Mature Model Management

Friday 27 May 2011
13:00-14:15, Turing
Steven Kelly

MetaCase
Modularization
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
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
Standard Practice Today

XML (1969)

VCS (1972)
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

- 🙄ly Adequately control a project
  - 😞 Free-for-all, merge by hand
XML under Version Control, v2.0

- © 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 anything else one might observe from the outer branches is merely a hallucination 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
The World is Not Binary

- We like having just 2 of things
  - Romeo and Juliet
  - Black and white
  - Good and evil

- Some relations *are* simple
  - A dog has a tail
  - Transition from Start to State

- But others *are* complex
  - Family has father, mother, kids
  - Inheritance, n-ary, assoc. class

- Binary makes life easy for us(?)
  - Good for teaching, prototypes(?)
  - When you have a hammer, everything looks like a nail...
  - thumbs too if you feel no pain!
N-ary: 1 relationship, 2..N objects

- Lemma 1: a line is a role
- Lemma 2: a junction is a relationship
- Works for binary too: one formalism enough for both
What’s in a Relationship?

- Increase in relationship-like concepts
  - Relational, network, binary, Ecore
  - ER
  - OPRR
  - GOPPRR

- We added concepts...
  - ...but we never said what the line was!
Object Binding, Property Holding

- A relationship binds objects together
  - e.g. ‘marriage’ for man & wife
- A relationship has properties
  - e.g. ‘date of marriage’
- Binding concerns objects & relationship
- Properties only concern the relationship
  - Roles too may have properties

Separate binding from property holding
  - Lines are the binding, blobs have properties
Graphs and Bindings

- Binding $\Rightarrow$ Relationship (Role Object)*
  - But where to store it? Or who ‘owns’ it?
- Graph ‘knows’ its contents
  - so can know facts involving several of them
- Bindings contained within a graph
- Graph $\Rightarrow$ Object* Binding*

- Object does not ‘know’ its relationships
  - Has different bindings in different graphs $\Rightarrow$ reuse
DOES IT MATTER?
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”
No really: DOES IT MATTER?
And with OPRR:

```
RelAB
  RoleA  A
    data
  RoleB  B
```
Meta-metamodel support for modularization

- No forced tree structure / strong containment
  - No forced backlinks
- Explicit concept of Graph
  - Links to subgraphs
  - Objects can be reused in multiple graphs
- Distinction between Relationship & Property → Object
  - Explicit concepts of Role and Port
  - N-ary relationships
- Common constraints expressed as data
  - Additional language to express arbitrary constraints
Integration

Integration Paradigms: Strings vs. Objects
Integrating Languages
Integrating Tools
Integration Paradigms:

1. String matching in files

- Strings are 1-dimensional character arrays
- Look for same sequence, “E”, “m”, “p” etc.
  - Or UUID, unique identifier in XML
- Inefficient, hard to see, fragile
  - but familiar!

```
class Employee...
class Manager extends Employee...
Developer extends Employee...
```
Integration Paradigms:

2. Direct references in repository

- Works like objects in memory
- Efficient: Direct pointer
- Visible: See referrers
- Robust: Change once
  - But less familiar!

```
class Employee

class Manager extends Employee

class Developer extends Employee
```

```
class Employee

... class Manager extends Employee...

Developer extends Employee
```
Integration Paradigms:

Tool support for direct references

- Concrete syntax: view
- UI: edit
- Cross-model references: link
- Disk representation: load

<table>
<thead>
<tr>
<th></th>
<th>view</th>
<th>edit</th>
<th>link</th>
<th>load</th>
</tr>
</thead>
<tbody>
<tr>
<td>XText</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EMF/GMF</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSL Tools</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPS</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MetaEdit+</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Integration Paradigms:

Summary

- We need both!
  - But text and graphical tools often only offer strings
- Use direct references whenever possible
  - Make most important references visible
- Use string matching if you need indirection
  - Deliberate breaking up into exchangeable modules
Integration

Integration Paradigms
Integrating Languages
Integrating Tools
Integrating languages:

Best integration = no integration

- General purpose languages need broad coverage
  - Too many concepts for one diagram type
  - Have to split into diagram type per aspect
  - Each diagram type has broad coverage for that aspect

- Reintegration of aspects across models is hard
  - For tool and modeler

- DSM languages need only support needed subset
  - Can often fit several aspects in one diagram type
  - Save reintegration costs
Integrating languages:

Relationships between models

- Diagram element links to new subdiagram
  - Allows modularization, scaling
- Metamodeler decides what is legal
  - Can one element link to several subdiagrams?
  - Can a subdiagram be reused by several elements?
    - The world is not a tree!
  - Is subdiagram type same as parent diagram?
  - Can an element have a different link when reused?
- Direct link or string indirection? Interface?
- Don’t show subdiagram in parent diagram!
  - Defeats the whole point of modularization
Integrating languages:

Sharing language concepts

- Reusing a concept in metamodels A and B
  - Allows reusing instances in models of type A & B
- Define different aspects for same element
  - Relationships with other elements in models
- Define and use a reusable element
  - Only exists once, not copies
  - Better than patterns or wizards
- Use directly in models or as element property
Integrating languages:

Create a metamodel from a model

- Some kinds of reuse are really instantiation
  - Define function => call function
  - Define class => create instance
  - Define configurable component => use & configure

- First model says what fields need filling in
  - And what the types of the content are

- But that’s just like a metamodel!

- Three levels of people
  - Metamodeller + Component definers + Modelers
    - base metamodel
    - component metamodel
    - how components extend base metamodel
Integration

Integration Paradigms
Integrating Languages
Integrating Tools
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
  - For external tools, not just plugins in same process

- Model and metamodel interchange
  - E.g. as XML
    - Although no working standards exist
Important features in MDD tools

- Productivity increase: 35%
- Easy to use: 25%
- Multi-platform: 20%
- Modern looking: 10%
- Eclipse integration: 5%
- VS integration: 0%
Variability

(demo of support for variability in Watch modeling language in MetaEdit+)
Variability

Routine configuration

Creative construction

Wizards

Feature-based configuration

Domain-Specific Language

Path through a decision tree
All choices known
Implementation available

Subtree of a feature tree
All features known
Feature implementations available

Subgraph of an infinite graph
Variant space known, variants not
New features can be implemented

© 2011 MetaCase
Co-Operation

(demo of multiple users working on the same model in MetaEdit+, with no conflicts or locking other users out)
Ladies and Gentleman: Mr. Risto Pohjonen

- MetaCase’s *real* MOVIES man
- Two music DVDs charted #1
- Finnish Emmy nomination
- Gold disk
Risto’s MetaEdit+ language for DVD menu structure
Multi-user environment: locking

- Locking mechanism secures user’s design elements from other users
- Opening an element for editing locks the element
  - Other users may still look at the element (e.g. open it), but are not allowed to modify it
- Fine granularity of locking
  - Down to single property of single object
- Try to avoid locking others’ data needlessly
  - Holding down shift when opening property dialogs or graphs prevents locking the selected element
  - Info tools show lock status and who has lock
Multi-user environment: basics

- Multi-user environment = several users sharing the same repository working on the same design space
  - Emphasis on reuse
- From users’ point of view the functionality of the multi-user version is basically the same as in the single-user version
- Most differences between single-user and multi-user version are on the administrative side:
  - True client/server architecture: server handles the repository sharing for clients
  - Most system administrator commands left to one true system administrator
Multi-user environment: commit

- Teamwork based on design transactions
  - ACID: Atomic, Consistent, Isolated and Durable
- Commit to make your changes available for others, and to see the changes committed by others
  - Real time updates not necessary – or desirable

USER 1

REPOSITORY

USER 2

S = START SESSION   C = COMMIT   A = ABANDON

Time
Single / Multi-user Combinations

a) small products, or for modules linked to form a larger product

b) integrated modules with reused elements

c) single product or module becomes too large for a single developer

d) pair modeling, division of roles
Evolution

(demo of language evolution in MetaEdit+, with multiple users’ models updating on the fly)
Most tools don’t support evolution

- Updating modelling language invalidates models
- Deploying updated language is a pain
- Upgrading tool framework invalidates tool

- Leads to language stagnation
- Stagnation is death for domain-specific languages
Scalability
It’s easy to build a simple modeling tool

- 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

- 1-3 = Coding
  - Language expressed throughout code
- 4-6 = Language Workbench
  - Language expressed as data

- 4: Language data is transformed into code
- 5: Language data is copied
- 6: Language data is live
It’s easy to build a simple language workbench

- 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
What is needed in a full language workbench (not so easy)

- 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)
Thank you!

Free MetaEdit+ evaluation download: **www.metacase.com**
Build your first DSM language and generator in an hour!

<plug>If you like it after 31 days, see 150€ Intro offer</plug>
And if you don’t like it, email me! 😊

**MetaCase**
www.metacase.com

Questions?
Literature and further links

  *also in German: Ausgereiftes Modellmanagement: Mehr als nur XML unter Versionskontrolle*, OBJEKTspektrum (Oktober 2010)

- DSM Forum, [www.dsmforum.org](http://www.dsmforum.org)
- Blogs: [www.metacase.com/blogs](http://www.metacase.com/blogs)
- Kärnä, J. et al., **Evaluating the use of DSM in practice**, DSM Workshop, OOPSLA 2009 [www.dsmforum.org/events/DSM09](http://www.dsmforum.org/events/DSM09)