Co-evolution in Industry and Research - Is it Convergent?

Steven Kelly, CTO, MetaCase

Models & Evolution Workshop

Tue 24 September 2024, 09:30

Contents

- We will look at the pressures that drive evolution in
  - modeling language
  - modeling tool
  - language workbench
- How do the evolutionary pressures differ between academia and industry, and are those differences necessary or useful?

Co-evolution: What else needs to change when a language changes

- Other parts of abstract syntax
- Constraints
- Notation
- Transformations: Generators, Internal, Importers
- Tooling
- Documentation
- Users’ brains
- Models


1. Beijing

Instantiation and Variation

- Each step can add order(s) of magnitude
- Tool → customers → languages → models

- Customer: mobile phone manufacturer
- apps → app powerset → variants → product instances
  - generators & frameworks for generation variants

- Orders of magnitude: 500 x 10 x 300,000 = 1.5 billion

- When the language changes, the models must work, code generation must work, new firmwares must work

Co-evolution concerns last longer than expected

- The language outlasted the business
- The tool use outlasted the largest company in Europe
  - What is the reason, and is it possible to make a research career based on it?

- None of its many hundreds of modelers has ever had to manually edit a model file to solve a co-evolution issue
- Nor have the metamodelers ever had to fix tool code after metamodel evolution.

2. San Diego


Crushing Dangerous Steel Parts with Hydraulic Press Compilation VOL 1
69600kg
0:08 / 8:14 • Deep frozen steel hammer

Evolutionary Pressures

Pressing Forward →

- Problem domain match
  - Additions
  - Corrections
  - Deletions
- Users
- Models
- Solution domain match
- Problem domain changes

← Pressing Back

- Tooling
- LangDev resources
  - Stagnation
- Sacred at birth
- Users
- Models
- Products in the field

Evolution Enablers

- Tooling
- Smart users
- Fearless Change (Mary-Lynn Manns, Linda Rising)
- Deprecation
  - Needs to be unintrusive, so they can concentrate on the actual work on the model

To be or not to be;
To deprecate or destroy?

- Where a language change reduces the set of legal models, it is rarely a good idea to adopt a strict formalist approach
  - e.g. deleting parts of models that no longer conform
- Non-conforming parts still contain info and earlier choices the modeler needs to use during model co-evolution
- Leave them: were legal when made, still generate correctly
- **Deprecate:** Allow old style but show warnings, guidance
- Follow experience with programming languages & libraries
  - Make co-evolution fully automatic if certain
  - Otherwise deprecate, provide update help


3. Cape Town

DevEnv support for co-evolution

- When building a language workbench, look for this
- Programming Language support
  - Change classes on the fly, existing instances update
- Not Java, C#, …
  - Smalltalk was a good choice!
- Database support (or other persistent format)
  - Change schema on the fly, existing stored data updates
- Support in prog.lang makes it natural to want this
  - Influences database to have support
    - and even better, as the database makes it multi-user
  - DB has classes as first class objects: Gemstone, Objectivity

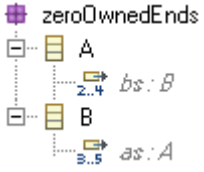
Standard Practice Today

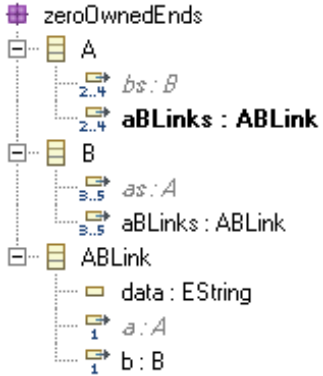
XML (1969)

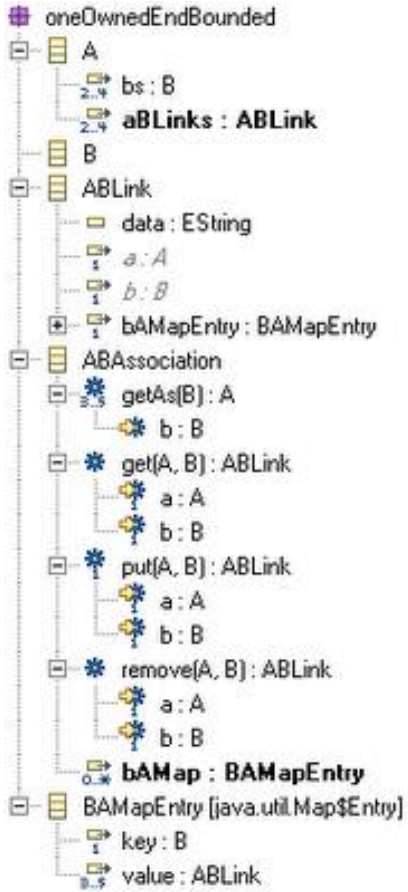
VCS (1972)

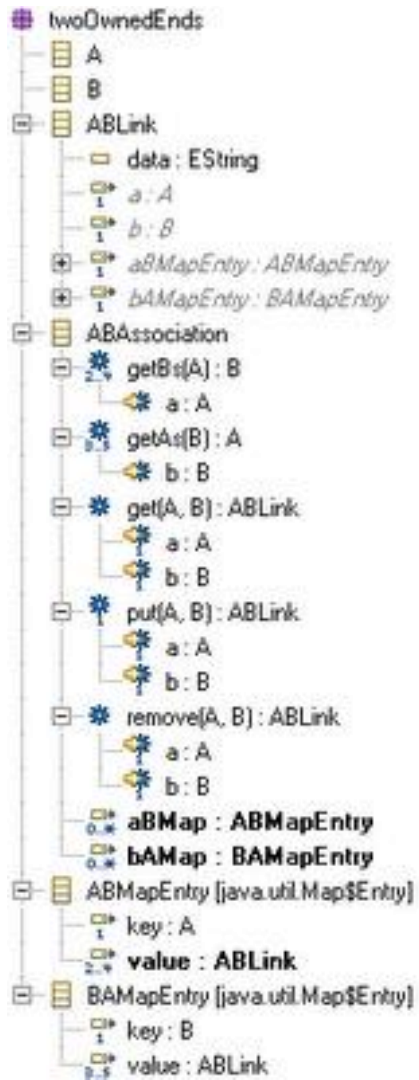
Design for evolution from the start

- Can't add in later as external tools, plugins
- Like n-ary relationships
  - If you start with binary you have to build a whole new level above what you intended as your platform
- Like multi-user collaboration
  - *Maybe* possible to take a tool and port it to a different language that supports collaboration+co-evolution natively









Modeling Associations With Ecore

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

And with OPRR:

RelAB — RoleA — A

data ＼ RoleB — B

Co-evolution scores in 3 tools

Location of Change ↓	*Nature of Change*			
	Add	Rename	Remove	Change
Metamodel	**5**	**2**	**4½**	**4½**
Constraints	**4½**	—	**4½**	**5**
Notation	**5**	**5**	**5**	**5**

← **MetaEdit+**

Location of Change ↓	*Nature of Change*			
	Add	Rename	Remove	Change
Metamodel	**2**	**1**	**1**	**1**
Constraints	**1**	—	**1**	**2**
Notation	**5**	—	**5**	**5**

← **Sirius**

Jjodel →

Location of Change ↓	*Nature of Change*			
	Add	Rename	Remove	Change
Metamodel	**5**	**2**	**3**	**5**
Constraints	**X**	**X**	**X**	**2**
Notation	**5**	—	**5**	**5**

"These differences do not seem to be explained by traditional distinctions between tools: industry / academic, open source / closed source, mature / research, commercial / free.

As both MetaEdit+ and Jjodel have had strong co-evolution support from the start, and EMF/Sirius has not improved significantly despite problems being noticed early on, it seems that the cause may be architectural, and thus hard to change via new versions.

One hypothesis is the more integrated nature of MetaEdit+ and Jjodel tools and development teams, compared with the more separate components and teams of EMF/Sirius.

Another hypothesis is the persistence formats, with MetaEdit+ and Jjodel staying closer to the in-memory structure of objects and keeping all artifacts integrated, whereas Sirius splits metamodels and models into several different types of loosely linked XML files." *A Framework for Evaluating…Co-Evolution…, SoSyM '24*

4. Arctic Circle

A Walk on the Wild Side

IS-SE border jumping:
MetaPHOR & MetaCase Consulting

Steven Kelly, 1998

KISS’98, Kilpisjärvi, Finland

Metamodelling and MetaCASE

- **Many Information System Development methods**
  - Structured, Object-Oriented, BPR
- **Not all methods can be supported by CASE tools**
  - Too many methods, evolving too fast
- **Separate CASE tool for each method → problems**
  - Isolates different parts of company, different projects, different phases
- **…Need a metaCASE tool to support *any* method**
  - Model methods = metamodelling,
    CASE tool follows metamodel
  - Support and integrate multiple methods at once
  - Easy addition of new methods, changes to existing ones

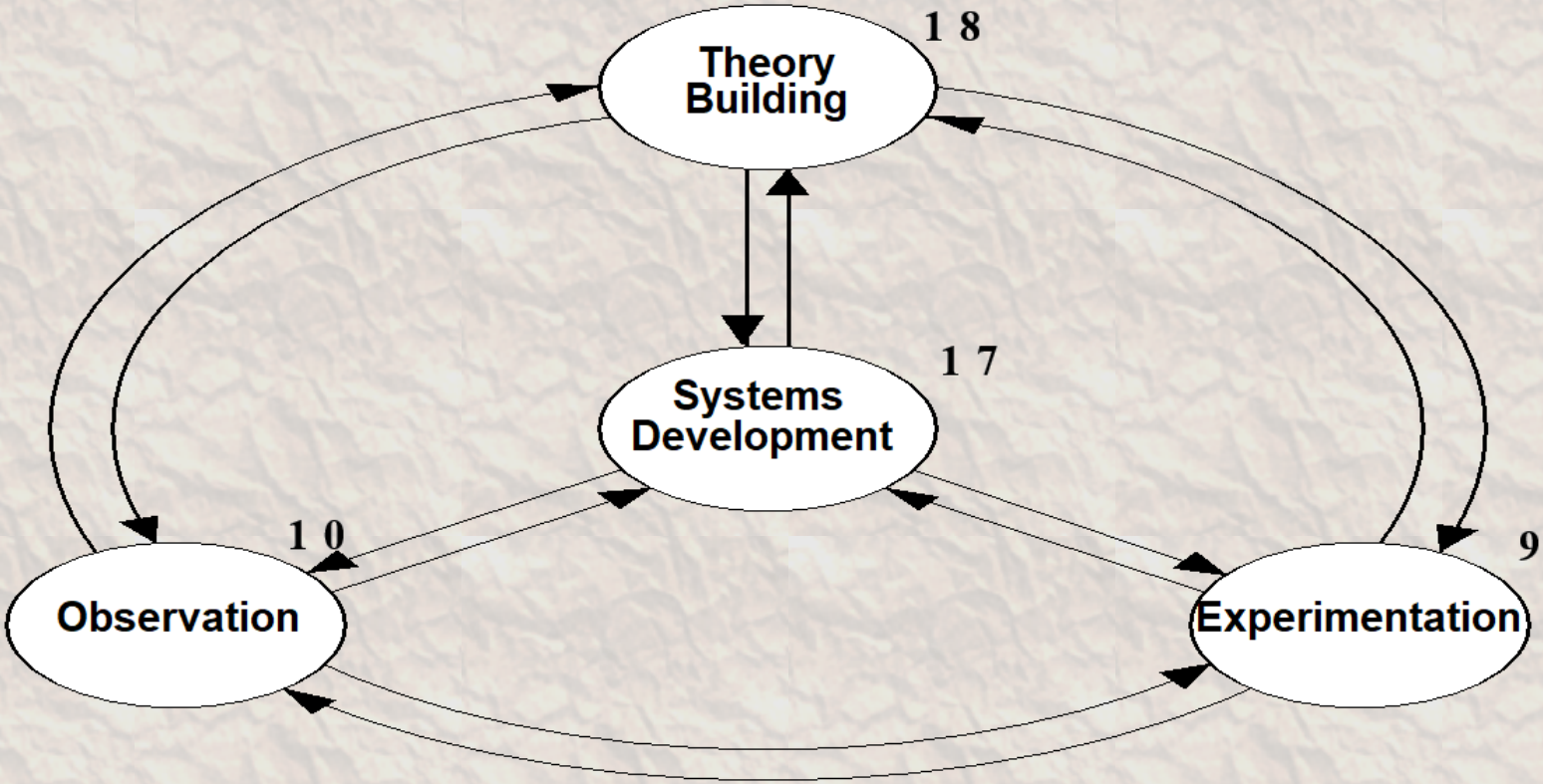
MetaEdit History

- **Windows only, initially only CASE tool**
  - Later Method WorkBench: assorted metaCASE add-ons
- **Core of 2 software engineers, 2 others later**
  - Large parts one man's work whilst on leave
  - Actor OO language, code file management later
- **Light on requirements, heavy on design**
  - Design largely hand-drawn
  - Documentation = pretty print code
- **Own code: 125 classes, 20.000 lines**
  - Total 380 classes, 70.000 lines, i.e. 70% reuse

MetaEdit+ History

- **Based on experience gained with MetaEdit**
  - Add multi-platform, multi-user, multi-method, multi-view
- **Completely new system: language, db, graphic library**
  - 3 major versions of each during project!
- **Core of 4 software engineers, about 10 others**
  - Envy code management system
  - Later modelled MetaEdit+ with MetaEdit+
  - Later some commercial development
- **Heavy on requirements, light on design**
- **Own code: 140 classes, 30.000 lines**
  - Total: 1450 classes, 396.000 lines, i.e. >90% reuse

Research Approach, Number of Papers



Nunamaker, Chen & Purdin, 1991,
Systems Development in Information Systems Research

Prerequisites

- **Big name professor**
  - Gets research funding, gives company respect
- **Good practices programming guru & SE project manager**
  - Guides, helps, does large amount of programming, integrates
- **Project supersecretary**
  - Handles finances, meetings, deals with bureaucracy, keeps professor in loop
- **Academic & commercial success with first project**
  - Published papers: needed for further research funding
  - Sold program (e.g. shareware): attracts commercial backing
- **Patient families…**

Lessons Learned

- **Systems Development is resource intensive**
  - Requires critical mass of skill
    (but not all need be skilled at everything)
  - Requires time and energy
- **Systems Development must follow good SE practices**
  - Design for expandability
  - Design must be elegant
  - Redesign, refactor and recode on the fly
- **Play to your team's strengths**
  - Good researchers, good designers and good programmers are often different people
- **Build shared understanding**
  - Your 'backup copy' if one member leaves

Instantiation, Variation, Forethought

- Academia doesn't have instantiation or variation
  - Have to do it in your head, before building (after: BOTTA)
- Industry often doesn't have the luxury of forethought
  - Take experienced people from line, form group to design next gen
- SE is a valid IS research method (Nunamaker) – but only if:
  - made easy to understand for outsiders
  - answers industry needs (even ones industry doesn't know yet)
- Students! Listen to industry greybeards:
  - won't be right about everything, but know lots that you don't
  - defending your new bits is good practice, suggests improvements
  - talking and learning is a great skill, particularly for smart people

MetaCase

Thank you!

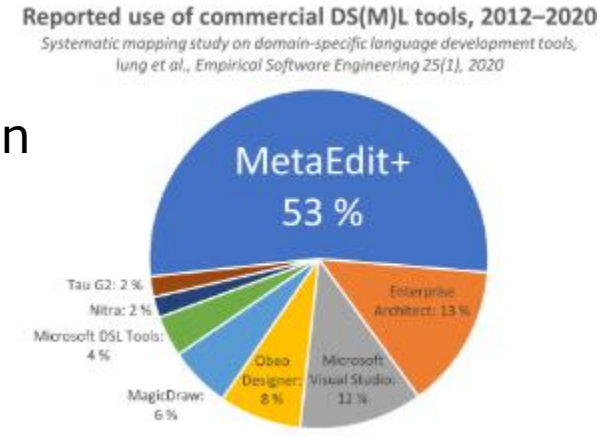
Questions? Experiences? Arguments?

Industry Day talk: **Today 12:07**

SoSyM  metacase.com/stevek_pubs.html

Academic Edition

- MetaEdit+ Workbench
  - a fully functional modeling and metamodeling toolset
- Permanent license
- Buy individually or in bundles of 10, 20, or 50 licenses
  - E.g. 40€/license in 50 bundle for class
- Many predefined metamodels
  - UML, BPMN, ER, SA/SD etc.
  - Domain-specific with full code generation
- metacase.com | Store | Academic



MetaEdit+