Goals

Greater Abstraction
... work on a higher level
... do more with less
... insulate from technology

Increase Value
... productivity
... quality
... responsiveness

Reduce lossiness
... domain → requirements
... requirements → design
... design → program
Goals

Not just any “MDD”
... domain-specific language
... domain-specific generator
... mostly made not bought
Let people do what they’re good at

MDD has several clearly defined Roles

Tech Experts
- evaluate technologies
- dig deep
- tune
- create templates
- spread knowledge

Language Designer
- works w/ domain expert abstractions, notations
- works w/ architect generators, interpreters
- requires “meta people”

App Developer
- cares about app domain
- uses DSLs + meta ware
- is isolated from technology

Flip side:
You actually need people who are good at this!
Only Gurus Allowed

Only gurus build languages 4%
I’m smart & need no help 12%

Worst practices: Initial conditions

Domain Users Programming?

Precision ≠ Programming

Domain Users Programming?

Precision ≠ Programming

… Scientists
… Insurance Mathematicians
… Logisticians
… Medical Doctors
… Hardware Folks

Domain Users Programming?

Domain Users Programming = Precision + X

Domain Users Programming?
Domain Users vs. Experts

Creating the Language vs. Using the Language

Creating: Domain Expert
Using: Domain User

Domain Users vs. Experts

Creating: Domain Expert
... senior
... complete
... big picture
... deep
... precise, formal
... guru

Using: Domain User
... not senior
... narrower
... shallow
Domain Dilettante
Insufficient understanding:
- Problem domain 17%
- Solution domain 5%

Never, ever delegate to interns!

Iterate!

Waterfall is bad!
With or Without MDD

Iterate!

Iterate!
Iterate!

Analysis Paralysis
Language must be known to be complete, fully implementable 8%

Instead: Prototype!
... meaningful
... 4-8 person weeks
... incremental
... external help?
MDD requires cross-project work.

A strict project-focused organization does not work.

Make room & budget for cross-cutting work.

Open Source-like Approach?

Suitability criteria

... Mature business area
... Work mostly in-house
... Experts available
3 Language Design

Sources for the Language

UML: Old Wineskins
Extend a large, general-purpose language  5%

Worst practices: Concept Source

+ !=

3GL: Visual Program
Traditional programming language + graphics  7%

Worst practices: Concept Source
Domain Analysis

Business DSLs

extracted from

<table>
<thead>
<tr>
<th>Domain Knowledge</th>
<th>Domain Artifacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Interview Experts</td>
<td>• Hardware Structure</td>
</tr>
<tr>
<td>• Mine existing formalisms</td>
<td>• Devices</td>
</tr>
<tr>
<td>• Word, Excel</td>
<td>• User Interface</td>
</tr>
</tbody>
</table>

Language Sources

Technical DSLs

extracted from

... framework
... library
... pattern (language)
... architecture

Technical DSLs

Code Components

The library is the language

32%

Worst practices: Concept Source

Library is the Language

...low-level details
...duplication
...can’t retarget
...level of abstraction?

Worst practices: Concept Source

Tool: hammer ⇒ nails

Tool’s technical limitations dictate language

14%
Worst practices: Concept Source

Limit Expressiveness

Too generic/specific
- Too few/generic 21%
- Too many/specific 8%
- Language for 1 model 7%

Too few concepts
- Not a DSL!
- Add heating, lights?
Precision

precisely what

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

Algorithmic Completeness

formally how

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

Limit Expressiveness

Use a 3GL if necessary
Generate APIs, Hooks

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!

No training
“Everyone understands the language like me” 21%

No training
“Everyone understands UML like me” 100%

Tutorials
... Concepts
... How to use Language
... How to integrate manual code
Example-Driven!

Language Definition captures the WHAT but not the WHY
Rationales
... why the concepts?
... why we generate
what we generate
... target platform decisions and idioms

Different Media

Notation

Notation, Notation, Notation
Domain Users care deeply about notation!

“UI” for the language

Textual
Semi-Graphical
Graphical
Form-Based
Matrices, Tables

Parts

Convertible

Embedded

Tool Specific!
Graphical vs. Textual

Graphical

Choices/Flow

Graphical

Relationships

Graphical

Timing

Textual

In all other cases!
Textual

Real simple...
...CVS/SVN Integration
... Diff/Merge
... Build automation
... Model Migration

Text + Visualization

Graphical vs. Textual

Predetermined Paradigm
Choosing wrong notation type because of blinkered view 7%

Graphical Syntax

Symbol should have 1:1 mapping to Domain concept

Worst practice: notation

Notation, ...
Symbol should call to mind Domain concept

Symbols should use full range of visual variables

<table>
<thead>
<tr>
<th>PLANAR VARIABLES</th>
<th>RETINAL VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horizontal Position</td>
<td>Shape</td>
</tr>
<tr>
<td>Vertical Position</td>
<td>Brightness</td>
</tr>
</tbody>
</table>

pictogram > geometric > photo

Simplistic symbols
Too simple/similar 25%
Downright ugly 5%

Color & label not enough
And by the way:

Notation, ...

And by the way:

why not just use text?

From Models To Code

Checks First and Separate
Concepts + Structure are not enough.

You need **constraints**.
Boolean expressions that validate the model beyond structure.

**Best:** constraints as data
- part of language

**Worst:** constraints as code
- requires impact analysis to scale
- but often needed

Checks first and separate

Model

G

Constraints

Code

... complex

Model

G

Constraints

Code

... complex

... duplication

Checks first and separate

Model

G

Constraints

Code

G'

Constraints

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

G

Constraints

Code

G'

Constraints

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'

Checks first and separate

Model

Constraints

G

G'

Code

Code'
check early. more semantics. better messages.

Constraints 1 ok implies constraints 2 ok implies constraints 3 ok

ERROR WARNING INFO

Rich Domain Specific Platform

Generated Application
- Domain Frameworks
- Libraries
- Middleware
- Operating System
- Drivers
Domain Frameworks
Libraries

Grown with the DSL!

Extreme Case

Generated Application
populates
Domain Frameworks

Care about generated code

Yes.
Can be regenerated.

Generated Code a Throwaway Product?

Generated Code a Throwaway Product?
But: must be...
... written (templates)
... understood
... debugged

But: must be...
... written (templates)
... understood
... debugged
if you don’t generate
100%: ... extended
... programmed against

Care!
... indent
... use good names
... document
... modularize

THE END.
...of part 1

come back tomorrow...
...for even better, worse and uglier stuff.
Questions?

MDD: the Best, the Worst and the Ugliest

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

Steven Kelly
CTO, MetaCase
www.metacase.com
stevek@metacase.com

Part 2

Quick Recap: Part 1
Breaking up Models

Viewpoints

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

\[ \text{viewpoint}_i \mapsto (\text{abstractions}_i, \text{notations}_i) \]
Well-defined Dependencies and Connection Points

Try to avoid overlap and the need for synchronization.

Tool Specific!

Model Integration

Decomposition

... Logical structure
... Sub-models
... Max 20 elements each
... Split 7±2 ways / level
Summary Model
... Top-level overview
... Shows all sub-models
... Shows sub-model links

Side-by-side view
... 2+ models on screen
... Reduces memory load
... User chooses
... User positions

Cross-model links
... Show referred objects
... Real object or pointer
... Use sparingly: coupling

Configuration Viewpoint
Teamwork
Partitions
Viewpoints
Multiple Targets
Multiple Configurations
Partitioning

MDD tools do not scale arbitrarily!

- performance
- overview
- search/find/organize
- teamwork

Partitions

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

Partition

- not transparent
- part of language design
- referencable elements
- “include path”

Cross Partition references

- lazy
- by proxy
- by name w/ linker
Partitioning Tool Specific!

Annotation Models

Worst practices: ignore use

Ignoring Use Process 42%
Copy entire diagram to handle minor variation?

Worst practices: ignore use

Worst practices: ignore use

Annotation Models

Ignoring Use Process 42%
Copy entire diagram to handle minor variation?

Annotation Models

Architecture DSL

Niche DSL

Business Domain 1 DSL

Code + other target platform artifacts
Niche DSL

Coding Domain 1 DSL

Annotation

Model

Architecture DSL

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 generators.
Interpretation vs. Generation

Interpretation

Generation

resulting code can be easily inspected

Generation

resulting code can be easily debugged

Generation

resulting code can be optimized and more efficient
Generation

Templates can be derived from existing code

Interpretation vs. Generation

Generation

work around limitations of target language

Interpretation vs. Generation

Generation

reuse

target runtime system

Interpretation vs. Generation

Interpretation

faster
turnaround

no regeneration

test

build
deploy

Interpretation vs. Generation

Combinations

<table>
<thead>
<tr>
<th>Generation</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>Java VM</td>
</tr>
<tr>
<td>DSL</td>
<td>?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Interpretation</th>
<th>Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bytecode</td>
<td>XML</td>
</tr>
<tr>
<td>JIT-Compiler</td>
<td>?</td>
</tr>
</tbody>
</table>
Integrating Generated and Manually Written Code

If at all possible, Do not modify Generated Code!

Protected Regions are a bad idea because generated code is not a throwaway anymore.

Protected Regions ... need to check in ... “sediment” of generated code!

Better Approach: \textbf{Hooks} in the generated code

Better Approach: \textbf{Hooks} in the generated code

extension points, base class, abstract methods & subclassing, empty callback methods, delegate, implement interfaces, \#include, reflection, AOP, design patterns, partial classes
Care About Generators

Generators
... important asset
... contain platform knowledge
... tend to grow more complex

Care!
... modularize
... functions
... naming
... polymorphism
... aspects
... refactor

Indent for the generator and then use a beautifier for the generated code.
Indent for the generator and then use a **beautifier** for the generated code.

except if you use a language with semantic whitespace!

Good Platform, Simpler Generator, Less Care.

Simplifying Generators

Reducing generator complexity is important.

Reducing generator complexity is important.

Separation of Concerns is the way to go.
Instead of putting **boilerplate** or **complex code** in generators as **fixed text**, refactor it into the domain framework.

Instead of putting **complex logic** into the generators, use **O-O principles & metaprogramming** to keep them flexible.

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

Architecture DSL

M2T

Code + other target platform artifacts

Business Domain 1 DSL

M2M

Business Domain 2 DSL

M2M

Business Domain 1 DSL

M2M

Business Domain 2 DSL

M2M

Niche DSL

M2T

Code + other target platform artifacts

Niche DSL

M2T

Code + other target platform artifacts
Don’t forget Testing

Limited Expressiveness. Reduced Need For Tests.

Constraint Checks. A Form of Test.

Testing Generators

Testing Generators

Reference Model Generator Code

Based On
Reference Test Cases Tests Binary
Testing Transformations

Reference Model → M2M → Result Model

Testing Transformations

Reference Model → M2M → Result Model

Based On
Reference Constraints

Tests

Testing Metaware

Reference Model
Reference Test Cases
Reference Constraints

... maintained!
... by metaware developers

Generating tests

Model

Generator
Code

Tests

Generator
Test Code

Self-Fulfilling ....!

Don't Forget Testing

Reference Model
Reference Test Cases
Reference Constraints

Generating tests

Model

Generator
Code

Tests

Test Model
Generator
Test Code

Don't Forget Testing

Reference Model
Reference Test Cases
Reference Constraints

Don't Forget Testing

Reference Model
Reference Test Cases
Reference Constraints
Worst practice: In Use

Ignoring real-life process of using language

42%

Multiple users
Multiple models
Multiple versions

All interlinked
Trying to model like you coded

modeling $\neq$ coding

Same old problems but new material

Old solutions don’t apply

... new processes

... new tools

Diff + merge:

Text easy, models hard

Multi-user editing:

Text hard, models easy

Co-Evolve Language and Concepts
Understanding the Domain

Building The Language

Building a language requires Formalization

requires frequent Evolution...

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

... hence flexible, agile Tooling!
Letting language stagnate after successful start

37%

Evolution

What to do with existing models if the language changes?

Might Require...

... config management
... version tag in models
... change tracking
... migration M2M

Think about...

... backward compatibility
... deprecation
... instrumentation
... viewpoints + partitions
Evolution

...of it all.

THE END.

Questions?