Language
Shapes (Architectural) Thought

Markus Völter
Language

Shapes

Thought

Sapir–Whorf hypothesis
aka Whorfianism
The principle of linguistic relativity holds that the structure of a language affects the ways in which its respective speakers conceptualize their world, i.e. their world view, or otherwise influences their cognitive processes.

Sapir–Whorf hypothesis
aka Whorfianism
The principle of linguistic relativity holds that the structure of an architecture modeling language affects the ways in which its users conceptualize an architecture.

*Sapir–Whorf hypothesis aka Whorfianism*
What is Software Architecture
... the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships between them.
A collection of software [..] components, connections, and constraints.

A collection of system stakeholders' need statements.

A rationale which demonstrates that [the system fulfils the needs]

Boehm et. al
... is its style and method of design and construction.

Hayes-Roth
...fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution

ISO/IEC/IEEE 42010
... the set of design decisions which, if made incorrectly, may cause your project to be cancelled.

Eoin Woods
... everything that must be consistent throughout a software system.

“strategic design” – Eric Evans

Mine
[Examples]

Class Structure
Components, Subsystems, Layers
Deployment onto Hardware

Dataflow Architecture (ETL)
Threading/Timing Architecture
Locking Protocol
It’s not about Granularity. And it’s not just structure. It’s about consistency.
Why would you model Architecture?
Communication
between people

„Diagramming“

Doesn‘t count :-}
[Communication]
[Analysis]
finding flaws early
predicting properties

Timing
Concurrency
Bus Utilization
model expected characteristics and check against implement’n
generate other artifacts, typically implementation code
Communication Analysis Checking Synthesis}

Model Purpose

Relevant for any Modeling Language
Drives Selection/Design of Language
Requires Tool Support!
Just pictures doesn’t cut it.
There’s more than code gen.
Purpose determines Language!
Separation Of Concerns
[Three Core Concerns]

- Type/Data Model
- System Model(s)
- Composition Model(s)
[Composition]

**context ComponentInstance inv:**
foreach of type's Component-InterfaceRequirements there must be a Wire of the same name.

**context Wire inv:**
the type of the target instance must provide the Interface pointed to by the Wire's cireq's target.
[Systems/Deployment]

Diagram:

- Configuration
  - name

- Component Instance
  - name

- Wire
  - name

- System
  - name

- Node
  - name

- Container
  - name
  - kind
[Additional Aspects*]
[Aspect: Persistence]

RefTable
- name

EntityTable
- name

Column
- name

Index
- name

DBPrimitive Type
- Query
  - name
  - expression

Entity Reference
- name
- isBidirectional
- targetMultiplicity
- sourceMultiplicity

Complex Type
- attribute
- name

Attribute
- name

Primitive Type
- attribute
- base

Type
- name

Entity
- entity

Data Transfer Object
- entity

Component
- name
[Separation?]

integrated into one fragment

separated into several fragments
Separation?

- Sufficiency
- Different Stakeholders
- Different Process Steps

1:n Relationships

- Well-defined interfaces
- Avoid Cycles
- Avoid Synchronization
Think in terms of Concerns. Separate them if necessary. Also support integration!
Established Formalisms
Communications
Simple Code Generation
Analysis/Verification

Code Generation

http://www.primordion.com
Composition | UML Comp. Struct.

Analysis

Code Generation

http://www.ual.es/
System | UML Deployment

Communication

Analysis

Code/Script Generation

http://www.ivoa.net/
thread CoinPublisher
  features
    acceptNotify: in event port;
  end CoinPublisher;

thread implementation CoinPublisher.impl
  calls(u: subprogram updateTotal);
  properties
    Compute_Execution_Time => 30ms .. 40ms;
    Dispatch_Protocol => ( Sporadic );
  annex behavior { **
    compute(5ms);
    compute(10ms);
    compute(15ms);
    raise(availableContent);
  } **;
end CoinPublisher.impl;
Composition | AUTOSAR

Analysis

Code/Script Generation

http://blog.lieberlieber.com/
package arpSafetyCar

interface clientServer ILifecycle {
  operation changeVehicleMode {
    in EVehicleMode vehicleMode out tBoolean success
  }
}

component application ModeManager {
  ports {
    receiver rMode requires IVehicleMode
  }
}
Adaptation | UML Profiles

Communication
Analysis
Code/Script Generation
Adaptation | UML Profiles

Analysis

Code/Script Generation

Components + Ports Ubiquitous
Graphical + Textual
Concerns mostly Separated!
Some means of Adaptation.
The majority of our interviewees were very successful with MDE but all of them either **built their own** modeling tools, made **heavy adaptations** of off-the-shelf tools, or spent a lot of time finding ways to **work around** tools. The only accounts of easy-to-use, intuitive tools came from those who had developed tools themselves for bespoke purposes. Indeed, this suggests that current tools are a barrier to success rather than an enabler.

---

Complexity problems are typically associated with off-the-shelf tools. Of particular note is accidental complexity – which can be introduced due to [...] [the] lack of flexibility to adapt the tools to a company’s own context [...]
Our interviews point to a strong need for tailoring of some sort: either tailor the tool to the process, tailor the process to the tool, or build your own tool that naturally fits your own process. Based on our data, it seems that, on balance, it is currently much easier to do the latter.

Profiles are Hard
not used right very often & unnecessarily complicated models
Profiles are Hard
May be misleading
models mean something else than what they appear to mean
Profiles are Hard
May be misleading
Models are not “intentional”

low-level abstractions make models hard to analyze
Profiles are Hard]
[May be misleading]
[Models are not “intentional”]
[Unintended Features]

...because profiles must limit existing functionality. Coverage!
Profiles are Hard
May be misleading
Models are not “intentional”
Unintended Features
Hard to include textual Aspects

There is no extensible way for textual syntax in UML
Profiles are Hard
May be misleading
Models are not “intentional”
Unintended Features
Hard to include textual Aspects
Many UML tools suck @ profiles

Magicdraw is the only exception
I have seen so far!
Profiles are Hard
May be misleading
Models are not “intentional”
Unintended Features
Hard to include textual Aspects
Many UML tools suck @ profiles
Standard Profiles are Complex

MARTE is 600 pages – how much of that stuff do you really need?
component DelayCalculator {
    provides aircraft: IAircraftStatus
    provides managementConsole: IManagementConsole
    requires screens[0..n]: IInfoScreen
}

component Manager {
    requires backend[1]: IManagementConsole
}

component InfoScreen {
    provides default: IInfoScreen
}

component AircraftModule {
    requires calculator[1]: IAircraftStatus
}

instance dc: DelayCalculator
instance screen1: InfoScreen
instance screen2: InfoScreen
connect dc.screens to (screen1.default, screen2.default)
namespace com.mycompany.datacenter {
    registered instance dc1: DelayCalculator {
        registration parameters {role = primary}
    }
    registered instance dc2: DelayCalculator {
        registration parameters {role = backup}
    }
}

namespace com.mycompany.production {
    instance dc: DelayCalculator
dynamic connect dc.screens every 60 query {
    type = IIInfoScreen
    status = active
}
}
interface IAircraftStatus {
  oneway message registerAircraft(aircraft: ID)
  oneway message unregisterAircraft(aircraft: ID)
  oneway message reportPosition(aircraft: ID, pos: Position)
  request-reply message reportProblem {
    request (aircraft: ID, problem: Problem, comment: String)
    reply (repairProcedure: ID)
  }
}

protocol initial = new {
  state new {
    registerAircraft => registered
  }
  state registered {
    unregisterAircraft => new
    reportPosition
    reportProblem
  }
}

}
struct FlightInfo {
   // ... attributes ...
}

replicated singleton flights {
   flights: FlightInfo[]
}

cOMPONENT DelayCalculator {
   publishes flights { publication = onchange }
}

cOMPONENT InfoScreen {
   consumes flights { init = all update = every(60) }
}
A DSL per Architecture/Platform

Really fits the A exactly.

But what about Effort?

What can we reuse? DSL-PLE?
[Candidates for Reuse]

Namespaces
Expressions
Data Types
Operations
Components

But still: extension, restriction & adaptation is required!
[More Candidates for Reuse]

- Tracing to Requirements
- Architecture Decisions
- Variability Support
- Documentation

Of Course: extension, restriction & adaptation is required!
Fine-grained Reuse as in OO
Handle Crosscuts
IDE Support
An extensible set of integrated languages for embedded software engineering.

### Specific Languages

- **C99**
- **Model Checking**
- **SMT Solving**
- **Dataflow Analysis**
- **Visualization**
- **PLE Variability**
- **Documentation**
- **Requirements & Tracing**
- **Reports & Assessments**

### Core

- **Components**
- **Physical Units**
- **State Machines**
- **State Machine Verification**
- **Decision Tables**
- **Component Contracts**

### Default Extensions

- **Test Support**
- **Decision Tables**
- **Logging & Tracing**

### User Extensions

- **Glossaries**
- **Use Cases & Scenarios**

### Backend Tool

- **C Compiler, Debugger and Importer**
- **NuSMV**
- **Yices**
- **CBMC**
- **PlantUML**
- **LaTeX**

### Platform

- JetBrains MPS

---

Implementation Concern | Analysis Concern | Process Concern
constant TAKEOFF = 100;  // implements PointsForTakeoff
constant HIGH_SPEED = 10; // implements FasterThan100
constant VERY_HIGH_SPEED = 20; // implements FasterThan200
constant LANDING = 100;    // implements FullStop

[verifiable]
exported statemachine FlightAnalyzer initial = beforeFlight {
    in event next(Trackpoint* tp) < no binding>
    in event reset() < no binding>
    out event crashNotification() => raiseAlarm
    readable var int16 points = 0
    state beforeFlight {
        // [ Here is a comment on a transition. ]
        on next [tp->alt == 0 m] => airborne
        exit { points := TAKEOFF; }  // implements PointsForTakeoff
    }
    state airborne {
        Error: type int16/[m/s] is not comparable with (uint8 || int8)
        on next [tp->alt == 0 m && tp->speed == 0] => crashed
    }
    state landing {
        on next [tp->speed == 0 mps] => landed
        [on next [tp->speed > 0 mps] => landing { points--; }]  // implements Sh...
    }
}
itemis France: Smart Meter

First significant mbeddr project
c.a. 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 16 384 200 | F +32 16 384 350 | 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 into other domains

insurance, financial, tax
The Language Workbench
Open Source
Apache 2.0
http://jetbrains.com/mps
+ Refactorings, Find Usages, Syntax Coloring, Debugging, ...
Projectional Editing
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]

Language Composition

Separate Files

Type System
Transformation
Constraints

In One File

Type System
Transformation
Constraints
Syntax
IDE
LWBs make Languages Easier
Multiple (Mixed) Notations
Language Extension and Composition
MPS works, but not the only one.
[Requirements]

1. Initially you have no points.
   InitialNoPoints /functional: tags
   [ When the game starts, you have no points. ]

2. Once a flight lifts off, you get 100 points
   PointsForTakeoff /functional: tags

3. The factor of points
   PointsFactor /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. ]

4. Points you get for each trackpoint
   InFlightPoints /functional: tags
Requirements + Components

1. Provides flight data
   FlightData /participant: tags
   [Lorem ipsum dolor sit amet, consectetur adipiscing elit.]

   component FlightDataProvider {
     data Flight
     owns x: Flight
     capability createFlight(): Flight
   }

4. Stores flights in memory
   InMemoryStore /participant: tags

   component InMemoryStore {
     collaborates with FlightDataProvider:
     owns flights: Flight
     capability store(Flight): status
     capability setup(): status
   }
[Requirements + Components]

- **Driver**
  - process()

- **Nuller**
  - process()
  - judge()

- **Judger**
  - pointsCollected
  - judge()

- **Interpolator**
  - process()

- **InMemoryStore**
  - store()
  - setup()

- **FlightDataProvider**
  - Flight
  - createFlight()
1.2.1 Describes the Interpolation

```java
Interpolation /scenario: tags
[ Text ]

scenario Interpolation
  UI {
    -> DataStore.getAFlight(): new Flight f
    -> DataStore.getAFlight() {
      return new Flight f2
    } DataStore.getAFlight
    -> Interpolator.process(received f2):
    loop over all the trackpoints in f {
      -> Judger.judge(new Trackpoint t)
    } loop
  }
```

[Diagram of collaboration and scenarios]
exported cs interface TrackpointStore1 {
    void store(Trackpoint* tp)
        pre(0) isEmpty()
        pre(1) tp != null
        post(2) !isEmpty()
    Trackpoint* get()
        pre(0) !isEmpty()
    Trackpoint* take()
        pre(0) !isEmpty()
        post(1) result != null
        post(2) isEmpty()
    query boolean isEmpty()
}
exported cs interface TrackpointStore2 {
    void store(Trackpoint* tp)
    protocol init(0) -> new full(1)
    Trackpoint* get()
    protocol full -> full
    Trackpoint* take()
    post(0) result != null
    protocol full -> init(0)
    query boolean isEmpty()
}
exported component InMemoryStorage extends nothing {

    provides TrackpointStore1 store

    Trackpoint* storedTP;

    void init() <= on init {
        storedTP = null;
        return;
    } runnable init

    void store_store(Trackpoint* tp) <= op store.store {
        return;
    } runnable store_store

    Trackpoint* store_get() <= op store.get {
        return storedTP;
    } runnable store_get

    Trackpoint* store_take() <= op store.take {
        Trackpoint* temp = storedTP;
        storedTP = null;
        return temp;
    } runnable store_take

    boolean store_isEmpty() <= op store.isEmpty {
        return storedTP == null;
    } runnable store_isEmpty
}
component InMemoryStorage
[Component Verification]

```c
[checked]
exported component InMemoryStorage extends nothing {

    provides TrackpointStore1 store

    Trackpoint* storedTP;

    void init() <= on init {
        storedTP = null;
        return;
    } runnable init

    void store_store(Trackpoint* tp) <= op store.store {
        return;
    } runnable store_store

    Trackpoint* store_get() <= op store.get {
        return storedTP;
    } runnable store_get

    Trackpoint* store_take() <= op store.take {
        Trackpoint* temp = storedTP;
        storedTP = null;
        return temp;
    } runnable store_take

    boolean store_isEmpty() <= op store.isEmpty {
        return storedTP == null;
    } runnable store_isEmpty
}

component InMemoryStorage

instances verificationInstances {
```
mock component StorageMock report messages: true {
    provides TrackpointStore1 store
    Trackpoint* lastTP;
    total no. of calls is 5
    sequence {
        step 0: store.isEmpty return true;
        step 1: store.store {
            assert 0: parameter tp: tp != null
        }
        do { lastTP = tp; }
        step 2: store.isEmpty return false;
        step 3: store.take return lastTP;
        step 4: store.store
    }
}
[Instantiation]

```java
instances interpolatorInstancesWithMock {
    instance StorageMock storeMock
    instance Interpolator ip(divident = 2)
    connect ip.store to storeMock.store
    adapt ipMock -> ip.processor
}
```
exported composite component MetrologyRawSignalSimulatorTestHarnessImpl {
    provides IMetrologyRawSignalSimulationRunner runner

    internal instances {
        instance MetrologyRawSignalSimulatorImpl signalSim
        instance GraphPlotterImpl plotter
        instance MetrologyRawSignalSimulationRunnerImpl runner

        connect runner.rawSignalSim to signalSim.rawSignalSim
        connect multi signalSim.sigRunHandler to runner.rawSigHandler
        connect runner.signalData to signalSim.signalData
        connect runner.graphPlotter to plotter.graphPlotter

        delegate runner to runner.rawSignalSimRunner
    }
}
[Tracing from Code]

```plaintext
[checked]
exported statemachine FlightAnalyzer initial = beforeFlight {
  in event next(Trackpoint* tp) <no binding>
  in event reset() <no binding>
  out event crashNotification() => raiseAlarm
  readable var int16 points = 0

  state beforeFlight
    entry {
      Kind: Implements
      on next 1st Target: Once you land successfully, you get another 100 points.
    }
    exit {
      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.
      For testing purposes, this one references @req(InFlightPoints)
    }

  state airborne
    on next [tp->alt == 0 m && tp->speed == 0 mps] -> crashed
    on next [tp->alt == 0 m && tp->speed > 0 mps] -> landing
    on next [tp->speed > 200 mps && tp->alt == 0 m] -> airborne { points += VERY_HIGH_SPEED; }
    on next [tp->speed > 100 mps && tp->speed <= 200 mps && tp->alt == 0 m] -> airborne
      { points += HIGH_SPEED; }
    on reset [ ] -> beforeFlight
}
```
Points you get for each trackpoint

InFlightPoints /functional: tags


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

tests:

PointForATrackpoint(500, 100) == 0
PointForATrackpoint(500, 100) == 0  # Error: failed, expected 210, but was 200
PointForATrackpoint(1100, 165) == 210
PointForATrackpoint(2100, 140) == 100
PointForATrackpoint(2100, 200) == 300
exported component Judge2 extends nothing {
    provides FlightJudger judger
    int16 points = 0;
    void judger_reset() <= op judger.reset {
        points = 0;
    } runnable judger_reset
    void judger_addTrackpoint(Trackpoint* tp) <= op judger.addTrackpoint {
        points += PointForATrackpoint(stripunit[tp->alt], stripunit[tp->speed]);
    } runnable judger_addTrackpoint
    int16 judger_getResult() <= op judger.getResult {
        return points;
    } runnable judger_getResult
} component Judge2
[PLE Variability]

feature model FlightProcessor

root ? {
  nullify
  normalizeSpeed xor {
    maxCustom [int16/ mps/ maxSpeed]
    max100
  }
}

configuration model cfgDoNothing configures FlightProcessor
  FlightProcessor_root {
  }

configuration model cfgNullifyOnly configures FlightProcessor
  FlightProcessor_root {
    nullify
  }

configuration model cfgNullifyMaxAt200 configures FlightProcessor
  FlightProcessor_root {
    nullify
    normalizeSpeed {
      maxCustom [maxSpeed = 200 mps]
    }
  }
[PLE Variability]

Trackpoint* process_trackpoint(Trackpoint* t) {
    {nullify}
    t->alt = 0 m;
    {max100}
    t->speed = 100 mps;
    {maxCustom}
    t->speed = maxCustom.maxSpeed;
    return t;
} process_trackpoint (function)

Trackpoint* process_trackpoint(Trackpoint* t) {
    t->alt = 0 m;
    return t;
} process_trackpoint (function)
## Controlled Names

<table>
<thead>
<tr>
<th>Name</th>
<th>Kind</th>
<th>Type</th>
<th>Unit</th>
<th>Value</th>
<th>Constraints</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLB_Time</td>
<td>quantity&lt;none&gt;</td>
<td>double</td>
<td>s</td>
<td>0.1 s</td>
<td>range 0.00 s .. 1.0e16 s</td>
<td>Time in seconds</td>
</tr>
<tr>
<td>Temperature_K</td>
<td>quantity&lt;none&gt;</td>
<td>double</td>
<td>K</td>
<td>300.0 K</td>
<td>range 223.0 K .. 1773.0 K</td>
<td>Temperature in Kelvin</td>
</tr>
<tr>
<td>Temperature_C</td>
<td>quantity&lt;none&gt;</td>
<td>double</td>
<td>degC</td>
<td>25.0 degC</td>
<td>range -50.0 degC .. 1250.0 degC</td>
<td>Temperature in Celsius</td>
</tr>
<tr>
<td>Torque</td>
<td>quantity&lt;none&gt;</td>
<td>double</td>
<td>Nm</td>
<td>0.0 Nm</td>
<td>&lt;no constraints&gt;</td>
<td>Torque in Nm</td>
</tr>
<tr>
<td>Inertia</td>
<td>quantity&lt;none&gt;</td>
<td>double</td>
<td>kgm2</td>
<td>0.0 kgm2</td>
<td>min 0.00</td>
<td>Inertia in kg m square</td>
</tr>
<tr>
<td>motor_speed</td>
<td>quantity&lt;none&gt;</td>
<td>double</td>
<td>radps</td>
<td>&lt;none&gt;</td>
<td>range 0.00 radps .. 100000.0 radps</td>
<td>Motor speed in rad per sec</td>
</tr>
<tr>
<td>shaft_speed</td>
<td>quantity&lt;none&gt;</td>
<td>double</td>
<td>radps</td>
<td>3.1 radps</td>
<td>range -20000.0 radps .. 20000.0 radps</td>
<td>Output Shaft Speed in rad per sec</td>
</tr>
<tr>
<td>motor_power</td>
<td>quantity&lt;none&gt;</td>
<td>double</td>
<td>W</td>
<td>2.1 W</td>
<td>range -100000.0 W .. 100000.0 W</td>
<td>Motor power in Watts</td>
</tr>
<tr>
<td>coolant_flowrate</td>
<td>quantity&lt;none&gt;</td>
<td>double</td>
<td>m3ps</td>
<td>2.5 m3ps</td>
<td>range 0.0 m3ps .. 3.0 m3ps</td>
<td>Coolant volume flow rate in m3 per sec</td>
</tr>
</tbody>
</table>

```cpp
exported double/s/ ->GLB_Time:ReqVars_StepInputErrorTol;
exported double/Hz/ ReqVars_Bandwidth;
exported double/degC/ ReqVars_MaximumTemperature;
exported double/radps/ ->motor_speed:ReqVars_NominalSpeed = 20.0 radps;
exported double/Nm/  ->Torque:ReqVars_NominalTorque;
exported double/degC/  ->Temperature_C:ReqVars_NominalAmbientTemperature = 25 degC;
exported double/rpm/  ReqVars_MaximumSpeed = 3500 rpm;
```
Architectural Abstractions first class
Code-integrated where useful
Analysis & Synthesis
Support Cross-Cutting Concerns
A different Perspective
[DSLs for Describing Architecture]
[DSLs as part of Systems]

System

DSL

expressed with

Model

describe
analyze
synthesize

Concern of System

An Architectural Decision!
[DSLs as part of Systems]

- Business Rules
- (Financial) Calculations
- Data Structures
- Mappings or Queries
- Validations
- Scientific Processes

Concern of System
[Examples]

① Insurance rules and products

\[
\text{local} = A1 \Rightarrow \sum_{i=1}^{NN} \left( \frac{(D(X + ANUI + i - 1) - D(X + ANUI + i)) \times \left(1 - \frac{TM18[i]}{TM17}\right)}{D(X + ANUI)} \right)
\]

② Tax/Benefits Rules (DTA Toeslagen)
Using DSLs is an Architectural Decision

Language Workbenches are the basis

Language Engineering is Efficient.
[Language-Oriented Applications]

System: LWB

DSL 1 → DSL 2 ← DSL 3
DSL 4 ← DSL 5 → ... → DSL N
If you have to build a tool, consider using an LWB as the foundation, and recasting the "application" as a set of languages.
If you have to build a tool, consider using an **LWB** as the foundation, and recasting the „application“ as a set of **languages**.
Generic Tools
Specific Languages
Tools are ways to work with Data.
work
{ author
  read
  analyze
  process
}
Data Formats are almost Languages.
almost
almost
Structure, Constraints, Semantics

Data Format
Structure, Constraints, Semantics

Language

Syntax + IDE

Data Format
almost

Structure, Constraints, Semantics

Data Format + Syntax + IDE

Language

Language Engineering

author
analyze
compose
execute
almost
Structure, Constraints, Semantics

Data Format  + Syntax + IDE

Language

Language Engineering
Language Workbenches

"Generic Tools"

author
analyze
compose
execute
A new Paradigm for Tools

„Language“ interpreted liberally – new styles for languages.
A new Paradigm for Tools

„Language“ interpreted liberally – new styles for languages.
Expressions  
„Code“  
Code Completion  
Error Highlighting  
Version Control  
Refactoring  
Debugging

Helper Buttons  
Tables  
Rigid Structures  
Tree Views  
Visualizations  
Live Interpretation  
Math Notation  
Graphical  
Prose + Code
Generic Tools
Specific Languages
Component Implementation

Component
name

Component Interface Requirement
name

providedInterface

interface
name

Operation
name

Parameter
name

required Interface

target

returnType

type

exception

Exception

Type
name
[Component Implementation]

1. Structural Element
2. Classification
3. Behavior Specification Formalism
A few more editing improvements in MPS.
More declarative languages to specify languages.

http://eelcovisser.org/wiki/projects/ldwb
Eelco’s Language Designer’s Workbench

**templates**

```plaintext
Definition.Function = <
  <Type> <ID> (<Param*; separator="","">) {
    <Statement*; separator="\n"> 
  }
>

Statement.If = <
  if(<Exp>)
  <Statement>
  else
  <Statement>
>
Statement.Return = <return <Exp>;;>

Exp.Add = <<<Exp> + <Exp>>>
Exp.Var = <<<ID>>>
```

**binding rules**

- `Param(t, name)`:
  - defines `Variable` name
- `Var(name)`:
  - refers to `Variable` name
- `Function(t, name, param*, s)`:
  - defines `Function` name
  - scopes `Variable`, `Function`
- `Call(name, exp*)`:
  - refers to `Function` name
More Competition.
More good LWBs.

http://languageworkbenches.net
Thank you!