Domain-Specific Modeling
Industrial strength MDD

30 September 2010
Steven Kelly

Outline
- Rise in productivity overdue
- Why not UML or MDA?
- DSM demo
- DSM cases
- How to build DSM languages
- Tools for DSM

Why not use standard UML?
- Using standard UML is really no faster than just coding
  - Scientific measurements from 48% slower to 10% faster
  - Recently ~15% in: WJ Dzidek, E Arisholm, LC Briand,
- 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
⇒ what doesn’t work
**Worst practices:**

**Concept Source**

**UML: Old Wineskins**

Extend a large, general-purpose language

![Image of a swiss army knife]

**3GL: Visual Program**

Traditional programming language + graphics

1 symbol => 1 keyword

![Image of a swiss army knife]

**MDA: UML+UML+UML**

Multiple semi-automatic transformations

![Image of a swiss army knife]

**MDA: UML+UML+UML**

Multiple semi-automatic transformations

Manufacturers’ claims:

+22% (Obeo) ... +35%(OpJ)

Not enough!

![Image of a swiss army knife]

**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

![Diagram of Booch, Rumbaugh and Selic's MDA process]

**Modeling functionality vs. modeling code**

- Map to code, implement
- Map to UML
- Generate code
- Domain Framework
**Domain-Specific Modeling demo**

**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

**Case: Polar Electro**
- DSM for heart rate monitors*
- Significantly improved developer productivity
  - "Measured productivity improvement is 1000% compared to earlier practice"
- Generated code = better quality
  - "Developers evaluated the generated code to be of better quality than previously manually written code"
- ROI achieved even during the first project
  - "When we compare the time to implement the DSM solution to the average time used to create applications it became early evident that the investment would pay back very quickly, even if there would be only one developer"

* Kärnä et al. Evaluating the use of DSM in practice, DSM'09, OOPSLA

**Sports computer applications**
- Example of application design

Kärnä, J., Using DSM for embedded UI application development, Thesis
Sport computer applications

- Heart rate: measuring, analysis and visualization
- Calorie calculation: current, cumulative, expenditure rate, active time
- Speed: current, average, maximum
- Distance: based on interval, trip, recovery
- Altimeter: vertical speed, altitude alarms, slope counter, graphical trend
- Cycling information: pedaling rate, cycling power
- Barometer: pressure drop alarm, graphical trend
- Exercise diaries
- Sensor connectivity: heart rate, speed, cadence, power, GPS
- Compass, Temperature, Odometer, Logbooks
- etc.

Survey results

- Perceived productivity improvement
- Error prevention
- Usability of tools
- Quality of code
- Ease of learning

Case: Home automation

- Advanced home automation systems
  - Lights, heating, cooling, etc
  - Energy saving, intrusion detection, earthquake alerts
- Use of these features require full-fledged user interfaces with menu navigation, data visualization and system configuration functions

Home automation application sample

Design flow

- Develop the same system twice
  - Real product
  - 2 teams
- "Productivity increases of factor 3 to 5"
  - Covering software design, implementation and configuration phases
  - Further gains expected for documentation and testing
**DSM Case Study: Nokia**

- DSM and related code generators for mobile phone*
  - In use since 1995, 100s of developers & phones
- Order of magnitude productivity gains (10x)
  - “A module that was expected to take 2 weeks... took 1 day from the start of the design to the finished product”
- Focus on designs rather than code
  - Domain-oriented method allows developers to concentrate on the required functionality
- Training time was reduced significantly
  - “Earlier it took 6 months for a new worker to become productive. Now it takes 2 weeks”

* MetaCase, Nokia case study, 1999

---

**Getting started**

A journey of 1000 miles begins with a single step

Setting off on the wrong foot can spoil the whole journey

---

**Productivity increase from DSM**

- Network
  - Increased
- Call processing services
  - Increased
- Touch-tone UI
  - Increased
- Host application
  - Increased
- Mobile phone software
  - Increased
- Phone radio/intercom
  - Increased
- Financial web applications
  - Increased

---

**How to implement DSM**

1. **Domain Idea**
   - Done a few times before!
   - DSM language
   - Code generator
   - Framework code
2. **Normal (many):**
   - Easy!
   - Model in DSM language
   - Generate code
   - Domain Framework

---

**Worst practices: Initial conditions**

- Only gurus build languages **4%**
- I’m smart & need no help **12%**
**Domain Dilettante**

Insufficient understanding:
- Problem domain 17%
- Solution domain 5%

**Analysis Paralysis**

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

**Sources for the Language**

**Tool: hammer ⇒ nails**

Tool’s technical limitations dictate language 14%

**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%
1. Domain expert’s concepts
- Concepts from domain
- Mostly made without help
- Simple code generation
- OK in established domain
- Usable by non-coders

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
- But works well with DSL or XML

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

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 or XML schema OK
- **3. Look and feel: common, easy, true DSM**
  - Found in very different domains
- **4. Variability space: adds power to handle complexity**

Best results combined 3 (L&F) and 4 (Variability)

---

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

---

Tools for DSM creation and use

- 6 ways to get the tools we need for DSM
  1. Write own modeling tool from scratch
  2. Write own modeling tool based on frameworks
  3. Metamodel, generate modeling tool skeleton, add code
  4. Metamodel, generate full modeling tool over a framework
  5. Metamodel, output configuration for generic modeling tool
  6. Integrated modeling and metamodeling environment
- Good tools minimize resource use (few man-weeks)
  - creating modeling tools and generators data-like, not code
  - guide in DSM creation
  - allow you to test DSM throughout domain design process
- Good tools allow DSM to change and reflect changes
  - to modeling tools
  - to design models already made

---

Currently many ‘starter’ tools, not suitable for industrial use

- Single user
- Single modelling language at a time
- Simple metamodels
  - Focus on objects, basic properties, binary relationships
- Simple notation
  - Single graphical element + label
- Simple template-based, text/XML-to-text generator
  - Or hand-written code to read models
- Resulting modelling tool primitive
  - Lacks majority of functions users expect in such a tool
- Upgrading modelling language invalidates models
- Upgrading tool framework invalidates tool
- Underlying architecture and datamodel flawed
The **CASE Repository: Welke 1988**
More than another database application
See http://www.dsmforum.org/papers/CASE_Repository.html

*There is increased awareness within the CASE arena of the need for a central repository of system description information. This is brought on by a growing recognition that only with a strong central repository can CASE toolsets:

The **Model Repository: Kelly 2008**
More than just **XML under version control**
See http://www.dsmforum.org/events/DSM08/keynote.html

*There is increased awareness within the modeling arena of the need for a central repository of system description information. This is brought on by a growing recognition that only with a strong central repository can modeling toolsets:

---

**Model Repository Requirements**

- ☑ Be integrated
- ☑ Cope with large projects
- ☑ Provide full life-cycle support
- ☑ Produce complete documentation
- ☑ Perform system-wide validation and verification
- ☑ Adequately control a project

---

**XML under Version Control**

- ☑ Be integrated
- ☑ Disconnected models
- ☑ Cope with large projects
- ☑ Fail to scale
- ☑ Provide full life-cycle support
- ☑ Filing cabinet is not a life cycle
- ☑ Produce complete documentation
- ☑ No support - luckily XML is human-readable
- ☑ Perform system-wide validation and verification
- ☑ No support - cope with problems later
- ☑ Adequately control a project
- ☑ Free-for-all, merge by hand

---

**XML under Version Control, v3.0**

- ☑ Be integrated - D.I.Y.
- ☑ Cope with large projects - D.I.Y.
- ☑ Provide full life-cycle support - D.I.Y.
- ☑ Produce complete documentation - D.I.Y.
- ☑ Perform system-wide validation and verification - D.I.Y.
- ☑ Adequately control a project - D.I.Y.

---

**XML under Version Control, v3.0**

- Seven easy steps to a solution:
  1. Wait for several projects to make D.I.Y. solutions
  2. Wait for community / evolution to identify the best
     - 6: one for each requirement
  3. Persuade authors to release as open source
     - Unencumbered license
  4. Persuade authors to integrate them
  5. Wait for results
  6. Persuade original tool owners to incorporate results
  7. Wait for integrated version

- Even if this worked, it is doomed from the start...

---

**Are you an Oglaroonian?**

- Natives to the small forest world of Oglaroon, Oglaroonians have taken what is a fairly universal trait among sentient species (to cope with the sheer infinite vastness of the universe by simply ignoring it) to its ultimate extreme.
- Despite the entire planet being habitable, Oglaroonians have managed to confine their global population to one small nut tree, in which they compose poetry, create art, and somehow fight wars.
- The consensus among those in power is that any trees one might observe from the outer branches are merely hallucinations brought on by eating too many oglanuts, and anyone who thinks differently is hurled out of the tree, presumably to his death.

-Douglas Adams-
The World is Not a Tree

- Car architecture diagram
  - CAN bus
  - Wheels
- Electronic circuit
- Function call tree
- Hierarchical database
  - Network database
  - Relational database

Trees make life easy for us
- Easy to parse e.g. XML
- Good for teaching, prototypes
When you have a hammer, everything looks like a nail

Aren’t All Meta-Languages Alike?

- Sapir-Worff hypothesis: (linguistics)
  language influences how we understand the world
- Alfred North Whitehead (mathematics)
  “by relieving the brain of all unnecessary work, a good notation sets it free to concentrate on more advanced problems”
- Ludwig Wittgenstein (philosophy)
  “The limits of my language indicate the limits of my world.”
- George Boole (computer science)
  “That language is an instrument of human reason, not merely a medium for the expression of thought, is a truth generally admitted”

Modeling Associations With Ecore

ed-metks.blogspot.com/2008/01/modeling-associations-with.ecore.html

And with OPRR:
RelAB RoleA A
RoleB B

What is needed in a mature tool? Generic modelling tool functions

- Basic modelling tool functionality
  - Multi-level undo/redo
  - Cut/copy/paste
    - For complex objects, specify depth to copy
  - Printing, exporting to various formats
  - Direct visual feedback when moving / scaling objects
  - User documentation and support
- Multiple simultaneous models, DSM languages, users
- Browsers by various links
  - Type, Graph contents, Graph hierarchy etc.
- Documentation generation

What is needed in a mature tool? Tool integration

- Standalone vs. IDE integration
  - As yet unsolved question
  - Integration adds high coupling, but sweetens the pill
  - Most IDE functions little use in DSM use
- Call external tools
  - E.g. autobuild
- Be called by external tools
  - E.g. via command line, as part of a nightly build
- API
- Model and metamodel interchange
  - E.g. as XML
    - Although no working standards exist

Important features in MDD tools
Multi-Everything: repository design

- Multi-parent (not tree)
- Multi-role (n-ary relationships)
- Multi-model (interlinked)
- Multi-occurrence (reuse objects)
- Multi-representation (same object in several places)
- Multi-form (diagram, matrix, table etc.)
- Multi-tool (open on same model)
- Multi-language (method integration)
- Multi-language version (evolution)
- Multi-platform (Windows, Linux, Mac etc.)
- Multi-user (simultaneous sharing, see others' updates)
- Multi-environment (integration, interchange)

Multi-Everything: DSM Solution Development Time

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

Days to create DSM language & generator

Language development
Generator development

Is DSM worth it?

- Good language workbenches enable industrial DSM
  - 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)

Literature and further links

- Blogs: www.metacase.com/blogs
- Kärnä, J. et al., Evaluating the use of DSM in practice, DSM Workshop, OOPSLA 2009 www.dsmforum.org/events/DSM09
Graphical Syntax

Symbols should use full range of visual variables

Notation literature

- Miller, George A., The Magical Number Seven, Plus or Minus Two Psychological Review, 63, 81-97. 1956, psychclassics.yorku.ca/Miller/