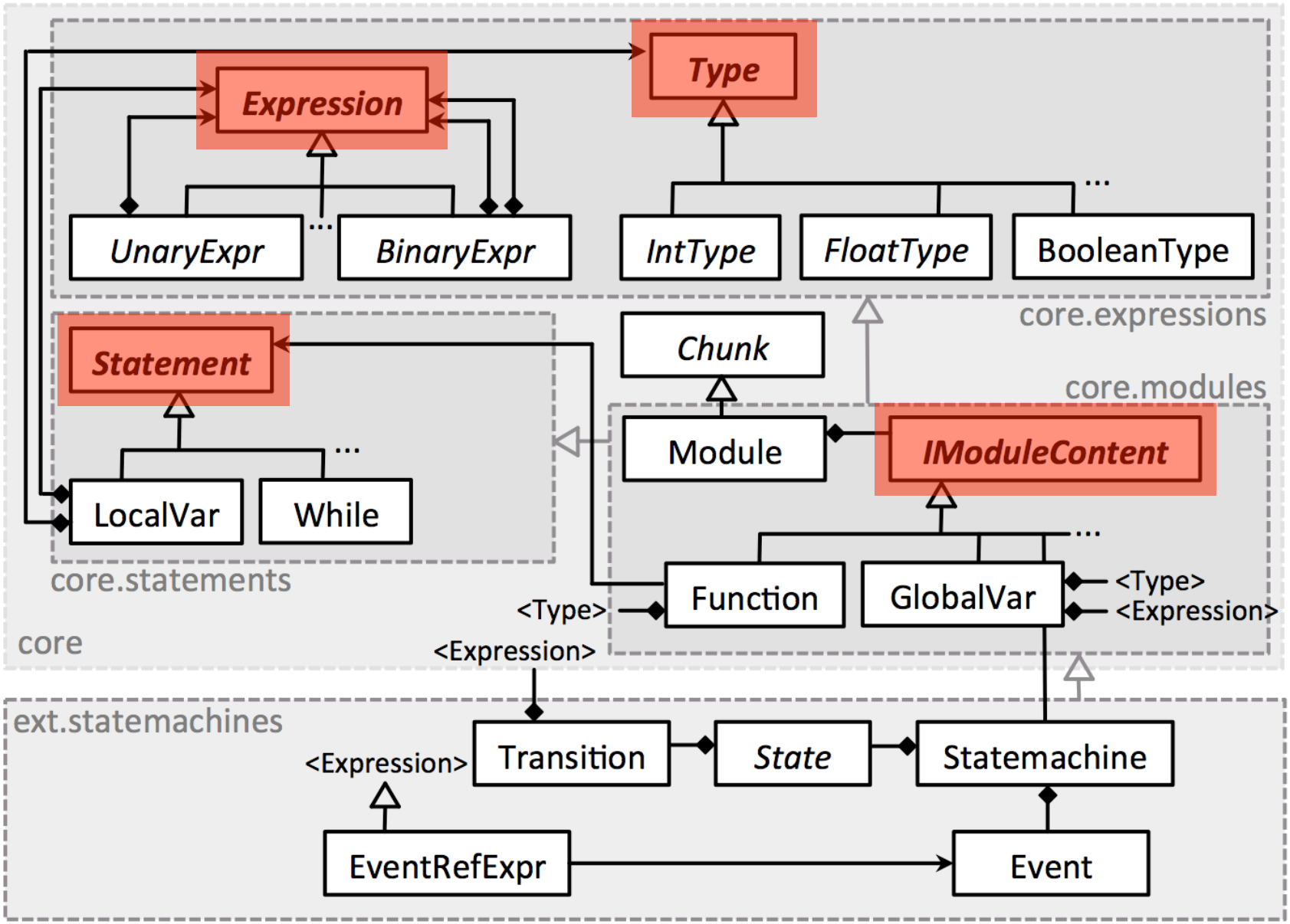


RQ 1

Language Modularity


User Extensions
Languages shipped with mbeddr
Testing
Logging
Utilities
Messaging
Components
Physical Units
State Machines
Concurrency
Importer
C99
Plattform
Libraries for web server, node navigation, additional
MPS
Syntax Highlighting, Code Completion, Goto Definition,
Foundation
C Compiler & Debugger
Implementation
Bottom-Up Development

Some Hooks are known a priori



Some had to be added a posteriori

IAssignmentLike: =, +=, *=, etc.

IFunctionLike: function, method

IBinaryLike: +-*/ but also . ?: !

IVoidTypeContainer: function, ftype

Conceptually this breaks modularity.

But it's like OO programming/frameworks.
So we deem it acceptable.

Overmodularization!

- core
  - solutions
  - com.mbeddr.core.base
  - com.mbeddr.core.checks
  - com.mbeddr.core.cinterpreter
  - com.mbeddr.core.dataflow.test
  - com.mbeddr.core.expressions
  - com.mbeddr.core.legacy
  - com.mbeddr.core.make
  - com.mbeddr.core.modules
  - com.mbeddr.core.modules.gen
  - com.mbeddr.core.pointers
  - com.mbeddr.core.statements
  - com.mbeddr.core.udt

We had a lot of circular dep's, solved via interfaces.

Not helpful.

Too much modularization is also not helpful.

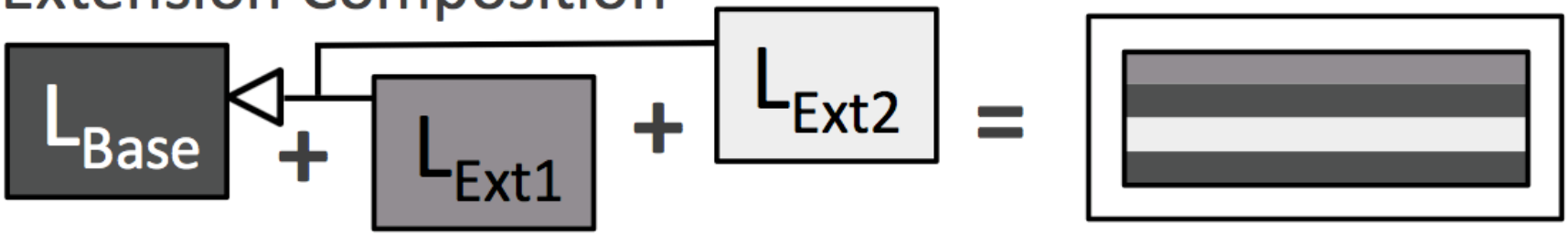
Refactoring tough because of MPS bugs in Refactorings.

We used mostly Extension (+EC)

Embedding

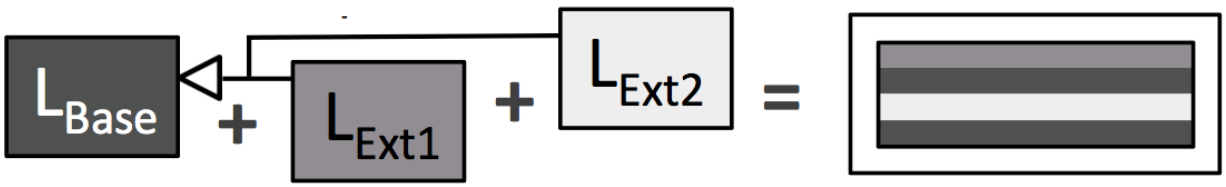
Extension

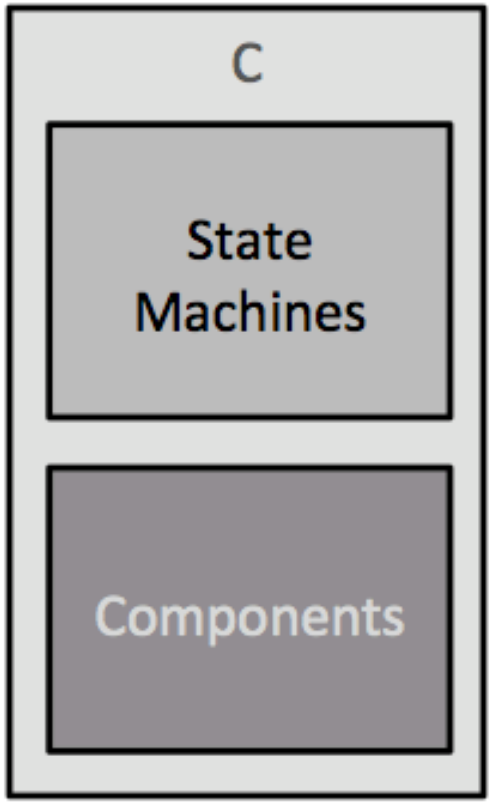
Extension Composition



Fundamentally it works and scales.

Is Extension Composition modular?





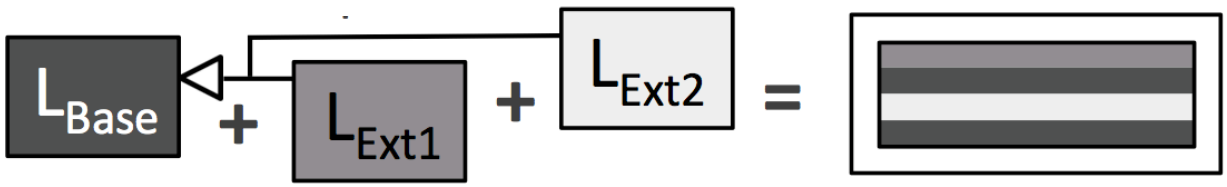
State Machines → C
Components → C

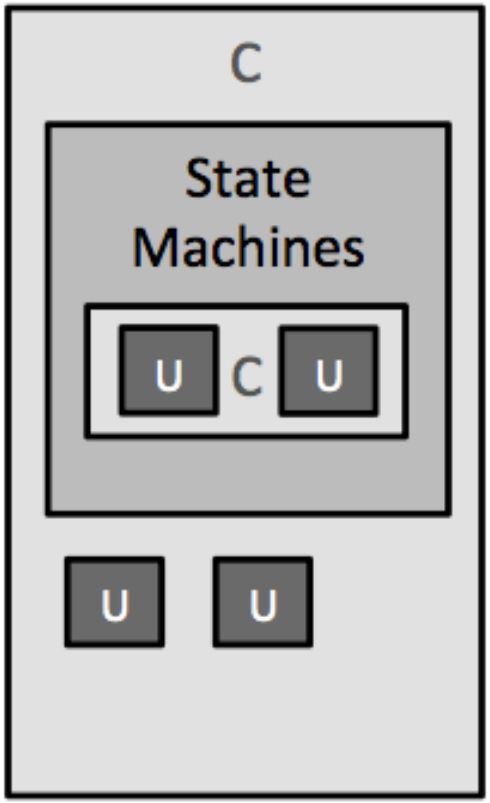
Multiple Extensions in C

No syntactic or IDE issues.
Type systems compose well.

BUT: Semantic Interactions cannot be statically guaranteed to be correct.

Is Extension Composition modular?





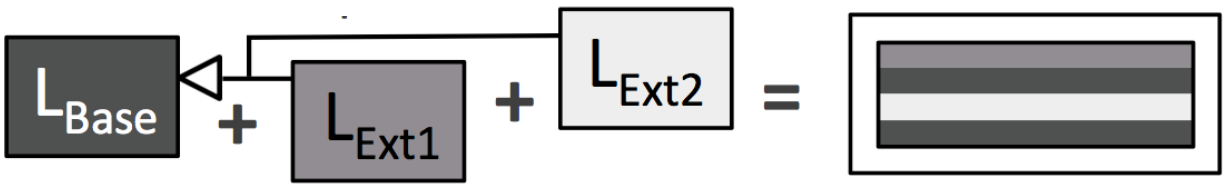
State Machines → C S
Units (U) → C S

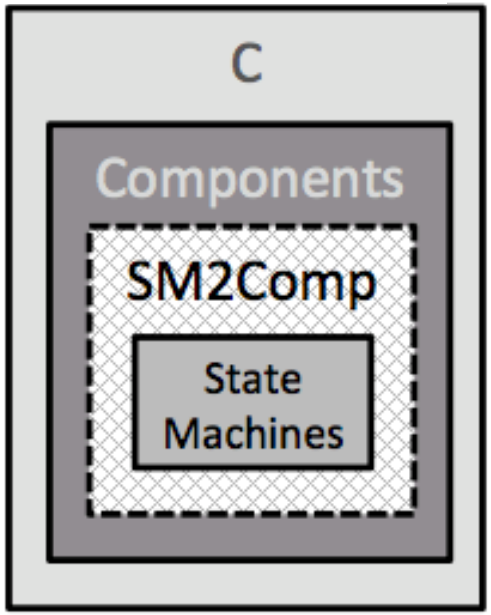
Nesting in C/in each other

No syntactic or IDE issues.
Type systems compose well.

BUT: Semantic Interactions cannot be statically guaranteed to be correct.

Is Extension Composition modular?





State Machines → C
Components → C
SM2Comp → C
SM2Comp → State Mach.
SM2Comp → Components

Nesting in each other

Adapter language Needed
Type system adapters needed

Adapter Language also deals with semantic interactions.

Specifically design integrations!
No modularity!

Is Extension Composition modular?

Yes, to the degree it can reasonably be expected to be (third case!)

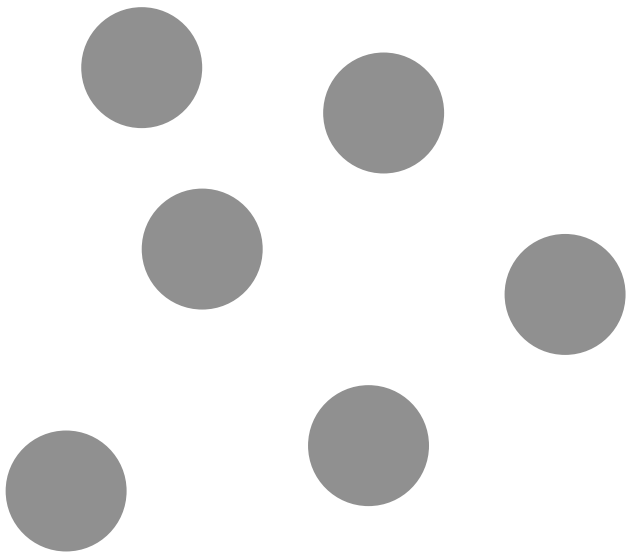
But with the limitations that semantics cannot be statically checked to be compatible.

Might be solved in the (far) future when all transformations are declarative and analyzable.

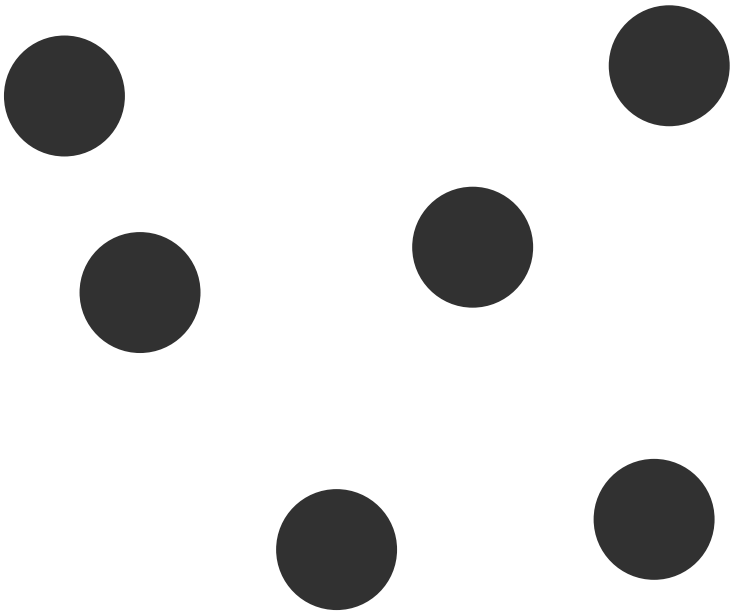
Composing Type Systems

Declarative

L1



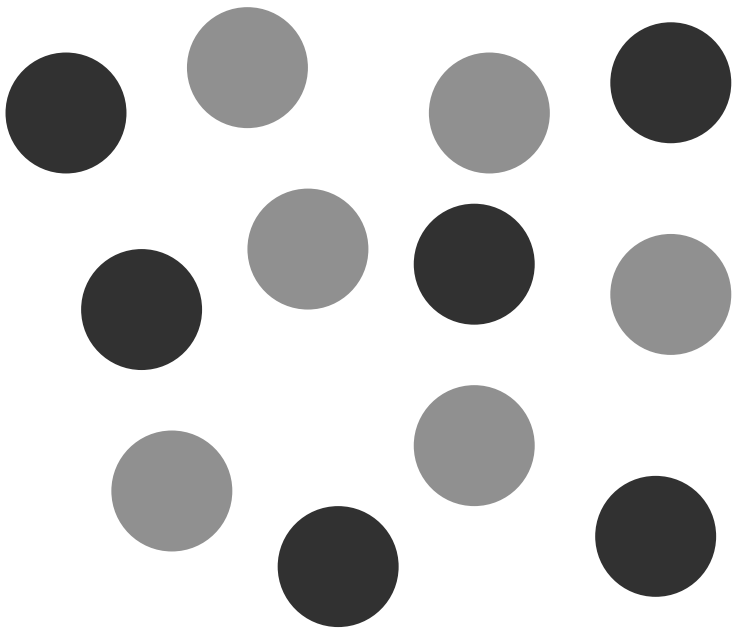
L2



Composing Type Systems

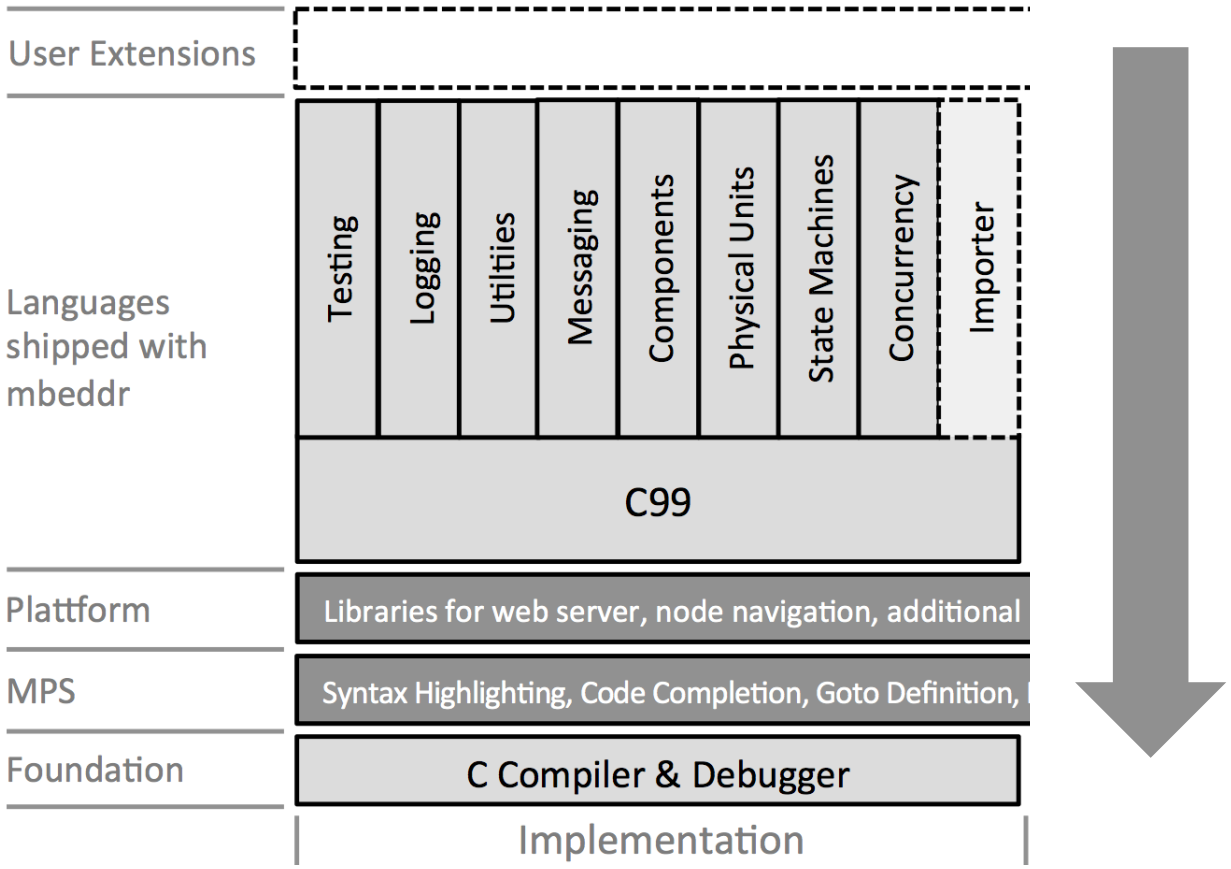
Declarative
Order Irrelevant

L1 + L2



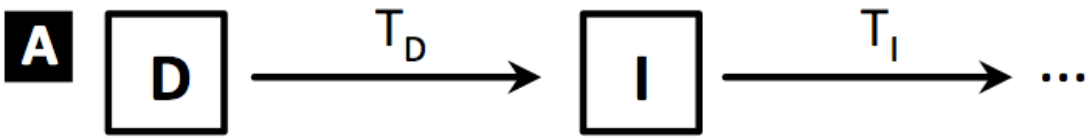
Composing Transformations

Declarative
Order Relevant



Composing Transformations

Declarative
Order Relevant

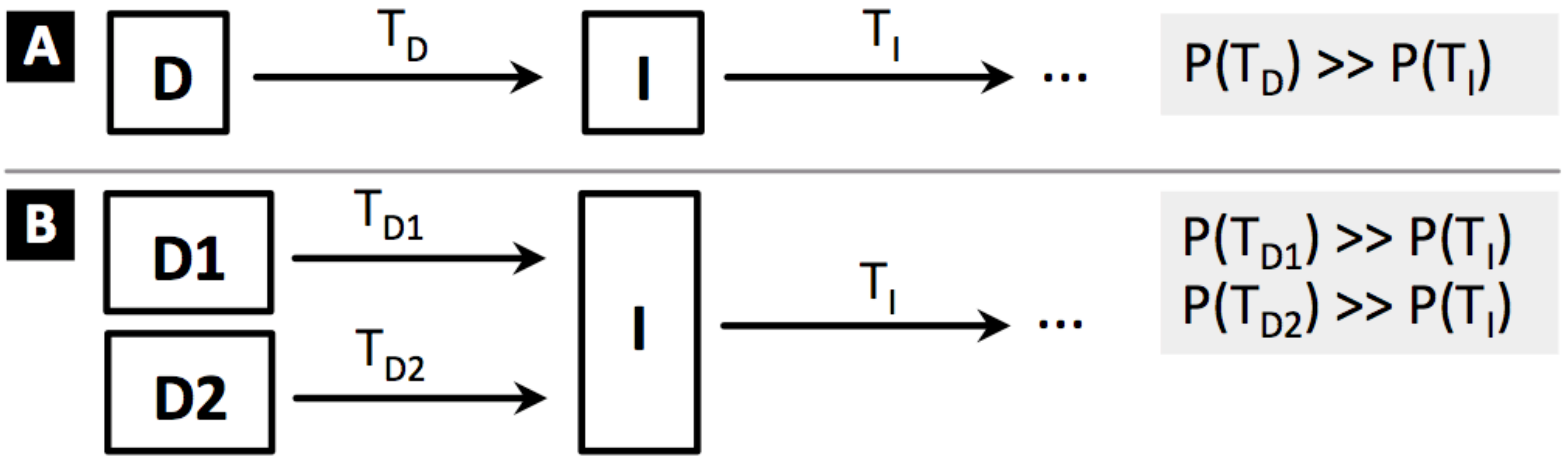


$P(T_D) >> P(T_I)$

Simple extension: D extends I

Composing Transformations

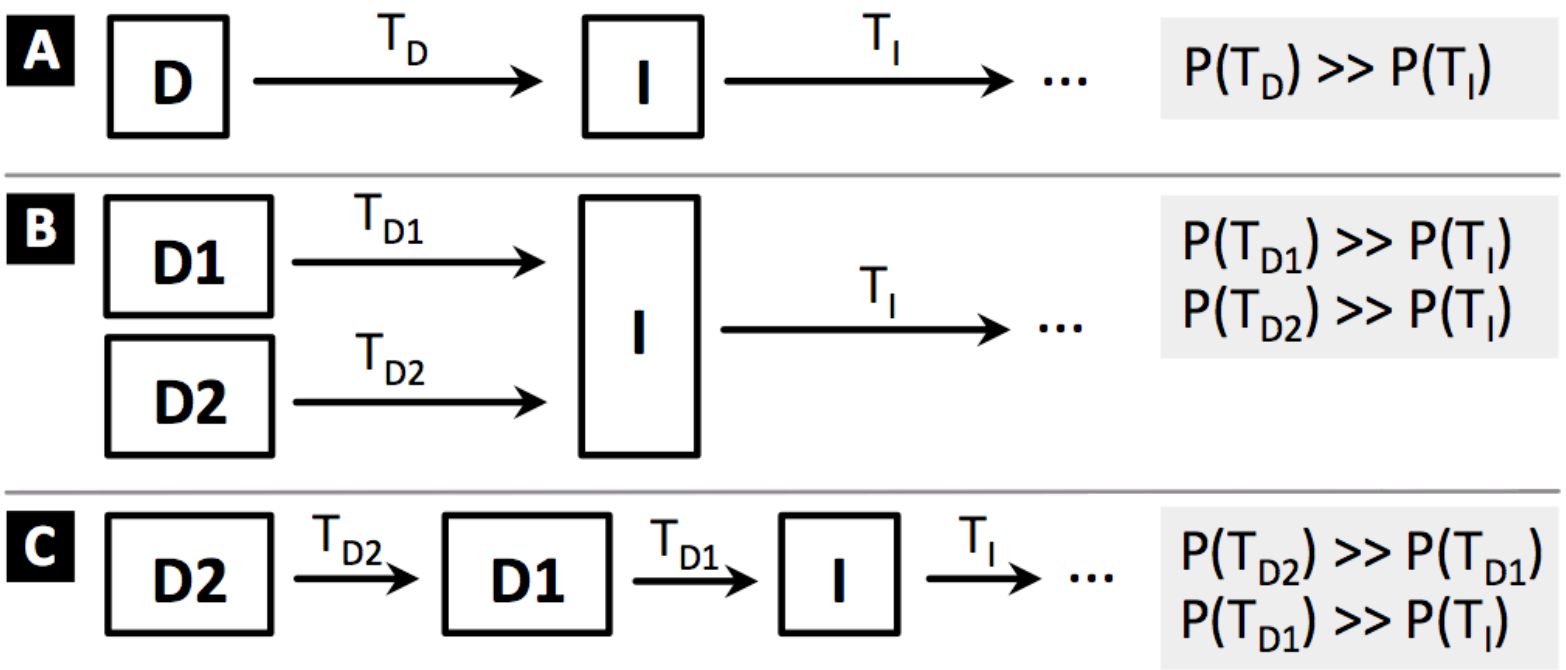
Declarative
Order Relevant



Independent extension:
D1 and D2 extend I

Composing Transformations

Declarative
Order Relevant

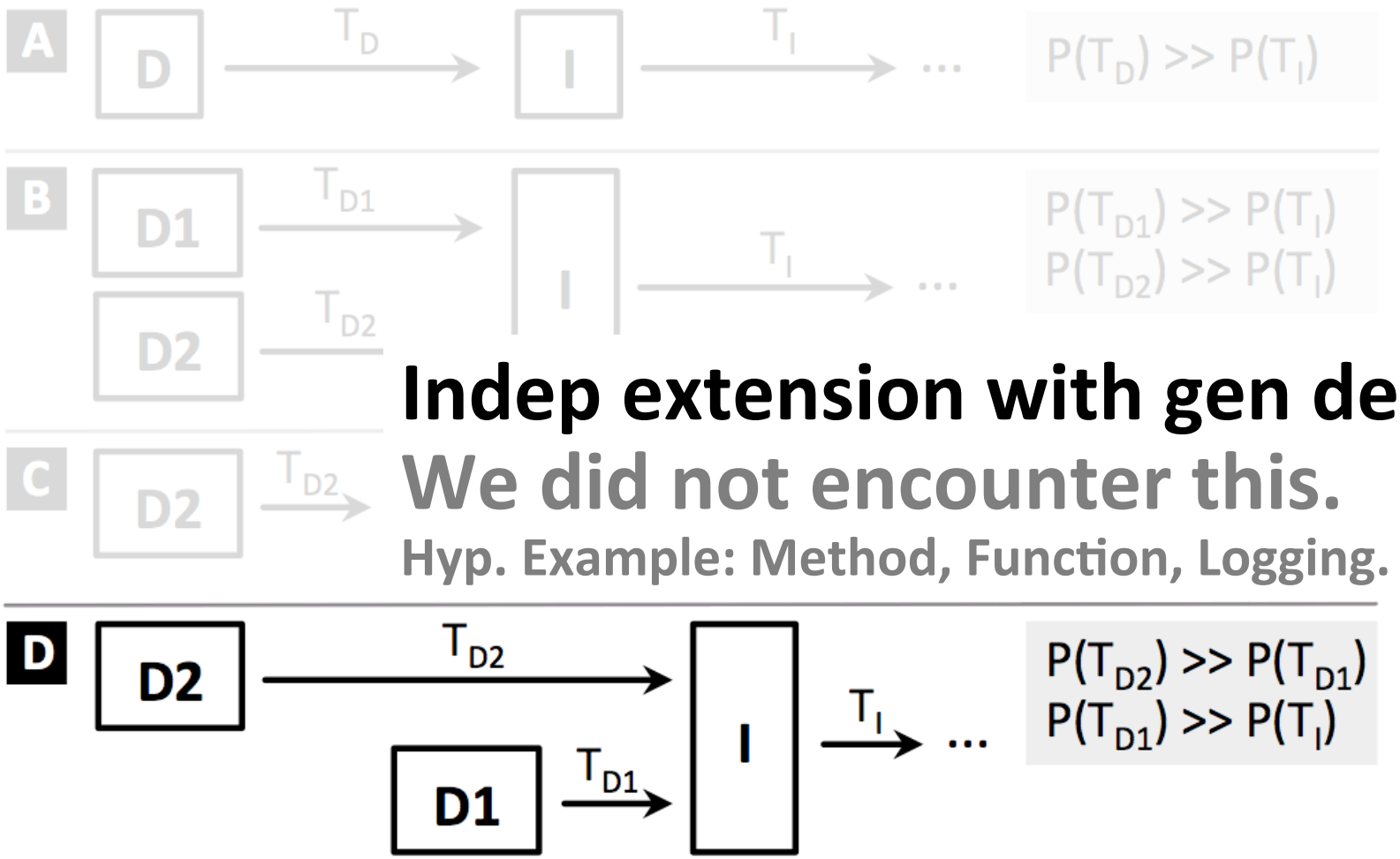


Stacked extension:

D2 extends D1 and extends I

Composing Transformations

Declarative
Order Relevant



Indep extension with gen dep:

We did not encounter this.

Hyp. Example: Method, Function, Logging.

Composing Transformations

Getting the ordering right can be a challenge. We discuss debugging later.

Once correct, the system works well.

RQ 1

Language Modularity

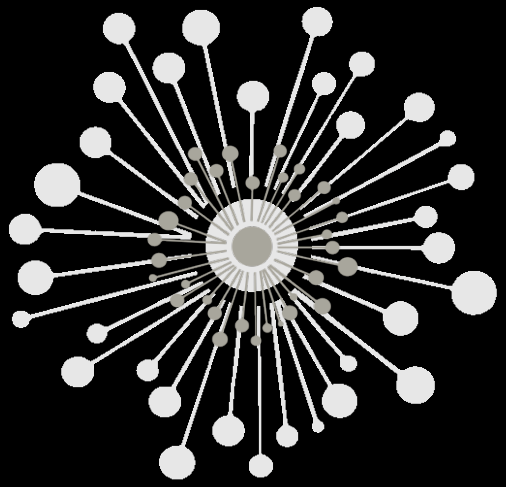
mbeddr's 34 C extensions indicate that MPS' **language modularity works**. It is useful for language understanding, testing and reuse.

In rare cases, modularity is compromised by **necessary changes to the base language** and unwanted dependencies between independent extensions.

Currently there is **no way to detect (unwanted) semantic interactions** between independent language extensions through analysis of their transformations.

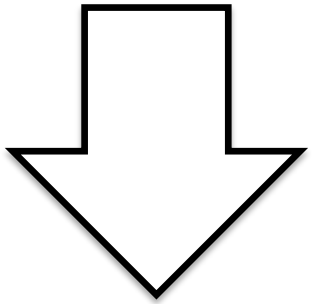
Lessons Learned about Language Engineering from the Development of mbeddr

5 What can we learn?



RQ 2 Projectional Editing

Projectional Editing



Language Composition

Syntactic Flexibility is actually useful for end users – see Smart Meter.

```
//[ A documentation comment with references
  [ to @arg(data) and @arg(dataLen)
void aSummingFunction(int8[] data, int8 dataLen) {
  int16 sum;
  for (int8 i = 0; i < dataLen; i++) {
    sum += data[i];
  } for
} aSummingFunction (function)
```

```
double midnight2(int32 a, int32 b, int32 c) {
```

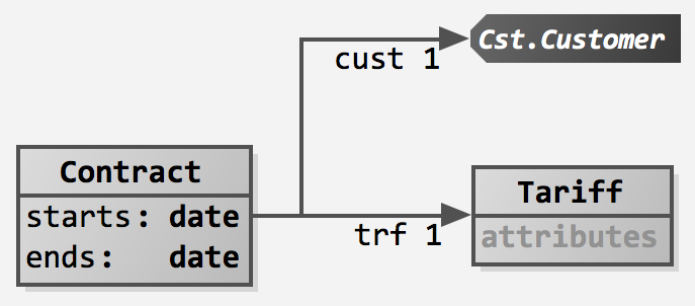
$$\text{return}\ \frac{-b + \sqrt{b^2 - \sum_{i=1}^{4} a * c}}{2 * a};$$

```
} midnight2 (function)
```

```
int16 decide(int8 spd, int8 alt) {
  return
```

	spd > 0	spd > 100
alt < 0	1	1
alt == 0	10	20
alt > 0	30	40
alt > 100	50	60

```
otherwise 0;
} decide (function)
```



Different Notations for one Abstraction.

State Machines

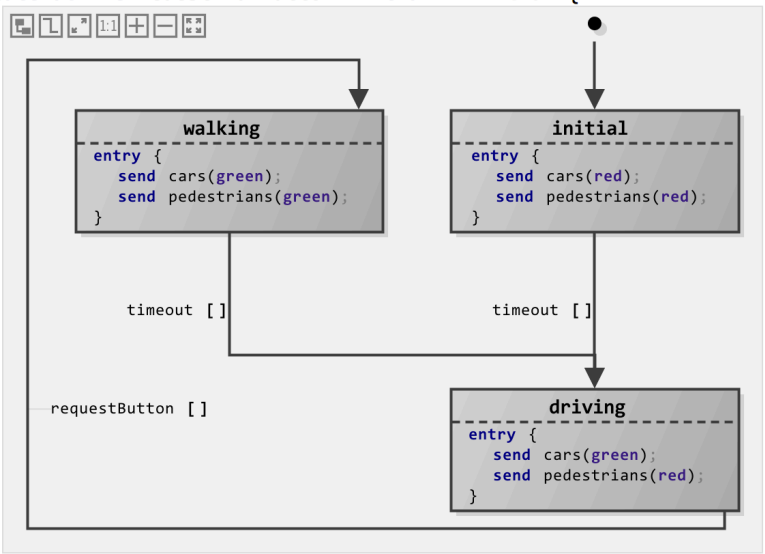
```
statemachine PedestrianButton initial = initial {
```

		Events	
		requestButton()	timeout()
States	initial		[ ] -> driving
	driving	[ ] -> walking	
	walking		[ ] -> driving

```
}
```

```
statemachine PedestrianButton initial = initial {
  in event requestButton() <no binding>
  in event timeout() <no binding>
  out event cars(Color carColor) <no binding>
  out event pedestrians(Color carColor) <no binding>
  state initial {
    entry {
      send cars(red);
      send pedestrians(red);
    }
  } state initial
  state driving {
    entry {
      send cars(green);
      send pedestrians(red);
    }
    on requestButton [ ] -> walking
  } state driving
  state walking {
    entry {
      send cars(green);
      send pedestrians(green);
    }
    on timeout [ ] -> driving
  } state walking
}
```

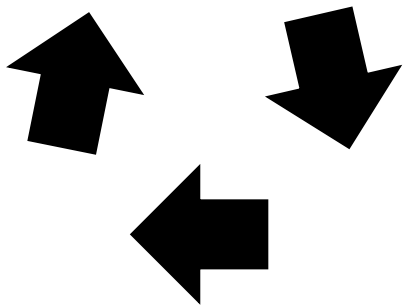
```
statemachine PedestrianButton initial = initial {
```




```
}
```

Different Notations for one Abstraction.

State Machines



But: type checks and some actions based on AST, not on what is shown.

This can be confusing to users.

Optionally projected syntax.

Requirements Traces

```
Node:        VERY_HIGH_SPEED [GlobalConstantDeclaration]
Kind:        implements
1st Target:  For each trackpoint where you go more than 200 mps, you get 20 points
             Lorem ipsum dolor sit amet, consectetur adipiscing elit.
             Praesent feugiat enim arcu, ut egestas velit. Suspendisse
             potenti. Etiam risus ante, bibendum ut mattis eget, convallis
             sit amet nunc.
```

```
#
#constant HIGH_SPEED = 10; [T]
#constant VERY_HIGH_SPEED = 20; [T]
#constant LANDING = 100; [T]
```

Decoupled Notational Primitives

Sum & Trees

```
int32 sumUpIntArray(int32[] arr, int32 size) {
  return  Σ_{i = 0}^{size}  arr[i] ;
} sumUpIntArray (function)
```

```
fun weighted(quarters: collection<FinData>) =
  Σ_{i: int = 0}^{quarters.size}  let [ √( q.pe / |q.rev| ) ]
                                  with q = quarters.at(i)
```

Math in C and KernelF

```
fun riskFactor(gender: int, age: int, weight: int) =
                                  <> default: LOW
        [gender == MALE]                          [gender == FEMALE]
   [age < 40]        [age > 40]            [age > 50 && overweight(weight)]
      LOW        [overweight(weight)]                  HIGH
                       MEDIUM
```

```
Scooter1
  Wheels
    c: front.Simple => rear.Simple
    c: front.sf.f11 => rear.sf.f11
    c: front.Brake.Simple => front.Simple
    c: front.PressureSensor[1].maxPressure > 1
    front: Wheel    rear: Wheel
  Engine
    int power
    c: power > 100
    c: Battery => true
    Electrical
      Battery
        int capacity
    Combustion
      Diesel    Gasoline
```

Trees for decisions and feature models

Developing new notations.

MPS is Bootstrapped.

```
LOOP
lower:        [> { name } : % varType % = % lower % <]
upper:        % upper %
body:         % body %
symbol:       SumSymbolSerif
parentheses:  (node)->boolean {
                Utils.hasFollowingExpression(node.body);
              }
```

```
graphical editor for concept State
  diagram.box

    editor   [/
               [> ---> [> { name } <] ---> <]
               [/
                 ------
                 (- % contents % /empty cell:  -)
               /]
             /]
    shape    StateShape(false)
    delete   thisNode.delete
```

```
table {
  column headers:
    query {
      getHeaders cols (node, editorContext)->join(string | EditorCell | node<> |
        node.colHeaders;
      }
      insert new header (node, index)->void {
        node.colHeaders.insert(index, new node<DecTabColHeader>());
      }
      on delete: (node, index)->void {
        node<DecTabColHeader> h = node.colHeaders.get(index);
        node.contents.where({~it => it.col == h; }).forEach({~it => it.delete; })
      }
    }
```

Editor Usability

Based on an empirical study.

People **prefer MPS** over conventional IDEs

MPS more is **more efficient** than normal IDEs

MPS more is **more productive** than normal IDEs

MPS makes it easier to create **correct programs**

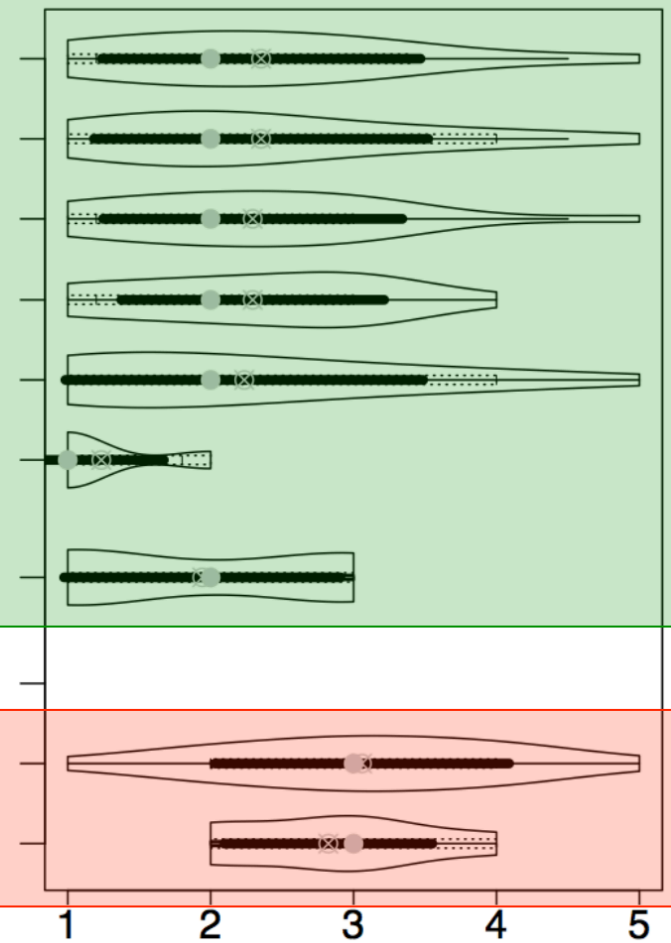
MPS enforces a **structurally correct AST**

People benefit from **language modularity**

People benefit from the **flexible notations**

The experience with **learning MPS** is mixed.

It takes some time to **get used to MPS**



Strongly agree ... Neutral ... Strongly disagree

Significantly Improved Editor Usability

Grammar Cells (see SLE Paper)

```
A  editor for concept GlobalVariableDeclaration
   [- flag{ exported } flag{ extern } wrap % type % { name } optional [- = % init % -] ; -]
B  editor for concept BinaryExpression
   rule: [- wrap % left % substitute constant wrap % right % -]
C  editor for concept NumberLiteral
   rule: [- wrap splittable{ value } -]
D  editor for concept ParensExpression
   rule: brackets[ ( % expression % ) ]
```

Top-Down Progr. via Target Creation

```
state initial {
  entry { ... }
  on timeout [ ] -> yellow
} state initial
state driving {
  entry { ... }
  on requestButton [
} state driving
state walking {
```

```
yellow  CompositeState (PlainCDemo.PedestrianButton.contents)
yellow  State (PlainCDemo.PedestrianButton.contents)
yellow  JunctionState (PlainCDemo.PedestrianButton.contents)
Default   Create Reference Target
```

Palettes and other input support utils.

HelloMath

```
int32 sumArr(int32[] arr, int32 size) {
  return Σ_{i=0}^{size} arr[i];
} sumUpIntArray (function)
```

Context Actions

abs, frac, i, log, pow, product, sqrt, sum

Generic Commenting

```
statemachine PedestrianButton {
  in event requestButton()
  in event timeout()
  out event cars(Color carColor)
  out event pedestrians(Color carColor)
  state initial {
    entry { ... }
    /*on timeout [ ] -> driving */
  } state initial
  /*state driving {
    entry { ... }
    on requestButton [ ] -> walking
  } state driving*/
  state walking {
    entry { ... }
    on timeout [ ] -> driving
  } state walking
}
```

But still issues/differences in Editor Usability

Selecting code is structural, not linear

Copy/paste of similar syntax but
different structure requires customization

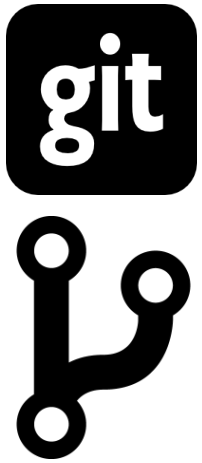
Grammar cells go a (very) long way, but
not *every* editing action is possible.

Pasting from text requires integration
of a parser (hooks exist), not automatic

Very few complaints since we did the improvements.

We consider the problem mostly solved.

Version Control Integration



Fundamentally File-based

We used git – worked well

diff/merge in MPS

Can't use outside tools (gerrit)

297d8500e0a08e7f8701b4a536b7d4a6be62cb64

```
statemachine PedestrianButton initial = initial {
                          Events
                          requestButton()  timeout()
   States  initial                          [ ] -> driving
           driving  [ ] -> walking
           walking                          [ ] -> driving
}


void normalizePosition(Position* p) {
  if (p->x > 100) {
    p->x = 100;

  } if
  if (p->y > 100) {
    p->y = 100;
  } if
```

Your version

```
statemachine PedestrianButton initial = initial {
                          Events
                          requestButton(int8 dt)  timeout()
   States  initial                                 [ ] -> driving
           driving  [dt > 100] -> walking
           walking                                 [ ] -> walking
}


void normalizePosition(Position* p) {
  if (p->x > 100) {
    p->x = 100;

  } if
  if (p->y > 100) {
    p->y = 100;
  } if
```

3 differences Deleted Changed Inserted

RQ 2 Projectional Editing

Two main benefits of projectional editing – **language modularity** and a range of combinable **notations** – have been **used extensively** in mbeddr. The anticipated benefits have been observed.

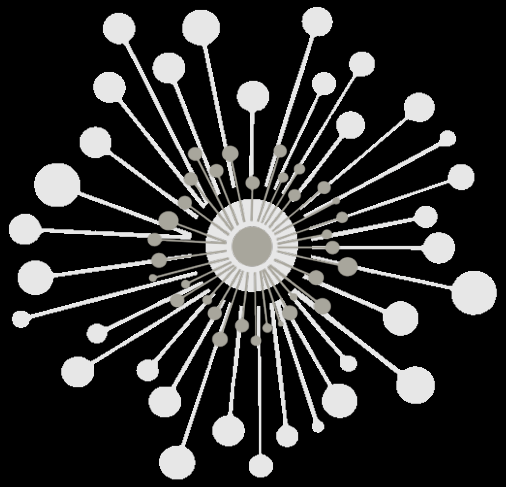
Editor can be flexibly extended with new notational styles with acceptable effort, as exemplified by math, tables and diagrams.

The ability to use **multiple and partial projections must be further improved** by integrating with other language aspects, in particular, editor actions and type checks.

Lessons Learned about Language Engineering from the Development of mbeddr

5

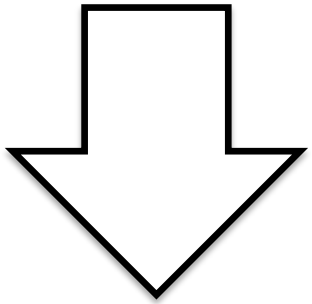
What can we learn?



RQ 3

Managing Complexity

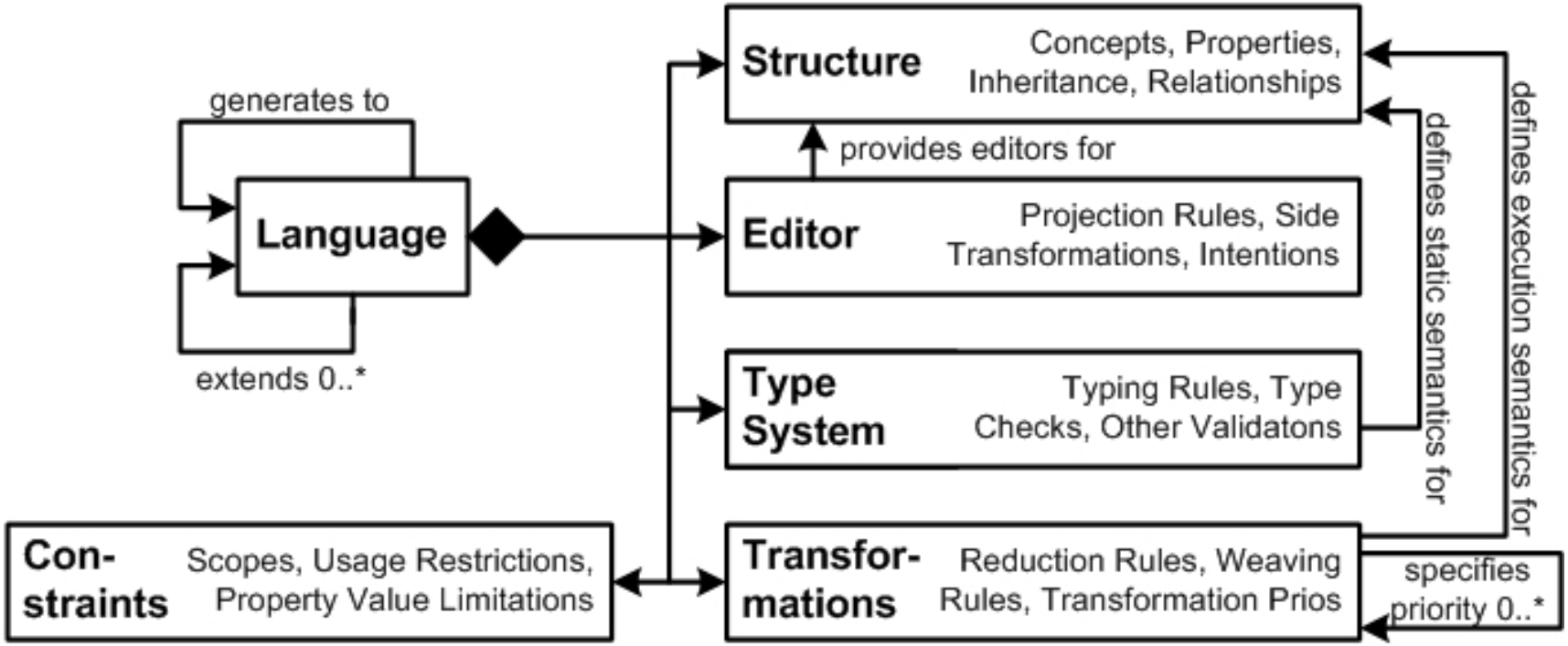
Projectional Editing



Reduced Complexity

Language modularization
No disambiguation code

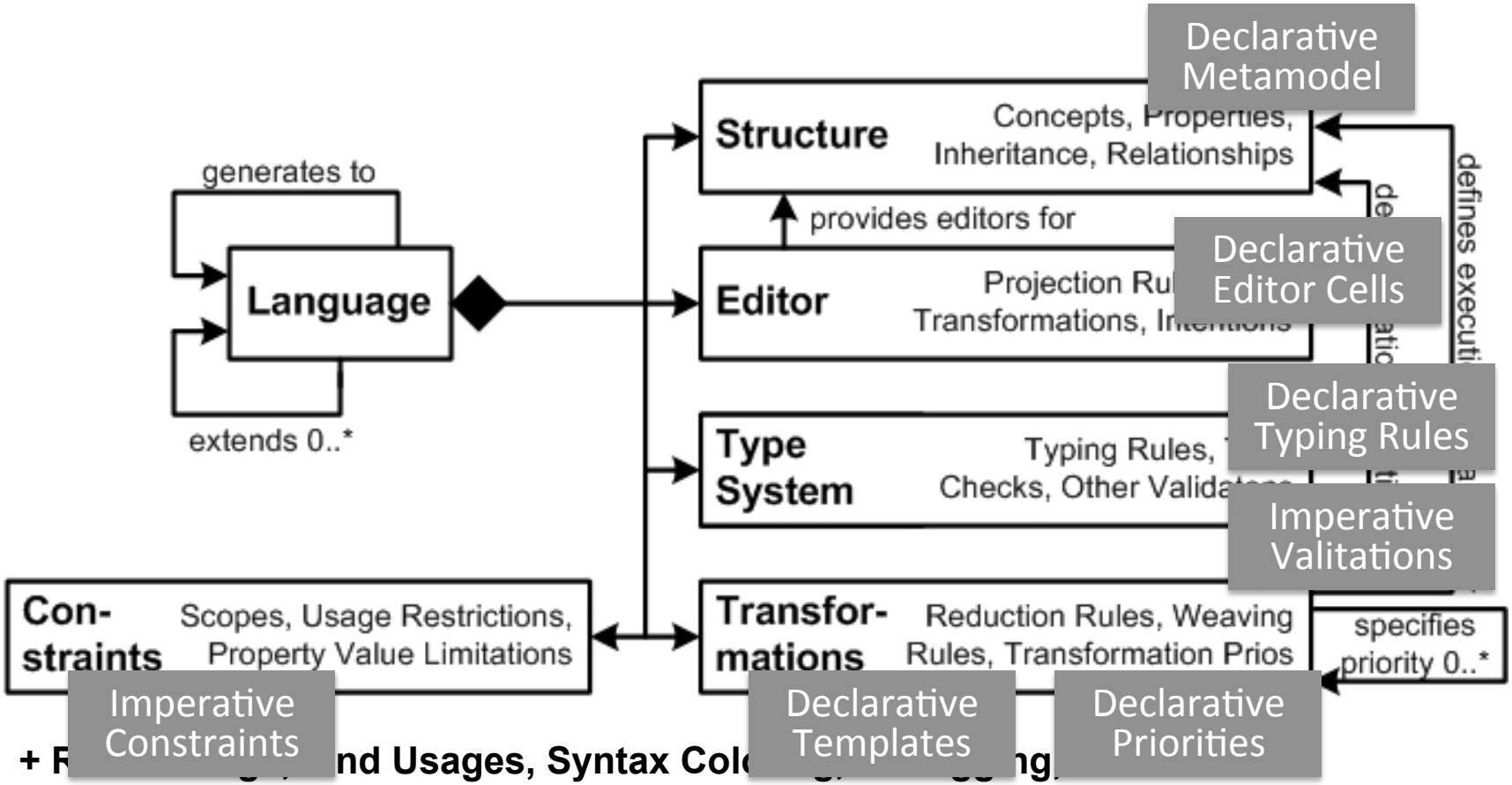
[Language Aspects]



+ Refactorings, Find Usages, Syntax Coloring, Debugging, ...

[Language Aspects]

A separate DSL for each aspect



[Language Aspects]

A separate DSL for each aspect

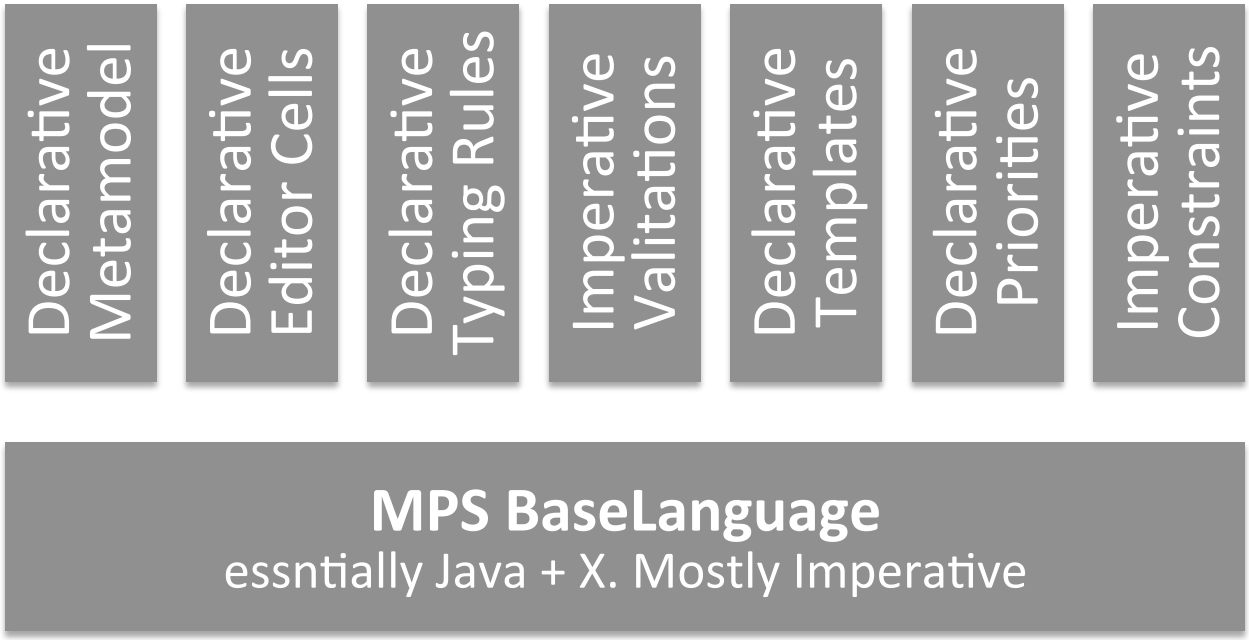
- Declarative Metamodel
- Declarative Editor Cells
- Declarative Typing Rules
- Imperative Valitations
- Declarative Templates
- Declarative Priorities
- Imperative Constraints

Each language optimized for its task.

In our experience, separate aspect DSL increase productivity – as DSLs should.

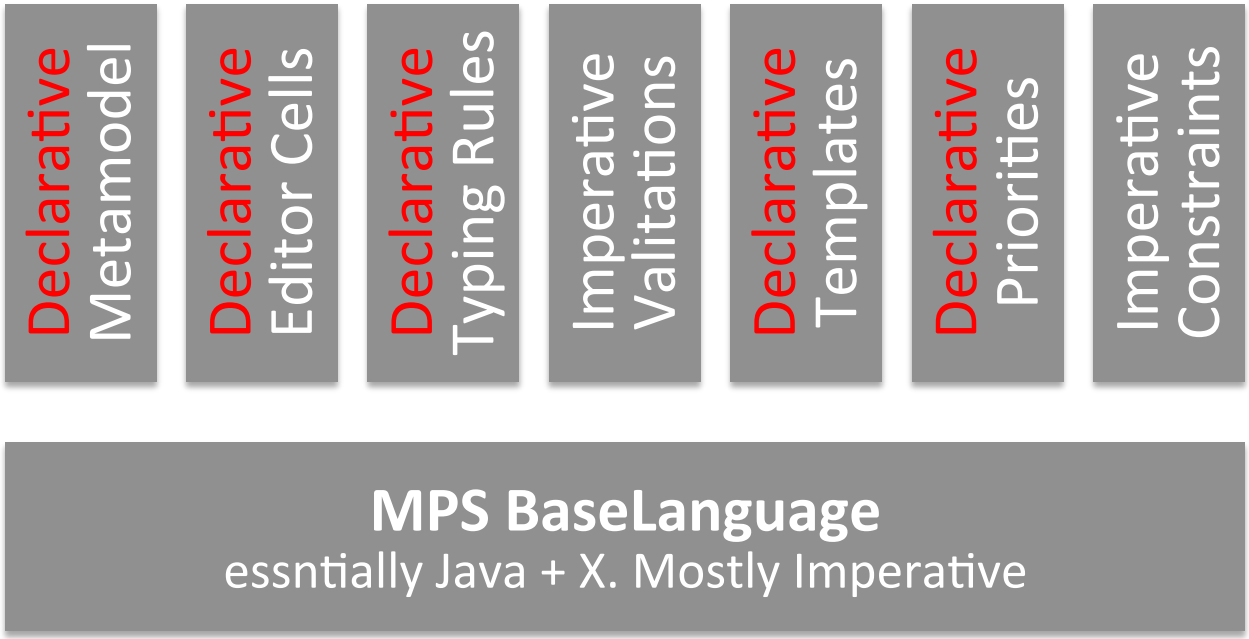
[Language Aspects]

A separate DSL for each aspect, but:



[Language Aspects]

A separate DSL for each aspect, but:



Declarative? { **Analysis?** **Optimization?**

[Language Aspects]

A separate DSL for each aspect, but:

Declarative? { **Analysis?** / **Optimization?**

The non-declarativeness leads to problems in analysis (impact, debugging, verifications) or optimization (caching).

[Language Aspects]

A separate DSL for each aspect. Debugging?

PlainCDemo

Position ▶ y

```
int8 add(int8 x, int8 y) {
  int8[3] a = {1, 2, 3};
  int8[3] b;
  b = a;
  return x + y;
} add (function)

void normalizePosition(Position* p) {
```

Cell Explorer

- EditorCell_Collection[refNodeList_contents] ([15, 75, 386, 532], baseLine = 90, ascent = 15, descent = 517)
  - **Node** PlainCDemo (ImplementationModule) (>Vertical)
  - Actions
  - EditorCell_Collection[Collection_czjnxc_a] ([15, 75, 136, 76], baseLine = 90, ascent = 15, descent = 61)
  - EditorCell_Constant[Constant_yww9py_a] ([15, 151, 2, 19], baseLine = 166, ascent = 15, descent = 4)
  - EditorCell_Collection[Collection_y1p9lm_a] ([15, 170, 208, 114], baseLine = 170, ascent = 0, descent = 114)
    - **Node** add (Function) (>Vertical)
    - Actions
    - EditorCell_Collection ([15, 170, 0, 0], baseLine = 170, ascent = 0, descent = 0)
    - EditorCell_Collection[Collection_y1p9lm_b0] ([15, 170, 208, 114], baseLine = 185, ascent = 15, descent = 99)
  - EditorCell_Constant[Constant_yww9py_a] ([15, 284, 2, 19], baseLine = 299, ascent = 15, descent = 4)
  - EditorCell_Collection[Collection_y1p9lm_a] ([15, 303, 296, 95], baseLine = 303, ascent = 0, descent = 95)
  - EditorCell_Constant[Constant_yww9py_a] ([15, 398, 2, 19], baseLine = 413, ascent = 15, descent = 4)
  - EditorCell_Collection[Collection_y1p9lm_a] ([15, 417, 386, 171], baseLine = 417, ascent = 0, descent = 171)

Editor

[Language Aspects]

A separate DSL for each aspect. Debugging?

Logical View

- mbeddr.tutorial.main.plainC@2_1
- mbeddr.tutorial.main.plainC@3_0
- mbeddr.tutorial.main.plainC@3_2
- mbeddr.tutorial.main.plainC@5_0
- mbeddr.tutorial.main.plainC@5_2
- mbeddr.tutorial.main.plainC@6_0
- mbeddr.tutorial.main.plainC@11_0
- mbeddr.tutorial.main.plainC@11_1
- mbeddr.tutorial.main.plainC@14_1
- mbeddr.tutorial.main.plainC@16_0
  - BuildConfiguration (m.t.m.plainC
  - NameShorteingRoot
  - PlainCDemo
  - SIUnits
  - TypeSizeConfiguration (m.t.m.pla
  - Unittest_Helper
  - Unittest_Helper_NonIsolated
  - alloca
  - setimp

PlainCDemo@5_0 | PlainCDemo

TrafficLights

```
statemachine TrafficLights initial = initial {
  in event pedestrian() <no binding>
  in event timeout() <no binding>
  state initial {
    on timeout [ ] -> cars
  } state initial
  state cars {
```

PlainCDemo@16_0

Warning: the node is in a transient model. Your changes won't be saved.

Warning: the node is in a read-only model. Your changes won't be saved

```
{ (invisble)
  switch (instance->__currentState) {
    case TrafficLights_initial__state: {
      switch (event) {
        case TrafficLights_timeout__event: {
          if (true) {
```

Generation Tracer

- Phase 11_0->11_1
  - [Statemachine] TrafficLights (6331712389840909340) (mbeddr.tutorial.main.plainC@11_0)
    - [StatementList] { (6183633367244809915) (com.mbeddr.ext.statemachines.generator.template.generator.template.main@
    - [TemplateCallMacro] Call (881042019525800918) (com.mbeddr.ext.statemachines.generator.template.generator.template
    - [StatementList] { (6331712389840912565) (mbeddr.tutorial.main.plainC@11_1)

Generation Tracer | Console | 3: Usages | 0: Messages | 9: Version Control | Cell Explorer | Documentation | 1 Event Log

Transformations

[Language Aspects]

A separate DSL for each aspect. Debugging?

PlainCDemo

add

```
PlainCDemo                                  cor
model   mbeddr.tutorial.main.plainC         imp

struct Position {
  uint16 x;
  uint16 y;
};

int8 add(int8 x, int8 y) {
  int8[3] a = {1, 2, 3};
  int8[3] b;
  b = a;
  return x + y;
} add (function)

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

exported int32 main(int32 argc, string[] a
  Position p = {10, ((uint16) add(20, 22))
  normalizePosition(&p);
  for (int8 i = 0;i < 100; i++ ) {
    add(i, 20);
  } for
  add(10, 20);
  return 0;
} main (function)
```

Typesystem Trace

- typeof_Function applied to node "add"
- typeof_ITyped applied to node "add"
- typeof_Function applied to node "normalizePosition"
- typeof_ITyped applied to node "normalizePosition"
- typeof_Function applied to node "main"
- typeof_ITyped applied to node "main"
- typeof_ITyped applied to node "x"
- typeof_ITyped applied to node "y"
- typeof_Type applied to node "int8 "
- typeof_ITyped applied to node "x"
- typeof_ITyped applied to node "y"
- typeof_Type applied to node "void "
- typeof_ITyped applied to node "p"
- typeof_Type applied to node "int32 "
- typeof_ITyped applied to node "argc"
- typeof_ITyped applied to node "argv"
- typeof_Type applied to node "uint16 "
- typeof_Type applied to node "uint16 "
- typeof_LocalVariableDeclaration applied to node "a"
- typeof_ITyped applied to node "a"
- typeof_LocalVariableDeclaration applied to node "b"
- typeof_ITyped applied to node "b"
- typeof_ExpressionStatement applied to node ";"
- typeof_ReturnStatement applied to node "return"
- typeof_Type applied to node "int8 "

- ; : z assigned
- = : a1 assigned
- z = a1 added

- Type system state
  - Solving inequalities in process
  - Inequalities by groups
    - u<=v
    - null<='x
  - Equations
    - j = i = uint16
    - s = t
    - q = r
    - w = v
    - y = x
    - o = p
    - e = f = int32
    - b = a = int8
    - z = a1
    - h = g = uint16
    - k = l
    - m = n
    - c = d = void

Type System

[Language Aspects]

A separate DSL for each aspect. Debugging?

```
System.out.println(...);
```

All in all, the debugging experience is not where it should be. Needs improvement.

All the rest…

[Language Aspects]

A separate DSL for each aspect, but missing aspects:

Declarative Metamodel

Declarative Editor Cells

Declarative Typing Rules

Imperative Valitations

Declarative Templates

Declarative Priorities

Imperative Constraints

MPS BaseLanguage
essntially Java + X. Mostly Imperative

Documentation

Debugger

Interpreter

...

Since MPS 3.3, one can define custom aspects.

MPS is bootstrapped.

meaning: it is built with itself.
it can be <u>extended</u> with itself.

MPS is bootstrapped.

meaning: it is built with itself.
it can be extended with itself.

```
!                    [expr <= $boolean]                    = !#expr
&&                   [left <= $boolean, right <= $boolean] = #left && #right
||                   [left <= $boolean, right <= $boolean] = #left || #right
<=>                  [left <= $boolean, right <= $boolean] {
                                                             boolean l = #left;
                                                             boolean r = #right;
                                                             if (l && r) { return true; }
                                                             if (!l && !r) { return true; }
                                                             return false;
                                                           }

IFunctionCall        []                                    = #->function(#args => node.function.args)
IFunctionLike        []                                    = node.body
ArgRef               []                                    = env[node.arg]
```

Declarative type mappings
Type Guards
Concise syntax, especially for recursive calls
Composable (just as languages)
Abstractions for environments and stacks

Interpreter

MPS is bootstrapped.

meaning: it is built with itself.
it can be extended with itself.

```
def EQ_ABSTRACT_SELF(class : ClassConcept) = {
  method := class.member
  assert method instanceOf InstanceMethodDeclaration

  // check method name and return type
  name := method.name
  assert method.returnType instanceOf
  assert eval(name.equals("equals"))

  // assert that the method has only o
  assert count getParameters(method) =

  // and check that the parameter's type is the same as the class
  parameter := getParameters(method)
  parameterType := parameter.type
  assert parameterType instanceOf ClassifierType
  assert class == parameterType.classifier
}
```

```
def CI_CONFUSED_INHERITANCE(class : ClassConcept) = {
  assert class.isFinal == true
  member := class.member
  assert member instanceOf FieldDeclaration
  assert member.visibility instanceOf ProtectedVisibility
}
```

Restricted, Declarative Language
Transformed to Patterns
Executed incrementally
Very fast!

IncA

MPS is bootstrapped.

meaning: it is built with itself.
it can be extended with itself.

```
<default> editor for concept SumExpression
  node cell layout:
    LOOP
    lower:        [> { name } : % varType % = % lower % <]
    upper:        % upper %
    body:         % body %
    symbol:       SumSymbolSerif
    parentheses:  (node)->boolean {
                    Utils.hasFollowingExpression(node.body);
                  }
```

New primitive editor cells for math, diagram, tables
More query-oriented than default MPS notations
Notation independent from Language

Editor

RQ 3

Managing Complexity

Using **a DSL for each language aspect works well** based on our experience, even though some aspects are missing and some are not declarative enough to support meaningful analyses.

The **support for debugging is spotty**: it works well for transformations, but debugging generator macros, behav- iors and type system rules is very tedious.

The ability to **extend** MPS' language definition **DSLs with MPS itself** is a powerful approach for managing complexity, and we have used it extensively, even though it has some limitations.

Lessons Learned about Language Engineering from the Development of mbeddr

5

What can we learn?

RQ 4

Tool Scalability

Editor Responsiveness

The bigger one root gets, the slower the editor gets because more "stuff" must be rendered:

- incrementally rendered, but still ...
- depends on the notation
- 3000 "lines" of C code

Split into several roots.
Impacts Language Design.

Editor Responsiveness

Code completion is extremely critical:

you can only enter what has been "put into" the code completion menu. **=> Tree Constraints Scopes**

```
statemachine TrafficLights initial = initial {
  in event pedestrian() <no binding>
  in event timeout() <no binding>
  state initial {
    on timeout [] -> cars
  } state initia
  state cars {
    on pedestria
  } state cars
  state pedestri
    on timeout [
  } state pedest
}

void normalizePo
  if (p->x > 100
    p->x = 100;
  } if
} normalizePosition (function)
```

add	^PlainCDemo.add (Function)
main	^PlainCDemo.main (Function)
normalizePosition	^PlainCDemo.normalizePosition (Function)
!	logical not
"	string literal
#concat	concatenating strings using the preprocessor
&	get address
'	character literal
(	parenthesize
((..)..)	downcast an expression
*	dereference

Default | Create Reference Target

Type Checking and Constraints

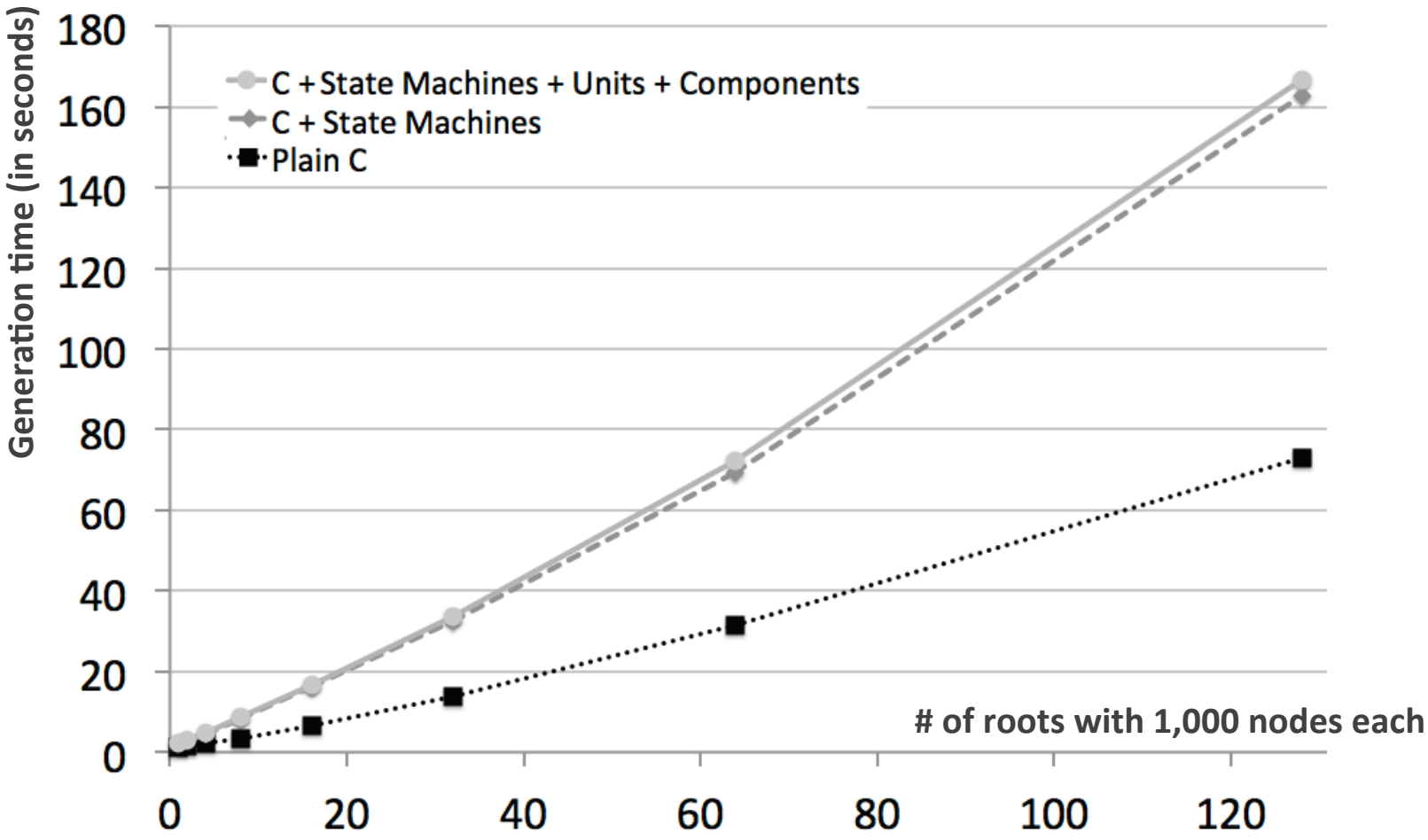
Type checking is incremental, and happens in the background.

Checks may run long – but are interruptable.

But both need locks on the repo: they can lead to performance issues.

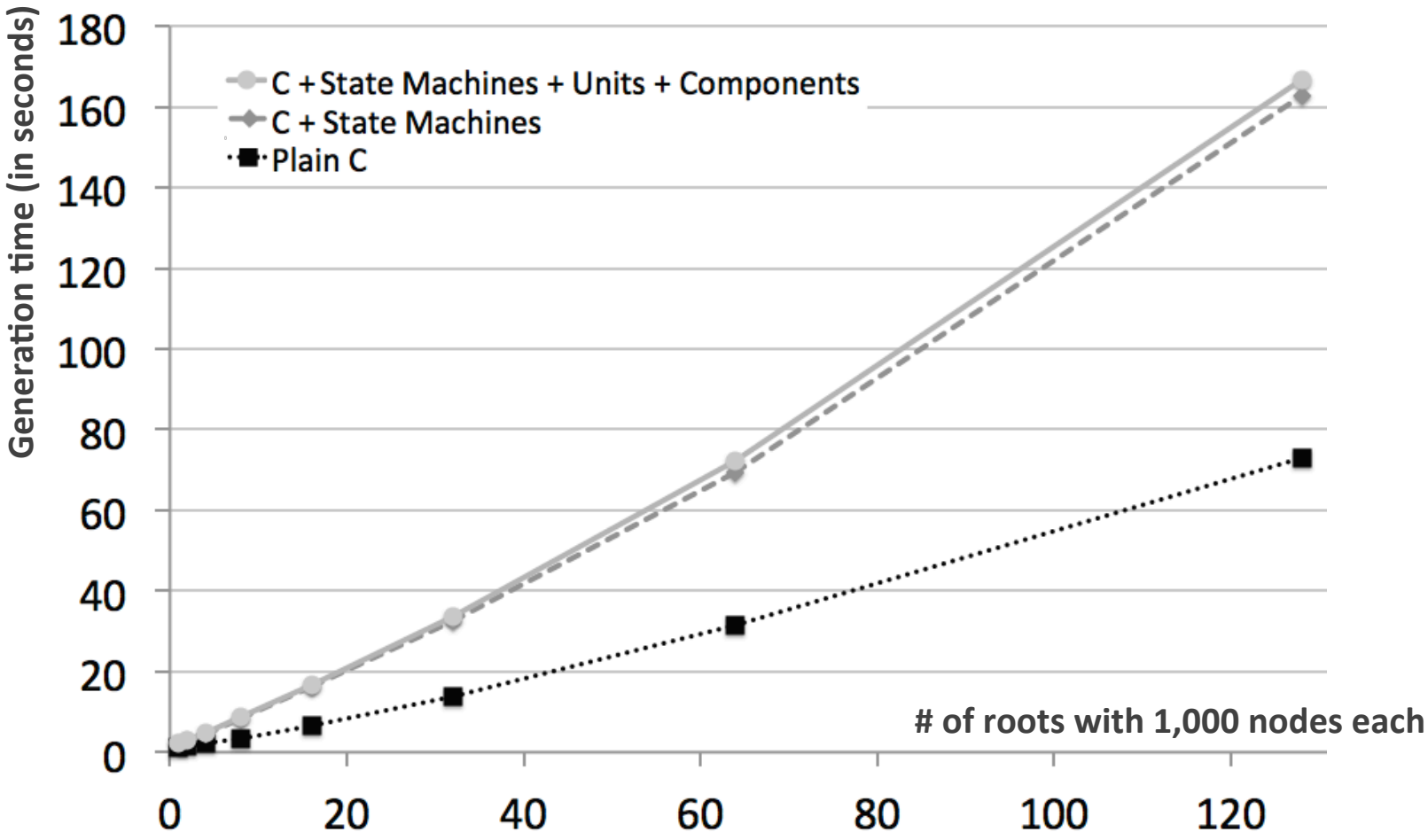
=> Cancellable Checks
Assessments
Selective Enabling
"Energy Saver Mode"

Transformation



The model with 128 roots corresponded to 68,400 LoC of generated C code (for Plain C), 202,000 LoC (C + State Machines) and 198,000 LoC (C + State Machines + Units + Components).

Transformation



Time ~ Size && Time ~ Intermediate Model Volume

RQ 4

Tool Scalability

With attention to root size and the distribution of code over models, then **systems of significant size can be built** with MPS.

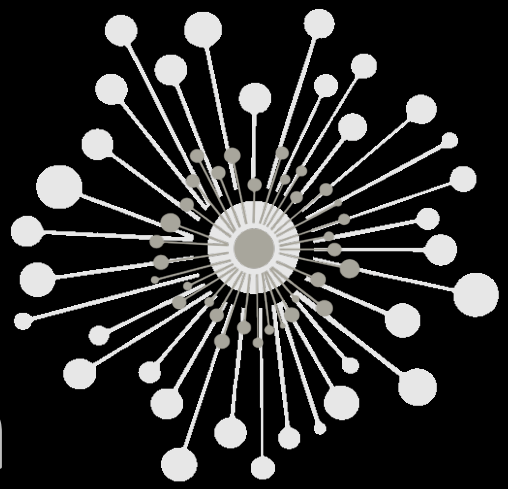
The performance of the **type system** (as it is evaluated in realtime in the editor) and **support for cross-model generation** are the two most critical ways of **improving MPS performance**.

During the **development of languages** we have not run into any problems regarding performance or scalability (of editor, type system or generator definitions).

Lessons Learned about Language Engineering from the Development of mbeddr

5

What can we learn?



RQ 5

Development Process

Testing the editor

No need to test if "parsing" works.

But you have to check if you can enter what you need to enter, where you think it belongs.

```
for ( int8 i = 0; i < 10

int8 add(int8 x, int8 y) {
  int8[3] a = {1, 2, 3};
  int8[3] b;
  struct
  return x + y;
} add (function)
```

Testing the editor

No need to test if “parsing” works.

And you have to test if the the tree is con-structure correctly when linear input is used.

```
Test case testSideTransformations
nodes
  ( [ AnImplementationModule   constraints          ] )
    [                          imports     nothing   ]
    [ ------------------------------------------------ ]
    [ void f() {                                       ]
    [   <expr  4 + 3 * 2>;                             ]
    [ } f (function)                                   ]

test methods
  test testPrecedence {
    assert expr.isInstanceOf(PlusExpression);
    assert expr.left.isInstanceOf(NumberLiteral);
    assert expr.right.isInstanceOf(MultiExpression);
  }
```

Testing the type system

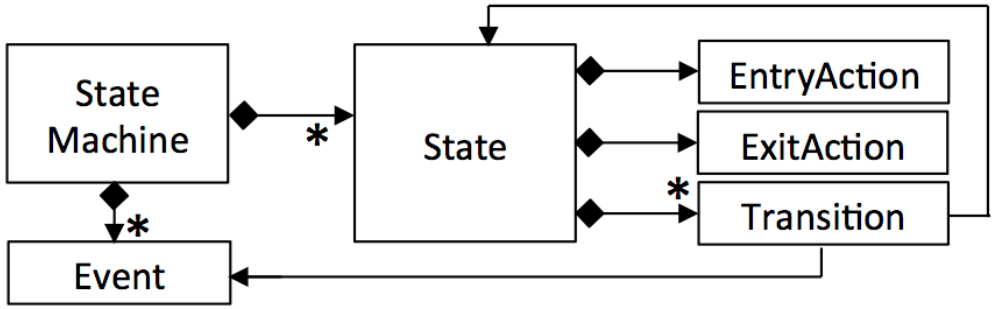
Do typing rules and constraints work?

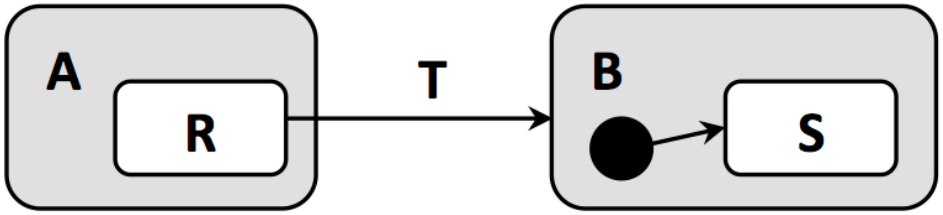
Essentially check if the red squigglies show up where you expect them.

```
Test case testingTypeChecksForLocalVariables
nodes
  ( [ AnImplementationModule    constraints          ] )
                                imports     nothing
    ---------------------------------------------------
    void f() {
      int8 anInt = 42;
      int8 aFloat = <check 3.3 has error>;
      int8 aString = <check "hello" has error>;
    } f (function)
```

Testing semantics

Whenever possible, test semantics by execution and test cases, and not by checking (the structure of) the generated code.






```
statemachine SM initial = S1 {
  in event e()
  in event f()
  state S1 { on e [] -> S2 }
  state S2 { on e [] -> S3 }
  state S3 { on e [] -> S1 }
}

exported testcase testModifyState {
  SM sm;
  assert(0) sm.isInState(S1);
  test statemachine sm {
    e → S2
    f → S2
    e → S3
    e → S1
  }
}
```

Testing semantics

Test case generation:

- Select a set of language extensions
- Randomly generate structurally sound programs
- If the type system finds errors in the program, discard the program (because errors are correctly reported)
- If the program has no errors, try to generate and compile it. If either generation or compilation fails, flag this particular generated program as a failed test.

Helps finding inconsistencies in the multi-stage transformation process.

Version Control

expressions.mps (/Users/markusvoelter/Documents/mbeddr/iets3.core/code/languages/org.iets3.core/solutions/sandbox.core.expr.os/models/sandbox/core/expr/os)

MPS model viewer | Include into commit

sandbox.core.expr.os.expressions

PaperCaseStudy

208c20a51ab2d32f6efc73faa9bc6741c429f07e

Your version

```
inData>) = Σ(i: number = 0, quarters.size) let √(q.pe / |q.rev|) with q = quarters.at(i)
```

```
inData>) = Σ(i: number = 0, quarters.size) √(quarters.at(i).pe / |quarters.at(i).rev|)
```

```
on: number) = cast<focus>(
```

	region == EUROPE	region.in[
time.range[0..6]	12	10
time.range[7..17]	20	22
time.range[18..19]	12	20
time.range[20..24]	12	

```
: number, weight: number) =  <> default: LO
[gender == MALE]
[age < 40]  Old
[age > 40]
```

Works. As long as you are willing to use MPS for diff/merge.

CI Server Integration

We successfully run all our builds, tests and packaging on


However, the effort to get the MPS builds to run there is too high

- the usual build dependency hell
- the MPS build language is not flexible enough
- maybe initially not enough experience with “build stuff”

Also, builds tend to take (too) long.

- a full mbeddr build takes 12 minutes on the server and ca. 30 minutes on a fast developer PC

Mixed bag!

Migration of Models

Languages have version numbers

Automated refactorings increment that number; so can developers.

Migration scripts bring models up to date based on the language versions.

This is already more than in most other tools we know about. But it is clumsy and sometimes slow.

Mixed bag!

RQ 5

Process Integration

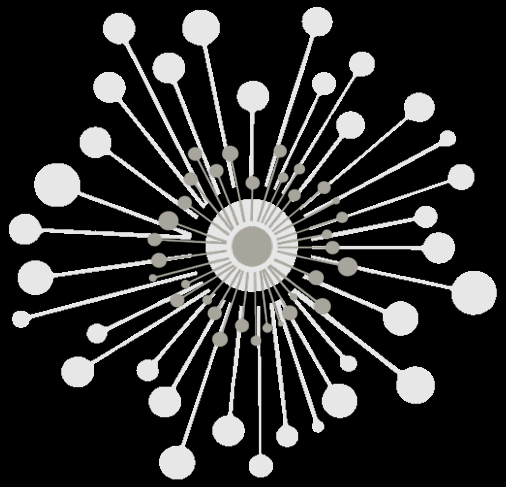
Except for the missing test support for model migrations and single-step transformations, **language testing works well**; we have achieved good coverage as demonstrated by a stable code base.

We have **successfully integrated** mbeddr's build, test and packaging with the Teamcity **CI server**, but the **effort was too high**, partially because of the inadequacy of MPS' build language.

Migrating models as languages change incompatibly **is feasible** with manually scripted migrations and their automatic execution based on implicitly-maintained language version numbers.

Lessons Learned about Language Engineering from the Development of mbeddr

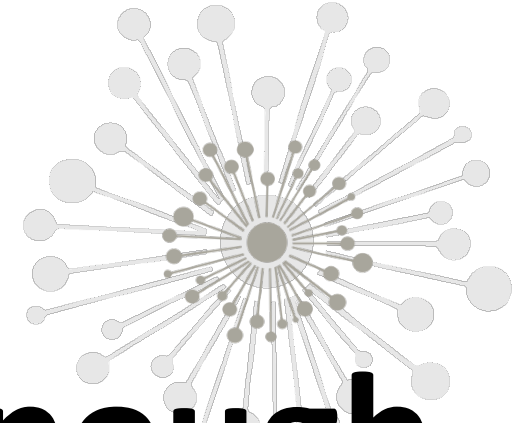
6



Concluding Thoughts

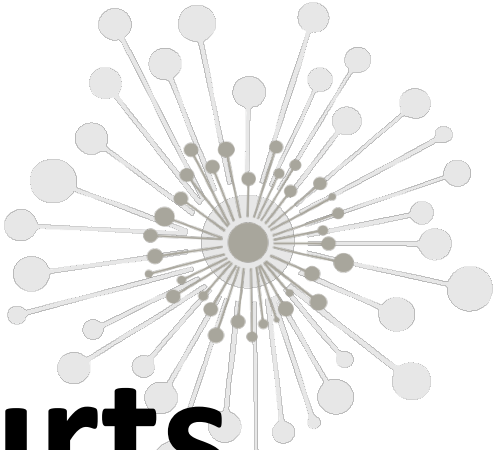
C is a strange language

The details were
harder to build than
we initially thought.
But MPS can do it!



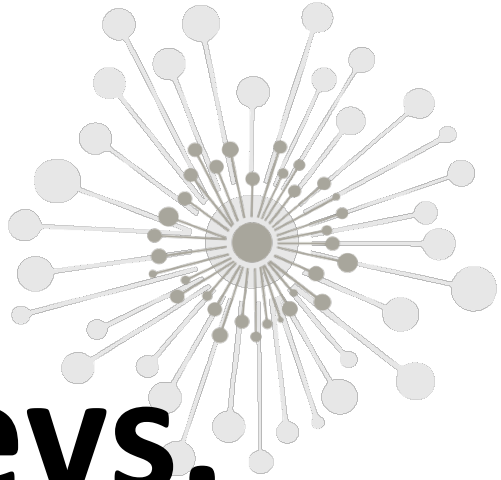
Language is not enough.

IDE, Debugger,
Analyses, Interpreter.



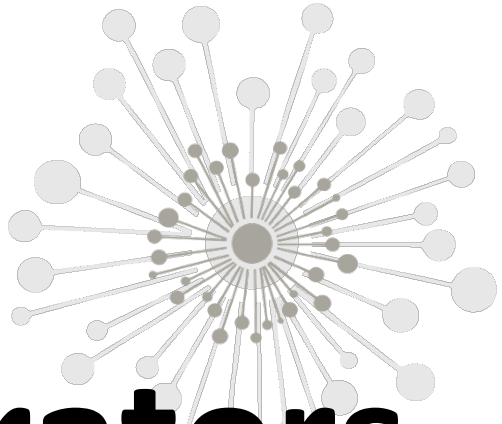
Tool Chrome hurts.

MPS „looks“ too complicated
for the prospective end user.



Onboarding devs.

It took 4 – 8 weeks to get competent new devs on board.



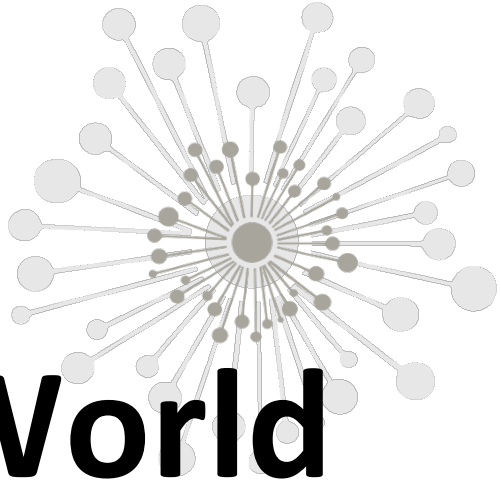
Optimizing Generators

is hard :-)

Modularity vs. Optimization

Global optimization are in conflict with modularity and extensibility.

Perhaps two generators?



Open vs. Closed World

How do you build trust into a system that is inherently extensible?

More Declarativeness

More declarariveness and inproved ability to analyze language definition code would help!

{S} spoofax

Shadow Models

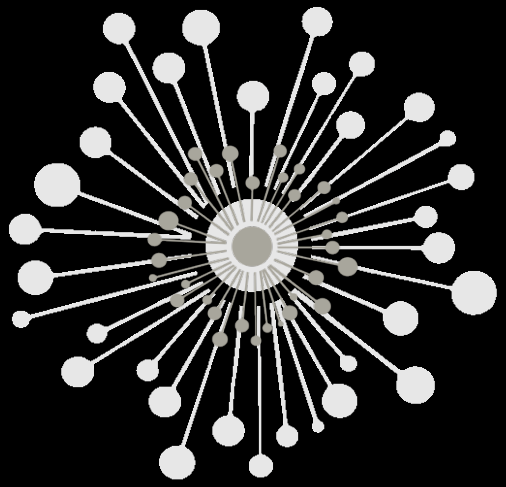
We need incremental, real-time, multi-step, in-IDE transformations for advanced analyses.

Change is hard.

Prospective users were hard to convince to adopt mbeddr.

Lessons Learned about Language Engineering from the Development of mbeddr

7



Wrap Up

MPS is a capable LWB.

Overall we are happy and continue to use it.

It is definitely ready for real-world use

mbeddr – and this evaluation – is testament to that.

Many architectural desicisons are good...

Projection, Aspect DSLs, Bootstrapping.

... but it has some weak points as well.

Internal complexity, imperativeness, build, fat Java client.

A good choice for LE projects today.
Great starting point for next gen LWBs :-)