Managing the Co-Evolution of Domain-Specific Languages and Models

Juha-Pekka Tolvanen

MetaCase
Jyväskylä, Finland
jpt@metacase.com

Steven Kelly

MetaCase
Jyväskylä, Finland
stevek@metacase.com

Abstract— **Refinement, enhancement and other maintenance tasks normally account for more work than the initial development phase. This applies to domain-specific languages and models too. This tutorial describes practices for managing the evolution of domain-specific modeling languages, while co-evolving the models that have already been created. The presented practices are field-tested in industry cases – some managed and refined over three decades. Participants will learn practices and patterns to form part of their toolbox for evolving their languages while in use alongside models. During the tutorial the practices learned are made concrete by applying them to sample cases.**

Keywords—domain-specific modeling, domain-specific language, evolution, maintenance, metamodel, model evolution

I. Tutorial Description

A. Short bios

Dr. Juha-Pekka Tolvanen is the CEO of MetaCase and co-founder of the DSM Forum. He has been involved in model-driven development and tools, notably metamodeling and code generators, since 1991. He has acted as a consultant world-wide for modeling language development, authored a book on Domain-Specific Modeling, and written over 70 articles for various software development magazines and conferences. Juha-Pekka holds a Ph.D. in computer science and he is an adjunct professor (docent on software development methods) at the University of Jyväskylä.

Dr. Steven Kelly is the CTO of MetaCase and co-founder of the DSM Forum. He has over twenty years of experience of consulting and building tools for Domain-Specific Modeling. Steven acts as an architect and lead developer of MetaEdit+, MetaCase's domain-specific modeling tool, he has seen it win or be a finalist in awards from SD Times, Byte, the Innosuomi prize for innovation awarded by the Finnish President. He is author of a book and over 50 articles. Steven has an M.A. (Hons.) in Mathematics and Computer Science from the University of Cambridge, and a Ph.D. from the University of Jyväskylä.

B. Proposed length

Half-day

C. Level of the tutorial

Advanced

D. Target audience

This tutorial is intended for all involved in creating Domain-Specific Modeling (DSM) languages and related generators. Language engineers will learn via scenarios from practice how to conduct and manage the evolution of the modeling languages and related generators.

Participants are expected to have experience on some language definition mechanism, mostly on using some metamodeling language.

II. Tutorial description in detail

Refinement, enhancement and other maintenance tasks normally account for more work than the initial development phase. This applies to domain-specific languages and models too. Compared to general purpose languages, domain-specific languages evolve more frequently — along with the changes in the domain and in the development needs [1,2]. This tutorial describes proven practices for managing evolution of modeling languages along with the models already created. In particular the latter is important as we normally don't want to lose the work done with earlier version of the modeling language or start updating models each time the language evolves.

The tutorial approaches the co-evolution of languages and models from two angles. The first angle is the nature of the change: adding, renaming/updating or deleting. The second angle is the part of the modeling language been changed: abstract syntax, concrete syntax or semantics. The practices presented have been learnt and proven in industry cases [2, 3] – the longest having been originally defined in the mid 90's and still in use today. We illustrate the cases of evolution using examples that are also available during the tutorial to try out. The tutorial concludes by describing change management and versioning practices for evolving modeling languages and their models. The following subsections describe the main aspects of the tutorial focusing on the two angles of evolution.

A. Adding new elements to the language

Adding new concepts to a modeling language (to its definition in a metamodel) is usually easy as it does not affect existing models. During the tutorial we'll update an existing language with new concepts such as objects, relationships and properties. For this and all other maintenance tasks the participants can propose their own changes to be considered during the tutorial making it more interactive.

If the addition is a constraint then its influence on the existing models needs to be checked, as there may be models that do not satisfy the new constraint. Unlike with general-purpose languages, you can often access all models made with the domain-specific language, as it has only been used in your organization. Modern tools can help here by providing automation and supporting features, e.g. accessing all models based on the language for instant update, or notifying language users when the type of change is contextual (i.e. models cannot be updated automatically but require input from the developer to consider the right type of change depending on the model data). For this purpose the language engineer may create a model checker that is run separately when moving to the new metamodel, or automatic annotations to highlight the elements requiring update, either as part of the notation or shown as a warnings list in editors. During the tutorial we'll create model checking reports for language users as well as conduct some changes directly based on the added constraints.

Adding new notational symbols and generators is usually safe: They update how existing models are shown or produce output as defined by the generator. We'll demonstrate these update tasks with some existing languages and maintenance cases.

B. *Renaming language elements*

Renaming or updating language elements in the metamodel often has an effect on concrete syntax and perhaps semantics. Moreover, it always has an effect on existing models. During the tutorial we'll rename a language concept in the metamodel and inspect its consequences to other parts of the DSM solution, such as constraints, concrete syntax and generators. We also inspect how model updates can or should be performed: fully automated in some tools and requiring using find/replace functions in others.

C. *Deleting language concepts*

Before deleting anything, the language engineer should first consider if it is better just to hide the language elements or make them no longer instantiable, rather than deleting the concepts and all their instances permanently. This is something that is normally difficult with textual programming languages but which modern tools for language development can provide. This approach allows existing model data to still be valid for example when generating code – after all, the generator support for them already exists and still works.

This approach of obsoleting rather than hard deletion allows language users to see and update design data from the past, while guiding them not to use the old language concept anymore. One bonus here is that if we later find that removal of language constructs was not a good idea we may bring the removed parts back into the language – and with good tool support this will also fully restore their instances. During the tutorial we will remove various kinds of language constructs from the metamodel and consider the influence of the change on existing models.

If language engineers are sure about the permanent deletion we should inspect how the deletion influences existing models. For example, it could be that the removed object type may have connections or constraints which then also need to be removed. Inspection of the influence of these changes is usually a better option than using brute-force automation for deletion. If the deletion involves decisions dependent on the model context, the language engineer can implement model check functionality similarly to that suggested earlier when adding new constraints.

Removing a language concept should normally also remove its notation shown in the model. If the removal is related to notation only it will be automatically reflected in symbols used in models.

D. *Change management and versioning*

The tutorial concludes by describing practices for versioning languages, generators and models. In practice the types of changes are evolutionary and not revolutionary: If the language were to change completely, it would be more the case that language engineers would create a new DSM solution.

If the step in the language maintenance is small it can be done directly in the production system – checking its influence before committing the changes and releasing the new version for all language users. However, if there are major changes then it is usually better to make them separately along with the test models – or even with a copy of the same models as in the production system.

E. *Intended outline*

The structure of the tutorial is based on the two dimensions outlined above: 1) the nature of change and 2) the part of the modeling solution changed. In the beginning we motivate the participant on the importance of maintenance phase compared to the initial language development phase. We also describe the general organization of the change management and show examples with existing DSM solutions.

Outline:

1. Welcome and motivation
2. Adding new elements
    a. Abstract syntax
    b. Concrete syntax
    c. Semantics
    d. Exercises
3. Renaming elements
    a. Abstract syntax
    b. Concrete syntax
    c. Semantics
    d. Exercises
4. Removing elements
    a. Abstract syntax
    b. Concrete syntax
    c. Semantics
    d. Exercises
5. Change management and versioning

III. Other tutorial details

A. Novelty of the tutorial

We are not aware of any tutorial addressing maintenance of domain-specific languages along with co-evolution of models made. Research work has been done on metamodel evolution but then independently from other parts of the language definition, like its constraints, notation or generators, or then independently of the maintenance of existing models.

This tutorial has not been given earlier, but the tutorial speakers have given tutorials on domain-specific modeling languages in conferences such as MODELS, ECOOP, OOPSLA, SPLC and Code Generation.

B. Required infrastructure

During the tutorial exercises for various language evolution scenario are examined. For their presentation and group discussion a flip chart or white board would be helpful. We will also demonstrate the evolution scenarios with example modeling languages, generators and models. If participants want to conduct the exercises hands-on with tools they should bring their computer and preferred tooling with them. For the rest the authors can provide tooling.

C. Sample slides

Sample slides are provided in a separate file.

References

[1] Sprinkle, J., Mernik, M., Tolvanen, J-P., Spinellis, D., “What kinds of nails need a domain-specific hammer?”, IEEE Software, July/Aug, 2009

[2] Kelly, S., Pohjonen, R., Worst Practices for Domain-Specific Modeling, IEEE Software, Volume: 26 Issue: 4, 2009

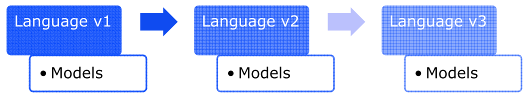
[3] Tolvanen, J.-P., Kelly, S., Model-Driven Development Challenges and Solutions - Experiences with Domain-Specific Modelling in Industry. In Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development, 2016

Agenda

- Introduction
- Co-evolution of languages and models
- Evolution scenario
  - Adding new elements and exercises
  - Renaming/updating elements and exercises
  - Removing elements and exercises
- Change management and versioning
- Summary

Co-evolution

- Domain-Specific Languages evolve as the domain evolves or development needs change
- DSL evolution is more frequent than GPL evolution
- Nobody wants to lose existing work done



Evolution scenario

- Two angles:
  - Nature of change
  - Part of language changed

Nature of change/ Part of language	Create	Rename/ Update	Delete
Metamodel			
Constraints			
Notation			
Generators			

Language for the exercises: IoT

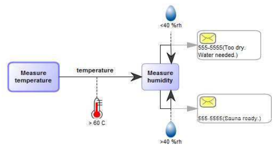
- An company makes IoT device and its applications
- The language need to be maintained as:
  - the domain (device) itself changes
  - we learn more about the domain
  - there are new development needs

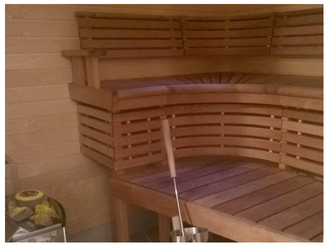
Nature of change/ Part of language	Create	Rename/ Update	Delete
Metamodel			
Constraints			
Notation			
Generators			

Domain in this tutorial: IoT device

- Sensors
  - altitude, movement (in 3 directions), humidity, location, luminance, pressure, speed, temperature, time, geofence, battery status…)
- Actions
  - sending SMS, send to cloud, push notification, saving logs

Sample 1: Sauna App





Task 1: Add new language elements

- IoT device supports now also measurement of humidity
  - It has new sensor
- Language evolution:
  - What parts of the language should be updated?
  - What kind of changes are needed?
- Model evolution:
  - Does it influence to the models?
  - If it does, how the models should be migrated?

Task 2: Add new constraints

- The device does not function properly if the temperature is over 60 degrees Celsius
  - We should not allow creating applications that expect warmer temperature
- Language evolution:
  - What parts of the language should be updated?
  - What kind of changes are needed?
- Model evolution:
  - Does it influence to the models?
  - If it does, how the models should be migrated?

Task 3: Add new notational element

- Notation should indicate if device expects too warm operational environment
- (code is not generated if too warm)
- Language evolution:
  - What parts of the language should be updated?
  - What kind of changes are needed?
- Model evolution:
  - Does it influence to the models?
  - If it does, how the models should be migrated?