Technological proposal for a feature modeling language

Mathieu Acher and Andrzej Wasowski

@acherm @AndrzejWasowski

Original abstract

Usage scenarios, semantics, expressiveness, and sharing facilities of feature models are all very important topics, but at the end of the day researchers and practitioners need a specific technology to concretely work with feature models. In this talk, we will briefly review some possible technologies (e.g., Java EMF and XMI). We will describe the kinds of tooling (e.g., type checker) and some requirements (e.g., readability) one can expect. Then we will specifically discuss the possibility to use YAML format and JSON schema. The hope is to open up to more diverse languages/ecosystems (e.g., JavaScript and Python) with a well-established technology. We will also report what we expect from other working groups (e.g., abstract syntax of the feature modeling language should be provided).

Abstract

“Usage scenarios, semantics, expressiveness, and sharing facilities of feature models are all very important topics, but at the end of the day researchers and practitioners need a specific technology to concretely work with feature models.”

#1 we start the workshop with a concrete solution ;)

#2 we are a good target

#3 technical realization is mandatory… (huge responsibility)

#4 there is a risk that MODEVAR initiative leads to nothing if we don’t have something concrete in the upcoming months. agree?

Agenda

- In this talk, we will briefly review some possible technologies (e.g., Java EMF and XMI).
- We will describe the kinds of tooling (e.g., type checker) and some requirements (e.g., readability) one can expect.
- Then we will specifically discuss the possibility to use YAML format and JSON schema.
- The hope is to open up to more diverse languages/ ecosystems (e.g., JavaScript and Python) with a well-established technology. We will also report what we expect from other working groups (e.g., abstract syntax of the feature modeling language should be provided).

Technological space

- We will briefly review some possible technologies (e.g., Java EMF and XMI).
- Parsers
  - Lex and yacc
  - ANTLR
- Language workbenches
  - MPS, Xtext, etc.
  - Obeo designer (graphical notation)
- Reuse of existing solutions
  - OCL?
  - Clafer?
  - FeatureIDE?
  - FAMILIAR?
  - FAMA?
  - …

My experience #1 (Mathieu)

FAMILIAR: A Domain-Specific Language for Large Scale Management of Feature Models

Mathieu Acher[a], Philippe Collet[b], Philippe Lahire[b], Robert B. France[c]

- FAMILIAR
  - A language for specifying feature models (OK)
  - A language for combining feature model operators (automated reasoning, slice, merge, diff, synthesis, etc.)

```
fmArch150 = FM ( Arch : Ar1 Ar2; Ar1: (Ar3|Ar4); Ar2 : (Ar5|Ar6); )
fmPlugin = FM ( Plugin : (Pl1|Pl2|Pl3)+ ; Pl1 -> Pl2 ; )
fmFull = aggregate { fmArch150 fmPlugin } withMapping
                    constraints (Ar3 -> Pl1 ; Pl2 -> Ar5; )
fmArch = slice fmFull including fmArch150.* // enforced FM
```

My experience #1 (Mathieu)

- FAMILIAR (we started with Xtext 0.8.2… Philippe can join the lamentation)
  - Eclipse lock-in
  - EMF lock-in
  - Java lock-in
  - Web attempt
  - Scala attempt

```
fmArch150 = FM ( Arch : Ar1 Ar2; Ar1: (Ar3|Ar4); Ar2 : (Ar5|Ar6); )
fmPlugin = FM ( Plugin : (Pl1|Pl2|Pl3)+ ; Pl1 -> Pl2 ; )
fmFull = aggregate { fmArch150 fmPlugin } withMapping
                    constraints (Ar3 -> Pl1 ; Pl2 -> Ar5; )
fmArch = slice fmFull including fmArch150.* // enforced FM
```

My experience #1 (Mathieu)

- Retrospectively
  - Lock-in frustrations (with PhD student, MSc student, industrial partners; personal)
    - Last frustration: I wanted to integrate a feature model into Python programs…
  - Opportunities
    - It would be so great to have a language you can use in different ecosystems

Metamorphic Domain-Specific Languages

A Journey Into the Shapes of a Language

Mathieu Acher Benoit Combemale
University of Rennes 1, Inria, IRISA, France
mathieu.acher@irisa.fr, benoit.combemale@inria.fr

Philippe Collet
Univ. Nice - Sophia Antipolis, I3S, France
philippe.collet@unice.fr

Abstract

External or internal domain-specific languages (DSLs) or (fluent) APIs? Whoever you are – a developer or a user of a DSL – you usually have to choose side; you should not! What about metamorphic DSLs that change their shape according to your needs? Our 4-years journey of providing the "right" support (in the domain of feature modeling), led us to develop an external DSL, different shapes of an internal API, and maintain all these languages. A key insight is that there is no one-size-fits-all solution or no clear superiority of a solution compared to another. On the contrary, we found that it does make sense to continue the maintenance of an external and internal DSL. Based on our experience and on an analysis of the DSL engineering field, the vision that we foresee for the future of software languages is their ability to be self-adaptable to the most appropriate shape (including the corresponding integrated development environment) according to a particular usage or task. We call metamorphic DSL such a language, able to change from one shape to another shape.

Categories and Subject Descriptors D. Software [*D.2 SOFTWARE ENGINEERING*]: D.2.6 Programming Environments; D. Software [*D.3 PROGRAMMING LANGUAGES*]: D.3.0 General

Keywords programming; domain-specific languages; metamorphic

1. Introduction

Domain-specific languages (DSLs) are more and more used to leverage business or technical domain expertise within complex software-intensive systems. DSLs are usually simple and little languages, focused on a particular problem or aspect of a (software) system. Outstanding examples of DSLs are plentiful: Makefiles for building software, Matlab for numeric computations, Graphviz language for drawing graphs, or SQL (Structured Query Language) for databases.

DSLs are found to be valuable because a well-designed DSL can be much easier to use than a traditional library. The case of SQL is a typical example. Before SQL was conceived, querying and updating relational databases with the available programming languages led to a huge semantic gap between data and control processing. With SQL, users can write a query in terms of an implicit algebra without knowing the internal layout of a database. Users can also benefit from performance optimization: a query optimizer can determine the most efficient way to execute a given query.

Another benefit of DSLs is their capacity at improving communication with domain experts [1–3], thus tackling one of the hardest problems in software development. But DSLs are also ordinary languages, in the sense that many difficult design decisions must be taken during their construction and maintenance [1]. They usually can take different shapes: plain old to more fluent Application Programming Interface (APIs) ; internal or embedded DSLs written inside an existing host language ; external DSLs with their own syntax and domain-specific tooling. To keep it simple, a useful and common distinction is to consider that a DSL can come in two main shapes [2]: external or internal. When an API is primarily designed to be readable and to "*flow*", we also consider it as a DSL.

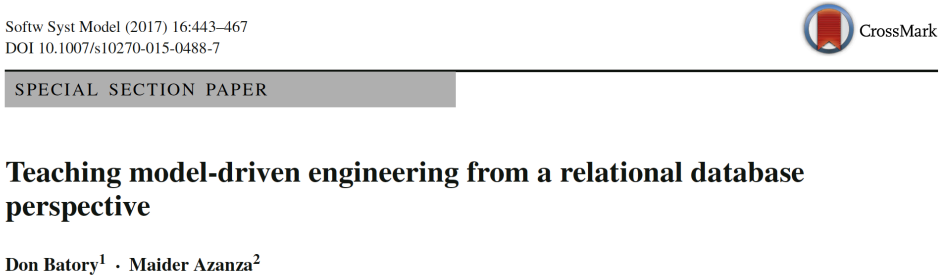
As for SQL – invented in 1974 and one of the first DSLs – it is interesting to note that it comes itself in different shapes. Figure 1 shows three of these shapes on the same basic query example. The top part of the figure shows the plain SQL variant, with a classical "*select, from, where*" clause. In the middle part of the figure, we show the same query written in Java with

Onward! 2014, October 20–24, 2014, Portland, OR, USA.

http://dx.doi.org/10.1145/2661136.2661159

My experience #2 (Mathieu)

- Teaching MDE/SLE/SPL since 2012
  - https://teaching.variability.io
  - Xtext
- Someone said: “The tools fostered a medieval mentality in students to use incantations to solve problems. Point here, click that, something happens. From a student’s perspective, this is gibberish.” and I fully agree



Softw Syst Model (2017) 16:443–467
DOI 10.1007/s10270-015-0488-7

CrossMark

SPECIAL SECTION PAPER

Teaching model-driven engineering from a relational database perspective

Don Batory[1] · Maider Azanza[2]

Requirements

- Language = Abstract syntax + Concrete Syntax + Services (tools)
- Is it our role? At least, when selecting a technology, we did think about services we want to build on top of the language
  - Basic editing services: autocompletion, syntax highlighting, etc.
  - Type checker
- Non-functional property: readability, learnability, interoperability, ability to integrate/use the language in different ecosystems

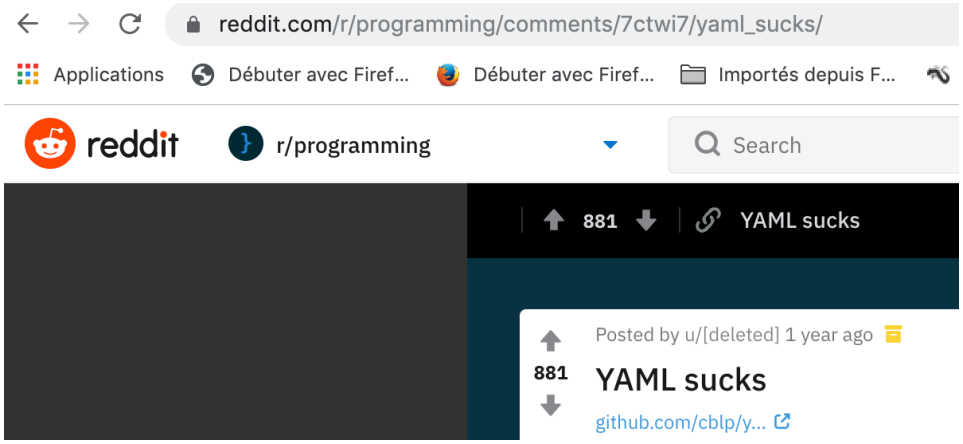
Proposal

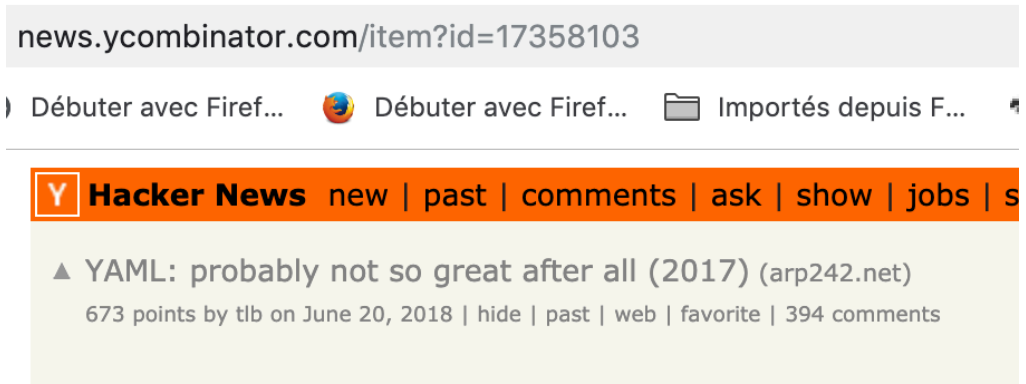
- YAML format (and JSON schema)

https://github.com/wasowski/fm-schema/

Proposal

- YAML format (and JSON schema)
- YAML???





Proposal

- YAML format (and JSON schema)
- YAML
  - **human readable** formats for **storing** structured data, with very good tool support for many programming languages (aka integration-friendly)
  - very lightweight and very easy to integrate into any tooling (industrial or research), which we thought was the most important objective for the software platform.

Proposal

- YAML format (and JSON schema)
- YAML+JSON=Wtf???
  - “a stricter version of YAML would make a lot of people's lives easier though.”
  - YAML does not seem to have a well-established schema definition language, but JSON Schema (https://json-schema.org/) can be used

A feature model in YAML

```yaml
- feature: &car
    parent: null
- feature: &engine
    parent: *car
- feature: &chasis
    parent: *car
```

(note: AST-based representation; CS is up to the community!)

Examples

```
- feature: &car
    parent: null
- feature: &engine
    parent: *car
- feature: &chasis
    parent: *car
- dependency:
     required: [ *car, *chasis ]
     excludes: [ *car, *chasis ]
```

Examples

```
feature: root
  - name: car
  - children
    feature:
      - name: engine
    feature
      - name chasis
```

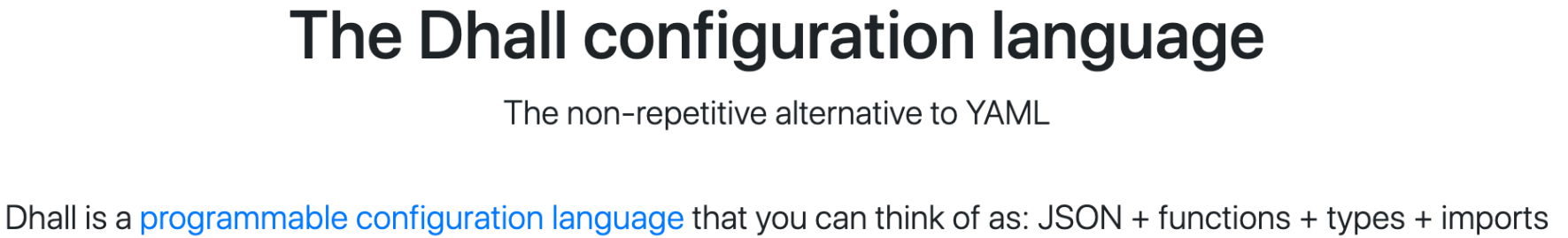
Examples

```
feature: car
  children
    - feature: engine
    - feature: chasis
```

```
feature: car
  children
    - group: hardware
      members:
        - feature: engine
        - feature: chasis
      min: 2
      max: 2
```

Related work (Dhall)

- https://dhall-lang.org
  - Almost 2000 stars on Github
  - Human and Integration friendly (thanks to YAML conversion)
- Our scope is different



Take-away

- The hope is to open up to more diverse languages/ecosystems (e.g., JavaScript and Python) with a well-established technology.
  - Playing the mainstream game? (Impact!)
  - Web!
  - Going beyond Java and Eclipse
- Schema to enforce consistency of instances (feature models)
- Concrete syntax?

Closed future

- What we expect from other working groups
  - abstract syntax and concrete syntax of the feature modeling language should be provided
  - list of services one can build on top of the languages
  - technical roadmap
  - critics/debates/approvals ;)