Building Domain-Specific Modelling Environments with Papyrus: An Experience Report

Guillaume Dupont
ECE, Concordia University
Montreal, QC
gdupont@encs.concordia.ca

Sadaf Mustafiz
ECE, Concordia University
Montreal, QC
sadaf.mustafiz@concordia.ca

Ferhat Khendek
ECE, Concordia University
Montreal, QC
ferhat.khendek@concordia.ca

Maria Toeroe
Ericsson Inc.
Montreal, QC
maria.toeroe@ericsson.com

ABSTRACT

Domain-specific modelling with domain-specific languages (DSL) is rapidly gaining popularity in both research and industry for representing models of a target domain. However for a broader adoption, MDE tools need to provide adequate support for building these DSLs as well as for using the DSLs.

We have been using the Papyrus modelling tool for domain-specific modelling. Papyrus implements the OMG UML 2 standard and uses the UML Profiling mechanism to model domain-specific notions. It has significantly evolved over the past years and is now considered to be industry ready. In this paper, we share our experiences on using the tool for domain-specific modelling. We base our work on the advanced customizability features for domain-specific modelling offered by Papyrus. However, we soon came to realize that creating a domain-specific modelling environment is a non-trivial task with Papyrus. We elaborate on the issues we encountered and on the steps taken to overcome them. As a possible solution for language engineers, we also developed a tool which can be used to automatically generate domain-specific modelling environments based on a given profile in Papyrus.

CCS CONCEPTS

• **Software and its engineering → Domain specific languages**;

KEYWORDS

domain-specific modelling, modelling tools, Papyrus, Eclipse

ACM Reference Format:
Guillaume Dupont, Sadaf Mustafiz, Ferhat Khendek, and Maria Toeroe. 2018. Building Domain-Specific Modelling Environments with Papyrus: An Experience Report. In *MiSE'18: MiSE'18:IEEE/ACM 10th International Workshop on Modelling in Software Engineering, May 27, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3193954.3193962

MiSE'18, May 27, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5735-7/18/05.
https://doi.org/10.1145/3193954.3193962

1 INTRODUCTION

Model-Driven Engineering (MDE) advocates domain-specific modelling (DSM) and the use of domain-specific modelling languages (DSML or simply DSL) to specify models pertaining to a certain domain. These domain-specific models need to conform to the language definition of the DSL. A modelling language is defined in terms of its abstract syntax, concrete syntax, and semantics [12]. A DSL is defined based on concepts and notations representing the domain [14]. These concepts and notations are the basis for specifying the abstract syntax and concrete syntax (a textual or visual representation) of the language.

With the advent of domain-specific modelling, various tools have come into existence to support the development of domain-specific modelling tools or workbenches. Such tools are often referred to as DSM environments, DSL editors, or DSL model creation tools, or simply as DSM tools. For complete DSM support, tooling is required for 1) creating the DSL itself, and 2) creating model instances of the DSL. Very few tools are available to provide the required level of support, and hence this shortfall has restrained broad adoption of DSLs [25]. Sirius[1], AToMPM[23], and the commercial tool MetaEdit+ [24] are examples of some existing DSM tools. Yet another DSM tool that has recently gained popularity and is rapidly evolving is *Papyrus*.

Papyrus[2] is an Eclipse-based modelling tool developed by the CEA (Commissariat a l'Energie Atomique) based on OMG UML 2 [2]. It rapidly gained in popularity and is widely used in the industry and is sometimes even recommended by standards. Among others, ETSI NFV (Network Functions Virtualization) [4] has made Papyrus it's tool of choice. The migration of industrial development from proprietary tooling to open-source Papyrus led to the evolution of the now industrial strength Papyrus tool [6]. For this reason, we decided to investigate the capabilities of Papyrus.

As stated in [10] regarding Eclipse Papyrus: *"It intends to provide an open, robust, highly scalable, and highly customizable tool for defining DSLs and corresponding tools. It does so using the UML profile mechanism as well as powerful UI customization facilities."*. Bordeleau [8] states that *"the broad adoption of MBE has been significantly limited by the fact that existing tools have failed to provide for better customisability and support for Domain-Specific Modeling*

[1] https://www.eclipse.org/sirius
[2] https://www.eclipse.org/papyrus/

Language (DSML) and to deliver capabilities to cover for the broad range of development aspects that are considered critical by end-users". Development of advanced customization and DSL capabilities of the Papyrus modelling tool was expected to address this need [8].

Papyrus has evolved over the past years and is now recognized as a solid tool for information modelling. Papyrus provides diagram editors for UML 2, SysML, and others. However, based on our experience, Papyrus does not currently seem to offer a simple and elegant DSM solution for creating user-defined diagram editors or modelling environments. Its domain-specific modelling capabilities at this time are still basic and needs to evolve to meet the expectations of language engineers. Papyrus defines DSLs based on UML profiles which imposes various restrictions making it difficult to define new DSLs in the tool. The advantages and drawbacks of defining DSLs based on UML profiles by making domain-specific extensions of the UML metamodel are discussed in [20]. The Papyrus International Consortium has addressed the need to offer improved support for customization and DSLs, as well as improved support for DSL toolsmiths. It also intends to *provide a set of tools and techniques that can be applied to customize Papyrus or develop DSL on top of Papyrus (including the recursive nature of DSL's).* These are part of the product goals for 2017 as stated in [5].

In this paper, we share our experiences on using Papyrus for creating DSLs and using these DSLs for domain-specific modelling. DSL development is not an easy task as it is: *"defining the concepts and relationships that lead to establishing a language to represent a particular domain requires time and effort"* [17]. On top of this, if the actual creation of the DSL with a tool and the use of the language as a model editor are also made difficult or cumbersome, it makes software engineers less receptive to the idea of taking the DSL path.

UML profiles are widely used and accepted for defining DSLs, as development time for DSLs can be greatly reduced by extending existing UML tools [20]. The basic usage is to define profiles and apply these profiles on their models to fulfill their domain-specific modelling needs. While time and effort for creating a DSM tool is not spent, this practice can be tedious and error-prone. When a DSL is defined and model instances of it need to be created, it brings about the need for a DSM tool or model editor to be developed in a simple and user-friendly manner.

While some tools (see Section 5) offer a relatively simple process requiring minimal effort and time for designing and using these DSLs, Papyrus still needs further work to make the tool amenable to DSL developers and users. The users are typically software engineers in the industry, or software engineering students and researchers in academia. While Papyrus supports development of DSM tools, the process is complex and barely intuitive making it rather difficult for a Papyrus newcomer to figure out what needs to be done. For one, there is a great need for adequate documentation to be available for users. Secondly, the process of creating modelling editors could possibly be made easier on the user side. In this paper, we aim to address the former by describing and providing clear guidelines for using the advanced features of Papyrus to create DSL editors. To address the latter issue, we went one step further and automated the process by developing a plugin generator for DSLs. *It should be noted that this work was carried out prior to the release of Papyrus Oxygen, and hence the Papyrus release we worked with and extended is Neon (2.0.2). Neon is still the recommended version at the time of writing of this paper.*

This paper is structured as follows: Section 2 provides essential background on metamodelling and profiles. Section 3 outlines how UML profiles are used for domain-specific modelling in Papyrus, and describes how we used the customization features of Papyrus to create a DSM tool and the issues encountered along the way. Section 4 presents a generator for creating DSM tools automatically from a profile in Papyrus. Finally, Section 5 talks about related work and Section 6 concludes with some final thoughts.

2 BACKGROUND

This section provides essential background on metamodels and profiles.

2.1 Metamodels

A model is a set of objects with relations between them which conform to what is referred to as a *metamodel* of the model. The metamodel represents, in essence, a *type* or a *template* of a model, and describes concepts and their relationships along with rules which need to be adhered by any instance of this metamodel.

The *Meta Object Facility* or *MOF* [3], a specification on how to represent and use metamodels and models, is often referred to as the *meta-metamodel*. On a concrete level, MOF has not been strictly implemented. However, one of its subset (*Essential MOF* or *EMOF* for short) has been broadly implemented under the *Eclipse Modelling Framework (EMF)* project as *Ecore* [22], and is what is traditionally used whenever dealing with models at the code level.

The OMG defined a modelling language (based on MOF), with the intention of creating more concrete metamodels for designing software and systems, the *Unified Modelling Language* or *UML* [2].

2.2 UML Profiles

UML introduced a concept called *profiles* [1], which is basically a way of extending the syntax and the semantics of UML. A profile takes the form of a standard UML *class diagram*, defining classes and relations between them such as realization, association, etc. However, some of the classes defined in the profile can be marked as being *stereotypes*. A stereotype is a special type of class which can be used to distinguish particular elements in other UML models. Additionally, it is possible to constraint the elements on which a stereotype can be applied by making it *extend* a UML meta-class, that is: indicating that a stereotype is actually extending the semantics of a particular type of UML element.

A model can use an existing profile, in which case it is said that the profile is being *applied* to it. When a profile is applied, it is then possible to apply the stereotypes of that profile to the objects of the model, giving them additional roles, meanings, etc.

2.3 Metamodels or Profiles?

It should be noted that even though they look quite similar to MOF metamodels, profiles are in fact very different. For instance, several profiles can be applied to a model, whereas a model can only conform to *one* metamodel. Similarly, it is possible to apply zero to many stereotypes to an object, whereas in a model, every object conforms to exactly one meta-class of its metamodel.

Profiles have been proposed as a convenient way for extending the reach and capabilities of UML, allowing complex structures to be expressed or creating subsets of it that fit a very special need. They can be used as a way of expressing metamodels, as long as their model instances have only one profile applied, with every object being stereotyped.

3 DOMAIN-SPECIFIC MODELLING SUPPORT IN PAPYRUS

Creating a domain-specific modelling environment involves defining the abstract syntax model (or metamodel) of the DSL, defining one or more concrete syntax models and associating them with the abstract syntax, and defining a semantics model (possibly with the use of model transformations) [12, 15]. This separation between the abstract syntax and concrete syntax of a language is important to make in any metamodelling tool both at a conceptual level and at the concrete level, since the abstract syntax is really independent of the concrete syntax. The abstract syntax is typically defined as an instance of a Class Diagram or Entity-Relationship Diagram (which represent the meta-metamodels). Depending on the support available, even concrete syntax models can be defined as instances of an embedded concrete syntax formalism. The concrete syntax may be textual or visual. In the latter case, a buttons toolbar or a palette can then be created based on the defined syntax.

If the models to be created using the modelling environment is not expected to be simulated, executed, or analyzed, it is not mandatory to define a semantics model. A constraints language (for instance, OCL) can be used to incorporate constraints at the syntax level based on which syntactic checks and validation can be carried out.

The steps vary from tool to tool and so does the complexity involved in the process. In this section, we discuss the domain-specific modelling support available in Papyrus.

3.1 Domain-Specific Modelling with Profiles

We outline here the process of creating DSLs using UML Profiles in Papyrus Neon. Papyrus was developed to be a UML-based modelling tool; and to that extent, it provides support for domain-specific languages (DSL), allowing a DSL designer to offer users a proper tool for using it. However, the process of defining a DSL is completely based on the use of profiles which make this a very cumbersome task.

3.1.1 Defining Abstract Syntax. Papyrus provides full support for UML 2 profiles [10]. It is possible to define a profile as a UML model, and then load this profile in other UML models (effectively *applying* it to that model), which allows for applying some of its stereotypes to elements of that model. It is even possible to define constraints in the profile using OCL.

3.1.2 Defining Concrete Syntax. Papyrus provides a way for applying a *style* to a diagram in a very CSS-like fashion, which allows specification of custom appearance depending on the type of element but also and above all, its stereotypes, effectively proposing a way to specify a more precise concrete syntax for our profile.

3.1.3 Defining Semantics. Operational or translational semantics for the DSLs need to be specified in order to execute or simulate the models. Moka [11], a recent addition to Papyrus, may be used for this purpose. It complies with the OMG standards fUML, PSCS, and Alf. However, we have not experimented with Moka as yet. *Note: In this paper, we focus on the modelling aspect: modelling of DSLs and modelling with DSLs. Execution and transformation of these models are outside the scope of this paper.*

3.1.4 Using Profiles for Creating Domain-Specific Models. In practice, with reference to the Papyrus Neon release, some desired features are missing and some issues with the tool exist, which results in suboptimal usability. To begin with, stereotypes need to be applied *one by one*, which quickly becomes cumbersome when dealing with models that have too many elements. Besides, the way profiles have been thought of makes it so that you can have a diagram with elements that do not have stereotypes, or that have several stereotypes (which would not be possible if we were using standard metamodels, assuming a stereotype is equivalent to a metaclass). This can easily lead to invalid models. This naive approach in our opinion is not ideal for domain-specific modelling. It is much desirable to have a modelling environment for domain-specific modelling which works as a model editor or tool for a specific DSL. We elaborate on our experience in using UML profiles for creating DSL modelling environments in the next section.

3.2 Creating a Plugin for a DSL

"One of the big strengths of Papyrus is that you can reuse Papyrus as a DSML environment. In order to do so, one has to write a Papyrus plug-in" [18].

The current support provided by Papyrus (version Neon) allows creation of domain-specific modelling environments. However, it is left to the designer to figure out how to go about creating such a tool. To the best of our knowledge, there is no available documentation providing an explicit process to help Papyrus newcomers on this regard. We have spent time and effort in understanding the customization and extension features of Papyrus, and we share here the process for creating a domain-specific modelling tool. We describe the steps we took for creating the environment with the use of the Papyrus plugin mechanism. The steps need to be followed in order and are mandatory, unless otherwise mentioned. The main elements based on which the plugin is built are shown in Figure 1.

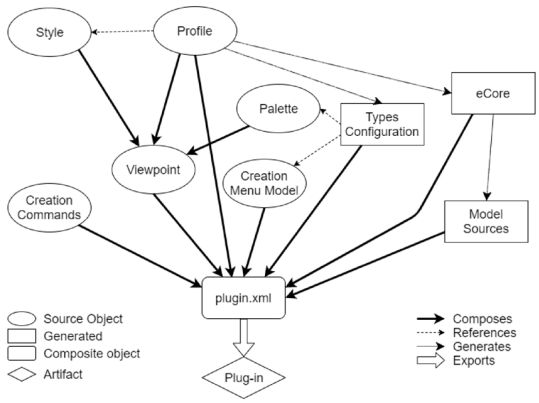


Figure 1: Creating a Papyrus DSL plugin.

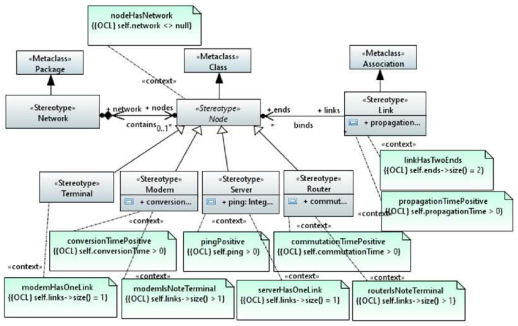


Figure 2: `SimpleNet profile`.

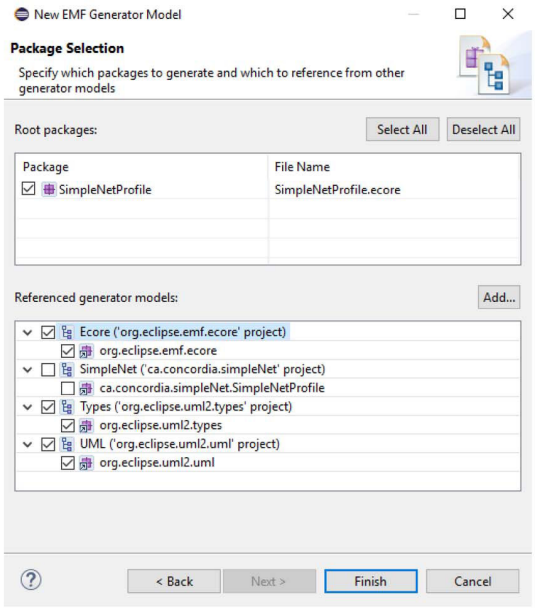


Figure 3: `.genmodel` generation dialog.

3.2.1 Defining the Profile. Papyrus implements the OMG UML specification and handles only UML models. Therefore, it is not possible to manipulate metamodels (like Ecore metamodels) directly with it. The base for this tool *must* be a UML profile, which needs to be created with Papyrus as the first step. We demonstrate the process with the use of a simple example DSL, *SimpleNet* (a DSL modelling simple networks, shown in Figure 2).

3.2.2 Generating the Model Sources. Papyrus needs each metaclass of the profile to be mapped to an existing Java class in order to manipulate them. That is why we have to generate the *source* of the model, via an EMF (Eclipse Modelling Framework) generator model (`.genmodel`). This is done via a dialog shown in Figure 3 and yields the architecture shown in Figure 4.

3.2.3 Creating the Commands. In order for us to create a new type of diagram, we need to provide Papyrus with two handlers,

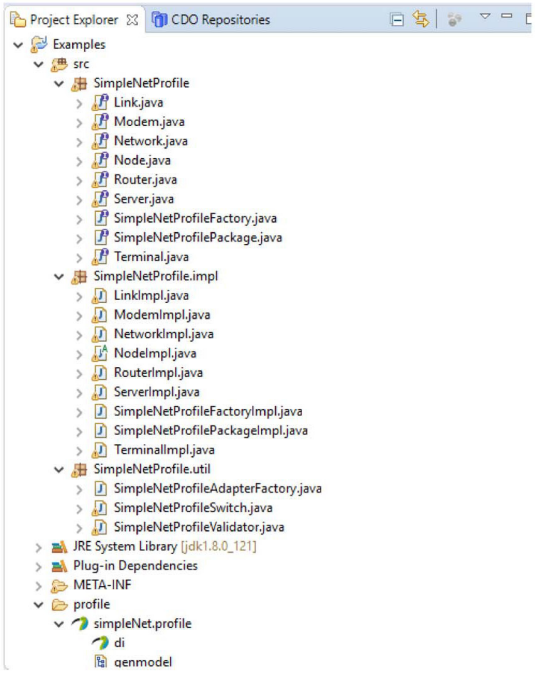


Figure 4: Generated sources for the `SimpleNet` example.

one for creating the base UML model for the diagram and the other one for creating the diagram.

3.2.4 Generating the Element Types Configuration. The mapping between the profile's stereotypes and the corresponding UML metaclasses as well as the one between these stereotypes and the UML diagram elements are stored inside two files that can actually be generated from the profile by Papyrus. These files are crucial as they define how Papyrus should handle the creation of elements in the diagram and map that to the creation of elements in the model. This is done via the dialog shown in Figure 5.

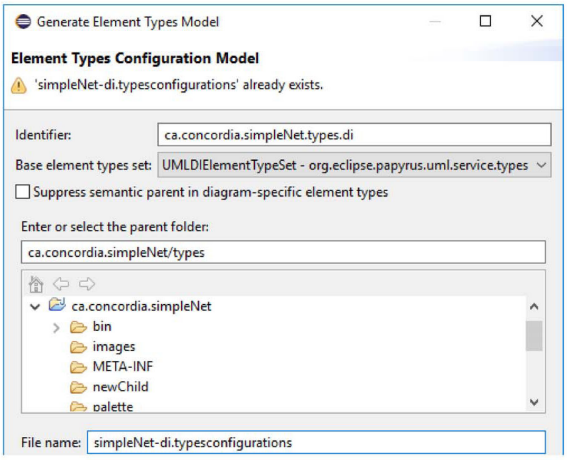


Figure 5: Element Types Configuration generation dialog.

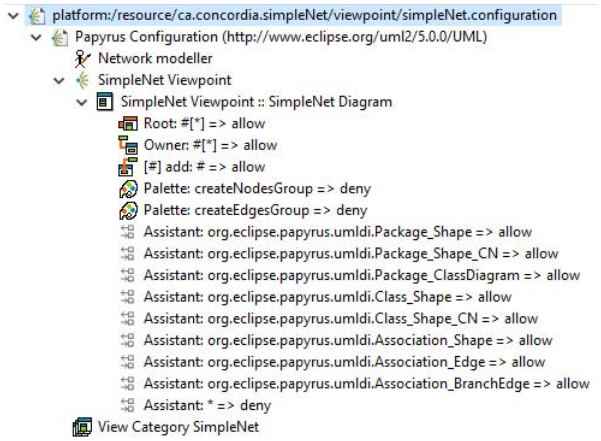


Figure 6: Viewpoint definition for the `SimpleNet` example.

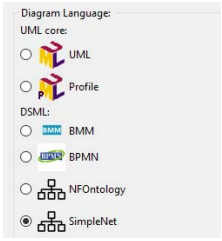


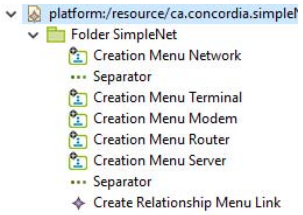
Figure 7: Modified model wizard with the new entry for the `SimpleNet` DSL.

3.2.5 Creating the Viewpoint. The viewpoint configuration model is a special type of file provided by Papyrus which allows us to define a new diagram category as well as diagram types for this category. It is also the file that binds together the style, the palette and the profile. Figure 6 shows an example of a viewpoint definition and Figure 7 shows the new defined DSL (*SimpleNet*) in the Papyrus new model wizard.

3.2.6 Defining the Style. Papyrus provides a way for customizing diagrams via a CSS-like style sheet, allowing the designer to refine the visual concrete syntax for the profile (adding stereotype-based colours for instance). This step is optional.

3.2.7 Creating the Context Menu Definition. It is possible to define a custom context menu that will be added to the "Add element" sub-menu inside the context menu of the model explorer. This menu is quite useful as it will append an element to the model with the correct profile and stereotype(s) applied. This step is optional. Figures 8a and 8b respectively show the definition of the contextual menu and its result in the tool.

3.2.8 Creating the Palette Definition. The palette designates the pop-up window on the left side of the workspace allowing the user to drag-and-drop elements into the diagram. Papyrus provides a way of customizing the palette, allowing the designer to restrict the elements a user can insert and automatically pre-configure those elements so that they have the correct stereotype. This step



(a) Context menu definition for the `SimpleNet` example.

(b) Result of the menu defined in Fig. 8a.

Figure 8: Contextual menu in Papyrus.

is optional (although recommended to be taken). Figures 9 and 10 show the definition of the palette and its result in the tool.

```
<?xml version="1.0" encoding="UTF-8"?>
<paletteconfiguration:PaletteConfiguration
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:paletteconfiguration="http://www.eclipse.org/papyrus/diagram/paletteconfiguration/0.7"
    id="ca.concordia.simpleNet.palette" label="SimpleNet"
    description="Palette configuration for SimpleNet profile">

    <drawerConfigurations
        id="ca.concordia.simpleNet.palette.networkdrawer"
        label="Network"
        description="Network">

        <ownedConfigurations
            xsi:type="paletteconfiguration:ToolConfiguration"
            id="ca.concordia.simpleNet.palette.nodes.Network"
            label="Network"
            description="Create a Network">
            <elementDescriptors
                elementTypeId="ca.concordia.simpleNet.types.di.Network_Package_Shape" />
            <elementDescriptors
                elementTypeId="ca.concordia.simpleNet.types.di.Network_Package_Shape_CN" />
            <icon
                pluginID="ca.concordia.simpleNet"
                iconPath="images/network16.png"/>
        </ownedConfigurations>
    </drawerConfigurations>
```

Figure 9: Palette definition for the `SimpleNet` example.

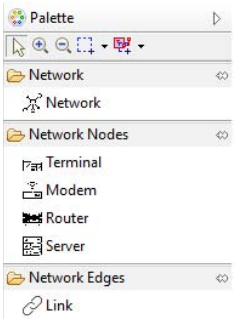


Figure 10: Result of the palette defined in Fig. 9.

3.2.9 Creating the Plug-in Definition. Overall, the extension mechanism is based on Eclipse's plug-in system. Most of the files are provided *via* extension points that are defined by Papyrus, and everything is bundled as an Eclipse plug-in (therefore defined by `plugin.xml`, `MANIFEST.MF` and `build.properties` files).

The resulting functional environment can be used to create model instances conforming to the user-defined domain-specific language. It is to be noted that the tool yields a *viewpoint* to Papyrus, and the user needs to specify that she actually wants to use this viewpoint instead of the default one via the "Preferences" dialog of Papyrus. This results in the revised model wizard with the new entry for the DSL created (as shown in Figure 7). The generated DSM tool is quite complete: from creation of the model to editing, with automatically applied profiles and stereotypes, concrete syntax, etc.

A video demonstrating the steps outlined in this section is available at https://users.encs.concordia.ca/~magic/dsmtoolgen-demo.php.

3.3 Issues Encountered

In essence, it is important to note that the creation of this tool presented so far brought a lot of errors, highlighting the problems that come with this method, as well as starting the reflection for potential improvements.

3.3.1 Absence of Documentation. One of the major issues that must be addressed regarding the use of Papyrus as a base for a tool is that it is quite difficult to find relevant pieces of work that indicate the way to go. In fact, except for the two project reports [13, 18] and the Papyrus *wiki* (and help contents), there was nothing clearly explaining the way Papyrus works in this regard and how to use it to create a full-featured environment. We would like to mention that following the completion of this work, there was a tutorial given by the Papyrus team at MoDELS 2017 (in September 2017) on developing DSM tools with Papyrus. This tutorial is not available online and we requested the presenters to share the tutorial material with us. While the presentation slides cover details that are not available elsewhere, this documentation was however the means of giving the tutorial and does not provide step by step details of the complete process of creating DSM tools.

We had asked a Software Engineering graduate student to walk through the Papyrus tutorial to create a DSM environment. We had also shared with him a draft of this paper. Based on the student's experience, we learned that he had tried to follow the tutorial slides but required clarifications and would have to explore on his own further to complete the given task if the paper was not available. The student found the process outlined in this paper easier to follow, and provided him with the necessary guidelines to build a DSM environment successfully.

Most of the steps of the method we came up with have been the result of the compilation of on-line resources, but also some digging into Papyrus' code (notably the code of already existing Papyrus-based tools such as SysML). In the end, we still had to figure out some of the steps by ourselves, and there are still some of them we are not quite sure if they are relevant or even what they are there for.

3.3.2 Platform Issues. The Eclipse platform being a (very) complex tool for which contributions come from a lot of different teams, it is not very surprising that it sometimes behaves in strange ways. Because Papyrus is a recent and evolving tool, it still presents a lot of bugs on very peculiar situations, which are unfortunately very likely to happen when using it as we did.

But the real issue that comes with this last assertion is that error tracking in Papyrus is not very meaningful. For instance, it took us several hours to figure out the source of *Unhandled event loop exception* errors appearing when trying to drag and drop elements from the palette was actually coming from the *element types configuration* files.

Moreover, sometimes some kind of spurious errors occur in Papyrus, which can corrupt files and even an entire workspace. It happened several times that the tool simply ceased to work for no apparent reason.

3.3.3 Missing Features. One of the biggest concern we are pointing out as developers is that the environment created with this method is *incomplete*; and more than that, it is *not* easily extensible. For instance, every element of a diagram must be mapped to a *stereotype*, meaning that every link of a model is actually a meta-class in the profile, generally extending a standard UML type of link (such as association, dependency, and so on).

But when it comes to creating the model, whenever we want to link two elements with a custom type of link (that is, a link with an associated stereotype), the link will be between those elements *as in UML*, and not *as in our profile*. This basically means that once we have linked two elements, we must manually set the attributes of this link so that it actually *does* the link between our two objects.

We came to know later that one way of going around this would be to extend the palette associated with the diagram and new child menus to create the UML elements with the stereotype pre-applied. This is however one that a novice Papyrus user might not be aware of or might find difficult to use. Making documentation available on these aspects would be beneficial for Papyrus users.

3.3.4 Structural Problems. The method for creating this modelling environment (as described earlier in this section) sometimes requires manual editing of automatically generated files (such as in the step for generating the Element Types Configuration) that must be taken *every time* something changes in the types of the profile.

In general, changing the environment's base profile often means recreating it, redoing essentially the same steps as before.

4 EXTENDED TOOL SUPPORT FOR DSL TOOLSMITHS

This section presents the work carried out for automating the generation of plugins for modelling environments.

4.1 Automated Generation of DSL Modelling Environments

From our experience on building modelling environments in Papyrus, we realized that many of the problems we encountered were coming from typographical mistakes or oversights which slowed

down the process and could have been avoided. From this observation, it appeared clear that the steps taken for creating a modelling environment from a UML profile were very likely to be the same (independently from the input), meaning these steps could probably be automated to some point.

It turned out every step involved in the creation of such a domain-specific modelling environment can be automated, and we achieved this with the use of an Acceleo[3] model-to-text transformation. The resulting tool is a generator for DSM tools (specifically Eclipse plug-ins). The starting point for the tool is the profile that serves as a base for the modelling environment, as well as a configuration file that holds every additional parameter needed for the generation (project name, custom files, etc.). The tool generates the whole project architecture, providing every file (whether mandatory or not) with a basic and working skeleton. The only thing that is left to do is to generate the sources (see step in Section 3.2.2) and the plug-in for the DSL is ready. It can of course be customized once it has been generated.

This approach has numerous advantages, starting with reduced probability of errors coming from oversight and typos, as well as a substantial reduction in development time, as we no longer need to write the same pieces of code over and over again. Moreover, as the tool essentially consists of various Acceleo templates, it is quite easy to extend, allowing its users to provide additional automatically generated elements (custom behaviors, property views specification, etc.).

The second part of the video available at the following link demonstrates our plugin generator tool: https://users.encs.concordia.ca/~magic/dsmtoolgen-demo.php.

4.2 Possible Extensions

While the plugin generator is completely functional, it is still possible to improve the tool in several ways. The tool could transform metamodels into profiles – this feature is one which Papyrus Oxygen claims to have support for and it would be useful to integrate this support into our tool. Another possible way the tool could be improved is by creating additional standard commands to be triggered when creating objects, for example to avoid having to fill in some attributes (e.g., ends of a link) when creating specific objects (e.g., a stereotyped link between two stereotyped classes). Customizing the properties view is also something we did not experiment with. However, as we understand it, this is something that can be done by using the property context files, which basically associates the elements attributes (UML or notation) with predefined field editors using the databinding APIs.

Another possible extension to the tool would be to customize the model explorer by using EMF facets to change the ownership structure in the model explorer, basically to change parent-child relations of the model explorer tree elements. Moreover, to allow syntactic or structural semantic checks to be enforced, one possibility is to integrate the MetaModel Agent[4] which includes a model validator to detect and correct DSL violations.

The tool developed is easily extensible, and hence it should not be difficult to incorporate these extra features.

4.3 Discussion

Based on our experience (as described in Section 3), building domain-specific modelling tools with Papyrus currently is non-trivial for practitioners and researchers without adequate Eclipse development skills.

To allow language engineers to model a DSL in an intuitive and easy manner, a tool needs to provide the basis for creating a sound language definition. This means that it should be possible to define a language in terms of its abstract syntax, concrete syntax (one or more), and semantics. To achieve this, one way would be to make Papyrus evolve by incorporating notions of metamodelling as part of the tool. The process of creating DSLs and diagram editors for them can then hopefully be much simplified, leading to a significant reduction in development time and effort hence making the life of DSL toolsmiths much easier.

5 RELATED WORK

We review here textual and graphical tools which primarily support creation of DSL editors or modelling tools, as well as ones which allow domain-specific modelling. There are several tools offering basic to advanced features for DSL toolsmiths. Some, for instance Xtext[5] and Jetbrains MPS[6], targets textual DSLs. Eclipse Sirius[1] allows the creation of graphical modelling workbenches by defining abstract and concrete syntaxes in a quick and easy manner on top of EMF (Eclipse Modelling Framework) and GMF (Graphical Modelling Framework). Sirius is used as a base for the commercial tool, Obeo Designer. Besides Sirius, other open-source tools with similar capabilities include GME (Generic Modeling Environment [16]) and EMF-based tools such as Eugenia, GMF, and Graphiti. DiaGen [19] and Tiger [7] also support graphical DSLs but there is no clear distinction made between the abstract and concrete syntax of a modelling language in these tools. A comparative analysis of tools supporting DSLs is provided in [21].

The Rational Software Architect (RSA) supports UML, profiles, and OCL. It also allows profile tooling to be generated, however the process is not completely stable as stated in [20]. It is also a proprietary tool making any DSL tooling based on RSA less accessible to the MDE community. AToM3 [9] and its newer web-based version, AToMPM [23], are metamodelling and model-transformation tools with advanced support for creating graphical modelling environments for DSLs as well as UML diagrams. The process of defining a DSL and then building a modelling environment for it is simple and clean and completely model-based. They do not however conform fully to the OMG UML 2.0 standard. Moreover, these are research tools and hence support and stability are not to the level required for industrial projects.

MetaEdit+ [24] is a commercial domain-specific modelling environment which offers users with an advanced and stable workbench tool to design the DSL and then with a modeller tool through which to use the DSL with diagramming editors, browsers, generators,

[3] Acceleo is a code generator which implements OMG's model-to-text specification (http://wiki.eclipse.org/Acceleo).
[4] http://www.adocus.com/en/products/metamodelagent/

[5] http://www.eclipse.org/Xtext
[6] https://www.jetbrains.com/mps

and multi-user support. The cost of the license however reduces accessibility and hence the userbase. The same applies for the Eclipse tool, Obeo Designer.

In contrast to some of the above tools, Papyrus is not in essence a metamodelling tool but is primarily an information modelling tool for UML. Metamodelling is achieved with the use of UML Profiles. In the context of Papyrus and DSL support in Papyrus, some initiations were made to develop DSL workbenches with plugins but these projects were left incomplete to the best of our knowledge [13, 18]. Papyrus-RT[7] supports DSL modelling however Papyrus-RT is an implementation of the UML-RT modelling language and hence supports only a subset of UML 2 and also introduces new concepts relevant for modelling of real-time systems.

6 CONCLUSION

In this paper, we shared our experiences on using Papyrus (Neon 2.0.X) for developing modelling environments (diagram editors) for domain-specific languages (DSL). Papyrus offers advanced features and customization options for DSL developers based on the UML profile mechanism. However, in our experience the features are not enough for designing and using DSLs in a simple and effective manner. We have been presented with various roadblocks when trying to develop a working and usable domain-specific modelling tool as discussed in the paper. The lack of documentation and experience reports on the topic slowed us down further. In our opinion, the tool needs to be further improved in this direction. Streamlining and documenting the creation process of DSLs would lead to Papyrus gaining a bigger userbase [18]. Documenting (and sharing) this process as well as the lessons learned was the main motivation behind this paper.

We also proposed a solution for building DSM tools in Papyrus. We have created a tool which takes as input a base profile and generates DSM environments as Papyrus plugins. The tool provides the ability to create a tooling palette for the DSL that can be used by the DSL users to create model instances of the language. The environment once built can be integrated into Papyrus such that it can be chosen as the DSL of choice when creating a new Papyrus project. This is a generic tool (Papyrus plugin) which can be used to generate new DSL editors in Papyrus. We discussed in the paper the issues we encountered along the way, how we addressed them, and we also described our main steps for coming up with the tool. This is to provide helpful guidelines for anyone working on creating modelling environments for DSLs using Papyrus.

While our solution allows DSL toolsmiths to create diagram editors with much greater ease and in a shorter time than currently possible with Papyrus, the basis of the DSL would still be UML profiles. In our opinion, Papyrus needs to evolve from being a tool for information modelling to a tool for metamodelling to provide much needed support for creating DSL workbenches.

ACKNOWLEDGMENTS

The authors would like to thank Antonio Robles from Ericsson for his valuable feedback on this work, and Omar Hassane for his help with the demo video. This work is supported by NSERC and Ericsson under Grant No.: IRCPJ 425135-16.

[7] https://eclipse.org/papyrus-rt/

REFERENCES

[1] 2013. *OMG Unified Modeling Language Specification Version 1.5.* Standard. Object Management Group (OMG).
[2] 2015. *OMG Unified Modeling Language™(OMG UML) Version 2.5.* Standard. Object Management Group (OMG).
[3] 2016. *OMG Meta Object Facility (MOF) Core Specification Version 2.5.1.* Standard. Object Management Group (OMG).
[4] 2017. *Network Functions Virtualisation (NFV) Release 2; Information Modeling; Papyrus Guidelines - ETSI GR NFV-IFA 016.* Standard. European Telecommunication Standards Institution (ETSI).
[5] 2017. Papyrus IC 2017 Plan. (2017). https://wiki.polarsys.org/Papyrus_IC_2017_Plan (Online; accessed December 2017).
[6] Ronan Barrett and Francis Bordeleau. 2015. 5 Years of 'Papyrusing' - Migrating Industrial Development from a Proprietary Commercial Tool to Papyrus (Invited Presentation). In *Proceedings of the International Workshop on Open Source Software for Model Driven Engineering co-located with MODELS 2015 (CEUR Workshop Proceedings)*, Vol. 1541. CEUR-WS.org, 3–12.
[7] Enrico Biermann, Karsten Ehrig, Claudia Ermel, Christian Köhler, Günter Kuhns, and Gabi Taentzer. 2006. *Tiger EMF model transformation framework (EMT) Version 1.2.0.* TU Berlin - EMT Project Team.
[8] Francis Bordeleau. 2014. Model-Based Engineering: A New Era Based on Papyrus and Open Source Tooling. In *1st Workshop on Open Source Software for Model Driven Engineering (OSS4MDE) co-located with MoDELS 2014 (CEUR Workshop Proceedings)*, Vol. 1290. CEUR-WS.org, 2–8.
[9] Juan de Lara and Hans Vangheluwe. 2002. AToM^3: A Tool for Multi-formalism and Meta-modelling. In *Fundamental Approaches to Software Engineering, 5th International Conference, FASE 2002, held as Part of ETAPS 2002 (LNCS)*, Vol. 2306. Springer, 174–188.
[10] Sébastien Gérard, Cédric Dumoulin, Patrick Tessier, and Bran Selic. 2010. Papyrus: A UML2 tool for domain-specific language modeling. In *Model-Based Engineering of Embedded Real-Time Systems*. Springer, 361–368.
[11] Sahar Guermazi, Jérémie Tatibouet, Arnaud Cuccuru, Saadia Dhouib, Sébastien Gérard, and Ed Seidewitz. 2015. Executable modeling with fUML and Alf in papyrus: tooling and experiments. *strategies* 11 (2015), 12.
[12] D. Harel and B. Rumpe. 2000. *Modeling Languages: Syntax, Semantics and All That Stuff, Part I: The Basic Stuff.* Technical Report. Jerusalem, Israel, Israel.
[13] Dominique Heer. 2017. *Implementation of a DSL in Papyrus.* Technical Report. University of Antwerp.
[14] Steven Kelly and Juha-Pekka Tolvanen. 2000. Visual domain-specific modelling: Benefits and experiences of using metaCASE tools. In *International Workshop on Model Engineering, at ECOOP*, Vol. 2000. 1–9.
[15] A.G. Kleppe. 2007. A Language Description is More than a Metamodel. In *Fourth International Workshop on Software Language Engineering (ATEM 2007) at MODELS 2007*. Nashville, TN, USA.
[16] Akos Ledeczi, Miklos Maroti, Arpad Bakay, Gabor Karsai, Jason Garrett, Charles Thomason, Greg Nordstrom, Jonathan Sprinkle, and Peter Volgyesi. 2001. The generic modeling environment. In *Workshop on Intelligent Signal Processing*, Vol. 17.
[17] Marjan Mernik, Jan Heering, and Anthony M. Sloane. 2005. When and How to Develop Domain-specific Languages. *ACM Comput. Surv.* 37, 4 (Dec. 2005), 316–344.
[18] Kristof De Middelaer. 2016. *Modelling with Papyrus: Domain-Specific Modelling Languages using UML Profiles.* Technical Report. University of Antwerp.
[19] Mark Minas and Gerhard Viehstaedt. 1995. DiaGen: A generator for diagram editors providing direct manipulation and execution of diagrams. In *11th IEEE International Symposium on Visual Languages*. IEEE, 203–210.
[20] L. Montrieux, Y. Yu, and M. Wermelinger. 2013. Developing a domain-specific plug-in for a modelling platform: The good, the bad, the ugly. In *2013 3rd International Workshop on Developing Tools as Plug-Ins (TOPI)*. 1–6.
[21] A. Ribeiro, L. de Sousa, and A. R. da Silva. 2016. Comparative analysis of workbenches to support DSMLs: Discussion with non-trivial Model-Driven Development needs. In *2016 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*. 323–330.
[22] David Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. 2009. *EMF: Eclipse Modeling Framework 2.0* (2nd ed.). Addison-Wesley Professional.
[23] Eugene Syriani, Hans Vangheluwe, Raphael Mannadiar, Conner Hansen, Simon Van Mierlo, and Hüseyin Ergin. 2013. AToMPM: A Web-based Modeling Environment. In *Joint Proceedings of MODELS'13 Invited Talks, Demonstration Session, Poster Session, and ACM Student Research Competition co-located with MODELS 2013. (CEUR Workshop Proceedings)*, Vol. 1115. CEUR-WS.org, 21–25.
[24] Juha-Pekka Tolvanen. 2016. MetaEdit+ for collaborative language engineering and language use (tool demo). In *Proceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering*. ACM, 41–45.
[25] Jon Whittle, John Hutchinson, Mark Rouncefield, Håkan Burden, and Rogardt Heldal. 2013. *Industrial Adoption of Model-Driven Engineering: Are the Tools Really the Problem?* Springer Berlin Heidelberg, Berlin, Heidelberg, 1–17.