Language Workbenches Opportunities and Challenges for V&V

Markus Völter

Bernd Kolb, Dan Ratiu, Zaur Molotnikov, Domenik Pavletic, Kolja Dumman, Sascha Lisson, Tamas Szabo, Niko Stotz,
Languages, Models, Programs

Language Workbenches

JetBrains MPS

mbeddr

V&V in mbeddr

Opportunities & Challenges
Languages, 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>
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

airborne

flying

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

landing

reset

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

landed

Model or Code?
Model or Code?

```plaintext
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?

```plaintext
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
with many first class concepts!
Small Language

with a few, orthogonal and powerful concepts
Modular Language

with many optional, composable modules
Language Workbenches
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 and modeling
Open Source
Apache 2.0
http://jetbrains.com/mps
Language Workbench

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

Parsing

Concrete Syntax

Abstract Syntax Tree

Projectional Editing

Concrete Syntax

Abstract Syntax Tree
[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

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

Tables

```plaintext
int16 decide(int8 spd, int8 alt) {
  return spd > 0 spd > 100 otherwise 0;
  alt < 0 1 1
  alt == 0 10 20
  alt > 0 30 40
  alt > 100 50 60
} decide (function)
```
[Projectional Editing]
Language Composition

Separate Files

L2 → L1

Type System
Transformation
Constraints

In One File

Type System
Transformation
Constraints
Syntax
IDE

50+ extensions to C
10+ extensions to requirements lang.
mbeddr
An extensible set of integrated languages for embedded software engineering.

<table>
<thead>
<tr>
<th>User Extensions</th>
<th>Default Extensions</th>
<th>Core</th>
<th>Platform</th>
<th>Backend Tool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Support</td>
<td>Decision Tables</td>
<td>C99</td>
<td>JetBrains MPS</td>
<td>C Compiler, Debugger and Importer</td>
</tr>
<tr>
<td>Decision Tables</td>
<td>Logging &amp; Tracing</td>
<td>State Machine Verification</td>
<td></td>
<td>NuSMV</td>
</tr>
<tr>
<td>Logging &amp; Tracing</td>
<td>State Machines</td>
<td>Decision Tables</td>
<td></td>
<td>Yices</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Component Contracts</td>
<td></td>
<td>CBMC</td>
</tr>
<tr>
<td>Components</td>
<td>Physical Units</td>
<td>Component Contracts</td>
<td></td>
<td>PlantUML</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Glossaries</td>
<td></td>
<td>LaTeX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Use Cases &amp; Scenarios</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>to be defined by users</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Implementation Concern | Analysis Concern | Process Concern

Specific Languages”
Research Project 2011 - 2013

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

Commercial Use and Extension

Research Platform
itemis France: Smart Meter

First significant mbeddr project
ca. 100,000 LoC
about to be finished
great modularity due to components
uses physical units extensively
great test coverage due to special extensions
LMS INTERNATIONAL
Researchpark Haasrode 1237 | Interleuvenlaan 68 | B-3001 Leuven [Belgium]
T +32 16 394 200 | F +32 16 394 350 | info@lmsintl.com | www.lmsintl.com

Worldwide
For the address of your local representative, please visit www.lmsintl.com/Lmsworldwide

LMS is a leading provider of test and mechatronic simulation software and engineering services in the automotive, aerospace and other advanced manufacturing industries. As a business segment within Siemens PLM Software, LMS provides a unique portfolio of products and services for manufacturing companies to manage the complexities of tomorrow's product development by incorporating model-based mechatronic simulation and advanced testing in the product development process. LMS tunes into mission-critical engineering attributes, ranging from system dynamics, structural integrity and sound quality to durability, safety and power consumption. With multi-domain and mechatronic simulation solutions, LMS addresses the complex engineering challenges associated with intelligent system design and model-based systems engineering. Thanks to its technology and more than 1250 dedicated people, LMS has become the partner of choice of more than 5000 manufacturing companies worldwide. LMS operates in more than 30 key locations around the world.
20+ Projects in various stages by various “Big Name” companies.

Branching out into the Domains Finance, Insurance, Requirements
[Accidental Benefits]

Better Abstractions & Notations
Easier to Review and Validate
Better Modularity (Interf. / Comp.)
Improved Testability
Domain-Specific Testing Support
Makes Tests easier to Validate
Domain-Specific Debugging
Abstractions are not Leaky!
V&amp;V in mbeddr
Unit Checking
Custom Constraints

```c
void calcVerticalSpeed(TrackpointWithVertical* prev,
    if (prev == null) {
        cur->vSpeed = 0 mps;
    } else {
        int16_t Error: type int16 /s · m^(−1)/ is not a subtype of int16 /mps/
        int8_t dTime = cur->time - prev->time;
        cur->vSpeed = dTime / dAlt;
    }
} calcVerticalSpeed (function)
```
DEMO

Units; Layers
Domain-Specific C Verification

Synergy between Language Engineering & Verification
Domain-Specific C Verification

Alternative 1

1. Make Program Expressive
2. Verify on Model Level
3. Report on Model Level
Domain-Specific C Verification

Alternative 2

1. Make Program Expressive
2. Translate to C
3. Verify on C Level
4. Lift Results Back
DEMO

Component Contracts
state machine VVI initial = Start
  in config(int lri, int vrp)
  in s
  in t
  out p => doPace()
  int c = 0
  int LRI = 0
  int VRP = 0
state Start {
  on config -> Start {
    LRI = lri;
    VRP = vrp; }
  on t -> Pace }
state Wait {
  on s [c <= VRP] -> Wait
  on s [c > VRP] -> Wait {
    c = 0; }
  on t [c < LRI] -> Wait {
    ++c; }
  on t [c == LRI] -> Pace }
state Pace {
  on entry {
    send p;
    c = 0; }
  on t -> Wait {
    ++c; }
}
typedef enum states {
  Start, Wait, Pace
} VVI_States;
typedef enum events {
  config, s, t
} Events_VVI;
void execute(Events_VVI e, int** args)
{
  switch (state) {
    case Start: {
      switch (e) {
        case config: {
          LRI = *args[0];
          VRP = *args[1];
          state = Start;
          return; }
        case t: {
          state = Pace;
          doPace();
          c = 0;
          return; }
        case s: {
          if (c <= VRP) {
            state = Wait;
            // ...
          }
        }
      }
    }
    case Wait: {
      switch (e) {
        case t: {
          if (c < LRI) {
            ++c;
            return; }
        }
        case s: {
          if (c == LRI) {
            state = Pace;
            doPace();
            c = 0;
            return; }
        }
      }
    }
  }
}
[DSCV] VCs

\[\text{after} \ \text{smIsInState}(\text{Wait}) \quad \text{until} \ \text{smIsInState}(\text{Pace}) \quad \text{exists} \ c \ == \ LRI\]

\[
\begin{align*}
\text{static int } q & = 0; \\
\text{static int } e & = 1; \\
\text{if} \ (\text{state} == \text{Pace}) \quad \{ \\
\quad q & = 0; \\
\quad \text{assert}(e); \\
\quad e & = 1; \\
\} \quad \text{if} \ (q) \quad \{ \\
\quad e & = (c == \text{LRI}) || e; \\
\} \quad \text{if} \ (\text{state} == \text{Wait} \\
\quad && \&\& \ !q) \\
\quad \{ q = 1; \ e = 0; \}
\end{align*}
\]

A

\[
\text{nondet_choice} \quad \{ \\
\quad \text{choice} \quad \{ \\
\quad \quad \text{smtrigger}(s); \\
\quad \} \quad \text{if} \ (\text{ch} == 0) \quad \{ \\
\quad \quad \text{execute}(s, 0); \\
\quad \}\quad \text{if} \ (\text{ch} == 1) \quad \{ \\
\quad \quad \text{// nothing} \\
\quad \}
\}
\]

B

\[
\begin{align*}
\text{smStateSubset} \ \text{Initial:} \\
\quad \text{smInState}(\text{Pace}) \\
\quad && \&\& \ \text{VRP} < \text{LRI}; \\
\quad \text{smNonDetInit}(\text{Initial}); \\
\quad \text{state} & = (\text{VVI_States}) \quad \text{nondet_int}(); \\
\quad \text{assume}(0 <= \text{state} \\
\quad && \&\& \ \text{state} <= 2); \\
\quad c & = \text{nondet_int}(); \\
\quad \text{LRI} & = \text{nondet_int}(); \\
\quad \text{VRP} & = \text{nondet_int}(); \\
\quad \text{assume}(\text{state} == \text{Pace} \\
\quad && \&\& \ \text{VRP} < \text{LRI});
\end{align*}
\]

C
induction on t
from Initial
until MAX_LRI {
    nondeterministically smtrigger(s);
    after smIsInState(Wait)
    until smIsInState(Pace)
    exists c == LRI }

smNonDetInit(Initial);
for (int step = 0; step < MAX_LRI; ++step) {
    nondeterministically smtrigger(s);
    after smIsInState(Wait)
    until smIsInState(Pace)
    exists c == LRI;
    smtrigger(t);
    after step > 1 until step == MAX_LRI
    exists smIsInStateSubset(Initial); }
Trustworthiness of Results
Hiding Low-Level Details
Efforts for Building Extensions
End-User Workflow
Extensibility
Other Languages
[DSCV vs. X]

Code-Level Verification Tools
High-Level Spec Languages
Modeling/Verification Tools
[Validation]

(Partially formal) Requirements

4 | Points you get for each trackpoint
InFlightPoints /functional: tags

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. Ut nec justo sapien, vel condimentum velit. Quisque venenatis faucibus tellus consequat rhoncus. Vestibulum dapibus dictum vulputate. Phasellus rhoncus quam eu dui dictum sollicitudin. Duis tempus justo magna. Nunc lobortis libero sed eros interdum aliquet ele. It uses @req(PointFactory) sdf @cfmod(ArchitecturalComponents) to calculate the total points.

calculation PointForATrackpoint: This rule computes the points awarded for a Trackpoint. It does so by taking into account the @alt and the @speed passed as arguments.

parameters:

| int16 alt: current altitude of the trackpoint |
| int16 speed: current speed of the trackpoint |

result = (BASEPOINTS * 1) *

<table>
<thead>
<tr>
<th>alt &gt; 2000</th>
<th>alt &gt; 1000</th>
<th>otherwise 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>speed &gt; 180</td>
<td>30</td>
<td>15</td>
</tr>
<tr>
<td>speed &gt; 130</td>
<td>10</td>
<td>20</td>
</tr>
</tbody>
</table>

tests: PointForATrackpoint(500, 100) == 0
PointForATrackpoint(500, 1200) == 0
PointForATrackpoint(1100, 165) == 200
PointForATrackpoint(2100, 140) == 100
PointForATrackpoint(2100, 200) == 300
[Validation]

Ubiquitous Tracing

```java
#constant TAKEOFF = 100; -> implements PointsForTakeoff
#constant HIGH_SPEED = 10; -> implements FasterThan100
#constant VERY_HIGH_SPEED = 20; -> implements FasterThan200
#constant LANDING = 100; -> implements FullStop
```

```java
exported statemachine FlightAnalyzer initial = beforeFlight {
    in event next(Trackpoint* tp) <no binding>
    [in event reset() <no binding>] -> implements ExampleWithDependencies
    [out event crashNotification() => raiseAlarm] -> implements ExampleWithDependencies
    readable var int16 points = 0
    state beforeFlight {
        entry { points = 0; }
        on next [tp->alt > 0 m] -> airborne
        [exit { points += TAKEOFF; }] -> implements PointsForTakeoff
    }
}
```
[Validation]

... also from Verification Conditions

// [ Assures that the atria are paced timely ]
[ after timesAP > 0 =>
  lastTimeA - lastTimeV <= AEI - 6; ]

Nicely integrated V&V
DEMO
Tracing
Easier to write verifiable code.
Easier to write the VCs.
Results are more understandable.

Some/many tool details hidden.

=> Potentially increases the use of verification in practice.
Identify interesting properties. Make them easily expressible. Integrate the analyses (tools). Lift the Results.

=> A systematic approach for integrating V&V into the IDEs
How do we verify the languages, tools, transformations?

Currently: testing with enough coverage is „good enough“.

=> In the future, a more formal way of lang definiton is needed.
5 Opportunities & Challenges
Good Experience.

Neutral Observation

Problem/Challenge
Default extensions are useful, in particular components, state machines and units.
Easy and useful to add customer/project specific extensions.
The Tracing problem is solved – at least from a technical perspective.
Decided not to make extensions BL independent, they are actually C extensions and cannot be used with other base languages.
Integration with analysis tools work and is useful, but performance and config of the analysis is still an issue. (leaky abstraction)
Do more verification on code level than on model level because of consistency problem with code.
Verification should be done incrementally – Verification-driven Development (VDD)
Splitting C into several languages not so useful – dependencies!
Modularity works in principle and practice
MPS’ approach scales to non-trivial and many languages.
Flexible notations actually work and are useful in practice.
Decoupling Notation from Language works.
MPS is easily extensible with new notational styles.
VCS integration works well (diff/merge)
MPS can be extended with the same means – bootstrapped.
MPS also supports debugging of DSLs – even though we had to extend the mechanism
No direct support for detecting semantic interactions between languages
Modularity: Sometimes base language requires change (introduction of abstract class or interface)
Ability to create additional language aspects missing (you can existing ones)
Debugger definition separate from generator; leads to duplication
many aspects of language definition too "procedural" and hence hard to analyze.
Due to the open world assumption of MPS, there is a "feeling of incompleteness" in aspects like e.g. in lifting analyses results.
Thank you!