MD* Best Practices

Markus Voelter
Independent Consultant/Itemis AG
www.voelter.de
voelter@acm.org

About

Markus Voelter
Independent Consultant/Itemis

Software Architecture
DSls & MDSD
Product Lines

About

Markus Voelter
Independent Consultant/Itemis

http://www.voelter.de
voelter@acm.org
skype: schogglad

Read the Paper!

http://www.voelter.de/data/articles/DSLBestPractices-Website.pdf

Limit Expressiveness

1 Designing the Language

30
Configuration

**select**
...from config space

... Properties
... Feature Models

Customization

**compose**
... creatively

... Vocabulary
... Sentences
... Box and Line

Precision

**precisely what**

... facts
... declarative
... domain experts can!

Algorithmic Completeness

**formally how**

... automation
... execution
... in model processors
... developers can!

Use a 3GL if necessary
Generate APIs, Hooks
Graphical vs. Textual

Domain Users care deeply about notation!

"UI" for the language

Graphical vs. Textual

Graphical

Sequence/Flow

A → B → C

Graphical

Relationships

Graphical

Textual

In all other cases!
Real simple...
...CVS/SVN Integration
... Diff/Merge
... Build automation
... Model Migration

Viewpoints

\[ \text{system} = \sum_{i=1}^{n} \text{viewpoint}_i \]

\[ \text{viewpoint}_i \Rightarrow (\text{abstractions}_i, \text{notations}_i) \]

Well-defined Dependencies and Connection Points
Try to avoid overlap and the need for synchronization.

Partitioning

MD* tools do not scale arbitrarily!

Partitions

... separate resources
... != logical structure
... units of check in/out
... processable separately

cross Partition references

... lazy
... by proxy
... by name w/ linker
Partition

... not transparent
... part of language design
... referencable elements
... „include path“

You can model everything with somehow!

Problem

Shoehorning domain abstractions into the generic language

Problem

Sidetracked by existing abstractions and notations

Theory

Notations/Abstractions extensible via Profiles

Practice

Very Limited Tool Support!
Meta Model Complexity!

But don’t reinvent the wheel either.

Where are standards useful?

People have to learn underlying concepts anyway.

Is UML with a profile still a standard language?

On which meta level do I want to standardize?
  M2 (UML), M3 (MOF)?
Isn’t a DSL based on MOF as „standard“ as a profile based on UML?

Introduction an intermediate language

UML Model
UML Metamodel
DS Model
DS Meta Model
Code

Separate viewpoints UML + DSL

UML Model
UML Metamodel
DS Model
DS Meta Model
Code

Learn from 3GLs

Namespaces
Scoping
Visibility
„Abstract“
Type Systems
Specialization ...

DSL ≠ 3GL

But:
Read this Book:

**Concepts, Techniques and Models of Computer Programming**
by Peter Van Roy and Seif Haridi

---

2. Processing the Models

**Interpretation vs. Generation**

**Interpretation**
resulting code can be easily inspected

**Generation**

---
**Generation**

resulting code can be easily **debugged**

**Interpretation vs. Generation**

resulting code can be **optimized** and more **efficient**

**Generation**

Templates can be **derived** from existing code

**Interpretation vs. Generation**

work around **limitations** of target language

**Generation**

no changes to target environment

(leaves no trace)

**Interpretation vs. Generation**

faster **turnaround**

no regeneration **test** **build** **deploy**
After code generation how do you make sure developers follow all the required procedures?

procedures?

... subclass
... overwrite
... naming conventions

Compiler Errors are not enough.

wrong abstraction level!

generate checks against the code base evaluated by the IDE

if (false) {
    GeneratedBaseClass x = new ManualSubclass();
}
Make Code True to Model

Analyses on the model can verify all kinds of properties about the system.

Iff the code is true to the model

Use a clever programming model that does not allow violations.

Model-2-Model
To Simplify Generators

generate the configuration for architecture analysis tools.
Reducing template complexity is important.

Separation of Concerns is the way to go.

Instead of putting complex logic into the templates, put it into an M2M that runs before code gen.

Model-2-Model For Simulation and Proof
Many useful formalisms already exist.

Simulation Proofs Properties

Use an M2M for this if possible.

Often the input is XML so you actually „generate code“

Cascading

PIM? PSM? PSSM?

Bottom Up
Works Better!
Annotation Models

Annotation Model references elements in base model.

Transformation takes additional information into account.

Make sure the annotation model only captures Exceptions from the default in the templates.
Classify Behaviour

Classify! ...
... state based 
... business rules 
... mathematics 
... or a specific DSL 
... 3GL code

Don’t forget Testing

Limited Expressiveness.
Reduced Need For Tests.

Constraint Checks.
A Form of Test.
Testing Generators

Reference Model → Generator → Code
Based On
Reference Test Cases → Tests
Binary

Testing Transformations

Reference Model → M2M → Result Model
Based On
Reference Constraints → Tests

Testing Metware

Reference Model
Reference Test Cases
Reference Constraints

... maintained!
... by metaware developers

Self-Fulfilling ....!

Model → Generator → Code
Tests

Generator → Test Code

Self-Fulfilling ....!

Don't Forget Testing

Don't Forget Testing

Don't Forget Testing

Don't Forget Testing
3 Process the Organization

Iterate!

Waterfall is bad! With or Without MD*

Co-Evolve Language and Concepts
Understanding the Domain

Building The Language

Building a language requires Formalization

requires frequent Evolution!

Building a language requires you to think and decide about the domain.

and flexible, agile Tooling!
Documentation is still necessary

The DSL and the „programs“ are documentation.

The DSL and the „programs“ are
documentation.

Not Quite!

Language Definition is not a
Teaching Tool!

Language Definition
captures the WHAT
but not the WHY

Tutorials
... Concepts
... How to use Language
... How to integrate manual code

Example-Driven!
Rationales
... why the concepts?
... why we generate
what we generate
... target platform decisions and idioms

Different Media

Compatible Organization
MD* requires cross-project work.

Compatible Organization
MD* requires cross-project work.
A strict project-focused organization does not work

Compatible Organization

Make room & budget for cross-cutting work.

Open Source?
THE END.
Thank you.
Questions?