Creating and using integrated domain-specific languages for different contexts

14 September, 2016
Juha-Pekka Tolvanen
jpt@metacase.com
Contents

- Introduction
- Case example
- Experiences
- Q&A
(DSL) modeling support consist of:

- Metamodels (abstract syntax)
- Notations (concrete syntax)
- Constraints and checks (semantics)
- Generators (semantics)
- Tooling
Integrated DSL example 1: ETSI’s SDL

- Languages integrated from the beginning!
  - Enables checking, consistency, collaboration, reuse etc.
Integrated DSL example 2: NWA+UML

- Some languages taken as given
  - Less control on semantics, checks, etc.
Integrated DSL example 3: this case!

- Some tools (+their languages) taken as given
  - Less control on functionality

- Link with existing requirements
- With code, etc in IDE
- With SysML
- With Simulink
- With UPPAAL
- Safety analysis in HipHOPS

Code, etc.
DSL case:
Automotive Embedded Systems

- Modeling solution targeting embedded system development in automotive
  - Covers different concerns: logical, physical, behavior, architecture, safety, dependability, timing, verification, testing, variation etc.
  - 24 (mostly) integrated languages, 600+ language elements
  - Generators for code, check, configuration, simulation, analysis, documentation, metrics: 400+ (sub)generators
  - Implemented in MetaEdit+ tool with a need to integrate with other languages and tools
  - Tool chain integration via API, command line & file exchange
Some of the DSLs fitted to V-model
Created DSL functionality covers

1. Editors for graphical, matrix, table and tree with text:
   - With checks, help system and basic generators like documentation, metrics etc.

2. Generators for and integration with:
   - Modeling and analysis tools (Simulink, HIPHOPS)
   - Programming environments (Visual Studio, Eclipse)
   - Requirements (Excel, ReqIF)
   - Documentation (Word, RTF, HTML)
   - Versioning systems (GiT, SVN)
   - File exchange format
   - + user defined integration: SPIN, Stateflow, Labview, UPPAAL, Modelica, AUTOSAR, T-VEC...
On language creation

- 3 language engineers
  - Shared metamodel definition with one remote engineer
  - 2 organizations involved in language creation
  - Consulting service on DSL creation

- 5+ generator developers from different organizations
  - Focused on different target tools/generation formats

- Language implementation started based on partial language definition (partial metamodel)
  - Rules, concrete syntax, model organization, reuse and checks were all added during the language creation
  - Initial metamodel was updated during implementation
DSL Development Effort

- Effort on creating modeling functionality: 73 days in total (covers metamodel, notation & constraints with tooling)

- 3 days per language (=73 days / 24 languages)

- The DSL development effort in the case is along the other publicly reported cases (Panasonic, Polar, Elektrobit...*)

* See references in the last slide for the public cases
DSL implementation time in different cases

- Call processing: 63 language concepts, XML generator
- Touch screen UI applications: 60 language concepts, C, HTML, build script generators
- Voice control application for microcontroller: 36 language concepts, Assembler generator
- Mobile phone applications: 77 language concepts, Python generator
- Automotive infotainment systems: Java generator for simulation
- Insurance product specification and management: 143 language concepts, J2EE generator
Language development process

- Language development was done in a period of 2,5 calendar years
  - First version in use Month 2 covering the most stable part
  - Gradually extended and modified based on changed requests

- Language updates released every 2-6 months
  - Updates versioned and provided in a single package covering whole modeling solution (no partial deliveries, like just constraints or generators)
Evolution: Updating languages

- Started with core part, supporting the immediate needs
  - Get users and results quickly
  - Extend other DSLs as needed

- Obtained feedback from language users (at the level of models, not metamodels)
  - Feedback support built-in to the tooling

- Treated all integrated languages as a whole
  - When one language changed reflections to other parts considered at the same time
  - DSL solution was delivered as one package to ensure that all languages are integrated too
Evolution 2: Updating models

- Model migration even more important than language updates: nobody likes loosing work when DSL changes!
  - Used a set of reference models to test the language changes before publishing the next version of DSLs

- Applied automated model updates as far as possible:
  - Non-destructive policy: This way models always open, editors always work for new DSL. Nobody loose his work!
  - Guidelines for updating when changes in semantics
  - Extended generators so that they accept also models made with earlier version if needed
Lessons learned

- Language engineering team does not need to be big
  - Enable collaboration among language engineers
- Deliver first the core parts to get results/show case early
  - Others will join and get interested
  - Get feedback from users (in-built assistance)
- Iterate quickly so DSL users get the functionality when they need it (not one month later)
- If building your first DSL, get support from someone having done it
Thank you!

Questions please?

For details, contact:
Juha-Pekka Tolvanen, jpt@metacase.com
References


References to publicly reported cases: