Complete Isolation of Business Logic using DSLs
Complete Isolation of Business Logic using DSLs

One of the most important architectural goals, IMHO!

I hope to help move it out of the niche it’s in right now.
1

Legacy Systems
Outdated Technology
Obscure Business Logic
You can’t understand/evolve/extract them independently.
Technology & Business Logic now have connected lifecycles.
If you could reliably extract the business logic and automatically transform it to run on a new technology platform, wouldn’t legacy systems lose much of their problematic nature?
If you could easily and reliably analyze and evolve business logic, wouldn’t legacy systems lose much of their problematic nature?
Business Logic

It’s what makes a business tick.
Distinguishes the business.

- Data Structures
- Business Rules
- (Financial) Calculations
- Mappings or Queries
- Validations
- Scientific Processes
- Contracts
- Processes
- UIs
Example
A real Example: Legacy

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

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

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 varibility
A real Example: Legacy

Specify/Program

Insurance Programs

„Pixelcrap“

Write formal code in a DSL mixed with tables and text
Printed, PDF

No tool support whatsoever
No testing (except inspection)

No reuse
No modularity
No variability
A real Example: Legacy

Specify/Program

Insurance Programs

„Pixelcrap“

C Code

Write formal code in a DSL mixed with tables and text
Printed, PDF
Developer reads „spec“
Very idiomatic implementation
Dev acts as a human compiler and implements it in C

No tool support whatsoever
No testing (except inspection)
No reuse
No modularity
No variability
A real Example: Legacy

Insurance Programs

„Pixelcrap“

C Code

Specify/Program

Implement

Debug

Write formal code in a DSL mixed with tables and text

Printed, PDF

No tool support whatsoever

Developer reads „spec“

No testing (except inspection)

Very idiomatic implementation

No reuse

Dev acts as a human compiler and implements it in C

No modularity

No variability

Debugging directly in C

Search-for-use by text search

Don’t trust the documents – may be outdated!
A real Example: Current

Specify/Program/Test/Debug

Insurance Programs

Write formal code in a DSL mixed with tables and text
Now with IDE support and executable tests

The same notation!
Funktionenmodell berbwvekFF

Formale Beschreibung

Funktion: berbwvekFF
Programmquelle: vmsctfal.c
Produkt-Typ: Fonds PK-Typ: Kapital-Konto
Status: 18.1

Parameter-Attribute

lkm_akt_param
lkm_faell_param
ber_zweck_param
kz_rzw_param

Verwendete VADM-Attribute

Keine verwendeten VADM-Attribute, werden automatisch hinzugefügt

Rückgabe-Attribut

bvvek

aufgerufene Funktionen

VTRKermbtbfaelFF (a)
berbewinzzFF (a; b; c)

Beschreibung

Die Funktion liefert den Barwert per @lkms akt_param des vorschüssigen Zahlungsstroms der Höhe 1 von Monat
@lkms akt_param bis @lkms faell_param – jeweils einschließlich. Zahlungszeitpunkte sind jeweils die Monatsbeginne, also
@lkms akt_param – 1# bis @lkms faell_param – 1#. Der Parameter @kz_ rzw_param steuert die zu berücksichtigende
Zahlweise des Zahlungsstroms. Möglicher sind die Zeit für die Ausprägungen 0 (Zahlungen zu den Beitragsfälligkeiten)
und 12 (monatliche Zahlungsweise).

Hilfsvariablen

kz_bf_hilf

Verarbeitungen

Schleife über lkm_faell_hilf = lkm_akt_param bis lkm_faell_param
Falls kz_rzw_param = 12
kz_bf_hilf = 1
sonst
kz_bf_hilf = VTRKermbtbfaelFF (lkms faell_hilf)
Ende Falls kz_rzw_param = 12
bvvek = bvvek + kz_bf_hilf * berbewinzzFF (lkms akt_param; lkm_faell_hilf = 1; ber_zweck_param)
Ende Schleife über lkm_akt_param bis lkm_faell_param
return bvvek
A real Example: Current

Funktionenmodell berbwvekFF

Formale Beschreibung

Funktion: berbwvekFF
Programmquelle: vmscfal1.c
Produkt-Typ: Fonds
PK-Typ: Kapital-Konto
Status: 18.1

Parameter-Attribute
lk_m_akt_param
lk_m_faell_param
ber_zweck_param
kz_rzw_param

Verwendete VADM-Attribute
Keine verwendeten VADM-Attribute, werden automatisch hinzugefügt

Rückgabe-Attribut
bwvek

aufgerufene Funktionen
VTRKermbgfaellFF (a)
berweinzelFF (a; b; c)

Beschreibung
Die Funktion liefert den Barwert per lkm_akt_param des vorschüssigen Zahlungssatzes lkm_akt_param bis lkm_faell_param – jeweils einschließlich. Zahlungszeitpunkte sind jeweils die Monatsbeginne, also lkm_akt_param - 1 bis lkm_faell_param - 1. Der Parameter kz_rzw_param steuert die zu berücksichtigende Zahlungsweise des Zahlungssatzes. Möglicherweise ist zur Zeit nur die Ausprägung 0 (Zahlungen zu den Beitragsfähigkeiten) und 12 (monatliche Zahlungsweise).

Hilfsvariablen
kz_bf_hilf

Verarbeitungen
Schleife über lkm_faell_hilf = lkm_akt_param bis lkm_faell_param

<table>
<thead>
<tr>
<th>Falls</th>
<th>Ausdruck</th>
</tr>
</thead>
<tbody>
<tr>
<td>kz_bf_hilf = 1</td>
<td>sonst</td>
</tr>
<tr>
<td>sonst</td>
<td>kz_bf_hilf = VTRKermbgfaellFF(lkm_faell_hilf)</td>
</tr>
</tbody>
</table>

Ende Falls kz_rzw_param = 12
bwvek = bwvek + kz_bf_hilf * berweinzelFF(lkm_akt_param; lkm_faell_hilf - 1, ber_zweck_param)

Ende Schleife über lkm_akt_param bis lkm_faell_param

return bwvek
A real Example: Current 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: Current 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: Current Insurance Programs

Specify/Program/Test/Debug

M3

Write formal code in a DSL mixed with tables and text

Generate

C Code

The same notation!

Exactly the same C code.
A real Example: Future

Incremental Refinement/Refactoring of languages:

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

Still exactly the same C code, or improved as needed.
Challenge: analyzing existing C Code

C is too flexible, too low-level and too „uncontrolled“ to implement analyzable business logic:

`malloc` vs. `free` mixed with business logic; pointer escaped.
Custom memory management inconsistent with standard `malloc/free` different, inconsistent `#defines` for YES/NO or TRUE/FALSE
Misuse of the preprocessor
No „architecture“, dependencies everywhere
Bad modularity, no fine-grained unit tests
No functional abstractions, sideeffects everywhere
Missing first-class abstractions for core domain concepts (date)
String and double comparisons with `==`

Obscure Business Logic
Example Domains

- Health and Medicine
- Automotive
- Aerospace
- Robotics
- Finance
- Embedded Software
- Science
- Government
- Law
- Law enforcement
Example Domains

- Algorithms for diagnosis and medicine dosage
- Specifying communication relationships between software components
- Satellite behavior and telemetry/telecommanding
- Robotics behavioral algorithms, movement, collision avoidance
- Insurance contracts/rules, product specification
- Embedded algorithms for math
- Biomedical analysis algorithms
- Tax and public benefits rules
- Precise specification of logistics contracts, interactive execution
- Digital Forensics, identifying „bad“ patterns in files
Employee of a user:

I am committing myself to implement our next [system] within one person year [instead of 50].
A customer:

Using a prototype language/tool they built, they could reimplement months of work in a few days. All tests ran.
Another case:

A customer had to schedule two weeks of work for their current supplier for a change that literally took minutes using the DSL.
Public Benefits Calculation:

We have been using such an approach for many years and could not imagine doing it any other way.
Outdated Technology
Non-Understandable Logic
Expensive to Evolve

Judge for yourself :-)
2

Separation of Concerns
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\cite{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.\cite{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).\cite{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.
Metamodel for Business Logic

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

Semantics
Metamodel for Business Logic

Semantics

Execution Engine

Tech Infrastructure

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

Well-defined meaning of this data structure

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

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

Well-defined meaning of this data structure

Technical Platform for correct, efficient and scalable execution
<table>
<thead>
<tr>
<th>Transformation</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ Code Inspection</td>
<td>+ Turnaround Time</td>
</tr>
<tr>
<td>+ Debugging</td>
<td>+ Runtime Change</td>
</tr>
<tr>
<td>+ Performance &amp; Optimization</td>
<td></td>
</tr>
<tr>
<td>+ Platform Conformance</td>
<td></td>
</tr>
</tbody>
</table>
Metamodel for Business Logic

Semantics

Tech Infrastructure

generate code, deploy

transfer data, interpret
Metamodel for Business Logic

Syntax
Semantics

Language

Syntax is critically important for

- Productivity
- Communication and Review
- Domain Expert Integration

- Only Buttons and Forms don’t work!

Tech Infrastructure

generate code, deploy
transfer data, interpret
Expressivity for Core Business Logic

User-Friendly Notation
Great Tool/IDE

Testing

Meaningful Analyses

Execution
Levels of Domain Expert Integration

- Generate derived artifacts
- Review the DSL sources
- Pair programming
- Independent Development

Domain expert integration is great, but even without it, the approach is useful to avoid the legacy trap.
Exchangable Technology
Understandable Business Logic
Metamodel for Business Logic

Syntax

Semantics

generate code, deploy

transfer data, interpret

Tech Infrastructure
**Metamodel for Business Logic**

**Syntax**

**Semantics**

- generate code, deploy
- transfer data, interpret

**Tech Infrastructure**
An old idea from the 1970s.

BUT...
Language Workbench
(Martin Fowler, 2004)

Freely define languages and integrate them
Language Workbench
(Martin Fowler, 2004)

powerful editing testing refactoring debugging groupware

language definition implies IDE definition
Language Workbench
(Martin Fowler, 2004)

There's no difference!
Language Workbench
(Martin Fowler, 2004)

Language Development is engineering, not science!
A Language Workbench – a tool for defining, composing and using ecosystems of languages.
Open Source
Apache 2.0
http://jetbrains.com/mps
Comprehensive Support for many aspects of Language Definition.

+ Refactorings, Find Usages, Syntax Coloring, Debugging, ...
[Projectional Editing]
What it is

Parsing

Projectional Editing

Concrete Syntax

Abstract Syntax Tree

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

```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 : (alt < 0) ? 10 : (alt == 0) ? 20 : (alt > 0) ? 30 : (alt > 100) ? 50 : 60;
} decide (function)
```

Graphical

```
Cst.Customer
\[\text{Contract} \rightarrow \text{Tariff}\]
```

```
\begin{array}{|c|c|}
\hline
\text{spd} > 0 & \text{spd} > 100 \\
\hline
\text{alt} < 0 & 1 \\
\text{alt} == 0 & 10 \\
\text{alt} > 0 & 30 \\
\text{alt} > 100 & 50 \\
\hline
\end{array}
```
[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.
[Projectional Editing]
Language Composition

Embedding

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

Extension

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

Extension Composition

\[ L_{\text{Base}} + L_{\text{Ext1}} + L_{\text{Ext2}} = \]
Other Language Workbenches

\{S\} spoofax

{\texttt{xtext}}

\textbf{Rascal}

\textbf{The Whole Platform}

TU Delft

itemis/Typefox

CWI Amsterdam

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{1}, Pedro J. Molina\textsuperscript{j}, Martin Palatnik\textsuperscript{f}, Risto Pohjonen\textsuperscript{h}, Eugen Schindler\textsuperscript{f}, Klemens Schindler\textsuperscript{f}, Riccardo Solmi\textsuperscript{1}, Vlad Vergu\textsuperscript{1}, Eelco Visser\textsuperscript{1}, Kevin van der Vlist\textsuperscript{k}, Guido Wachsmuth\textsuperscript{1}, Jimi van der Woning\textsuperscript{1}

\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}Icine, Sevilla, Spain
\textsuperscript{k}Sogyo, De Bilt, The Netherlands
\textsuperscript{l}Young Colfield, Amsterdam, The Netherlands
4 Precision vs. Programming
Precision ≠ Programming

{ Formulas, Rules
Data Structures
Tables
Values }
Precision

Performance
Scalability
Robustness
Deployment

Programming

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

\{ Greek Letters, Analyses, Proofs \} = Formalization
Why a DSL?
Isn’t a framework or code-populated model good enough?
Metamodel for Business Logic

Syntax

Semantics

generate code, deploy
transfer data, interpret

Tech Infrastructure
For (fine-grained) behaviors, this is just way too tedious. Expressions!

Domain-Specific Syntax lets domain experts contribute.

Static Checks provide more static assurances.

Language can evolve over time (in contrast to PL) to cover additional things first-class.
Imagine you are a Compiler
How would you parallelize these two pieces of code?

```java
int[] arr = ...
for (int i=0; i<arr.size(); i++) {
    sum += arr[i];
}
```

```java
int[] arr = ...
List<int> l = ...
for (int i=0; i<arr.size(); i++) {
    l.add( arr[i] );
}
```
Imagine you are a Compiler
How would you parallelize these two pieces of code?

```java
int[] arr = ...  
for (int i=0; i<arr.size(); i++) {
    sum += arr[i];
}
```

Overspecification!
Requires Semantic Analysis!

```java
int[] arr = ...  
List<int> l = ...  
for (int i=0; i<arr.size(); i++) {
    l.add( arr[i] );
}
```
Imagine you are a Compiler
How would you parallelize these two pieces of code?

First-Class Abstractions.  
Directly represents Semantics.

```java
for (int i in arr) {
    sum += i;
}
```

```java
seqfor (int i in arr) {
    l.add( arr[i] );
}
```
Def: Domain-Specific Language

A DSL is a language for a domain D that provides linguistic abstractions for common patterns and idioms of a language at D-1 when used within the domain D.

A good DSL does not require the use of patterns and idioms to express semantically interesting concepts in D. Processing tools do not have to do “semantic recovery” on D programs.

As you understand D over time, you add additional first-class abstractions to the DSL.
Integration on the Platform
Metamodel for Business Logic

Syntax

Semantics

Language Workbench

generate code, deploy

transfer data, interpret

Tech Infrastructure
Tech Infrastructure
Persistence/Database
Sensors
Transactions
Permissions

... typical technical services, also found in app servers etc.

Service 1

...

Service N
Evolution
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 Simple, because it well-defined domain semantics and free from „technology stuff“
Today‘s software is tomorrow‘s legacy system.

No, it is not.
Some Lessons Learned
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 works 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.
Business L vs. Programming L
Structure/Guid.  +  Mixed  -  Text
Notation  +  *  Clean  1  Powerful
Views  -  L  E
IDE/Tool
Learn/Effective
Business oriented languages are very different from what we have learned about languages for developers. LWBs let you build such languages.
Examples
## Rigid Structures

<table>
<thead>
<tr>
<th>Variables:</th>
<th>Parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRMI</td>
<td>int</td>
</tr>
<tr>
<td>FR</td>
<td>int</td>
</tr>
<tr>
<td>NN</td>
<td>int</td>
</tr>
<tr>
<td>TT</td>
<td>int</td>
</tr>
<tr>
<td>J</td>
<td>int</td>
</tr>
<tr>
<td>A3</td>
<td>int</td>
</tr>
<tr>
<td>G3</td>
<td>int</td>
</tr>
<tr>
<td>ANUI</td>
<td>int</td>
</tr>
<tr>
<td>X</td>
<td>int</td>
</tr>
</tbody>
</table>

**Business objects**

- person : Person

**Variables:**

- PRMI : int
- FR : int
- NN : int
- TT : int
- J : int
- A3 : int
- G3 : int
- ANUI : int
- X : int

**Parent**

- Libraries

- Standard
- Extra

**Rule Set Type** DemoRuleSetType

**Business objects**

- <no business objects>

**Variables:**

- <no variables>

**Parent**

- <no parent>

**Rule Set Type** DemoRuleSetType
Prose-Like Language for Calc Rules

bloedverwanten : lijst van Burgers zijn gedefinieerd als {
   Een bloedverwant is een Burger die
   bloedverwant in rechte lijn is of die
   bloedverwant in tweede graad zijlijn is
   Einde declaratie
}

bloedverwanten in rechte lijn : lijst van Burgers zijn gedefinieerd als {
   Een bloedverwant in rechte lijn is een Burger die
   nakomeling is of die
   voorouder is
   Einde declaratie
}

bloedverwanten in tweede graad zijlijn : lijst van Burgers zijn gedefinieerd als {
   Een bloedverwant in tweede graad zijlijn is een ouder.kind met
   ouder.kind ongelijk het actuele voorkomen
   Einde declaratie
   ' dus: broer of zus (incl. erkend kind van ouder)
}

bloed- of aanverwanten in rechte lijn : lijst van Burgers zijn gedefinieerd als {
   Een bloed- of aanverwant in rechte lijn is een Burger die
   bloedverwant in rechte lijn is of die
   aanverwant in rechte lijn is
   Einde declaratie
}
Diagrams for Data Modeling

Contract
starts: date
ends: date

Tariff
applicableTariff 1
attributes

Customer.Customer
customer 1
contracts 0..*

BillingRegion
- code [key]: string
- name: string
- baseMinPrice: float
- maxRebateFactor: float
# Tables for Reference Data

## Core Data Default Regions for entity BillingRegion

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Base Price</th>
<th>Min Price</th>
<th>Max Price</th>
<th>Rebate</th>
<th>Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>BW</td>
<td>Baden Württemberg</td>
<td>0.20</td>
<td>0.20</td>
<td>0.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BY</td>
<td>Bayern</td>
<td>0.20</td>
<td>0.20</td>
<td>0.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BE</td>
<td>Berlin</td>
<td>0.15</td>
<td>0.15</td>
<td>0.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BB</td>
<td>Brandenburg</td>
<td>0.10</td>
<td>0.10</td>
<td>0.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HB</td>
<td>Bremen</td>
<td>0.20</td>
<td>0.20</td>
<td>0.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HH</td>
<td>Hamburg</td>
<td>0.15</td>
<td>0.15</td>
<td>0.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HE</td>
<td>Hessen</td>
<td>0.15</td>
<td>0.15</td>
<td>0.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MV</td>
<td>Mecklenburg-Vorpommern</td>
<td>0.10</td>
<td>0.10</td>
<td>0.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NI</td>
<td>Niedersachsen</td>
<td>0.15</td>
<td>0.15</td>
<td>0.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NW</td>
<td>Nordrhein-Westfalen</td>
<td>0.15</td>
<td>0.15</td>
<td>0.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RP</td>
<td>Rheinland-Pfalz</td>
<td>0.15</td>
<td>0.15</td>
<td>0.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SL</td>
<td>Saarland</td>
<td>0.15</td>
<td>0.15</td>
<td>0.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SN</td>
<td>Sachsen</td>
<td>0.10</td>
<td>0.10</td>
<td>0.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST</td>
<td>Sachsen-Anhalt</td>
<td>0.10</td>
<td>0.10</td>
<td>0.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SH</td>
<td>Schleswig-Holstein</td>
<td>0.15</td>
<td>0.15</td>
<td>0.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TH</td>
<td>Thüringen</td>
<td>0.10</td>
<td>0.10</td>
<td>0.7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Funktionenmodell berbwvekFF

Formale Beschreibung

Funktion: berbwvekFF
Programmquelle: vmstcf01.c
Produkt-Typ: Fonds PK-Typ: Kapital-Konto
Status: 18.1

Parameter-Attribute
liek_akt_param
liek_faell_param
ber_zweck_param
kz_rzw_param

Verwendete VADM-Attribute
Keine verwendeten VADM-Attribute, werden automatisch hinzugefügt

Rückgabe-Attribut
bwvek

aufgerufene Funktionen
VTRKernmbtgfaellFF (a)
berbweinze1FF (a; b; c)

Beschreibung
Die Funktion liefert den Barwert per @liek_akt_param des vorschüssigen Zahlungsstroms der Höhe 1 von Monat @liek_akt_param bis @liek_faell_param jeweils einschließlich. Zahlungszeitpunkte sind jeweils die Monatsbeginne, also @liek_akt_param - 1# bis @liek_faell_param - 1#. Der Parameter @kz_rzw_param steuert die zu berücksichtigende Zahlweise des Zahlungsstroms. Möglicher sind zur Zeit nur die Ausprägungen 0 (Zahlungen zu den Beitragsfällen) und 12 (monatliche Zahlungsweise).

Hilfsvariablen
kz_bf_hilf

Verarbeitungen
Schleife über liek_faell_hilf = liek_akt_param bis liek_faell_param
Falls kz_rzw_param = 12
  kz_bf_hilf = 1
sonst
  kz_bf_hilf = VTRKernmbtgfaellFF (liek_faell_hilf)
Ende Falls

bwvek = bwvek + kz_bf_hilf * berbweinze1FF (liek_akt_param; liek_faell_hilf - 1; ber_zweck_param)

Ende Schleife über liek_akt_param bis liek_faell_param

return bwvek
Funktionenmodell GeometrischesMittel

Formale Beschreibung

Funktion: GeometrischesMittel
Programmquelle: Programmquelle auswählen
Produkt-Typ: Produkt-Typen auswählen
PK-Typ: PK-Typen auswählen
Status: Status auswählen

Parameter-Attribute

Verwendete VADM-Attribute
Keine verwendeten VADM-Attribute, werden automatisch hinzugefügt

Rückgabe-Attribut
result

auferufene Funktionen
Keine aufgerufenen Funktionen, werden automatisch hinzugefügt

Beschreibung
Berechnet das geometrische Mittel der Parameter

Hilfsvariablen
Hilfsvariable hinzufügen

Error: Quadratwurzel ist nur für positive Zahlen erlaubt

result = sqrt(a^2)
result = sqrt(a + b)

return result
Decision Mechanisms, directly in Expressions

\[
\text{val } c2: \text{ int } = \text{ split three}\begin{cases} 
< 0 & \Rightarrow 0 \\
0..3 & \Rightarrow 42 \\
> 3 & \Rightarrow 44
\end{cases}
\]

\[
\text{fun } \text{pricePerMin} (\text{time: int, region: int}) =
\]

<table>
<thead>
<tr>
<th>time.range[0..6]</th>
<th>region == EUROPE</th>
<th>region in [US, CAN, ASIA]</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>20</td>
<td></td>
</tr>
</tbody>
</table>

\[
\text{fun } \text{riskFactor} (\text{gender: int, age: int, weight: int}) =
\]

\[
< > \text{default: LOW}
\]

- [gender == MALE]
  - [age < 40] LOW
  - [age > 40] [overweight(weight)] MEDIUM

- [gender == FEMALE]
  - [age > 50 && overweight(weight)] HIGH
Extension function can be called in dot-notation, perfectly suitable for developers.
For non-programmers, a more prose-like notation is helpful. Notice the prose-call facility is a modular extension of the expression language.
Influences on the Language
Domain Structure

Non Functionals
Permissions, IP, Sharing

User Skills

Sep. of Concerns
Different Views

Educate,
Put results in context

Get a better tool :-)  

Refactor towards Structure

Model Purpose
Analyze, Generate

Tool Capabilities
Notations, Editing, Scale

Software Engineering Practices
The Language is not Enough
We tried it before, and it failed.
The UML tool was a bad choice
  -> ok, choose a better one :-(

Hard to represent business logic in UML.
  -> oh, really?? Who would have thunk.

Generate Class-Skeletons, fill in app logic.
  -> how and why does this solve the challenges??

Round-Tripping did not work.
  -> never works, but why use it?

Such an approach is completely pointless!!
Rule Language

No tests and debuggers for end users

-> hard to be sure about things

Language not expressive enough (tables)
Tool too limited to enhance expressivity

-> tedious to express many algorithms

Parts still had to be programmed manually

-> overall process more complex, not simpler

The right direction, but not good enough.
Drawbacks
You need inhouse expertise for language engineering

or a very close and trusted vendor who does it for you.
If you use this approach for real, you should have language engineering expertise in house.
You will invest a lot into a particular tool.

You can easily export models, but no portability for language definitions.
10
Summary
Thank you!
voelter@acm.org
www.voelter.de
@markusvoelter
[open source]
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.**