Build your own
Language
Why and How?
Motivation
I just finished v3 of the requirements document. But I am sure it will take another two months of ping-pong with IT to get the damn thing to run.

Aargh, another half-baked requirements document. Those guys always rely on us to “debug” it and make it work.
The IT guys have decided to port the system to mobile phones. We have to do another re-write/-understand of all the Fachlichkeit. Again!!

Well, yes, but we have to keep up with the evolving technologies and new platforms. No way around it!
Decouple Fachlichkeit and Technology!
so you can evolve both independently.

Represent Fachlichkeit precisely/formally,
so you can analyze, test, simulate.

Use “friendly“ languages,
so domain experts can contribute directly.

DSL
Domain Specific Language
Examples
Insurance Contracts

Write formal code in a DSL mixed with tables and text.

Now with IDE support and executable tests.

The same notation!
## Unterhaltsvorschuss

<table>
<thead>
<tr>
<th>Zeitangabe:</th>
<th>laufend</th>
</tr>
</thead>
<tbody>
<tr>
<td>Häufigkeit:</td>
<td>monatlich einmal</td>
</tr>
<tr>
<td>Leistungskontext:</td>
<td>Leer</td>
</tr>
<tr>
<td>Zählar:</td>
<td>uvg</td>
</tr>
<tr>
<td>Anspruch Beginn:</td>
<td>Anfang – Unbegrenzt: junger Mensch.geburtsdatum</td>
</tr>
<tr>
<td>Anspruch Ende:</td>
<td>01.01.1800 - 31.12.9999 : min(junger Mensch.geburtsdatum + 12 Jahre, datum + 72 Monate – Anzahl Monate mit uvg)</td>
</tr>
<tr>
<td>Zeitraum für Berechnung:</td>
<td>Anfang – Unbegrenzt: {standardzeitraum, standardzeitraum}</td>
</tr>
<tr>
<td>zweckgebundene Leistung:</td>
<td>☐</td>
</tr>
<tr>
<td>dem Grunde nach:</td>
<td>☐</td>
</tr>
<tr>
<td>Zeitraumbezogene Daten</td>
<td></td>
</tr>
<tr>
<td>nullwerte Anzeigen:</td>
<td>boolean = 01.01.1800 - 31.05.2016 : true</td>
</tr>
<tr>
<td>01.06.2016 - Unbegrenzt : false</td>
<td></td>
</tr>
<tr>
<td>berechnungsart</td>
<td>berechnungsarttyp = 01.01.1800 - 31.12.9999 : dreißigstel</td>
</tr>
<tr>
<td>Bezugsobjekte:</td>
<td></td>
</tr>
<tr>
<td>Attribute:</td>
<td>bemerkung : string wird validiert</td>
</tr>
<tr>
<td>antragsdatum : Datum</td>
<td></td>
</tr>
</tbody>
</table>

Public Benefits
composite block[plusOffset: number, minusOffset: number]

plusMinus_Composite_Offset(a: number, b: number) -> (sum, difference)
# Tachograph Rules

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&lt; TimePeriodObjectTypA1 &gt;</td>
</tr>
<tr>
<td></td>
<td>TimePeriodObjectTypA4 &gt;</td>
</tr>
<tr>
<td></td>
<td>TimePeriodObjectTypA6</td>
</tr>
<tr>
<td></td>
<td>&lt; TimePeriodSpecifier2::Duration = 24 Hours</td>
</tr>
<tr>
<td></td>
<td>&lt; TimePeriodSpecifier3::Duration = 15 Minutes</td>
</tr>
<tr>
<td></td>
<td>△ TimeSpikeObjectTypA5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Database</th>
<th>databaseOneAndMoreIterationsHappy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Begin</td>
</tr>
<tr>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td>eTimePeriodObjectTypA</td>
<td>50</td>
</tr>
<tr>
<td>eTimeSpikeObjectTypA</td>
<td>86020</td>
</tr>
<tr>
<td>eTimePeriodObjectTypA</td>
<td>86020</td>
</tr>
</tbody>
</table>
Math

```kotlin
fun midnight1(a: number, b: number, c: number) = (-b + sqrt(pow2(b) - 4 * a * c)) / (2 * a)

fun midnight2(a: number, b: number, c: number) {
    val bSquared = pow2(b)
    val sqrtPart = sqrt(bSquared - 4 * a * c)
    (-b + sqrtPart) / (2 * a)
}

fun midnight3(a: number, b: number, c: number) {
    -b + \sqrt{b^2 - 4 * a * c} / (2 * a)
}
```

Insurance Math

<table>
<thead>
<tr>
<th>D : Kommutationswerte</th>
<th>l : Lebende im Jahr x</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ergebnistyp:</strong></td>
<td><strong>Ergebnistyp:</strong></td>
</tr>
<tr>
<td>number[3]</td>
<td>number[0]</td>
</tr>
<tr>
<td><strong>Laufvariable:</strong></td>
<td><strong>Laufvariable:</strong></td>
</tr>
<tr>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td><strong>Parameter:</strong></td>
<td><strong>Parameter:</strong></td>
</tr>
<tr>
<td>i</td>
<td>geschlecht</td>
</tr>
<tr>
<td></td>
<td>q</td>
</tr>
<tr>
<td>(D_x := l_x * \frac{1}{(1 + i)^x})</td>
<td>(l_0 := \text{startwertLebende})</td>
</tr>
<tr>
<td></td>
<td>(l_x := l_{x-1} * (1 - q.lookup(x, geschlecht)))</td>
</tr>
</tbody>
</table>
Satellite Software

Activity `enableTcs with Numeric Id 1` is commandable by TC(150,1)
Short Description: enable thermal control

Description: The thermal control heats the system if it is too cold. The switching histeresis can be configured.

Constraints:

- : TCSCONTR.inMode(OFF) // switching on is possible only if the TCS is off

In-Parameter:

- int16[^\*/C/]: upperThreshold: constrained: << no constraint >> // upper switching threshold
- int16[^\*/C/]: lowerThreshold: constrained: lowerThreshold < upperThreshold // lower switching threshold
- component<<TemperatureAcquisition>> acq: constrained: << no constraint >> // acquisition component instance to use

```c
REQUEST acq.startAcquisition ( << ... >> ) -- ( << ... >> )
on error do nothing special
on error abort
UPTH = upperThreshold;
LOTH = lowerThreshold;
DELAY for 10 s
TCSCONTR.setMode(ON);
TELEMETRY (150,11)

Description: Report switching on in a dedicated packet that reports the initial temperature.

[initialTemp : int32[^\*/C/] = PUS150.AVTEMP // initial temperature when starting thermal control ]
```

Activity `disableTcs with Numeric Id 2` is commandable by TC(150,2)
Short Description: disable thermal control

Description:

Constraints:

- : TCSCONTR.inMode(ON) // switching off is possible only if the TCS is on

In-Parameter:

- << ... >>

```c
TCSCONTR.setMode(OFF);
REQUEST TACQA.stopAcquisition ( << ... >> ) -- ( << ... >> )
on error do nothing special
REQUEST TACQB.stopAcquisition ( << ... >> ) -- ( << ... >> )
on error do nothing special
```

} Component ThermalControlSystem
Healthcare

**decision table**

```plaintext
BpScoreDecisionTable(sys: bpRange, dia: bpRange) =
```

<table>
<thead>
<tr>
<th>sys</th>
<th>&lt;= 50</th>
<th>[51..90]</th>
<th>[91..95]</th>
<th>[96..100]</th>
<th>[101..109]</th>
<th>&gt;= 110</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;= 90</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>[91..140]</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>[141..150]</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>[151..160]</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>[161..179]</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>&gt;= 180</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

**decision tree**

```plaintext
DiarrheaStoolsDecisionTree(score: DiarrheaStoolsOverBaseline, 
patientHasAnySymptom: boolean, goToStartBrat: boolean)
```

- `DiarrheaReco1`
- `DiarrheaReco3`
- `DiarrheaReco2`
- `DiarrheaRecoSBrat`
- `DiarrheaRecoCBrat`

```plaintext
type temperature: number[36|42][1]
type measuredTemp: number[35|43][2]
```

Error: type number[32.55|39.99][4] is not a subtype of number[36|42][1]

```plaintext
val T_measured: measuredTemp = 42.22
val T_calibrated: temperature = T_measured * 0.93
```
Healthcare

PASS

function test gradeStools
    given 7 expected 3
    given 6 expected 2
    given 5 expected 2
    given 4 expected 2

PASS

function test DiarrheaStoolsDecisionTree
    given false, 1, true, false expected DiarrheaUSRecoLevel1Symptom
    given false, 9, false, false expected DiarrheaUSRecoGrade3

PASS

function test checkScreeningQuestion
    given answers to DiarrheaScreeningQuestionnaire
        dietarySupplements: false
        medication: true
        hospitalized: false
    }
Using language workbenches and domain-specific languages for safety-critical software development

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.
MPS Demo
(Meta-) Tooling

Language Workbench

Open Source, by JetBrains

Very Powerful

Used for years by itemis and others

Vast Experience
(Meta-) Tooling

Language Workbench
Open Source, by JetBrains
Very Powerful
Used for years by itemis and others
Vast Experience
MPS: Language Toolkit

- 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 Priors
- Constraints: Scopes, Usage Restrictions, Property Value Limitations

Generates to:
- Language
  - Extends 0..*

Defines execution semantics for:
- Structure
- Editor
- Type System
- Transformations

Provides editors for:
- Structure

 Specifies priority 0..*
MPS: Notational Freedom
MPS: Language Composition

**Embedding**

\[ L_{\text{Host}} + L_{\text{Adapt}} + L_{\text{Emb}} = \]

**Extension**

\[ L_{\text{Base}} + L_{\text{Ext}} = \]

**Extension Composition**

\[ L_{\text{Base}} + L_{\text{Ext1}} + L_{\text{Ext2}} = \]
Embedding/Extending the KernelF functional language is key to DSL development productivity.
Other Language Workbenches

{S} spoofax
TU Delft

xtext
itemis/Typefox

Rascal
CWI Amsterdam

The Whole Platform
Solmi/Persiani
Evaluating and Comparing Language Workbenches
Existing Results and Benchmarks for the Future

Sebastian Erdweg\textsuperscript{d}, Tijs van der Storm\textsuperscript{a}, Markus Völter\textsuperscript{e}, Laurence Tratt\textsuperscript{b}, Remi Bosman\textsuperscript{f}, William R. Cook\textsuperscript{c}, Albert Gerritsen\textsuperscript{f}, Angelo Hulshout\textsuperscript{g}, Steven Kelly\textsuperscript{h}, Alex Loh\textsuperscript{c}, Gabriël Konat\textsuperscript{l}, Pedro J. Molina\textsuperscript{j}, Martin Palatnik\textsuperscript{f}, Risto Pohjonen\textsuperscript{h}, Eugen Schindler\textsuperscript{f}, Klemens Schindler\textsuperscript{f}, Riccardo Solmi\textsuperscript{l}, Vlad Vergu\textsuperscript{l}, Eelco Visser\textsuperscript{l}, Kevin van der Vlist\textsuperscript{k}, Guido Wachsmuth\textsuperscript{l}, Jimi van der Woning\textsuperscript{l}

\textsuperscript{a}CWI, The Netherlands
\textsuperscript{b}King’s College London, UK
\textsuperscript{c}University of Texas at Austin, US
\textsuperscript{d}TU Darmstadt, Germany
\textsuperscript{e}voelter.de, Stuttgart, Germany
\textsuperscript{f}Sioux, Eindhoven, The Netherlands
\textsuperscript{g}Delphino Consultancy
\textsuperscript{h}MetaCase, Jyväskylä, Finland
\textsuperscript{i}TU Delft, The Netherlands
\textsuperscript{j}Icinetic, Sevilla, Spain
\textsuperscript{k}Sogyo, De Bilt, The Netherlands
\textsuperscript{l}Young Colfield, Amsterdam, The Netherlands
Lessons Learned
A Language is not Enough
Great IDE
Analyses
Refactorings
Testing
Debuggers

Language
Abstractions
Notations
Syntax Coloring
Code Completion
Goto Definition

Aligned with Processes
Report Back
Write Tests
Run them

GOOD
Great

GREAT
Influences on the Language
Domain Structure

Non Functionals
  Permissions, IP, Sharing

User Skills

Model Purpose
  Analyze, Generate

Tool Capabilities
  Notations, Editing, Scale

Software Engineering Practices

Sep. of Concerns
  Different Views

Get a better tool :-(

Educate,
  Put results in context

Refactor towards Structure
How to make People precise?
Precision ≠ Programming

{ Formulas, Rules
  Data Structures
  Tables
  Values

Performance
Scalability
Robustness
Deployment

Programing
Training is required.

ProgrammingBasics
How to think like a programmer.

What is this?
This is a tutorial on how to think like a programmer, and to learn some programming along the way. It teaches you fundamental ideas and concepts present in all programming systems, from “real” programming languages over scripting languages and configuration files to domain-specific languages.

Table of Contents
Part 1: The Basics
1. Values and Expressions
2. Testing Programs
3. Types
4. Functions

Part 2: Making Programming Useful
1. Structured Values
2. Collections
3. Decisions and Calculations
4. Instantiation
Skills?
Organizations do not have the necessary skills. True.

But...

Is this the next legacy system?
Today’s software is tomorrow’s legacy system.

Or is it?
Existing models become incompatible with new language

⇒ Language Versions
Migration Scripts
Runtime Tech outdated, uncool or slow

⇒ Keep Lang Technology
Keep Models
Build new Generator
Language Tech outdated, uncool

⇒ Build new Tool

Migrate Data Feasible, because it well-defined domain semantics and free from „technology stuff“
Today's software is tomorrow's legacy system.

No, it is not.
In conflict with Agile?
MD* and Agile is in Conflict.

---

1. Project 1
   - Language Development
   - System Development
   - Depend on, use

2. Project 1
   - Language Development
   - System Development
   - Dep’d on, use

   Later:
   - ...
MD* and Agile is in Conflict. “

Manage like any other intra-project dependency.

Evolution of client code is *easier* than for F/L/P because of migration support!
“MD* and Agile is in Conflict. “

Manage like any other 3rd party dependency:
Development Roadmap
Issue Tracker
Release Notes

...
MD* and Agile is in Conflict. “

Models and DSLs are an **Enabler** for Agility: Integration of Domain Experts „Living“ Requirements Decoupled Fachlichkeit & Technik

WTF?
Leading LWBs are so productive, you can literally sit with the domain experts and interactively prototype languages (and then clean up later).

I’ve looked at the implementation of the language in MPS, but I didn’t find much. Is this all there is? Where’s the magic?

[Customer]
Leading LWBs are so productive, you can literally sit with the domain experts and interactively prototype languages (and then clean up later).

I’ve looked at the implementation of the language in MPS, but I didn’t find much. Is this all there is? Where’s the magic?

[Customer]
What about CI?
You integrate like any other automatable CI step.
Wrap Up
A Language Stack for Implementing Contracts

The term Smart Contract is used for arbitrary programs that run on the distributed, trustworthy computing infrastructure provided by a blockchain. However, the sweet spot for such programs is actual contracts, i.e., long-running, collaborative processes involving several parties who may or may not trust each other. To implement such contracts effectively, we need much more than the Blockchain: contracts must be expressed in a way so that the relevant stakeholders, who are not typically programmers, can understand the them; contracts must be functionally correct, i.e., they must behave in exactly the way the stakeholders expect; and they must be protected against being gamed, for example, through sybil attacks. The trust in the execution of the contract, mostly through non-repudiability, is then provided by the blockchain. In this talk, I discuss research into how to formally model contracts, I present languages that are suitable for representing contracts in a way that is lawyer-accessible and prevents some aspects of gaming, and I discuss how such approaches lead to improved correctness through correctness-by-construction and simplified verification.
Lessons learned from developing mbeddr: a case study in language engineering with MPS

Authors
Markus Voelter, Bernd Kolb, Tamás Szabó, Daniel Ratiu, Arie van Deursen

Abstract
Language workbenches are touted as a promising technology to engineer languages for use in a wide range of domains, from programming to science to business. However, not many real-world case studies exist that evaluate the suitability of language workbench technology for this task. This paper contains such a case study. In particular, we evaluate the development of mbeddr, a collection of integrated languages and language extensions built with the JetBrains MPS language workbench. mbeddr consists of 81 languages, with their IDE support, 34 of them C extensions. The mbeddr languages use a wide variety of notations—textual, tabular, symbolic and graphical—and the C extensions are modular; new extensions can be added without changing the existing implementation of C. mbeddr’s development has spanned 10 person-years so far, and the tool is used in practice and continues to be developed. This makes mbeddr a meaningful case study of non-trivial size and complexity. The evaluation is centered around five research questions: language modularity, notational freedom and projectional editing, mechanisms for managing complexity, performance and scalability issues and the consequences for the development process. We draw generally positive conclusions; language engineering with MPS is ready for real-world use. However, we also identify a number of areas for improvement in the state of the art in language engineering in general, and in MPS in particular.
Separation of concerns is key to avoid the legacy trap.

DSLs can isolate business logic completely from technical concerns.

DSLs can help integrate domain experts with communication/review or even coding.

Language Workbenches enable DSLs by reducing effort to build, compose and maintain them.

DSLs are not in conflict with Agile. ... to the contrary, DSLs are a powerful enabler!