

MDE for large projects : today needs and tomorrow standards

by Etienne Juliot & Stéphane Lacrampe
etienne.juliot@obeo.fr / stephane.lacrampe@obeo.fr

1 - Technical space projection with Acceleo

MDE (Model Driven Engineering) brings lots of new efficient paradigms like models transformation, reverse engineering. However, MDE is sometimes too complicated for some companies. Acceleo has been designed by Obeo to bring a new solution for code generation with an easy-to-use point of view.

It has been designed to improve software development productivity, durability and agility, but also to make MDA easier to learn and use. Therefore, our experience feedback could be useful for EMP to find new solutions and also to detect existing problems with current Eclipse's frameworks.

Acceleo is an innovative MDA code generator and is built upon all OMG concepts (like working at the metamodel level). Acceleo is really customizable like a "with box" and can be used for PIM-to-code or PSM-to-code generation.

It makes it possible for architects and developers to embed complex and powerful architecture patterns in generators and apply them automatically to models.

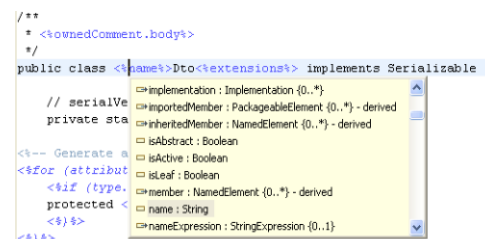
Acceleo is an Eclipse plug-in and leverage on the Eclipse platform. It is also strongly based on the EMF framework. Acceleo can generate any technology as output, like main J2EE frameworks (Hibernate, Struts, JSF, EJB, Spring...) but also .Net, Php ..

Acceleo software is divided in three main parts :

- *Importing* : Acceleo is not a modeller, but it is able to import model from any modeller, as it can read the XMI standard format. As for now, EMF deals only with UML2 and UML2.1. So, Acceleo embark the MDR framework (NetBeans project) and is delivered with UML1.4 and UML1.3 meta-model and a bridge to EMF. Last but not least, Acceleo can work with any meta-model, so it is ready for new UML versions or DSL/GMF technology for example.

- *Edition* : Acceleo contains a full and professional editing environment for module making embed in Eclipse :

Illustration 1 : It contains the Template Editor, which is a complete editing environment with eclipse regular features : syntax colouring, in-line completion for meta-model elements, scripts, services... And also error detection for compilation and runtime errors !



```

/**
 * <ownedComment.body*>
 */
public class <name>Dto<extensions*> implements Serializable
{
    // serializable
    private sta
    <!-- Generate a
    <$for (attribut
    <$if (type,
    protected <
    <$) $>
    <$) $>

```

Illustration 1: Template editor

Illustration 2 : It also contains the Reflective Editor for instant and real-time preview of generation results. Preview shows detailed information on what every model element contribute to the generation output, making very quick and easy template development and tuning.

- **Execution** : the execution part is the heart of Acceleo, it can apply Acceleo modules on models to generate code. Special care has been put on performance to be able to scale with real industrial projects. Also, everything is ready to use it for real projects thanks to the launcher engine, and iterative work is managed thanks to the merge engine.

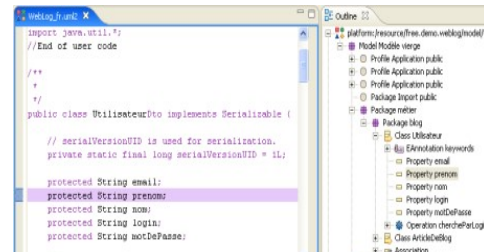


Illustration 2: Reflective editor

The merge engine is a very important feature. Acceleo team is made of experienced MDA users that believes that 100% code generation is not always the most efficient choice. Hand-coding is often faster and more maintainable than modelling. Acceleo deals with incremental generation, by defining specific coding zones with user tags. These zones are conserved when generating again.

Acceleo offers an appropriate solution to avoid the extremist policy "Everything has to be within the model". We prefer "Use tools for what they are best for".

Acceleo contains additional features such as :

- **Execution chains** (Illustration 3): these files, based on an extensible meta-model, are used for generating multiple source files based on multiple templates. It acts as a "glue" for all generation inputs necessary for a project, specifying which model to use, where and what to generate, etc...
- **Tree-like syntax** : Acceleo syntax is very easy to use, and has been specially designed to keep generation template as readable and easy to maintain as possible. It leverage on the fact that a model has a tree structure, so navigation through the meta-model becomes very natural, making it possible for anybody to write generation template.
- **Services** : Keeping Acceleo syntax very simple is one of our main concerns, but what happens if you want to do more complicated things than a if or a for ? Well, there is a very powerful and well designed language to do complex things which is Java. With Acceleo, you can apply on a model element any built-in Java services (ex : toUpperCase(...)) or design your own Java services for your needs. So you may extend Acceleo as you like.

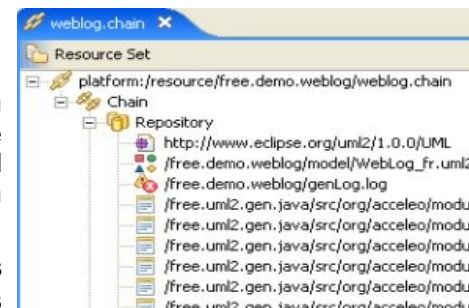


Illustration 3: Execution chain

Acceleo is natively built on top of many Eclipse projects and overlap some existing projects (like Jet, Xpand). Every code generator have its own features, advantages and disadvantages. A good strategy would be to create some bridges, some generic API and some standards to allow interoperability between tools. We hope M2T will help this to become a reality.

2 - Models management needs

Eclipse platform already provides lots of very professional tools and frameworks to manage models. The features list begins to be quite important with meta-models creation, models persistence in XMI files, models visualization, ... But some limitations slow down its usage for very large scale projects.

2.1 - Performance

Creating a « Helloworld » example using EMF and other Eclipse based model tools is very efficient.

Creating a model with only few simultaneous designers isn't really a problem and works fine.

But for big projects with many hundreds elements, some problems begin to appear. These problems can be fixed with some technical workarounds built on top of Eclipse's frameworks or with some methodology process.

Obeo go really further by giving models the central place of our retro-engineering platform Obeo Agility. Our models repository works fine with very large models. However, dealing with several thousand elements becomes really hard with EMF.

Our kind of performance needs are for example to improve memory load. We work with > 1 Go models and classical methods can't be applied. So, some advanced features like lazy loading, load balancing or DB storage need to be made by Eclipse products. To be widely use, they must also be completely integrated inside existing Eclipse framework like EMF and GMF.

2.2 - Scalability and extensibility

Since Eclipse 1.0, lots of great features contributed to the platform success. Some of these features are really useful for a daily use of source code development.

By using models, several "standard" features have disappeared and created some difficulties in getting a good productivity for a daily use of models.

These kind of missing features are :

- Refactoring
Example : a model A references a model B. If B's filename changes, link between A and B is broken.
- Extension
If I want to have a generic metamodel which can be extended, it could be useful to reuse the Eclipse's extension mechanism. So, metamodel would be easy to extend.
- Wizard
Models and metamodels can be created and filled with EMF's introspective editor. It could be useful if the creation of SWT wizards in order to manage these models could be made with an integrated system (a kind of GMF framework for wizards creation)
- File splitting
The ability to work on a model store on multiple ecore files isn't easy with EMF. If some very deep models used lots of contain relations, it would be very useful to have easy to do features to create new file which would store a part of the master model.

3 - M2T and M2M interoperability

When you want to build a full MDE software factory, you have already a large choice of tools. Each of these tools has its own advantages and its users community. And now, which question may we ask ? Do we need some universal tools inside Eclipse (like Corba try to be ...) ? Or do we need interoperability and common utilities ?

Eclipse platform already provides Ecore standard to exchange models between tools. This first step guaranties durability of the file format. With the use of Ecore models, a MDE chain can be composed of steps executed by products which hasn't been design to work together.

But it isn't enough to provide a transparent use of MDE tools. In a perfect world, if an application wants to use two different generators G1 and G2, a transformer T1 and a transformer T2, this application just needs to call two generic API to launch the generations and transformations, without any direct dependences.

We hope M2T and M2M will provide this interoperability layer and will act as a glue.

Today, there already are many model-to-text engines (Jet, Xpand, Acceleo) and model-to-model engines (QVT, ATL, Obeo Transfo). Each tool has a specific syntax and some specifics features. And it's better to have the right tool at the right place than trying to do the perfect and utopian tool for every need.

M2T and M2M could provide some features to insure interoperability between engine. For example, a common API to execute generation / transformation or a common mechanism to import templates.

With a common layer, we can imagine to improve extension of Eclipse Modeling Project. GMF or other EMF based tools could remove direct dependences to Jet and only rely on M2T. The extensions and plugins architecture would do the real link.

Moreover, users could override some generative behavior by choosing tool of his choice. After all, isn't it one of the goal of the Eclipse platform : be open and extensible ?