Domain-Specific Modelling: The Killer App for Method Engineering?

12th September 2007
9:30 — 10:30
Steven Kelly, MetaCase

Exponential growth is not smooth

- Moore: 18 months
- Computer: 33
- Programmer: 88
- 1980s PC revolution
- 2000 mobile revolution
- Computers/user:
  - 1970: 1:10
  - 2010: 10:1
- Customise with software
- Software crisis worsening!
How has productivity improved?

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

- New programming languages have not increased productivity
- UML and visualization of code have not increased productivity

- Abstraction of development can be raised above current level...
- ... and still generate full production code (and ignore it!)

*Software Productivity Research & Capers Jones, 2002

Contingency theory and software development

- One size does not fit all: Diversity due to:
  - type of systems built
  - technology used to implement them
  - organizations, cultures, tools etc.

- Laboratory studies show that developers understand and use methods differently
  - Extend, give new meanings, create own interpretations etc. for modeling constructs (e.g. in studies by Wijers, Verhoef)

- Contingency theory advocates flexible languages:
  no single language gives best result in all situations
  - IFIP WG conferences (Olle et al. 1982, -83, -86, -88)

- Empirical studies show that companies prefer own methods
  - 2/3 use internal, home-grown methods, Russo et al., Fitzgerald

- Create your own modeling language and generators?
  - Domain-Specific Modelling
What is Domain-Specific Modelling?

Introducing Domain-Specific Modelling (DSM)
Where to apply – and why
Real life examples from various domains

Modelling functionality vs. modelling code

Domain Idea

- Solve problem in domain terms
- Map to code, implement
- Map to UML
- No need to map!

Domain Framework

- Model in DSM language
- Generate code
- Domain Framework

Assembler

- Map to code, implement
- Generate, Add bodies

Finished Product

- Code
- UML Model
Why is the vision possible (now)?

- Need to fit only **one** company’s requirements!
- Modelling is Domain-Specific (unlike 90’s UML)
  - Works for one company / product group, application domain / framework / product family
  - Language has concepts people are already familiar with
  - Models used to solve the problem, not to visualize code
- Generator is Domain-Specific (unlike 80’s CASE)
  - Generate just the code needed from models
    - Efficient full code
    - No manual coding afterwards
    - No reason for round-tripping
  - Generator links to existing primitives/components/platform services etc.
  - Can generate 3GL, Assembler, object-oriented, XML, etc.

Real-life cases, various domains

- Insurance products & eCommerce   Java
- VoiceMenu for microcontroller   Assembler
- Business Process Modelling for XPDL   XML
- Call Processing Language   XML
- Web application   Java, XML
- Robotics   C
- Infotainment systems
- Enterprise apps in Smartphones   Python
- Symbian native applications   C++
- IMS Service Creation
Business Process Modelling for XPDL ⇒ XML

© 2007 MetaCase

Call Processing Language ⇒ XML

© 2007 MetaCase
Infotainment systems

Enterprise apps in Smartphones ➔ Python
Symbian native applications ⇒ C++

IMS Service Creation
Where to apply?

- Repetitive development tasks
  - Large portion of the work similar to earlier products
  - Several products made in parallel
  - Useful to narrow down the design space
- Domain expertise needed
  - Non-programmers can participate
- These normally include:
  - Product Family
  - Platform-based development
  - Configuration
  - Business rule definitions
  - Embedded devices

Tools for DSM:
Building a modeling language — i.e. real Method Engineering!
How to implement DSM
Done a few times before!

Domain Idea

Expert (few)

Normal (many)

Easy!

Model in DSM language

Generate code

Domain Framework

Generate code

Framework code

Finished Product

Tools for DSM creation and use

- 6 ways to get the tools we need for DSM
  1. Write own modelling tool from scratch
  2. Write own modelling tool based on frameworks
  3. Metamodel, generate modelling tool skeleton, add code
  4. Metamodel, generate full modelling tool over a framework
  5. Metamodel, output configuration for generic modelling tool
  6. Integrated modelling and metamodeling environment

- Good tools minimize resource use (few man-weeks)
  - create modelling tools automatically
  - guide in DSM creation
  - allow you to test DSM throughout domain design process

- Good tools allow DSML to evolve
A Brief History of Tools

- Tools for textual languages (late 70’s ->)
  - SEM (Teichroew and Yamato)
  - Others include Plexsys, Metaplex, Quickspec, PSL/PSA
- Tools for graphical languages (mid 80’s)
  - Swedish Ramatic: set theory for metamodeling
  - British Eclipse (not THAT Eclipse!): directed graphs
- Tools for graphical metamodeling (late 80’s)
  - Finnish Metamodeling Editor MetaEdit: OPRR
  - + dozens of others over the years: MetaView, Kogge, Virtual Software Factory, Paradigm+ SDK, Customizer in Excelerator, IPSYS ToolBuilder, ConceptBase, Dome, GME etc.
    - Most of the tools focus on initial language specification and editor construction

Currently many 'version 1.0' tools — again!

- Single user
- Single modelling language at a time
- Simple metamodels
  - Focus on objects, basic properties, binary relationships
- Simple notation
  - Single graphical element + label
- Generator is text-to-text
  - Or hand-written code to read models
- Resulting modelling tool primitive
  - Missing majority of functions users expect in such a tool
- Upgrading modelling language invalidates models
- Upgrading tool framework invalidates tool
Comparison of EMF+GEF and MetaEdit+ for DSM, MDSD Workshop, OOPSLA '04
© 2007 MetaCase

1 hour            10kLOC=1yr=2000h

package org.eclipse.gef.examples.logicdesigner.model;
import org.eclipse.gef.examples.logicdesigner.LogicMessages;
import org.eclipse.ui.views.properties.IPropertyDescriptor;
import org.eclipse.ui.views.properties.PropertyDescriptor;
import org.eclipse.ui.views.properties.TextPropertyDescriptor;
public class LED
extends LogicSubpart
{
public static String P_VALUE = "value";
protected static IPropertyDescriptor[] newDescriptors = null;
static{
PropertyDescriptor pValueProp = new TextPropertyDescriptor(P_VALUE,
LogicMessages.PropertyDescriptor_LED_Value);
pValueProp.setValidator(LogicNumberCellEditorValidator.instance());
if(descriptors!=null){
newDescriptors = new IPropertyDescriptor[descriptors.length+1];
for(int i=0;i<descriptors.length;i++)
newDescriptors[i] = descriptors[i];
newDescriptors[descriptors.length] = pValueProp;
} else
newDescriptors = new IPropertyDescriptor[]{pValueProp};
}

public Object getPropertyValue(Object propName) {
if (P_VALUE.equals(propName))
return new Integer(getValue()).toString();
if( ID_SIZE.equals(propName)){
return new String("(+getSize().width,+getSize().height)");}
return super.getPropertyValue(propName);}

public void resetPropertyValue(Object id){
if (P_VALUE.equals(id))
setValue(0);
else
super.resetPropertyValue(id);}

public void setPropertyValue(Object id, Object value){
if (P_VALUE.equals(id))
setValue(Integer.parseInt((String)value));
else
super.setPropertyValue(id,value);
}
}
What is needed in a mature tool?

- Meta-metamodel concepts
  - Explicit concepts for Graph, Role, Port
  - N-ary relationships
- Constraints
  - Easy & powerful for relationships, property values
  - Language for arbitrary constraints
- Model integration
  - Objects as property values
  - Reuse of objects across models
  - Links to subgraphs
- Notation and modelling tool functionality
  - Models automatically update when metamodel changes
- Purpose-built generator language and tools

Research Issues and Challenges

- Reuse
  - Whole models and individual model elements
  - Sequential and parallel
  - Tool support for modeller vs. rules by metamodeller
- Versioning
  - Metamodel level, upgrading the language
  - Model level, with graphical diff
- Scaling
  - Multiple users, concurrency vs. split & merge
  - Multiple projects, sequential and parallel

- These all conflict with each other!
  - At least if we try to apply familiar code-based practices
Roadmap hints for ME research

1. Understanding the state of the art
   - mature tools
   - previous research
2. Empirical research on the use of DSM in the field
3. Condensing 1 & 2 into advice for method engineers
4. Hypothesizing from 1 & 2 a set of new requirements for the next generation of tools

Prerequisites for success:
- Don’t skip steps!
- Industry maturing past “everything is UML”
- Academia going beyond “ME = mix&match existing bits”

Thank you!

Try MetaEdit+ for free, buy your own for 150€, get 20 for 50€ each!
Further reading

- Pohjonen, R., Kelly, S., Domain-Specific Modelling, Dr. Dobb's, 8, 2002.