Implementation of a DSL in Papyrus

Dominique Heer

University of Antwerp

Abstract

This paper evaluates Papyrus, a UML2 tool for domain-specific language modeling. By using an example from the image processing domain, Papyrus' abilities to serve as a graphical modeling environment are examined. In detail, UML's extension mechanisms are exploited to implement a domain-specific modeling language for creating simple image processing pipelines. In addition, this project is compared to another project realizing the same DSL with another tool, JetBrains MPS, a metaprogramming system.

Keywords:
Papyrus, UML, UML Profiles, MPS, Image Processing, DSL, DSML, MDE

1. Introduction

This section gives a brief overview of model-driven engineering and the problems and challenges that lead to the development of the open-source graphical modeling environment Papyrus.

1.1. Overview

It should be no surprise that more and more companies integrate model-driven engineering (MDE) in their software development life cycle, for it has a number of advantages when compared to other methodologies which focus on the computing aspects and not on domain models. For example, MDE captures domain knowledge on a high level of abstraction, and the resulting models can serve as a documentation. In addition, it is less error-prone and validation becomes easier because it is executed on the model and not on code-level.

Email address: dominique.heer@student.uantwerpen.be ()

As a logical consequence, many modeling languages exist that visualize the design of a system, with UML (Unified Modeling Language) being the most prevalent one. Standardized by the Open Management Group (OMG), UML2 features fourteen different diagram types. However, it has been found that for some applications UML is too general, and further specification and adaption is needed. To support this, UML profiles (see 2.4) have been added. Profiles allow users to extend the meta-model of UML, thus creating new elements suitable for specific domains.

Furthermore, many tools exist that aim to support the MDE development process. However, in the last 25 years, it turned out that not a single company was able to build a MDE tool which captures all requirements needed by their customers. In addition, researchers work with their own tools, while companies use proprietary platforms. This leads to the fact that technology and knowledge exchange becomes unnecessarily difficult. At last, most commercial tools do not offer sufficient support for DSMLs (Bordeleau (2014)).

To tackle the problems described above, CEA, Ericsson and other companies have worked together to build an open source modeling tool called Papyrus which is based on the Eclipse platform. Although the main focus of Papyrus is UML modeling, it can be customized in various ways (see section 2). In this paper, Papyrus' abilities to serve as a graphical modeling environment for a specific domain is explored and evaluated. In detail, a DSL (domain-specific language) for image processing pipelines is implemented (see 3 and 4). In section 5, the experiences made are then compared to another project which focused on the implementation of the same DSL in another tool, MPS[1]

1.2. Related Work

The idea to implement a domain-specific language for image processing pipelines is not new: in Hartmann et al. (2015), IPOL, a DSL for image processing applications, is described. IPOL is a textual language based on XML. It allows the user to specify the complete pipeline, that is, images sources (sensors), processing blocks, and a target display (sinks). By abstracting the development of a pipeline, the design becomes language- and hardware-independent and thus can be easier analyzed and maintained.

[1] `https://www.jetbrains.com/mps`

Furthermore, Membarth et al. (2016) implemented HIPA^{cc}, a DSL embedded in C++ for describing image processing algorithms. They found that developers oftentimes have difficulties choosing the right programming language, especially when it comes to parallel computation, and therefore a DSL for modeling image processing algorithms would be a compelling solution. In addition, they implemented a source-to-source compiler which translates from the DSL to the target architecture.

Profiles in UML have also been broadly discussed. Fuentes-Fernández and Vallecillo-Moreno (2004) give a short and informative introduction to UML Profiles with special focus on the extension mechanisms and how to use them. Accordingly, Atkinson and Kühne (2002) describe some flaws of the UML extension mechanism and how to overcome them.

2. Papyrus

In the following sections, the Papyrus tool is introduced.

2.1. Overview

Within the Eclipse project, a variety of tools exist that address the development of domain-specific languages, such as Sirius[2], Xtext[3] and, in a broader sense, Papyrus. Whereas Sirius allows to easily create graphical modeling workbenches, and Xtext supports the development of text-based DSLs, Papyrus on the other hand is primarily used for UML2 modeling, since it implements 100% of the OMG specification. However, by making use of UML profiles (see 2.4), it can be extended and diagrams can be customized. In addition, Papyrus supports code generation from UML diagrams 2.5.

2.2. History

As described in section 1.1, most commercial modeling tools did not meet all the requirements of the users. Therefore, CEA List, a French research institute, started to develop Papyrus, but soon other companies showed interest in the collaborative development of a modeling tool. In 2008, a proposal was submitted to the Eclipse foundation with the goal to implement Papyrus on

[2] `http://www.eclipse.org/sirius`
[3] `http://www.eclipse.org/Xtext`

top of the Eclipse framework. According to the proposal[4], this was a natural choice because Eclipse is open source, offers powerful frameworks (EMF, GMF, see 2.3) and has a vibrant user community.

In October 2016, Papyrus version 2.0.1 was released.

2.3. Architecture

Papyrus is a sub-project of Eclipse's Model Development Tools (MDT) based on the Graphical Modeling Framework (GMF) and the UML2 metamodel. The latter is in turn based on EMF (Eclipse Modeling Framework), a framework which allows to model a data model and generate code from it. EMF models can be constrained by using OCL, the Object Constraint Language. GMF on the other hand provides components for developing graphical modeling editors based on EMF.

Figure 1 gives a graphical overview of Papyrus' architecture.

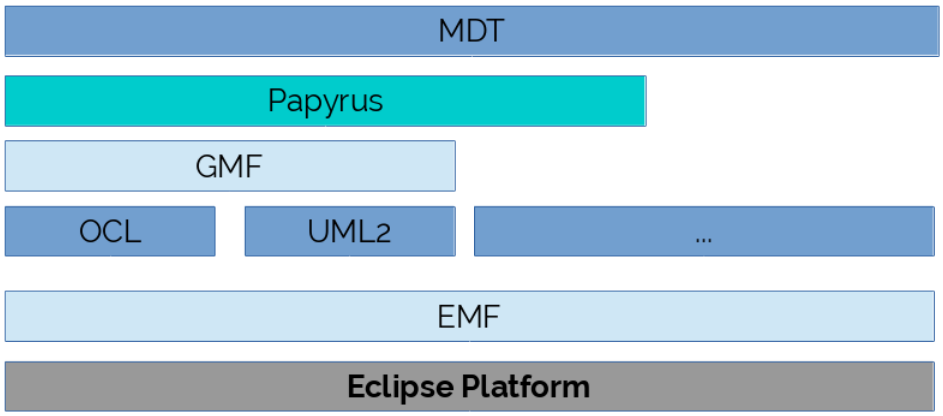

Figure 1: The Papyrus architecture

2.4. UML Profiling

To uphold its reputation as a flexible graphical modeling environment, Papyrus has to allow users the creation of custom DSLs for specific domains. In some domains, it is not necessarily useful to use the UML standard, but

[4] `http://wiki.eclipse.org/MDT/Papyrus-Proposal`

rather a specialized language. For these cases, UML features extension mechanisms to customize and extend the standard diagrams. These mechanisms are defined as follows:

1. stereotypes: allow the user to extend UMLs vocabulary by altering the meaning, syntax and visual representation of UML model elements
2. constraints: impose restrictions on the metamodel
3. tagged values: further enrich stereotypes by adding attributes

These three mechanisms describe a so-called UML profile. Papyrus has extensive support for UML profiles. For example, the SysML modeling language[5], which is based on UML and focuses on systems engineering, was implemented within Papyrus by using UML profiles.

Fuentes-Fernández and Vallecillo-Moreno (2004) provides a more detailed overview and example-based introduction to UML profiles.

In the scope of this project, profiles will be used to add new diagram types to support the image processing pipeline DSL (see 3 and 4.1).

2.5. Code Generation

Despite its ability to edit UML and SysML diagrams, Papyrus also allows the user to automatically generate code. As of writing this, Java and C++ are integrated, with support for Ada and C planned. In addition, custom code generators can be added. These generators are developed in the Xtend[6] programming language, a Java dialect which aims to improve Java by adding both new functionality (macros, powerful switch statements, operator overloading, etc) and by removing boiler plate. Xtend compiles into Java code and can therefore be used together with any Java library. Furthermore, it seamlessly integrates into the Eclipse IDE.

In addition to Xtext, plugins exist that make it easy to add custom code generators. Within the Eclipse project, Acceleo[7] is a famous choice, because it built on top of EMF and OCL, and implements the OMG standard for model-to-text generation (MOFM2T[8]). Acceleo supports code generation for every metamodel which is based on EMF, and comes with a powerful editor.

[5] `http://www.eclipse.org/papyrus/components/sysml`
[6] `http://www.eclipse.org/xtend`
[7] `http://www.eclipse.org/acceleo`
[8] `http://www.omg.org/spec/MOFM2T/1.0`

The ultimate goal of this project is to implement a model-to-code generator for the image processing pipeline DSL. Whether Xtend or a plugin like Acceleo are more practicable is discussed in 4.2.

3. Case study

3.1. The Domain-Specific Language

For the case study, a simplified image processing pipeline was chosen. Such a pipeline consists of exactly one input image, several processing blocks, exactly one output image and multiple connectors which connect the elements. Processing blocks can be for example image scaling, noise reduction, filters, effects or colorspace conversions. Hereby, it is possible that a processing block has several parameters. Every processing block has exactly one incoming and one outgoing connection. A source image has zero incoming and one outgoing connection, while an output image has only one incoming connection and no outgoing connections. Technically, such a pipeline is a linked list, with nodes representing the components (source image, processing blocks, target image), and pointers representing the data flow.

Figure 2 shows an example for a graphical representation of an image processing pipeline. This pipeline features an input image, two processing blocks and an output image.

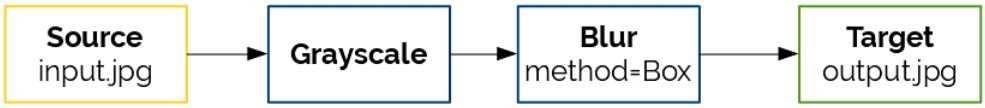

Figure 2: A simple example for an image processing pipeline

Hereby, the following restrictions are imposed on the image processing pipeline DSL:

- there is exactly one input image
- there is exactly one output image
- the input and output images are of type JEPG or PNG
- at least one processing block must be present
- at least three processing blocks have to be implemented

Purpose of this case study is to serve as an example for implementing a DSL in Papyrus. In addition, the same DSL is implemented in the scope of another project by using Jetbrain's MPS tool. Both tools, MPS and Papyrus, will be compared in section 5 based on the experiences made.

3.2. Generated Code

The model-to-text transformation has to generate valid Java source code which can be compiled and executed. The usage of external libraries for image reading, writing and manipulation is allowed. The resulting code has to preserve the order of the processing blocks and must take their parameters defined in the DSL into account.

The generated code for the example shown in figure 2 could hence look like this:

```
1  public class Pipeline {
2
3    private static void saveImage(
4      Image image, String filename) {
5      // TODO: implement
6    }
7
8    public static void main(String[] args) {
9      Image image = new Image("input.jpg");
10     Filter filter;
11     filter = new GrayscaleFilter();
12     image = filter.apply(image);
13     filter = new BlurFilter(BlurMethod.BOX);
14     image = filter.apply(image);
15     saveImage(image, "output.jpg");
16   }
17 }
```

4. Implementation in Papyrus

This section explains the implementation of the graphical DSML in Papyrus. The implementation process can be divided into two separate tasks: define the UML profile, and develop a code generator which automatically generates valid Java code from a given UML model.

4.1. UML profile

Papyrus makes it easy to create profiles. When adding a new Papyrus model, the user can choose between a UML model and a UML profile 3.

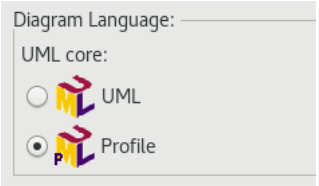

Figure 3: Papyrus allows the user to either create UML models or UML profiles

The profile editor then features several components that can be added, with profiles, packages, stereotypes, meta-class imports, constraints, extensions and generalizations being the most important ones.

For the image processing DSML, the UML class diagram was chosen to be extended. It features Class objects, which translate to input images, processing blocks and output images, as well as Realization edges, which translate to the connections between the images and processing blocks in the custom modeling language.

To extend the Class meta-class, it has to be imported. Then, a stereotype is added and connected to the meta-class by using an extension relationship (see figure 4).

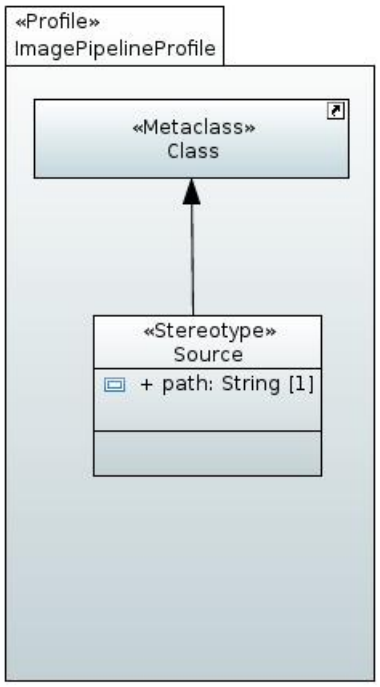

Figure 4: A new diagram type called *Source* is added; it extends the UML Class meta-class

Every stereotype can be enriched with tagged values (called properties in Papyrus). Due to some limitations, these properties can only be Booleans, Integers, Real numbers or Strings. Custom enumerations can be defined, though, but they are not saved in the corresponding UML model file (which is needed later for code generation).

In addition, constraints can be defined in the Object Constraint Language (OCL). These constraints can either be written in a separate file and added to a stereotype, or they can be directly added in the editor. Whether or not a model violates a constraint can then be checked by validating it.

The complete UML profile for the image processing DSML can be seen in figure 5. It is also possible to define custom icons and shapes for each stereotype, thus altering the visual representation of a UML element.

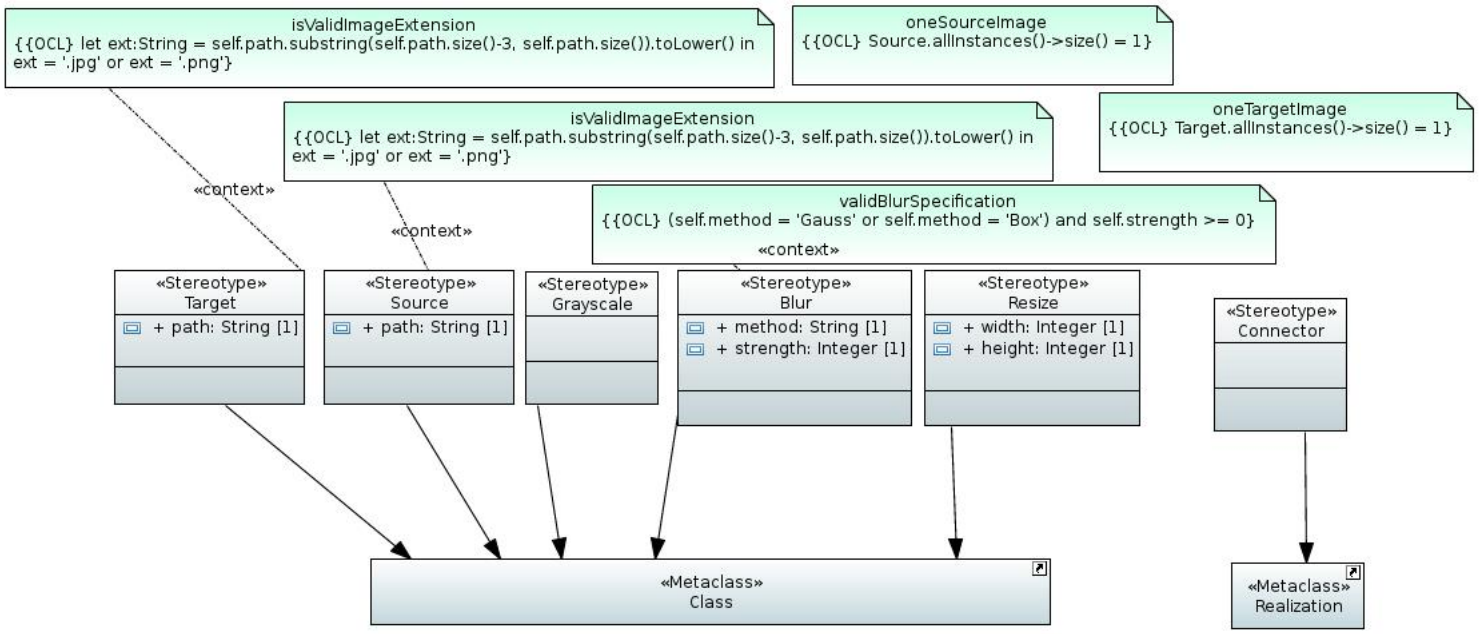

Figure 5: The complete UML profile for the custom DSML, featuring stereotypes, attributes and constraints

Section 4.3 describes how to add more processing blocks.

4.2. Code Generator

As explained in section 2.5, there are two ways of generating code: by either using Papyrus' internal code generation facility, or by using a plugin. The former would require the development of a Papyrus plugin-in that adds a new menu entry as well as a code generator written in the Xtend programming language. This approach is appropriate if one wants to add a code generator for a new programming language (apart from C/C++ and Java since Papyrus already supports those), and is very powerful and flexible.

Using a plugin like Acceleo, however, can be less cumbersome. Acceleo is tailored for simple code generation and uses a template-based approach, thus allowing the user to generate any textual language from a UML model. Setting up a project for a code generator is a matter of minutes. Therefore, and also because of lacking documentation about the Xtend approach, Acceleo was chosen.

To start the development of the model-to-text generator, the Acceleo plugin has to be installed first. It features a custom Eclipse perspective

and allows the creation of a new Acceleo project. When creating a new project, the metamodel of the input model has to be specified (Eclipse's implementation of the UML2 standard in the case of the image processing DSL). In addition, the input of the template has to be defined, which is in this case the whole model. Figure 6 gives an overview.

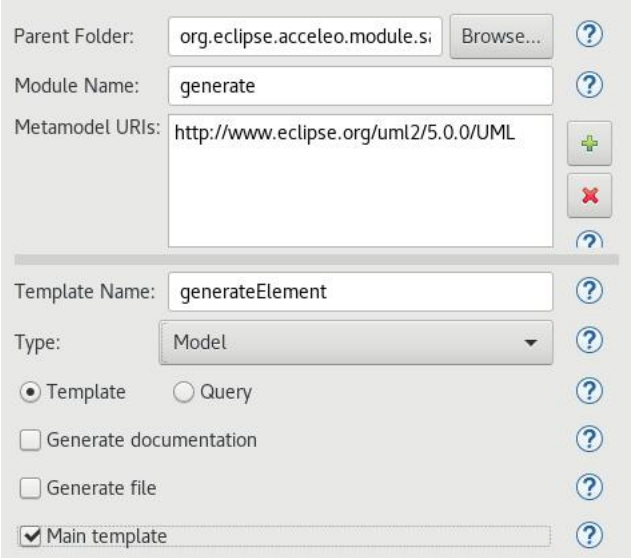

Figure 6: The definition of a new Acceleo project for the image pipeline DSL

After the project was created, the template has to be defined. In Acceleo, square brackets mark the beginning and the end of an expression that has to be evaluated, whereby expressions are based on the OCL language.

The following listing shows a code snippet of the template and demonstrates how to iterate over all classes in the model, and if the applied stereotype is of type *Source*, a function is called that generates Java code for loading an image.

```
1  [module generate('http://www.eclipse.org/uml2/5.0.0/UML')]
2  [template public generateElement(model: Model)]
3
4  [for (c: Class | model.eAllContents(Class))]
5    [for (stereotype: Stereotype | c.getAppliedStereotypes())]
6      [if (stereotype.name = 'Source')]
7        [c.genCodeLoadImage()/]
8      [/if]
9    [/for]
10 [/for]
```

The fully developed code generator works as follows: at first, a Java file called *Pipeline.java* is generated. It contains a single class which contains in turn the main function. Then, the Java code for loading the source image is generated. Afterwards, all connections between the blocks are traversed, and, depending on the type of the processing block, the respective Java code is generated. Hereby, an image processing library is used [9] that features several filters. At last, code for writing the output file is generated.

To run the code generator, a new runtime configuration has to be added. Here, the input model can be specified, which is the UML file of a pipeline model created by a user.

4.3. Extensibility

An important question is how easy it is to extend the existing DSML. For example, if a new processing block has to be added, which are the steps that have to be taken?

In short, the following procedure must be undergone every time a new element should be added to the DSML:

1. extend the image pipeline UML profile
2. extend the code generator
3. add the new element to a model

In detail, the UML profile is extended by adding another stereotype. As an example, a processing block called *DiffuseFilter* is to be added to the DSML. As can be seen in figure 7, a new stereotype was defined featuring an

[9] `http://www.jhlabs.com/ip/filters`

integer property called *strength* and an OCL constraint *validDiffuseSpecification* that checks if the strength property is valid, that is, if it is a positive integer. In addition, the new element extends the UML metaclass *Class*.

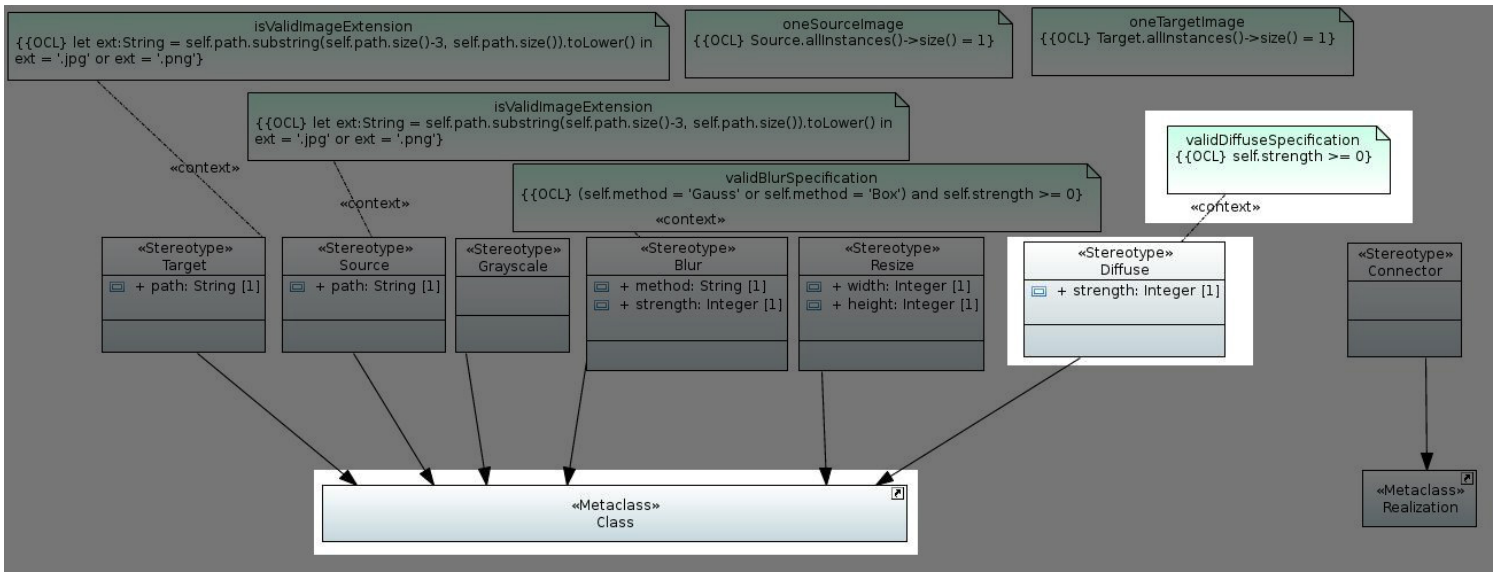

Figure 7: A new element is added to the DSML

As a next step, the code generator is adapted in a way so that it supports code generation for the new element. Fortunately, this process is quite easy, because it only requires to add another *else if* statement:

```
1  ...
2  [elseif (stereotype.name = 'Diffuse')]
3    System.out.print("Applying Diffuse Filter ... ");
4    filter = new DiffuseFilter();
5    filter.setScale([clazz.getValue(
6         clazz.getAppliedStereotype('ImagePipeline::Diffuse'),
7         'strength')/]);
8    image = filter.filter(image, null);
9    System.out.println("done");
10 [/if]
11 ...
```

In line 4, the *DiffuseFilter* is created. In line 5, the *strength* attribute of the stereotype is passed as an argument to the *setScale* function of the filter.

Finally, the filter is applied.

The last step consists of adding the new element to a model. First, an UML class object is added to the model and the stereotype is applied (see figure 8). Then, the new processing block is connected to the other pipeline elements by using Realiztion arrows (see figure 9). The *Connector* stereotype is in turn applied to the arrows.

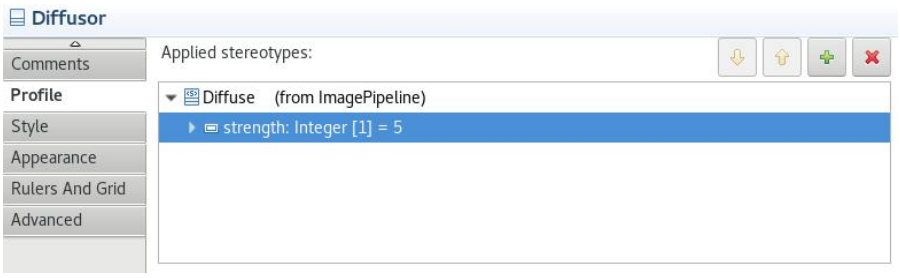

Figure 8: The *DiffuseFilter* stereotype is added to the UML class object

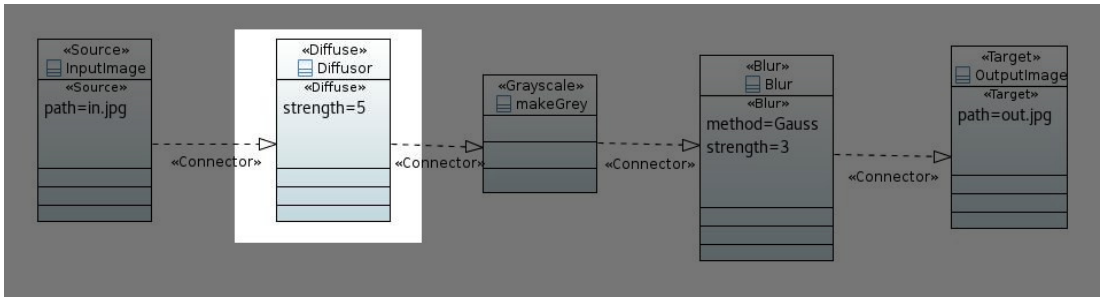

Figure 9: The diffuse filter is added to a pipeline model

5. Comparison with MPS

In this section, the experiences made with Papyrus are compared to the experiences made with another tool, Jetbrains MPS, which was used in another project to implement the same image processing DSL.

5.1. Tool complexity

This section deals with the question of how difficult the tools were to install, setup and use.

The **Papyrus** project provides compressed packages to download for Windows, Linux and Mac OSX. No installation is needed, the tool can immediately be used after downloading. Since it is based on the Eclipse platform, it resembles the famous Eclipse Java IDE very much, making it easy and intuitive to use.

While there exist many tutorials about how to get started with UML modeling, extensive documentation about how to implement custom DSLs is missing. Fortunately, a user guide describes the definition of UML profiles, but does not go into greater detail.

Also, documentation about how to extend or create a new code generator is outdated. On the positive side, however, there is a vibrant community that provides a lot of information in the official forums.

JetBrains MPS on the other hand is easy to install, also implemented in Java, and a comprehensive and official documentation exists as well as numerous screencasts and tutorials to guide the beginner through the complete process of defining a new language. One of its distinctive features is the projectional editor. Such an editor manipulates the abstract syntax tree directly rather than the concrete syntax. This might cause initial confusion, but is nowadays considered mature enough to be used productively.

5.2. Tool scope

This section deals with the question of what the tools are capable of.

Although **Papyrus** is mainly regarded as a tool for UML2 modeling, it is possible to create graphical DSLs for a lot of different domains. However, when using the UML profiles approach, these DSLs have to be abstracted from UML. In addition, by using plugins like Acceleo, code generation for almost any programming language is possible. By developing a plugin, Papyrus can be turned into a workbench explicitly tailored to the DSL.

MPS, on the other hand, is a language workbench and mainly provides support for textual domain-specific modeling languages, although graphical constructs are possible. The mbeddr[10] project is a good example. In addition, only Java is natively supported as a target language for code generation.

5.3. Usability of the DSL

This section elaborates on the question of how usable the developed DSLs are, that is, how easy it is for users to create models in this language.

The most significant difference between the DSL created with Papyrus and the one created with MPS is that the former is graphical, while the latter is textual (see figure 10). Intuitively, one would prefer a graphical DSL to a textual language for the domain of image processing.

However, without a plugin which turns Papyrus into a workbench for the DSL, it is cumbersome for users to actually use the DSL. For every element, the stereotype has to be applied manually.

```
Pipeline MyPipeline
  load image.png
  adjust brightness value 1.2
  grayscale
  blur method gauss strength 2
  save image.jpg
```

Figure 10: An example of how a pipeline model looks in the DSL implemented in MPS

The DSL implemented in MPS on the other side is, despite of the fact that it is text-based, easy to use and to understand. Hence, it is not considered a significant drawback that is is not graphical.

5.4. Extensibility of the DSL

In this section, the extensibility of the DSLs is discussed: how easily can the language be extended with additional elements?

Section 4.3 already discussed the extensbility of the DSL implemented in **Papyrus**. To summarize, it is easy to add new elements to the language: a new stereotype has to be defined, and the code generator template has to be adjusted.

[10] `http://mbeddr.com`

To add a new concept in **MPS**, the abstract syntax has to be defined first. This is done by adding a new concept (similar to a class in object-oriented programming) to the language which then is enriched by an editor for the concrete syntax. Finally, a template for the generator has to be written and augmented with macros.

It can be seen that the process of adding new elements to the DSLs is quite similar in both Papyrus and MPS.

5.5. Summary

Both Papyrus and MPS are useful tools for creating domain specific languages. Obviously, the main factor when it comes to the decision about which tool to use is whether a graphical or a textual DSL is needed. In addition, it is easier in MPS to come up with a tailored workbench for the DSL since it provides to the user a powerful editor. In Papyrus, on the other side, a plugin has to be developed first.

6. Conclusion and Future Work

In this paper, the graphical modeling tool Papyrus was evaluated by creating a custom DSML for a simple image processing pipeline. First, the DSML itself was specified. Then, the language was implemented by means of UML profiles, making use of stereotypes, attributes and constraints. At last, a code generator was developed which performs a model-to-text transformation. The result is valid Java code which can be compiled and executed.

Papyrus has proven to be a useful and powerful tool for creating domain-specific modeling languages. Although its main purpose is to serve as a UML modeling tool, it can be customized and extended. In addition, it has been shown that a custom DSL can easily be extended 4.3.

However, as Papyrus main purpose is not the creation of custom DSLs, there might exist tools which are more suitable for the creation of custom graphical modeling workbenches (Eclipse Sirius[11], DiaGen[12], Tiger[13]).

The comparison with JetBrains MPS showed that both tools are suitable for the task of creating DSLs. MPS, on the one side, is primarily used for textual languages and can be turned easily into a workbench for the implemented DSL. On the other hand, Papyrus is to be preferred for graphical languages and supports code generation for multiple programming languages. It can also be customized with plugins.

Future work might include the development of a Papyrus plugin which bundles the implemented UML profile 4.1 and the code generator 4.2 in a way so that a ready-to-use graphical modeling workbench results. By using such a plugin, the user would be presented with a palette containing graphical representations of the building blocks of the DSML. The development of such a plugin was started but not finished (see figure 11).

[11] `http://www.eclipse.org/sirius`

[12] `https://www.unibw.de/inf2/DiaGen`

[13] `http://www.user.tu-berlin.de/o.runge/tfs/projekte/tiger`

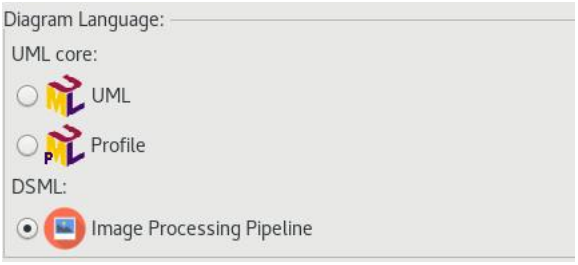

Figure 11: A Papyrus plugin could bundle the UML profile and the code generator so that the user can choose the Image Processing Pipeline DSML

7. References

Atkinson, C., Kühne, T., Jul. 2002. Profiles in a strict metamodeling framework. Sci. Comput. Program. 44 (1), 5–22.

Bordeleau, F., 2014. Model-based engineering: A new era based on papyrus and open source tooling. In: OSS4MDE@MoDELS.

Fuentes-Fernández, L., Vallecillo-Moreno, A., 2004. An introduction to uml profiles.

Hartmann, C., Reichenbach, M., Fey, D., 2015. Ipol - a domain specific language for image processing applications. Proceedings of the International Symposium on International Conference on Systems (ICONS 2015), 40–43.

Membarth, R., Reiche, O., Hannig, F., Teich, J., K"orner, M., Eckert, W., 2016. Hipacc: A domain-specific language and compiler for image processing. IEEE Transactions on Parallel and Distributed Systems 27 (1), 210–224.