Supporting Diverse Notations In MPS’ Projectional Editor

Markus Völter and Sascha Lisson

with Bernd Kolb, Domenik Pavletic, Kolja Dumman, Tamas Szabo, Niko Stotz, Dan Ratiu, Zaur Molotnikov,
Languages, Notations, Models, Programs
<table>
<thead>
<tr>
<th></th>
<th>more in GPLs</th>
<th>more in DSL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain Size</td>
<td>large and complex</td>
<td>smaller and well-defined</td>
</tr>
<tr>
<td>Designed by</td>
<td>guru or committee</td>
<td>a few engineers and domain experts</td>
</tr>
<tr>
<td>Language Size</td>
<td>large</td>
<td>small</td>
</tr>
<tr>
<td>Turing-completeness</td>
<td>almost always</td>
<td>often not</td>
</tr>
<tr>
<td>User Community</td>
<td>large, anonymous and widespread</td>
<td>small, accessible and local</td>
</tr>
<tr>
<td>In-language abstraction</td>
<td>sophisticated</td>
<td>limited</td>
</tr>
<tr>
<td>Lifespan</td>
<td>years to decades</td>
<td>months to years (driven by context)</td>
</tr>
<tr>
<td>Evolution</td>
<td>slow, often standardized</td>
<td>fast-paced</td>
</tr>
<tr>
<td>Incompatible Changes</td>
<td>almost impossible</td>
<td>feasible</td>
</tr>
</tbody>
</table>
Model or Code?

```c
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;
}
```
beforeFlight

next [alt > 0]

reset

crashed

next [alt == 0 && speed == 0]

airborne

flying

next [alt == 0 && speed > 0]

landing

next [alt == 0 && speed == 0]

landed

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?

```c
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?
GEMOC 2014

To cope with complexity, modern software-intensive systems are often split in different concerns, which serve diverse stakeholder groups and thus must address a variety of stakeholder concerns. These different concerns are often associated with specialized description languages and technologies, which are based on concern-specific problems and solution concepts.
To cope with complexity, modern software-intensive systems are often split in different concerns, which serve diverse stakeholder groups and thus must address a variety of stakeholder concerns. These different concerns are often associated with specialized description languages, which concern-specific solution concepts.
To cope with complexity, modern software-intensive systems are often split in different concerns, which serve diverse stakeholder groups and thus must address a variety of stakeholder concerns. These different concerns are often associated with specialized description languages, which are concern-specific notations.
[Diverse Notations]

Regular Code/Text

Graphical
[Diverse Notations]

Regular Code/Text

Mathematical

Tables

Graphical
[Diverse Notations]

Regular Code/Text

Mathematical

Tables

Graphical
[Diverse Notations]
An extensible set of integrated languages for embedded software engineering.

Specific Languages

<table>
<thead>
<tr>
<th>Core</th>
<th>Default Extensions</th>
<th>User Extensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>C99</td>
<td>Test Support</td>
<td>Glossaries</td>
</tr>
<tr>
<td>Model Checking</td>
<td>Decision Tables</td>
<td>Use Cases &amp; Scenarios</td>
</tr>
<tr>
<td>SMT Solving</td>
<td>Logging &amp; Tracing</td>
<td></td>
</tr>
<tr>
<td>Dataflow Analysis</td>
<td>State Machine Verification</td>
<td></td>
</tr>
<tr>
<td>Visualization</td>
<td>Component Contracts</td>
<td></td>
</tr>
<tr>
<td>PLE Variability</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Documentation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Requirements &amp; Tracing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reports &amp; Assessments</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Platform

JetBrains MPS

Backend Tool

C Compiler, Debugger and Importer
NuSMV
Yices
CBMC
PlantUML
LaTeX

Implementation Concern
Analysis Concern
Process Concern
Research Project 2011 - 2013

Open Source @ eclipse.org
Eclipse Public License 1.0
http://mbeddr.com

Commercial Use and Extension

Research Platform
JetBrains MPS
Open Source
Apache 2.0
http://jetbrains.com/mps
Language Workbench

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

Diagram:
- Language
  - Structure: Concepts, Properties, Inheritance, Relationships
  - Editor: Projection Rules, Side Transformations, Intentions
  - Type System: Typing Rules, Type Checks, Other Validations
  - Transformations: Reduction Rules, Weaving Rules, Transformation Priorities
- Constraints: Scopes, Usage Restrictions, Property Value Limitations

Language generates to Language extends 0..*
Projectional Editing
[Projectional Editing]

Parsing

- Concrete Syntax
- Abstract Syntax Tree

Projectional Editing

- Concrete Syntax
- Abstract Syntax Tree
[Projectional Editing]
Language Composition

Separate Files
Type System
Transformation
Constraints

In One File
Type System
Transformation
Constraints
Syntax
IDE
[Projectional Editing]
Language Composition

Separate Files
Type System
Transformation
Constraints

In One File
Type System
Transformation
Constraints
Syntax
IDE

50+ extensions to C
10+ extensions to requirements lang.

mbeddr
[Projectional Editing]
Syntactic Flexibility

Regular Code/Text

Mathematical

Tables

Graphical
Projectional Editing
Syntactic Flexibility

Regular Code/Text

```c
// 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];
    }
}
aSummingFunction (function)
```

Mathematical

```c
double midnight2(int32 a, int32 b, int32 c) {
    return \[-b + \sqrt{b^2 - \sum_{i=1}^{4} a * c}\] / 2 * a;
}
midnight2 (function)
```

Tables

```c
int16 decide(int8 spd, int8 alt) {
    return spd > 0 ? spd > 100 ? 1 : 100 : 0;
    alt < 0 : 1
    alt == 0 : 10
    alt > 0 : 30, 40
    alt > 100 : 50, 60
}
```

decide (function)

Graphical

```
cust 1
```
```
Contract
starts: date
ends: date
```
```
Tariff
attributes
```
```
Cst.Customer
```
Thank you!