Domain-Specific Modeling
76 cases of MDD that works

17 November 2009
15:00-16:30 GMT
Steven Kelly
Outline

- Why Domain-Specific Modeling?
- Examples
- How to build
- Is this for you?
A rise in productivity is overdue

- "The entire history of software engineering is that of the rise in levels of abstraction"
  Grady Booch

- New programming languages have not increased productivity

- Abstraction of development can be raised above current level...

- ... without losing control or accepting substandard results

*Software Productivity Research & Capers Jones, 2002*
Enterprise smartphone app demo
Why not use standard UML?

- Using standard UML is really no faster than just coding
  - Scientific measurements from 48% slower to 10% faster

![Bar chart showing mean time to correctly complete 5 tasks for UML+Java and Java.]

- We need something more than standard UML!
Abstraction benefits

... work on a higher level

... do more with less

... insulate from technology
Worst Practices for Domain-Specific Modeling
Steven Kelly, Risto Pohjonen
Free from: www.metacase.com/stevek.html

- 76 DSM cases
- 15 years
- 4 continents
- several tools
- 100 DSL creators
- 3–300 modelers
UML: Old Wineskins

Extend a large, general-purpose language
3GL: Visual Program

Traditional programming language + graphics

1 symbol => 1 keyword 😞
MDA: UML+UML+UML

Multiple semi-automatic transformations
MDA: UML+UML+UML

Multiple semi-automatic transformations

Manufacturers’ claims:
+22% (Obeo) ... +35% (OpJ)

Not enough!
Booch, Rumbaugh and Selic say:

- “the full value of MDA is only achieved when the modeling concepts map directly to domain concepts rather than computer technology concepts”
  - An MDA Manifesto, MDA Journal, May 2004

- Use language of problem domain
- Generate language of solution domain
Modeling functionality vs. modeling code

Domain Idea

- Solve problem in domain terms
  - Map to code, implement
  - Map to UML

Finished Product

- Assembler
- Code
- UML Model

Model in DSM language

No need to map!

Generate code

Domain Framework
Case: Financial web application

- Developing portal for insurances and financial products
- Need to specify several hundred financial products
- Insurance experts visually specify insurance products and generate code to the portal

- Comparison to hand-writing Java after first 30 products = DSM 3-5 times faster, 50% fewer errors
public class Basis extends ProductRepository
{
    public Basis(String name)
    {
        super(name);
        PRODUCT_NAME = Basis;
        NofPackage productpackage = createProduct();
        this.addNofPackage(productpackage);
    }

    public Basis()
    {
        // name of namespace ProductRepository not used
        this(Basis);
    }

    private NofPackage createProduct()
    {
        productpackage_ = new NofPackage(PRODUCT_NAME);
        // Global Instances, will be re-used by each section
        NofAttribute attribute;
        NofAssociation nofAssociation;
        Constant constant;
        AssociationEnd end1;
        AssociationEnd end2;
        Reference reference;

        // ********************************************
        // Tags
        // ********************************************
        beitragsicht_ = new Tag("Tarifierung", NofModelConstants.TAGID_TARIF);
        productpackage_.addContainedTag(beitragsicht_);
        selektionssichttrue_ = new Tag("Selektion_true", NofModelConstants.TAGID_AUSWISSEND);
        selektionssichttrue_.addTagValue("True");
        productpackage_.addContainedTag(selektionssichttrue_);
        angebotsicht_ = new Tag("Angebot", NofModelConstants.TAGID_ANGEBOT);
        productpackage_.addContainedTag(angebotsicht_);

        // ********************************************
        // Exceptions
        // ********************************************
        NofException Exception1 = new NofException("Exception1");
        parameter_ = new Parameter("ExceptionParam1", new DataType("Number"));
        addParameter(parameter_);
        parameter_ = new Parameter("ExceptionParam2", new DataType("String"));
        addParameter(parameter_);
        Exception1.setExceptionParameters(addParameter(parameter_, parameter_));
    }
}
Case: Call Processing Services

- Specify services than can run safely on Internet telephony servers
- Designs can be considered valid and well-formed right from the design stage
- Language uses concepts familiar to the service developer
  - Switches, Locations and Signaling actions etc.
- Generate full service from the model

➤ Creation of new services 6 times faster compared to manual practices

© 2008 MetaCase
<?xml version="1.0" ?>
<!DOCTYPE cpl PUBLIC "-//IETF//DTD RFCxxxx CPL 1.0//EN" "cpl.dtd" -->
  <cpl>
    <subaction id="voicemail">
      <location url="sip:jones@voicemail.example.com">
        <redirect />
      </location>
    </subaction>
    <incoming>
      <address-switch field="origin" subfield="host">
        <address subdomain-of="example.com">
          <location url="sip:jones@example.com" priority="3" clear="No">
            <proxy timeout="10" recurse="No" ordering="Parallel">
              <busy>
                <sub ref="voicemail" />
              </busy>
              <failure>
                <sub ref="voicemail" />
              </failure>
              <noanswer>
                <sub ref="voicemail" />
              </noanswer>
            </proxy>
          </location>
        </address>
      </address-switch>
    </incoming>
  </cpl>
Case: Process of medical mixing

- Single fixed physical machine
- Product variability in software: how to mix
- Platform has 11 cups, syringe to transfer between
- Problems in hand-coded software quality:
  - Syringe broken, operator died, patient treatment error
- Competition by ICT to find best DSM solution
  - Reduced from 277 pieces of input to 29
class DNAMixingMachine extends MixingMachine
{
    int dna_consultation()
    {
        double i, j;
        shut(0);
        move(-4);
        if (scan() == 9) return -1;
        suck(9); move(5); blow(9);
        move(-2); suck(30); move(1); blow(30);
        move(-3);
        if (scan() < 6) return -1;
        i-scan(); suck(1); move(5); open(0); blow(1);
        move(-3); suck(30); move(1); blow(30);
        move(1); fill(1); suck(3); move(3); fill(0); blow(3);
        move(-5); suck(30); move(1); blow(30);
        move(2); fill(2); suck(2);
        move(2); fill(0); blow(2);
        wait(8);
        move(-5); suck(30); move(1); blow(30);
        move(i); fill(1); suck(3); move(3); fill(0); blow(3);
        move(-5); suck(30); move(1); blow(30);
        move(2); fill(2); suck(2);
        move(2); fill(0); blow(2);
        wait(8);
        move(-5); suck(30); move(1); blow(30);
        move(i); fill(1); suck(3); move(3); fill(0); blow(3);
        move(-5); suck(30); move(1); blow(30);
        move(2); fill(2); suck(2);
        move(2); fill(0); blow(2);
        wait(8);
        move(-5); suck(30); move(1); blow(30);
        move(2); i-scan(1); suck(1); move(3);
        if (i < -10) { blow(1); j--; }
        else { blow(10); j = 10; }
        move(-1); blow(i-j);
        wait(7);
        move(-5); suck(30); move(1); blow(30);
        move(4); suck(3+2+3+2+3+2+1-j); move(1); blow(3+2+3+2+3+2+1-j);
        wait(12);
        suck(3+2+3+2+3+2+1); shut(1); move(1); blow(3+2+3+2+3+2+1);
        move(7); open(1); suck(30); move(1); blow(30);
        return 0;
    }
}
Productivity increase from DSM

Domain

- Heart rate monitor: 1000%
- Call processing services: 600%
- Touch screen UI applications: 500%
- Home automation: 600%
- Mobile phone software: 1000%
- Phone switch features: 750%
- Financial web application: 500%
DSM consistently 5-10x faster
DSM Solution Development Time

- Heart rate monitor
- Call processing services
- Touch screen UI applications
- Home automation
- Mobile phone applications
- Automotive infotainment systems
- Financial web application

Days to create DSM language & generator

© 2009 MetaCase
How to implement DSM

Domain Idea

DSM language

Code generator

Framework code

Finished Product

Domain Idea

DSM language

Code generator

Framework code

Finished Product

Done a few times before!

Normal (many)

Expert (few)

Easy!

Model in DSM language

Generate code

Domain Framework

© 2009 MetaCase
Getting started

A journey of 1000 miles begins with a single step

Setting off on the wrong foot can spoil the whole journey
Only Gurus Allowed

Only gurus build languages 4%

I’m smart & need no help 12%
Domain Dilettante

Insufficient understanding:
- Problem domain 17%
- Solution domain 5%
Never, ever delegate to interns!
Analysis Paralysis

Language must be known to be complete, fully implementable 8%
Sources for the Language
Tool: hammer $\Rightarrow$ nails

Tool’s technical limitations dictate language 14%
Approaches to identify concepts

■ “What concepts should my language have?”
  - Hard problem for DSM beginners
  - Analysed 23 cases to find good toolbox of approaches*

■ 4 main approaches:

<table>
<thead>
<tr>
<th>Problem domain</th>
<th>Solution domain/ generation target</th>
<th>Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Telecom services</td>
<td>Configuration scripts</td>
<td>1</td>
</tr>
<tr>
<td>Insurance products</td>
<td>J2EE</td>
<td>1</td>
</tr>
<tr>
<td>Business processes</td>
<td>Rule engine language</td>
<td>1</td>
</tr>
<tr>
<td>Industrial automation</td>
<td>3 GL</td>
<td>1, (2)</td>
</tr>
<tr>
<td>Platform installation</td>
<td>XML</td>
<td>1, (2)</td>
</tr>
<tr>
<td>Medical device configuration</td>
<td>XML</td>
<td>1, (2)</td>
</tr>
<tr>
<td>Machine control</td>
<td>3 GL</td>
<td>1, 2</td>
</tr>
<tr>
<td>Call processing</td>
<td>CPL</td>
<td>2, (1)</td>
</tr>
<tr>
<td>Geographic Information System</td>
<td>3 GL, propriety rule language, data structures</td>
<td>2</td>
</tr>
<tr>
<td>SIM card profiles</td>
<td>Configuration scripts and parameters</td>
<td>2</td>
</tr>
<tr>
<td>Phone switch services</td>
<td>CPL, Voice XML, 3 GL</td>
<td>2, (3)</td>
</tr>
<tr>
<td>eCommerce marketplaces</td>
<td>J2EE, XML</td>
<td>2, (3)</td>
</tr>
<tr>
<td>SIM card applications</td>
<td>3 GL</td>
<td>3</td>
</tr>
<tr>
<td>Applications in microcontroller</td>
<td>8-bit assembler</td>
<td>3</td>
</tr>
<tr>
<td>Household appliance features</td>
<td>3 GL</td>
<td>3</td>
</tr>
<tr>
<td>Smartphone UI applications</td>
<td>Scripting language</td>
<td>3</td>
</tr>
<tr>
<td>ERP configuration</td>
<td>3 GL</td>
<td>3, 4</td>
</tr>
<tr>
<td>Handheld device applications</td>
<td>3 GL</td>
<td>3, 4</td>
</tr>
<tr>
<td>Phone UI applications</td>
<td>C</td>
<td>4, (3)</td>
</tr>
<tr>
<td>Phone UI applications</td>
<td>C++</td>
<td>4, (3)</td>
</tr>
<tr>
<td>Phone UI applications</td>
<td>C</td>
<td>4, (3)</td>
</tr>
<tr>
<td>Phone UI applications</td>
<td>C++</td>
<td>4, (3)</td>
</tr>
</tbody>
</table>
1. Domain expert’s concepts

- Concepts from domain
- Mostly made without help
- Simple code generation
- OK in established domain
- Usable by non-coders

Insurance products/J2EE
2. Generation output

- Modelling constructs come from code artefacts
- Static parts are easy
  - Data structures
  - Core XML elements
- Dynamic behaviour harder
  - Avoid “graphical 3GL”
  - Need domain framework
- Danger: low level of abstraction
  - Little productivity gain
- But works well with DSL or XML

Internet telephony/CPL
3. Look and feel of end system

- Best for physical end product
  - UI on PC, embedded, speech
- Often state machine basis
  - Extend with data & control flow
  - Power of relationships
- Visible domain concepts
  - Easy to identify
  - High level of abstraction
- Domain framework hides code
  - Don’t write code in models...
  - ...unless you really have to!
- Generators considered easy

Smartphone apps/Python
4. Variability space

- Language concepts capture variability space
- Modeler makes variant choices
  - Composition, relationships, values
- Infinite variability space (Czarnecki)
  - Not just feature tree: unbounded product family
- Static variance easy, dynamic harder
- Predict future variability ⇒ high level of abstraction
Evaluation of the Approaches

- Hierarchy of approaches
  - From less to more experienced DSM practitioners
- 1. Domain expert’s concepts – "we just did it"
- 2. Generation output
  - Generic/ad hoc language not so good
  - Established DSL good
- 3. Look and feel: common, easy, true DSM
- 4. Variability space: adds power to handle complexity
  - Found in very different domains

Best results combined 3 (L&F) and 4 (Variability)
Too generic/specific

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

Too simple/similar 25%
Downright ugly 5%
Not like this...

Not a DSL!

Add heating, lights?
What about the generator?

Generator translates the models into the required output
1. crawls through the models → navigation based on metamodel
2. extracts required information → access data in models
3. outputs it into the code → mixing fixed text and model data
4. with translation where necessary → e.g. space to underscore, XML legal
How to make a generator

- Make generated code based on current hand-written code
  - Removes risk of slow, bloated or unreadable code
  - Follow good coding standards, include comments, have data to link back to models (e.g. in comment or via simulator)

- Make generation process complete, target 100% output
  - Never modify the generated code
    - Correct the generator or framework instead
    - Or use add-in hand-coded functions

- Put domain rules up-front to the language
  - Generator definition becomes easier when the input is correct

- Try to generate as little code as possible
  - Glue code only, rest in domain framework or platform

- Keep generator as simple as possible
  - Push low-level implementation issues to the framework
  - Keep generator modular to reflect changes
Is DSM worth it?

- Language workbenches make moving to DSM feasible
  - Can focus on language design, not on creating tooling
- Creation of languages does not take much time
Is DSM for you?

- You probably can’t know yet!
  - Have to try it out for your situation
- If it does fit your situation, the 500-1000% makes it probably the single most important thing you can do
  - More than SOA, cloud, Ruby, agile, ...

How to see quickly if it would work for you:

- Look for repetition in your work, e.g.:
  - many screens and database tables
  - many products in family
  - etc.
- Try out a tool
  - Tutorials
  - Have a go with your domain (ask for hints)
Thank you!

Questions?

MetaCase

www.metacase.com
Literature and further links

- Blogs: www.metacase.com/blogs