What's new in the MPS Ecosystem
DSL Development

GPL Extension
Reuse GPL incl. Expressions and TS
Add/Embed DS-extensions
Compatible notational style
Reduce to GPL

New Language
Analyze Domain to find Abstractions
Define suitable, new notations
Rely on existing behavioral paradigm
Reuse standard expression language
Interpret/Generate to one or more GPLs

Formalization
Use existing notation from domain
Clean up and formalize
Generate/Interpret
Often import existing „models“
New Language

Analyze Domain to find Abstractions
Define suitable, new notations
Rely on existing behavioral paradigm
Reuse standard expression language
Interpret/Generate to one or more GPLs
Functional Features

Functional, no state at its core.
Purity + Effect Tracking
The usual types, literals and op’s
Various Conditionals
Functions and Blocks

Error Handling

```
try <e> => <s> error <E-1> => <e-1> ... error <E-n> => <e-n>
```

Immutable Collections and higher-order functions
Enums, tuples, records, all immutable
Constraints on types and functions

+ Algebraic Data Types
+ Datetime
+ Temporal Data
A purely functional language only heats up the processor.

-- Anonymous
**Functional Features**

- Functional, no state at its core.
- Purity + Effect Tracking
- The usual types, literals and op’s
- Various Conditionals
- Functions and Blocks
- Error Handling

```plaintext
attempt<T|E-1,... E-n>
try <e> => <s> error <E-1> => <e-1> ... error <E-n> => <e-n>
```

- Immutable Collections and higher-order functions
- Enums, tuples, records, all immutable
- Constraints on types and functions

**Stateful Features**

- Boxes (like Clojure’s `ref`)
- Transactional Memory
- State Machines
- Interactors
Tooling: Live Execution

```kotlin
val aBool: boolean = true
val anInt: number = 42
val aReal: number{2} = 33.33
val aString: string = "H"

fun f(n: number) where [post res < 10] = 2 * n

test suite PaperDescription

execute automatically : false
Only local declarations: true

test case BasicOperators [fail] {
    assert 42 + 33 equals 75 [1 ms] <number[75|75]{0}>
    assert 42 + 2 * 3 equals 48 [1 ms] <number[48|48]{0}>
    assert aReal + anInt equals 75.33 [1 ms] <number[-∞|∞]{2}>
    assert aBool && true equals true [0 ms] <boolean>
    assert aString + ", W" equals "H, W" [1 ms] <string>
    confail f(20)
    assert if aBool then 42 else 33 equals 44 actual: 42 <number[33|42]{0}>
}
```
Tooling: Coverage

assessment: InterpreterCoverage
query: interpreter test coverage in current model
          problems only: □
          languages: language/org.iets3.core.expr.base/
          ignore: {33 ignored concepts}
sorted: □ must be ok: □ hide ok ones: □
last updated: 2 minutes ago by markusvoelter

<table>
<thead>
<tr>
<th>org.iets3.core.expr.base</th>
</tr>
</thead>
<tbody>
<tr>
<td>TupleValue</td>
</tr>
<tr>
<td>SomeValExpr</td>
</tr>
<tr>
<td>LogicalImpliesExpression</td>
</tr>
<tr>
<td>ErrorExpression</td>
</tr>
<tr>
<td>SuccessExpression</td>
</tr>
<tr>
<td>ParenExpression</td>
</tr>
<tr>
<td>TryExpression</td>
</tr>
</tbody>
</table>

total 38, new 38, ok 0
coverage 95 %
Tooling: Debugging

```scala
assert { val v = getR().addr.first if v.street.length < 3 then "LessThan3" else "Bigger" }
```

- **assert**
  ```scala
  assert { val v = getR().addr.first if v.street.length < 3 then "LessThan3" else "Bigger" } [AssertTestTree]
  ```

- **actual**
  ```scala
  val v = getR().addr.first if v.street.length < 3 then "LessThan3" else "Bigger" ⇒ Bigger : String [ValExpression]
  ```

- **val v ⇒ #Address[Oetz, #City[70327, Street...]] : RecordValue (4 ms) [ValExpression]

- **if v.street.length < 3 then "LessThan3" else "Bigger" ⇒ Bigger : String [IfExpression]
  ```scala
  cond: v.street.length < 3 ⇒ false : Boolean (1 ms) [LessExpression]
  ```

- **else: Bigger ⇒ Bigger : String [StringLiteral]

- **expected: getOetz() ⇒ Oetz : String [FunctionCall]
Tooling: REPL

```
repl Time_repl_0 for node artificialclock(0) in Time

downstream updates □

Reevaluate All  Clear  Clear Tail  Close & Return

[0]  ─ ─ ─ artificialclock(0) : artificialclock, effects[reads]
       artificialclock:5

[1]  ─ ─ $0.advance(5) : artificialclock, effects[modifies]
       artificialclock:5

[2]  ─ ─ $0.time : number[0|∞], effects[reads]
       5

[3]  ─ ─ <no expression>
       <no result found>
```
Tooling: Test Case Generation

Random values + Extreme Values

test case TestFunctions [success] {
  vectors function add -> producer: random 25
  results: true

<table>
<thead>
<tr>
<th>a</th>
<th>a2</th>
<th>b</th>
<th>c</th>
<th>s</th>
<th>res</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>3</td>
<td>-3.06</td>
<td>false BLUE</td>
<td>&quot;</td>
<td>ok</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2.74</td>
<td>false BLUE</td>
<td>&quot;M/Yh-0I/ac&quot;</td>
<td>2</td>
<td>ok</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0.22</td>
<td>false BLUE</td>
<td>&quot;</td>
<td>1.22</td>
<td>ok</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>-0.45</td>
<td>true BLUE</td>
<td>&quot;7l:6h7sg!afLmULGU8wtI00H9&quot;</td>
<td>not executed</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1.38</td>
<td>false RED</td>
<td>&quot;Mtoa7J6GuWTye2f2-fLhD$hj8C2K&quot;</td>
<td>not executed</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>-2.63</td>
<td>false BLUE</td>
<td>&quot;n66r7E (f0J$qaQMj$&quot;</td>
<td>not executed</td>
<td></td>
</tr>
</tbody>
</table>

vectors function plus -> producer: eqclass
results: true

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-10</td>
<td>GREEN</td>
</tr>
<tr>
<td>1</td>
<td>-10</td>
<td>BLUE</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>GREEN</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>BLUE</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>GREEN</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>BLUE</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>[PRE] b &gt; 0</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>[POST] res == a * b</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>[POST] res == a * b</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
<td>[POST] res == a * b</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>[PRE] b &gt; 0</td>
</tr>
<tr>
<td>11</td>
<td>9</td>
<td>[POST] res == a * b</td>
</tr>
</tbody>
</table>
fun doodle(a: number[1|5]) = if true then a else a * 1

fun add(a: number[1|5]) = alt \[ a > 3 \implies a + 1 \\
\text{otherwise} \implies \text{doodle}(a) \]

test case TestFunctions [incomplete] {
  vectors function add
  results: true
  mutator: # of mutations 20
    keep all: false
    \rightarrow \text{ParenExpression}
    \rightarrow \text{NumberLiteral}
    \rightarrow \text{LogicalNotExpression}
    \rightarrow \text{ParenExpression}
    \rightarrow \text{ParenExpression}
}

\rightarrow \text{producer: random 30}
\{ 30 entries \}
fun add(a: number[1|5]) = alt
                    \[
                    \begin{cases}
                    ((a) + 1) > 3 & \Rightarrow a + 1 \\
                    a & \\
                    \text{otherwise} & \Rightarrow \text{doodle}(a)
                    \end{cases}
                    \]

fun doodle(a: number[1|5]) = if true then a else a * 2

fun doodle(a: number[1|5]) = if true then a else a * \[
                    \begin{cases}
                    (a \ast 1 + 1) & \\
                    a \ast 1 &
                    \end{cases}
                    \]

fun doodle(a: number[1\|5]) = if !(true) then a else a \ast 1

true
## Example Kf Extensions

*Used in Systems Engineering, Healthcare, Finance and Smart Contracts (Experimental)*

<table>
<thead>
<tr>
<th>IDE</th>
<th>Salary/Tax</th>
<th>Smart Contracts</th>
<th>Healthcare</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Alternative Rules</td>
<td>Live Values</td>
<td>Simulator with Phone UI</td>
</tr>
<tr>
<td></td>
<td>Rules for Data Items</td>
<td>Diff of Live Values</td>
<td>Debugger for State Machine</td>
</tr>
<tr>
<td></td>
<td>Projection for a given Date</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Structure</td>
<td>Basic Data</td>
<td></td>
<td>Time Series Components</td>
</tr>
<tr>
<td></td>
<td>Result Data.</td>
<td></td>
<td>Test Scenarios</td>
</tr>
<tr>
<td>Behaviors</td>
<td>Calculation Rules &amp; Dependencies</td>
<td></td>
<td>(Special) State Machines</td>
</tr>
<tr>
<td></td>
<td>Variants</td>
<td></td>
<td>Test Generation &amp; Mutation</td>
</tr>
<tr>
<td></td>
<td>Validity</td>
<td></td>
<td>Coverage Measurement Utils</td>
</tr>
<tr>
<td>Functional &amp; Types</td>
<td>Date Types</td>
<td></td>
<td>Time &amp; Duration Types</td>
</tr>
<tr>
<td></td>
<td>Currency Types</td>
<td></td>
<td>Decision Trees and Tables</td>
</tr>
<tr>
<td></td>
<td>Temporal Types</td>
<td></td>
<td>Sentence-like Function Calls</td>
</tr>
</tbody>
</table>

Language-Specific Extensions to Functional Debugging/Tracing and Testing
Languages Used

**Configuration**
for visualizations, simulations and documentation

**State Machines**
components, parameters, instantiation, states, transitions, events

**Test and Scenarios**
test vector generation, algo instantiation, user interaction/events

**KernelF Extensions**
decision trees and tables, dealing with time

**KernelF Expressions**
binary ops, if-then-else, primitive types and literals, collections
What good is all the abstraction if we cannot trust the translation to the implementation?
What good is all the abstraction if we cannot trust the translation to the implementation?

System Architecture & Safety Standards

Tools may introduce additional systematic errors if faulty. Safety standards require reliable mitigation of such errors.

DO-178C  EN50129  IEC62304  ISO26262
What good is all the abstraction if we cannot trust the translation to the implementation?
Unqualified Tools!

End-to-end testing required. How to do this without exploding effort?

Automated Redundancy

catch errors in redundant path while reducing manual effort. + specific risk mitigations
Modeling Architecture

Model the Algo/System with the DSL and also model the tests/verification. Then translate both and execute on the level of the implementation.

+ Risk Analysis  + Mitigations
Risk Analysis

- Compensating
- Related

Requirements
- Model $M_S$
- Model $M_T$
- Language
- Language

- Random Error
- Systematic Error
- Error
- Insufficient Resources

- Trafo $T_S$
- Trafo $T_T$

Transformation Engine
- Error

- CSC Program
- CSC Tests

Runtime System
- Error

Random Error
Mitigations – Safe Modeling Architecture
Using language workbenches and domain-specific languages for safety-critical software development

Authors
Markus Voelter, Bernd Kolb, Klaus Birken, Federico Tomassetti, Patrick Alff, Laurent Wiart, Andreas Wortmann, Arne Nordmann

Regular Paper
First Online: 17 May 2018

Abstract

Language workbenches support the efficient creation, integration, and use of domain-specific languages. Typically, they execute models by code generation to programming language code. This can lead to increased productivity and higher quality. However, in safety-/mission-critical environments, generated code may not be considered trustworthy, because of the lack of trust in the generation mechanisms. This makes it harder to justify the use of language workbenches in such an environment. In this paper, we demonstrate an approach to use such tools in critical environments. We argue that models created with domain-specific languages are easier to validate and that the additional risk resulting from the transformation to code can be mitigated by a suitably designed transformation and verification architecture. We validate the approach with an industrial case study from the healthcare domain. We also discuss the degree to which the approach is appropriate for critical software in space, automotive, and robotics systems.
Realtime Incremental Transformations
What is Shadow Models?

Functional **transformation language** with support for fixpoints

**Incremental** execution upon change of input model

Unidirectional, but with first-class support for **lifting** results

Fully integrated into **MPS IDE**

**Code:** https://github.com/JetBrains/MPS-extensions/tree/master/code/shadowmodels

**Docs:** https://jetbrains.github.io/MPS-extensions/extensions/shadowmodels/
User **edits** the input model

A **delta** is propagated into the transformation engine

A **change** on the shadow model is produced

This **triggers** analysis or update of results in Shadow Model

Results/messages are **lifted** back to input level

Results are **annotated** to the input model
```
program Example2 { 
  enum Color = red, green, yellow, yellow; 
  fun decide(a: int, b: int) = alt [ 
    a > b => 1, 
    a == b => 2, 
    otherwise => 3 
  ]; 
  | ?maybe? | 2 | 4 | 6 |
}
```
Example Shadow

```kotlin
program Example2 {

val Color_red  = -309736043
val Color_green = -1313361145
val Color_yellow = -1555955472

fun decide(a: int, b: int) = if a > b
    then 1
    else if a == b then 2 else 3

val res = if ?maybe?
    then if ?maybe?
        then 1
        else if ?maybe? then 4 else 
    else if ?maybe? then if ?maybe?
        then 3
        else if ?maybe? then 4 else 
    else if ?maybe? then if ?maybe?
        then 5
        else if ?maybe? then 6 else 
    else 
}
```
Example Use Cases

- Mapping of system models to **model checkers** for interactive verification of temporal properties
- **Flattening** of component hierarchies for type checking
- Flattening of hierarchical **feature models** for path expression evaluation
- Weaving in of **safety concerns** into C code.
- **Desugaring** of business DSLs to a core functional language; that core language would then feature an interpreter, a compiler and an integration with an SMT solver
Use Case: Kf2

- Minimal, Expressive Core
  - Functional
  - Reactive
- Sugar on top
- Interpreter, Generator and Verifier below
- Realtime-Trafo with Shadow Models
void PIControlTask() {
    // sample current spacings between the cars
    float volatile curr_d = readDistanceSensor();
    // sample current speed of the car
    float volatile curr_v = readSpeedSensor();
    // error integration using Newton method
    static float error_integral = 0.0f;
    error_integral += (curr_d - DESIRED_DISTANCE);
    // applying control law
    float throttle_u = PI(curr_d, curr_v, error_integral);
    // writing to the actuator
    writeActuator(throttle_u);
} PIControlTask (function)
void PIControlTask()
{
    // sample current spacings between the cars
    float volatile curr_d = readDistanceSensor();
    // sample current speed of the car
    float volatile curr_v = readSpeedSensor();
    // error integration using Newton method
    static float error_integral = 0.0f;
    error_integral += (curr_d - DESIRED_DISTANCE);
    // applying control law
    float throttle_u = PI(curr_d, curr_v, error_integral);
    // writing to the actuator
    writeActuator(throttle_u);
} PIControlTask (function)
Use Case: Safety Patterns

```c
void PIControlTask()
{
    // sample current spacings between the cars
    float volatile curr_d = readDistanceSensor();
    // sample current speed of the car
    float volatile curr_v = readSpeedSensor();
    // error integration using Newton method
    static float error_integral = 0.0f;
    error_integral += (curr_d - DESIRED_DISTANCE);
    // applying control law
    float throttle_u = PI(curr_d, curr_v, error_integral);
    // writing to the actuator
    writeActuator(throttle_u);
} PIControlTask (function)
```

SAFETY-PATTERN: Safe Function - StatementDuplication

### Configuration

**Delta constraints**

```c
const
    // mmap of custom VP
    #alias NOTIFY_DC_DETECTION = hex<ffffff06u>;
```

**Delta function**

```c
    static int8 cnt = 0;
    if (cnt == 10) {
    if ( Delta > 8 ) || ( Delta < -8 )) {
        //if (int8 volatile) (NOTIFY_DC_DETECTION)) = 1;
    if
    } else {
        cnt++;
    if
    }
```

**Required capabilities**

Avoid pass-by-reference

void PIControlTask()
{
    // sample current spacings between the cars
    float volatile curr_d = readDistanceSensor();
    // sample current speed of the car
    float volatile curr_v = readSpeedSensor();
    // error integration using Newton method
    static float error_integral = 0.0f;
    error_integral += (curr_d - DESIRED_DISTANCE);
    // applying control law
    float throttle_u = PI(curr_d, curr_v, error_integral);
    // writing to the actuator
    writeActuator(throttle_u);
} PIControlTask (function)

void PIControlTask()
{
    // sample current spacings between the cars
    float volatile curr_d = readDistanceSensor();
    // sample current speed of the car
    float volatile curr_v = readSpeedSensor();
    // error integration using Newton method
    static float error_integral = 0.0f;
    error_integral += (curr_d - DESIRED_DISTANCE);
    // applying control law
    float throttle_u = PI(curr_d, curr_v, error_integral);
    // writing to the actuator
    // sample current spacings between the cars
    float volatile curr_d_dup = readDistanceSensor();
    // sample current speed of the car
    float volatile curr_v_dup = readSpeedSensor();
    // error integration using Newton method
    static float error_integral_dup = 0.0f;
    error_integral_dup += (curr_d_dup - DESIRED_DISTANCE);
    // applying control law
    float throttle_u_dup = PI(curr_d_dup, curr_v_dup, error_integral_dup);
    // writing to the actuator
    
    float delta = throttle_u_dup - throttle_u;
    static int8 cnt = 0;
    if (cnt == 10) {
        if ((DELTA > 8) || (DELTA < -8)) {
            *((int8 volatile*) (NOTIFY_DC_DETECTION)) = 1;
        } else {
            cnt++;
        }
    } else {
        cnt++;
    }
    writeActuator(throttle_u_dup);

    Required_capabilities
    Avoid pass-by-reference
Use Case: Interactive FM Specialization

Klaus Birken

![Diagram showing Vehicle, Assistants, ParkAssistant, RearCamera, LaneAssistant, Engine, Combustion, Electrical, Hybrid.]

**Configuration Cfg10 for Vehicle**
- **M Assistants**
  - O ParkAssistant
  - O RearCamera
  - O LaneAssistant
- **M Engine**
  - A Combustion
  - A Electrical
  - A Hybrid
Use Case: Interactive FM Specialization

Klaus Birken

Configuration Cfg10 for Vehicle
- M Assistants
  - O ParkAssistant
  - O RearCamera
  - O LaneAssistant
- M Engine
  - A Combustion
  - A Electrical
  - A Hybrid

Uncheck Hybrid
Use Case: Interactive FM Specialization

Klaus Birken

Configuration Cfg10 for Vehicle
- M Assistants
  - O ParkAssistant
  - O RearCamera
  - O LaneAssistant
- M Engine
  - A Combustion
  - X A Electrical
  - X A Hybrid

Vehicle_modulo_Cfg10
- Assistants
  - c: RearCamera => ParkAssistant

ParkAssistant  RearCamera  LaneAssistant

Check
Combustion
Use Case: Interactive FM Specialization

Klaus Birken

Configuration Cfg10 for Vehicle

- M Assistants
  - O ParkAssistant
  - O RearCamera
  - ? O LaneAssistant
- M Engine
  - A Combustion
  - X A Electrical
  - X A Hybrid

Check RearCamera

Vehicle_modulo_Cfg10

- Assistants
  - LaneAssistant
Current State (March 2019)

Still under active development.
Primarily by Sascha Lisson.

Used for initial use cases at itemis Stuttgart.
Dclare for MPS
Declarative Rules for MPS

(Modeling Value Group)
Unidirectional.
Bidirectional requires complementary rules

Granularity is per node/property combination

Dynamic read-tracking

No scalability numbers yet.

Open Source.

https://github.com/ModelingValueGroup/DclareForMPS
Plain Text Gen

„Template-engine-style“ text generator on top of MPS textgen

Abstractions for words, lines

Paste in literal text then selectively replace with string-typed expressions

(Schindler Universe)
Meetup 2019 tentatively scheduled:
Oct 10/11 in Amsterdam

Community Guidelines online:
https://confluence.jetbrains.com/display/MPS/Community+Guidelines+of+MPS

Support Contract available for serious users
https://www.jetbrains.com/mps/support/partnership.html

Continued evolution of MPS:
• Development has obviously slowed down.
• Many of the team members work on apps on top of MPS (or support paying customers).
• Team is planning to hire two additional devs.
• Future evolution increasingly aligned with the needs of (support-paying) users.
• A process for coordinating them is being designed.