Stepwise Adoption of Continuous Delivery in Model-Driven Engineering

Jokin Garcia[1] and Jordi Cabot[2]

[1] IK4-IKERLAN, Arrasate, Spain
jgarcia@ikerlan.es
[2] ICREA-UOC, Barcelona, Spain
jordi.cabot@icrea.cat

1 Extended Abstract

Continuous Delivery (CD) and, in general, *Continuous Software Engineering* (CSE) is becoming the norm. Still, current practices and available integration platforms are too code-oriented. They are not well adapted to work with other, non text-based, software artifacts typically produced during early phases of the software engineering life-cycle. This is especially problematic for teams adopting a *Model-Driven Engineering* (MDE) approach to software development where several (meta)models (and model transformations) are built and executed as part of the development process. Typically, (part of) the code is automatically generated from such models. Therefore, in a complete CD process, changes in a model should trigger changes on the generated code when appropriate.

A step further would be to apply CD practices to the development of modeling artefacts themselves. Analogously to "traditional" CD, where the goal is to have the mainline codebase always in a deployable state, the aim would be to have the modeling infrastructure always ready to be used. Those models could be the final product themselves or an intermediate artifact in a complete CSE process as described above.

Either way, a tighter integration between CD and MDE would benefit software practitioners by providing them with complete CSE, covering also analysis and design stages of the process. In this extended abstract we give some firsts details on both scenarios.

1.1 Enabling modeling support in CD processes

In a standard CD pipeline models are regarded as plain text. The CD platform does not "understand" them and therefore cannot parse and discover nor trigger their possible dependencies. Dependencies between jobs have to be manually added and co-evolution after a model change is limited to alerting developers that some element (at the end of a model dependency) needs to be manually reviewed.

The first step is then to make CD model-aware. In order to be part of a CD pipeline, MDE tools must be wrapped as tasks capable of being executed

standalone (i.e. without human intervention) when called by the CD server. Thus, a barebone integration of MDE in CD requires to find at least one MDE tool for each major MDE activity (model-to-model transformation, model-to-text generation, model comparison,...) that offers some kind of external interface (via API or shell access). Good news is that we have been able to find at least one example of each tool to assemble such MDE-aware CD platform but plenty of work needs to be done towards enhancing MDE tools with APIs that match well with the expectations and standard vocabulary of CD processes.

1.2 Evolution of MDE artefacts as part of a CD process

Modeling artefacts change often in any development project. Including the metamodels/DSLs themselves. And this necessarily impacts a number of other coupled modeling artefacts (e.g. changing a metamodel would impact all transformations using that metamodel as transformation source or target). This (co)evolution problem has been well studied in the research literature and it's addressed by performing four key phases: 1) change detection between two versions of the artifact; 2) impact analysis that classifies changes into: Non Breaking Changes (NBC), Breaking and Resolvable Changes (BRC) and Breaking and Unresolvable Changes (BUC); 3) synchronization of artifacts (models, transformations) to metamodel changes and 4) testing the results.

Nevertheless, current approaches mostly focus on single co-evolution scenarios (metamodels vs models, metamodels vs transformations, etc). No holistic and global approach considering evolution impacts across a rich modeling ecosystem has been proposed so far. We believe a CD infrastructure could help on this challenge. A CD-based co-evolution would benefit from the following advantages: reactivity (the co-evolution process is triggered automatically when a change is committed, instead of requiring a manual launch by the modeler which is error-prone); parallelization (different co-evolution solutions may need to be applied in cascade, some of them could run in parallel), time-saving (co-evolution and testing tasks are executed only if an impact analysis determines that it is actually needed); visibility (of the process and its state) and flexibility (the automation enables non-expert modelers to control the CSE). We have started to adapt current MDE coevolution algorithms to work on top of the Jenkins platform.

1.3 Automating the process

With the two previous steps, we would get a rich CD process where modeling artefacts are first-class citizens of a CSE process same as code-based ones. The last step would be to be able to automatically define and configure the CD pipeline itself based on the analysis of the dependencies between all artifacts in order to minimize the work of preparing a CD infrastructure. For some, this information is more or less explicit (a transformation configuration launch) and could be used as starting point but for others a static analysis of the artifacts is required. This is still an open research problem.