The FLFS project

FLFS involves three INRIA research teams: ASCOLA, AtlanMod, and Phoenix. Its purpose is to place domain expertise at the centre of software development process. FLFS addresses the current limitations of software engineering with regard to large-scale software production, robustness, reliability, maintenance, and evolution. Our main contribution is to parameterize software development process with a specific domain of expertise. This approach covers all the stages of software development and combines the following three emerging paradigms:

- **Domain-Specific Modelling**, also known as Model Driven Engineering.
- **Domain-Specific Languages (DSL)**, instead of General-Purpose Languages.
- **Generative Programming (GP)**, in particular Aspect-Oriented Programming as a mean to transform models and programs.

This poster presents one of the FLFS contributions: AML, a DSL for expressing matching algorithms, and its application to software evolution.

A Software Evolution Problem

Let us illustrate an instance of the software evolution problem. The Netbeans Java API evolve from version 1.12 to version 1.15. Programmers have to adapt their programs to the new API. How can programmers get out of the tedious refactoring task?

A Solution Integrating Models, GP, and DSL

We represent software artifacts (e.g., programs) using models. The models conform to a metamodel (e.g., the Netbeans Java API).

AML: A Domain Specific Language to Manage Software Evolution

AML is a DSL that facilitates:

- **Simple development of matching algorithms**. There is not an unique algorithm that matches all models perfectly. In practice, given a pair of models, programmers have to change a matching algorithm in order to get accurate results. AML allows to adapt matching algorithms in a straightforward way.

- **Reutilization of matching techniques**, Matching algorithms often combine two kinds of matching techniques:
  - Generic techniques base their judgments on standard kinds of features, e.g., labels.
  - Narrowed techniques refer to a particular domain and improve the results of generic techniques.

AML enables to develop both Generic and Narrowed techniques. Generic techniques can match models related to any application domain.

Fundamentals behind AML

In runtime, an AML algorithm is a chain of model transformations (Figure 3). Each transformation:

- **Instruments a matching technique**.
- **Takes as input a set of models**: the models to be matched (i.e., Left Model and Right Model), an Equal model that refers to Left and Right model, and a set of additional models.
- **Yields an Equal model**: Each Equal model basically contains a set of mappings.

AML Tools

- **The software evolution use case**: [http://www.eclipse.org/m2m/atl/usecases/ModelAdaptation](http://www.eclipse.org/m2m/atl/usecases/ModelAdaptation)

AML Results on Software Evolution (Garces, ECMDA, 2009)

We have validated our solution on two concrete examples:

- A Petri Net metamodel from the research literature: 6 versions.
- The Netbeans Java metamodel: 8 versions.

Accuracy

Figure 4 shows that AML achieves a high accuracy. The percentage of correct mappings (or equivalences) and changes ranges from 99%-100%, and 90%-100%.

Performance

Petri Nets: ~1s

Netbeans Java: ~10s

In the Java scenario, we have used the adaptation transformation to migrate the source code of GNU Classpath. The transformation generates a model with 69791 lines, and takes 38 s.