Let's build
Fachlichkeit
like they build
Rockets

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

Bernd Kolb
kolb@itemis.de
www.itemis.de
@berndkolb
Motivation
The Story of Paula and Knut (maybe your story)

“"I just finished the third revision of the specification. But I'm sure it will take another two month of annoying ping-pong with the IT."”

Paula 45, Product Engineer

“"Once again the spec was crap. Another 3 month wasted...””

Knut 32, Software Engineer
Examples
Healthcare
Context & Motivation

Mobile Apps that help patients with treatments
Monitor side-effects and recommend actions
Manage dosage of medications
Context & Motivation

Mobile Apps that help patients w/ treatments
Monitor side-effects and recommend actions
Manage dosage of medications

“Algorithms“ for recommendations and dosage at the core of these apps.

Safety-critical, since they could hurt patients.

Customer develops many different apps/algos like this, efficiency of algo development is key.
Health care professionals directly „code“ algos, using a suitable language. Avoids indirections through requirements docs. Speed up dev significantly.

Pretty typical DSL-based dev-approach.
Some Language Impressions I

decision table 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 DiarrheaStoolsDecisionTree(score: DiarrheaStoolsOverBaseline, patientHasAnySymptom: boolean, goToStartBrat: boolean)

patientHasAnySymptom

score >= 7

score in [4..6]

goToStartBrat

DiarrheaReco1

DiarrheaReco3

DiarrheaReco2

DiarrheaRecoSBrat

DiarrheaRecoCBrat

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}

val T_measured: measuredTemp = 42.22
val T_calibrated: temperature = T_measured * 0.93
Some Language Impressions II

**PASS**

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

**PASS**

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

**PASS**

```java
function test checkScreeningQuestion
  given answers to DiarrheaScreeningQuestionnaire{
    dietarySupplements: false
    medication: true
    hospitalized: false
    expected true
  }
```
Insurance
Context & Motivation

Specify/Program

Insurance Programs

Write formal code in a DSL mixed with tables and text
No tool support whatsoever
No testing (except inspection)
No reuse
No modularity
No variability
Context & Motivation

Specify/Program

Insurance Programs

Write formal code in a DSL mixed with tables and text
No tool support whatsoever
No testing (except inspection)
No reuse
No modularity
No variability
A real Example: Legacy

Write formal code in a DSL mixed with tables and text

No tool support whatsoever
No testing (except inspection)

No reuse
No modularity
No variability

Printed, PDF

Developer reads „spec“
Very idiomatic implementation

Dev acts as a human compiler and implements it in C

Spec/Program

„Pixelcrap“

C Code

Specify/Program

Debug
Solution Approach

**Insurance Programs**

Write formal code in a DSL mixed with tables and text

Now with IDE support and executable tests

The same notation!
A real Example: Future

Specify/Program/Test/Debug

Insurance Programs

Generate

C Code

Still exactly the same C code, or improved as needed.

Incremental Refinement/Refactoring of languages:

- Partially automated migration of models
- Add model natural notations (insurance-specific, math)
- Add Support for modularity, reuse, variants
Salary
Context & Motivation

Calculate Salaries + Taxes for Employees
Various deployment platforms

Complex Business Logic
Variability over 16 States
Based on evolving Law

Dates & Currencies
Temporal Data + Arithmetics
Reactive Rules
Currencies and Dates

```kotlin
fun printDate(d: date) {...}
val today = /2018 01 23/ // date type inferred
{ printDate(today) }

val nextWeek: date = today + 7
val lastYear: date = today - 365 // ignoring leap years for now :-(
val howLongIsAYear: number = lastYear - today

val salary : TT[currency] = TT | /2017 01 01/ => 5.000 EUR |
| /2017 05 01/ => 6.000 EUR |
```
Temporal Data

\begin{align*}
\text{val salary} & \cdot 2017\ 10\ 07/ = \text{TT} \quad | \quad /2017\ 01\ 01/ & \Rightarrow & 5.000\ \text{EUR} \\
& \quad | \quad /2017\ 05\ 01/ & \Rightarrow & 6.000\ \text{EUR} \\
\text{val salary} & \cdot 2017\ 11\ 05/ = \text{TT} \quad | \quad /2017\ 01\ 01/ & \Rightarrow & 5.000\ \text{EUR} \\
& \quad | \quad /2017\ 05\ 01/ & \Rightarrow & 5.500\ \text{EUR}
\end{align*}

\text{val salary : TT[currency] = TT} \quad | \quad /2017\ 01\ 01/ & \Rightarrow & 5.000\ \text{EUR} \\
& \quad | \quad /2017\ 05\ 01/ & \Rightarrow & 6.000\ \text{EUR}

\begin{align*}
\begin{array}{c}
a + s \\
\hline
a_1 \\
\hline
a_2 \\
\hline
a_3
\end{array}
& \quad + \\
\begin{array}{c}
\hline
\hline
\hline
\end{array}
\begin{array}{c}
a + s \\
\hline
a_1 + s \\
\hline
a_2 + s \\
\hline
a_3 + s
\end{array}
\end{align*}

\begin{align*}
\begin{array}{c}
a \\
\hline
a_1 \\
\hline
a_2 \\
\hline
a_3
\end{array}
& \quad + \\
\begin{array}{c}
a \\
\hline
b_1 \\
\hline
b_2
\end{array}
= \\
\begin{array}{c}
a + b \\
\hline
a_1 + b_1 \\
\hline
a_2 + b_1 \\
\hline
a_3 + b_2
\end{array}
\end{align*}
Result Data and Rules

```
result data [monthly] Salary {
  employment  -> Employment // basic data
  amount      : currency       
}

calculation for Tax
  depends Salary foreach person.employments as salaries

  calculate [monthly] {
    val factor = // do some weird tax math
    val total  := salaries.amount.sum
    amount    := total * factor
    employment:= ctx.employment
  }

result data [monthly] Tax {
  person  -> Person // basic data
  amount  : currency       
}
```

// depends on Salaries of all employments
// of the Tax bill’s person
// in the respective time

// sum up all salaries in current month
// populate fields of the result data item
// ctx is available in all calculations
Result Data and Rules

calculation for SalaryReport // data structure indexed to an Employment
depends Salary as s
              Salary[month.prev] as s_last

calculate [monthly] {
    currentSalary := s.amount
    lastMonthsSalary := s_last.amount
    delta := s.amount - s_last.amount
}

calculation for Salary
depends ...
calculate [monthly] {
    val e = ctx.employment
    val totalHoursWorked = e.workedHours.reduce(SUM)
    val averageWage = e.wage.reduce(WEIGHTED_AVERAGE)
    val religion = e.person.religion.reduce(LAST, increment.year)
IDE Support
Approach in a Nutshell
Integration of Fachler
It’s what makes a business tick. Distinguishes the business.

- Business Rules
- (Financial) Calculations
- Data Structures
- Mappings or Queries
- Validations
- Scientific Processes
- Contracts
- Processes
- UIs
It's what makes a business tick. Distinguishes the business.

Contributed not by developers

... but typically implemented in software
SO HOW DOES IT GET INTO THE SOFTWARE?

Contributed not by developers

... but typically implemented in software
Reality
Goal!?
Let Business/Domain people contribute directly!
Give them expressive, productive tools to do so!
Expressivity for Core Domain Knowledge

User-Friendly Notation
Great Tool/IDE

Testing

Meaningful Analyses

Synthesis of Software
Separation from Technology
Outdated Technology
Obscure Business Logic
Fachlichkeit „burried“ in implementation code.
Technology & Business Logic now have connected lifecycles.
Goal: Separate the Lifecycles

Fachlichkeit  Technology
Technical
Separation of concerns

From Wikipedia, the free encyclopedia

In computer science, separation of concerns (SoC) is a design principle for separating a computer program into distinct sections, such that each section addresses a separate concern. A concern is a set of information that affects the code of a computer program. A concern can be as general as the details of the hardware the code is being optimized for, or as specific as the name of a class to instantiate. A program that embodies SoC well is called a modular[1] program. Modularity, and hence separation of concerns, is achieved by encapsulating information inside a section of code that has a well-defined interface. Encapsulation is a means of information hiding.[2] Layered designs in information systems are another embodiment of separation of concerns (e.g., presentation layer, business logic layer, data access layer, persistence layer).[3]

The value of separation of concerns is simplifying development and maintenance of computer programs. When concerns are well-separated, individual sections can be reused, as well as developed and updated independently. Of special value is the ability to later improve or modify one section of code without having to know the details of other sections, and without having to make corresponding changes to those sections.
Outdated Technology
Non-Understandable Logic
Expensive to Evolve

Business Logic

Technology
Metamodel for Business Logic

Clearly defined data structure to express all business-relevant structures, behaviors and non-functional concerns.
Metamodel for Business Logic

- Data Structures
- Behavioral Rules
- Expressions
- Validations
- Special Types (e.g. temporal)
...
Domains often have a rich language/vocabulary anyway; it just needs to be formalized. (DDD: Ubiquitous Language)
Metamodel for Business Logic

Semantics

Clearly defined data structure to express all business-relevant structures, behaviors and non-functional concerns.

Well-defined meaning of this data structure
Metamodel for Business Logic

Semantics

Clearly defined data structure to express all business-relevant structures, behaviors and non-functional concerns.

Well-defined meaning of this data structure

- IDE Support is possible
- Evolution is possible
- Portability is possible

- Type Checking
- Solver-Integration
- Model Checking
- Contracts
Metamodel for Business Logic

Clearly defined data structure to express all business-relevant structures, behaviors and non-functional concerns.

Semantics

Well-defined meaning of this data structure

Execution Engine

Technical Platform for correct, efficient and scalable execution

Tech Infrastructure
Metamodel for Business Logic

Semantics

- Clearly defined data structure to express all business-relevant structures, behaviors and non-functional concerns.
- Well-defined meaning of this data structure

Tech Infrastructure

- Technical Platform for correct, efficient and scalable execution
Metamodel for Business Logic

Semantics

Tech Infrastructure

- generate code, deploy
- transfer data, interpret
Metamodel for Business Logic

Syntax

Semantics

Language

generate code, deploy

transfer data, interpret

Tech Infrastructure
Metamodel for Business Logic

Syntax

Semantics

Syntax is critically important for

Productivity
Communication and Review
Domain Expert Integration

generate code, deploy
transfer data, interpret

Only Buttons and Forms don‘t work!

Tech Infrastructure
Metamodel for Business Logic

Syntax
Semantics

Language

generate code, deploy
transfer data, interpret

Tech Infrastructure
Metamodel for Business Logic

Syntax

Semantics

Language Workbench

generate code, deploy

transfer data, interpret

Tech Infrastructure
A Language Workbench – a tool for defining, composing and using ecosystems of languages.
Other Language Workbenches

{S} spoofax

TU Delft

xtex

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{i}, 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}Icistic, Sevilla, Spain
\textsuperscript{k}Sogyo, De Bilt, The Netherlands
\textsuperscript{l}Young Colfield, Amsterdam, The Netherlands
Lessons Learned
A Language is not Enough
Language Design::More than Lang

- Language
  - Abstractions
  - Notations

- Great IDE
  - Syntax Coloring
  - Code Completion
  - Goto Definition

- Analyses
  - Relevant
  - Good Errors

- Refactorings
  - Aligned with Processes

- Testing
  - Write Tests
  - Run them
  - Report Back

- GREAT

- Debuggers
  - Animate Execution
  - Simulators
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) {
    \[-b + \sqrt{b^2 - 4 * a * c} \over 2 * a\]
}
Feature Models

Component Architectures

- FourCylEngine
  - DrivingCommands_in
  - RoadConditions_in

DriveTrainController
  - EngineStatus_out
  - SpeedFromEngine_out

Gearbox
  - Gear_out
composite block[plusOffset: number, minusOffset: number]

plusMinus_Composite_Offset(a: number, b: number) -> (sum, difference)
Unterhaltsvorschuss

Zeitangabe: laufend
Häufigkeit: monatlich einmal
Leistungskontext: Leer
Leistungsart: Leer
Zählaus: uvg

Anspruch Beginn: Anfang – Unbegrenzt: junger Mensch.geburtsdatum
Anspruch Ende: 01.01.1800 – 31.12.9999 : min(junger Mensch.geburtsdatum + 12 Jahre ,
datum + 72 Monate – Anzahl Monate mit uvg)

Zeitraum für Berechnung: Anfang – Unbegrenzt: {standardzeitraum, standardzeitraum}
zweckgebundene Leistung: ☐
dem Grunde nach: ☐

Zeitraumbezogene Daten
nullwerte Anzeigen: boolean = 01.01.1800 – 31.05.2016 : true
01.06.2016 – Unbegrenzt : false
berechnungsart : berechnungsarttyp = 01.01.1800 – 31.12.9999 : drittgastel

Bezugsobjekte:
Attribute: bemerkung : string wird validiert
antragsdatum : Datum
Unterhaltsvorschuss

Zeitangabe: laufend
Häufigkeit: monatlich einmal
Leistungskontext: Leer
Leistungsart: Leer
Zählart: uvg

Anspruch Beginn: Anfang - Unbegrenzt: junger Mensch.geburtsdatum
Anspruch Ende: 01.01.1800 - 31.12.9999: min(junger Mensch.geburtsdatum + 12 Jahre, datum + 72 Monate - Anzahl Monate mit uvg)

Zeitraum für Berechnung: Anfang - Unbegrenzt: {standardzeitraum, standardzeitraum}

zweckgebundene Leistung: ☐
dem Grunde nach: ☐

Zeitraumbezogene Daten
nullwerte Anzeigen: boolean = 01.01.1800 - 31.05.2016: true
01.06.2016 - Unbegrenzt: false
berechnungsart: berechnungsarttyp = 01.01.1800 - 31.12.9999: dreißigstel

Bezugssobjekte:
Attribute: bemerkung: string wird validiert
antragsdatum: Datum
Influences on the Language
Language Design::Influences

Domain Structure

Non Functionals
Permissions, IP, Sharing

User Skills

Model Purpose
Analyze, Generate

Tool Capabilities
Notations, Editing, Scale

Software Engineering Practices

Get a better tool :-)
Language Design::Influences

- Domain Structure
- Non Functionals: Permissions, IP, Sharing
- User Skills
- Model Purpose: Analyze, Generate
- Tool Capabilities: Notations, Editing, Scale
- Software Engineering Practices
- Style!
- Sep. of Concerns: Different Views
- Refactor towards Structure
- Get a better tool :-)
How to make People precise?
Precision

\{ Formulas, Rules, Data Structures, Tables, Values \}

Performance, Scalability, Robustness, Deployment

\{ Programming \}
Does this scale?
Does the approach scale?

If **structure**, **formalization**, and **tool support** don’t scale, then what will??

What are the alternatives?
- Excel?
- Wikis?
- Prose Documents?
Do the tools scale?

In terms of overall system size?
Yes, the system has to be broken down into models of manageable size, as usual. This requires some thought.

In terms of team size?
Yes, since we rely on established version control systems (git) to deal with groupware aspects; and yes, diff/merge work as expected.

In terms of language complexity?
Yes, in particular, since you can modularize the language definitions.
Can I find the people to do this?

Yes, but it is a significant change, so:

- it may be a significant education/training effort.
- a few people might not get it
- a few people may not want to do it.
This is a threat!
Precision and Formality
Different Processes
Higher Efficiency

- New Skills
- Role Change
- Job Loss

Automation
Focus on Engineering
Empower Business Ppl

- Job Loss
- Role Change
- Less Importance
Some people are afraid of this. Take them seriously.
A change of culture that must be managed!
Is this the next legacy system?
Today’s software is tomorrow’s legacy system. Or is it?
Today’s software is tomorrow’s legacy system.

Business change is hard
Technology change is hard
Separation of Concerns
Keep BL free of technology
Make it „portable“
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 Simple, 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. ”
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
MD* and Agile is in Conflict.

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]
MD* and Agile is in Conflict.

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]
Skills?
Organizations do not have the necessary skills. True. But...

Rockets ????
Further domain-specific extensions to C. Developed by end-user lang engineer.

<table>
<thead>
<tr>
<th>User Extensions</th>
<th>User-defined Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Languages shipped with mbeddr</td>
<td>C99</td>
</tr>
<tr>
<td>Plattform</td>
<td>Libraries for web server, node navigation, additional notations, pattern matching, palettes, XML processing, debugging...</td>
</tr>
<tr>
<td>MPS</td>
<td>Syntax Highlighting, Code Completion, Goto Definition, Find Usages, Type Checking, Data Flow Analysis, Refactoring, Versioning, Debugging</td>
</tr>
<tr>
<td>Foundation</td>
<td>C Compiler &amp; Debugger</td>
</tr>
<tr>
<td></td>
<td>PlantUML</td>
</tr>
<tr>
<td></td>
<td>Latex</td>
</tr>
<tr>
<td></td>
<td>HTML</td>
</tr>
<tr>
<td></td>
<td>CBMC</td>
</tr>
<tr>
<td></td>
<td>Z3</td>
</tr>
<tr>
<td></td>
<td>Sat4J</td>
</tr>
<tr>
<td></td>
<td>Implementation</td>
</tr>
<tr>
<td></td>
<td>Process</td>
</tr>
<tr>
<td></td>
<td>Analysis</td>
</tr>
</tbody>
</table>
Infrastructure Specifics in C

```c
#include TEMP_BUFFER_SIZE = 10;

TACQA = Instance of Temperature Acquisition with mnemonic tail A and the Numeric Id 350
  [SENSOR = SensorA]
TACQB = Instance of Temperature Acquisition with mnemonic tail B and the Numeric Id 351
  [SENSOR = SensorB]

Component Temperature Acquisition with Base Mnemonic: TACQ
  Short Description: acquisition of temperatures

Description: The components acquires the measurements of an assigned set of thermistors
{
  Attribute (hidden) int32/rawTemp/[TEMP_BUFFER_SIZE] MEASURED = <no init>; // measured raw values
  Attribute (hidden) uint32 ACQCTN = 0; // index for filling data acquisition buffer
  Attribute (readwrite) tempSensor SENSOR (Id=2) = <no init>; // selected sensor for this component

Mode Chart TCSACQ (Id=3) initial = OFF {
  Trigger tcsAcquisition
  Mode OFF {
    << ... >>
  }
  Mode ON {
    entry { ACQCTN = 0; }
    on trigger tcsAcquisition {
      // measure a value
      MEASURED[ACQCTN] = readTemperature(SENSOR);
      ACQCTN = (ACQCTN + 1) % TEMP_BUFFER_SIZE;

      // calculate average of the @top(TEMP_BUFFER_SIZE) latest measurements and convert to °C
      TEMP_BUFFER_SIZE - 1
      AVTEMP = convert[\sum_{idx = 0}^{\text{TEMP_BUFFER_SIZE} - 1} \frac{MEASURED[idx]}{\text{TEMP_BUFFER_SIZE} \to °C};
    }
  }
}

Activity startAcquisition with Numeric Id 1
  ... { TCSACQ.setMode(ON); }

Activity stopAcquisition with Numeric Id 2
  ... { TCSACQ.setMode(OFF); }
} Component Temperature Acquisition
```
Infrastructure Specifics in C

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 hysteresis can be configured.
  Constraints:
  0: 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

  { REQUEST acq.startAcquisition ( << ... >> ) --> ( << ... >> )
    on error do nothing special
    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.
  }

Activity disableTcs with Numeric Id 2 is commandable by TC(150,2)
  Short Description: disable thermal control
  Description:
  Constraints:
  0: TCSCONTR.inMode(ON) // switching off is possible only if the TCS is on
  In-Parameter:
  << ... >>

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

} Component ThermalControlSystem
TCSCONTR
thermal control

OFF
thermal control is inactive

TC(150,1)
enableTcs
TC(150,2)
disableTcs

ON
thermal control is active

trigger tcsControl
periodically triggered for altering the heater power state according to the measured values
exit
disable all heaters

PUS128 TelemetryService

enableTcs

TACQ_startAcquisition

TM(150,11); queue=10

ProcessTelemetry

TCSACQ->ON

DELAY 10 s
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**

**Migrating to a new LWB is feasible b/c semantics of all models are known, by definition.**