Reviewer: 1

This paper proposes an approach with its associated tool (Jorvik) to facilitating the creation of Papyrus-based UML profiles based on annotated Ecore metamodels. The main motivation is twofold: (i) facilitating the creation of UML profile, especially by automating repetitive tasks (the given example is the creation of OCL constraints to constrain the source and target ends of a stereotyped association), and (ii) facilitate the creation of Papyrus graphical editors. The approach consists of various steps where an annotated Ecore metamodel (using Emfatic) is first translated into a UML profile (after some validation rules are performed to assess that the Ecore metamodel is well-formed), then, various artifacts are produced to get "distributable" Papyrus graphical editors in order to graphically create models based on the profile definition. Annotations used in Emfatic allow for defining whether an Ecore EClass should be displayed as a Node or an Edge and whether an Ecore EReference should be displayed as an edge or not. Shapes can be associated with nodes while icons can be associated with Node and Edge to form the Papyrus graphical toolbar. For non-trivial transformations, the approach allows users to "polish" the transformations rules by adding their own using the EGL transformation engine from Epsilon. Finally, the tool also allows for transforming the produced UML models back to EMF models conform to the Ecore metamodel.

The approach and the associated tools have been tested on various DSLs. The running example is the Simple Development Process Language (SPDL). Other DSLs have been experienced, including the Archimate UML profile, and other, relatively smaller academic profiles. The authors also conducted an evaluation to assess: (i) the completeness of their implementation, and (ii) the productivity gain Jorvik and the proposed approach are expected to offer. This last point has been evaluated through a user experiment with two Ph.D. students with different levels of knowledge in modeling. The tool is available on Github (no license is provided for now, so I would not consider Jorvik being open source for the moment).

The paper is motivated and fits under the scope of SoSyM. The implementation seems interesting, although I could not get the chance to make it work, due to the absence of documentation. However, while I strongly agree with the authors that the process of creating UML profiles and their corresponding graphical editors in Papyrus clearly needs to be facilitated, I have several concerns regarding the proposed contribution and the structure of the paper.

In more detail:

- The paper is presented as an extension of the ECMFA 2018 paper that was presented by the same authors. The extension made to the ECMFA 2018 paper appears incremental to me. Three out of the four points in the introduction section are related to the implementation. The main contribution is the refactoring of Jorvik to be compliant with Papyrus 3.0. Presented as it is, I hardly consider it as a good increment for the ECMFA paper. What about Papyrus 4.0? 5.0? etc. What does not transpire in the paper is that before Papyrus 3.0, there was no standard for creating profile-based DSLs in Papyrus and their associated tools and the different features of Papyrus to customize the tool based on a profile definition was always unstable, deprecated, or buggy. What changed is that since Papyrus Oxygen, there is an effort

of standardizing how Papyrus can facilitate the specification of UML-based architecture description languages [1]. This is to my mind the motivation that should explain why an effort to refactor Jorvik to conform with Papyrus 3.0 is of interest. This aspect must be clearly better explained.

- Thank you very much for your comment. In the paper we stated that our refactoring work on Jorvik is applicable to “Papyrus 3.0 and above” – we changed this to “Papyrus 3.0+” to make this more obvious to the readers. We have added some content in the introduction section to explain why an effort to refactor Jorvik to conform to Papyrus 3.0+ is needed and why it is accounted as a major contribution to the journal paper.. However, for the overall idea of the paper, it is about demonstrating how the MDE paradigm helps improve the productivity through software engineering processes. Therefore, over stating the migration from Papyrus 3.0- to Papyrus 3.0+ may mislead the reader.

- Section 2.1 introduces one motivation behind Jorvik, that is: Jorvik could reduce the development cost of doing repetitive tasks when creating UML diagrams. The given example shows that Jorvik can help by automating the creation of OCL constraints that are used to constrain the way an association to which a stereotype A_to_B is applied must connect two nodes. The authors argue that OCL constraints must be created for each stereotype applied to the Connector meta-classes which is a repetitive task that is automated by the automatic transformation provided by Jorvik. The example is not convincing to me. Let's consider the following PlantUML-based profile (http://plantuml.com):

```
@startuml
class Model << Metaclass >>
class Association << Metaclass >>
class Class << Metaclass >>
class A << Stereotype >>
class B << Stereotype >>
class "Model" as Model_S << Stereotype >>
class A_to_B << Stereotype >>

Model_S --> Model
A --> Class
B --> Class
A_to_B --> Association
A_to_B ..> B : target >
A_to_B ..> A : source >
@enduml
```

Where two stereotypes A and B extend (denoted "-->" in the above code) the Class UML meta-element and one stereotype A_to_B that extends the Association meta-element. The particularity of this profile is that dependency relations (denoted "..>" in the above code) are added in order to define for each connector stereotype the source end and the target end (the same way it is done textually in Listing 1 on page 10 in the paper). The Papyrus editor supports the creation of such dependency

relations in a profile diagram. Then, a generic OCL constraint to check the association's end types and the navigability of the association (the two OCL constraints that are generated by Jorvik and shown in Listings 5 and 6 on pages 16--17 of the paper) can be written:

```
context Model inv:
let profile : Profile = self.getAllAppliedProfiles()->select(profile : Profile
| profile.name = 'ab')->asOrderedSet()->first(),
    dependencies : Set(Dependency) = profile.packagedElement->select(pe |
pe.oclIsTypeOf(Dependency)),
    association : Set(Association) = Association.allInstances() in

    association->forAll(association |
        let source : Class = association.ownedEnd->asOrderedSet()->first().type
          , target : Class =
association.memberEnd->excluding(association.ownedEnd)->asOrderedSet()->first()
.type in
        association.getAppliedStereotypes()->forAll(stereotype |
        let isSourceExist : Boolean = dependencies->exists(d |
(d.client->includes(stereotype))._and(d.name = 'source'))
          , isTargetExist : Boolean = dependencies->exists(d |
(d.client->includes(stereotype))._and(d.name = 'target')) in
          isSourceExist._and(isTargetExist) implies (
              let depS : Dependency = dependencies->select(d |
(d.client->includes(stereotype))._and(d.name = 'source'))->asOrderedSet()->first()
              , depT : Dependency = dependencies->select(d |
(d.client->includes(stereotype))._and(d.name = 'target'))->asOrderedSet()->first()
              , sourceS : Stereotype =
depS.supplier->asOrderedSet()->first().oclAsType(Stereotype)
              ,    targetS : Stereotype =
depT.supplier->asOrderedSet()->first().oclAsType(Stereotype) in
              target.isStereotypeApplied(targetS)._and(source.isStereotypeApplied(sour
ceS))
          )
    ))
```

And I am done. I do not need to generate multiple OCL constraints from the annotated Ecore meta-model. The OCL seems complex but it took me about 20 minutes to create it and is generic for all profiles (it could certainly be improved for performance). The above OCL constraint has been tested in the interactive Xtext OCL console for validating UML models created in Papyrus (I have not conducted stong experiments to assess performance bottleneck though). So it appears to me that the example that is given to motivate the need for Jorvik is not convincing.

- Thank you very much for your example. We appreciate the detailed example provided to support your argument, however, we would like to point out that the UML profile provided in the example may not be well formed. Introducing dependencies between an Association and its connecting Classes is not typical in UML profiling. Thus,

by creating the UML profile in your way, some people may argue that the produced UML profile is incorrect. Therefore, the OCL constraint you provided may not apply.

- A major issue is that Jorvik appears to be really limited. It does not support user OCL constraint definitions (besides the two OCL constraints for the navigability and the end types of associations that are automatically generated from the EGL scripts), composite shapes, bi-directional associations, to name a few.
- Thank you for your comment, we would like point out that the users can define their own OCL constraint in the generated UML profile if they like (in Papyrus' UML Profile editor). We provide the OCL constraints for navigability and end types just to demonstrate what can be achieved. In fact, supporting user-defined OCL in EMF annotation is rather easy for us to implement. However, the users cannot test the OCL in the EMF annotation, because there is no static analysis support in EMF editors (i.e. Emfatic) for OCL. Thus, at the current state, it is easier for the users to define their own OCL constraints in the UML profile where auto correct and static code checkers are available.

We also would like to invite you to consider the fact that Jorvik, in its current shape, is not a complete product, there are a lot of space for new features to be developed (which may include user-defined OCL in annotations of the EMF metamodel). In addition, we would like to get more feedback for Jorvik before we plan to add new features to it. This paper is essentially an evaluation of the MDE paradigm in the sense that it can be used to automate labour intensive and error-prone processes, i.e.. UML profiling and creating its supporting editor.

Besides, it only supports SVG-based nodes and simple edges with basic styling properties. Most of the features available in industrially approved tools to build graphical editors (e.g., Sirius) are missing (e.g., conditional styles, positioning constraints, partition shapes, and so on). It left a bitter taste in my mouth where Jorvik appears to me as limited as the old GMF tool framework that was enough to create simple "nodes and edges", but where the generated code must be edited to support advanced diagramming features (including composite shapes). Here, customizing the standard code generation seems even more painful since the user has to write complex "polishing" transformation rules to support those advanced features. The entire paper misses proper positioning with respect to about 10 years of experience that were capitalized in tools such as Sirius to build usable graphical editors.
- Thank you, we would like point out that we did not build Jorvik from scratch so that it generates editors that create simple "nodes and edges". Please consider the fact that Jorvik is an extension to Papyrus (provided that Papyrus already provides graphical modelling editors for UML). Therefore Jorvik inherits the limitations of Papyrus. Such limitations include the restrictions on what users can change of the visual representation, as the graphical elements must "play nice" with the underlying Papyrus implementation.

For example, a Papyrus profile editor can provide custom shapes for nodes, but these shapes are added by Papyrus to an edit part that is not used for containment. Thus, Jorvik is constrained by the limitations that Papyrus places on editors. If we were to represent containment via visual nesting, the contained element would not be placed inside the custom shape. In other words, Papyrus does not let us modify the edit part for it to handle containment (unless we change Papyrus' implementation in its source code). This is explained in section 4.12.

As we mentioned in the paper, Papyrus is the leading open source tool for UML modelling, which also enables its users to define UML profiles and create editors for their defined profiles. Our motivation is to demonstrate that MDE can be used to automate the creation of UML profiles and their supporting editors, which is useful if developers seek open source solutions for UML profiling. Please bear in mind that this paper is not to sell Jorvik as a product. In Jorvik we provide, as much as possible, facilities that help users of Jorvik towards their goal, there is still vast potentials for new features.

- A second major issue I have is related to the proposed process. The proposed process forces the user to start with an annotated Ecore metamodel to generate the UML profile. What if the UML profile already exists? Could the tool start from the UML profile to generate "distributable" Papyrus graphical editors?
- Thank you for your comment, this is a very good point. We would like to add support to generate UML profile editors from UML profiles. Right now we provide a work-around solution – an UML profile can be used to extract an annotated Ecore file, which can be used to generate the UML profile editor for that profile. We list in the future work that we would support the generation of editors from UML profiles.

Besides, the approach seems counter-productive for me. Let's consider a UML profile that extends the UML state machines to, e.g., annotate transitions and states with temporal constraint information (e.g., the system described by the state machines should never stay in a state more than 10 seconds). Following the author's approach, I should define the following metamodel:

```
@Node(base="State", shape="..." icon="...")
class State {
    attr String name;
    attr String temporalConstraint;
}

@Edge(base="Transition", source="src", target="tar" shape="..." icon="...")
class Transition {
    attr String temporalConstraint;
    ref State[1] src;
    ref State[1] tar;
}
```

Two problems occur:

a) how does "src" and "tar" map to the original "source" and "target" attributes of the Transition UML meta-element? The same problem appears in Listing 1 on page 10 of the paper where the two attributes "src" and "tar" of the Role association must conform somehow to the ownedEnd and memberEnd attributes of the Association UML meta-element. Nothing is said on this point.
b) Assuming the above metamodel to create state machines and assuming a model created by the Papyrus graphical editors generated by Jorvik. The concept of State machines or regions are not represented in the annotated Ecore meta-model, but they exist in the UML meta-model. Therefore, this meta-model should be enough to create a UML model consisting of a state machine, a region, and two states connected through a transition (where the transitions and states are "stereotyped" with the above definitions). The problem appears when I want to transform back my UML model into its corresponding EMF representation conform to the above metamodel. Since there is no notion of state machines nor region in the Ecore metamodel, the transformation would fail. Therefore, two cases can occur: (i) the generated graphical editors are limited to the concepts present in the annotated Ecore metamodel, and in that case, we lose the benefits of using UML profiles (which consists in reusing all the UML meta-elements), or (ii) all the UML meta-elements must be modeled in the annotated Ecore metamodel, which is again counter-productive (one needs to have a complete understanding of the UML meta-model and "replicate" the UML meta-elements onto the Ecore meta-model). In both cases, we lose the benefits of using UML profiles, therefore, the proposed process is not convincing to me. The contrary (starting with a UML profile annotated with diagram annotations and generating the corresponding Ecore metamodel to allow UML models to be transformed into a corresponding EMF representation) appears to be much more interesting.
- Thank you for your comment, this is a very accurate observation. Jorvik as it currently stands only supports the following edge types: Association, Dependency, Generalization, InformationFlow, Usage and Realization (we have added the description of this in the paper). Among these types, Association extends UML::Relationship, and others all inherits UML::DirectedRelationship. We used an ad-hoc approach to use “source” and “target” to denote the direction of the edge (which is natural to DirectedRelationships), this information is used to generate the OCL constraints. We did not use “ownedEnd” and “memberEnd” for Association, in the sense that we think it may confuse EMF users in the first instance. We can definitely change our keywords in this aspect if deemed necessary.

Nonetheless, in the example, Transition extends RedefinableElement, which in turn extends NamedElement. Current keywords are not applicable to Transition and we will add this in the future work. We thank you for your observation, please consider the fact that the Jorvik is under development and is not a complete product yet.

- Another major issue is related to the user experiment that was conducted. The authors argue that Jorvik improves productivity gain by generating most of the artifacts needed by Papyrus to create graphical editors. But the comparison does not

seem fair for me. First, the authors compare the number of Lines of Code (LOC) hand-written of the annotated Ecore metamodel used by Jorvik with the number of LOC of the UML profile created by Papyrus. However, the second one is graphically created so the code is not hand-written. Besides, Papyrus relies on an XML serialization which explains why the code is much larger than the code of the annotated EMF metamodel written using Emfatic which proposes a lighter, human-readable notation. It is like comparing apples and oranges. Second, the evaluation does not consider any "polishing" transformation that should be created to overcome the limitations of Jorvik in order to get the same result in both approaches. I am pretty sure considering them would balance the evaluation in favor of Papyrus.
- Thank you for your comment, we realised the unfairness to compare LoC for XMI based models, therefore we included the counts of model elements required next to the LoC. May be our description is not abundantly clear. We have re-written in the section to make the numbers clear. In this section, before we made any changes, there are a couple of sentence to describe how much more effort is needed to write polishing transformations in order to make re-produce Archimate. We have re-written the paragraph with regard to this, too.

With Jorvik and polishing transformation, we conclude we create 63.5% less objects and 63.7% less CSS code than creating the editors manually.

- The experiment section introduces the concept of "Default" and "Essential" knowledge. It is said that the Default knowledge is ground knowledge while Essential knowledge is "key" knowledge which is not easily accessible online. The difference is rather confusing and only burdens the reading of the evaluation. Besides, there is no example of Default and Essential knowledge for the different tasks provided in Table 5. The authors should either remove this distinction from the paper or clearly explain for each task of Table 5 the two distinct pieces of information they provided to the users.
- Thank you. It was our mistake that we did not point to an example of the default knowledge and the essential knowledge. In principle, the default knowledge is, to our best knowledge, what is accessible online towards completing the tasks we design. The essential knowledge is the knowledge that we collected and is sufficient to guide the participants to complete the tasks. We have added references to examples for default knowledge and essential knowledge. We hope that with the examples, the difference between the default knowledge and the essential knowledge is clear.

- Section 6 (Evaluation) provides an unfair comparison between the different pieces of work in the literature [2,13,14,27,28] and Jorvik. The criticisms about the different work are also true about Jorvik. Especially: (i) Jorvik also involves non-trivial human-driven tasks for creating the polished transformation rules (for which no evaluation has been made to show the complexity of creating these rules) that are required for building "real" graphical editors, and (ii) Jorvik has limited capabilities (no user-defined OCL constraints, no composite shapes, etc.). Second, the entire section 6.1 (UML Profiles) is not related work but rather describes the context of the paper. It

should be merged with Section 2.1. Third, a related work on tools (e.g., Sirius) for building graphical editor based on DSL definitions is completely missing.

- Thanks for your comment. The comparison does not necessarily mean criticism. We appreciate the work mentioned in the section, we are simply pointing out that we are creating an approach to generate not only UML profiles, but also their supporting editors (with their OCL constraints as well) from a light weight metamodeling approach. Some experienced modeller would argue that using Emfatic (text editor for EMF metamodeling, as shown in Listing 1 is rather more efficient than creating an EMF metamodel from either a tree editor or a graphical editor. On the other hand, to create UML profiles, we typically need to import meta elements from UML, create Stereotypes, map the Stereotypes to their corresponding UML elements and creating references among the Stereotypes for DSLs, in addition, we may also want to add OCL constraints to the UML profiling. The workload for the above scenario can be reduced by Jorvik. With regard to user-defined OCL constraints, as we have commented before, that 1) the user can add OCL constraints to their generated UML profile and 2) Jorvik is not a complete product. With regard to composite shapes, we have explained in the paper that Jorvik inherits limitations of Papyrus, and there is nothing we could do to overcome such limitations (apart from changing Papyrus' implementation in the source code level).

We respectly think that the related work section 6.1 is necessary, should the reader require more context in UML profiling. We added a couple of sentences to the introduction to emphasis the importance of UML profiling. However, this paper is about using MDE to automate the process of creating UML profiles and their supporting editors. Therefore, we think that moving the entire section 6.1 to the introduction is inappropriate and may mislead the readers.

With regard to Sirius, we think there are clear differences in what we are doing compared to Sirius. For Sirius, the users create graphical modelling facilities based on readily created DSL (EMF based) using tree editors. For Jorvik, we generate UML profiles and its supporting editors. We Sirius and Jorvik have different focus. We will include Sirius in related work.

- The section describing the implementation is unnecessarily long and complex. Around 10 pages are dedicated to the details of the implementation. In comparison, Section 3 (which is supposed to be the core of the paper) is only 5-pages long. Details about artifacts that must be generated for Eclipse and Papyrus to work (manifest files, plugin.xml, the locations of the SVG files, and so on) are superfluous and burden the reading of the paper. This section must be considerably reduced.
- Thanks for your comment, we understand what you are coming from. We included as much details as possible (with as little redundancy as possible) in the implementation section, for 1) there is no readily available documentations from Papyrus to guide the users to create Papyrus editors that support their UML profiles, the implementation

section provides a complete story on what the users should do if they wish to create their own editors. And this paper may be their only reference if they choose to develop their own editors. And 2) the implementation section provides sufficient (but not excessive) details to give enough context when we discuss our findings in our experiment (where the readers need to refer to each implementation steps in Section 4 in order to grasp what is being done). We have accepted your opinion on reducing the section and we have tried to reduce this section as much as we can.

As a conclusion, I think the paper needs major changes to be accepted.

Smaller issues below:

=== Section 2.2 ===

- The meaning of "distributable" is missing - addressed

- Section 2.2 puts a strong emphasis on the palette definition while, by experience, the most time-consuming task is on the graphical representation (via the unclear "ElementTypeConfigurations" element). The "ElementTypeConfigurations" model is mentioned but its role is not explained; - among the mentioned 3 approaches for defining creation facilities (i.e. creation tools in the palette to create UML instances which apply corresponding Stereotypes in the UML profile automatically), only the third option (creating a Papyrus editor, which includes creating architecture model, palette configuration model, elementTypesConfiguration model, etc) involves the creation of the ElementTypeConfigurations. We have added a couple of sentence to make this clear. We have also explained what ElementTypesConfiguration models are for.

=== Section 3 ===

- The motivation behind the backward transformation from a UML model to an EMF one is missing. Why do we need this feature? – this is a feature to support the change propagation from the UML profile back to the EMF metamodel, we have added a sentence to clarify this.
- Listing 1: the @Edge annotation applied to EMF classes seem to work only when the EMF class has two attributes with a multiplicity set to 1. If this is the case, it should be stated in the paper – this is not the case, we have added an explanation to the paper.
- Fig. 3 is never referenced. – It is mentioned in the paragraph right above the figure.

=== Section 4.2 ===

- The footnote 6 seems to me to be an important limitation of the tool that could be easily checked by Jorvik. – this is not the limitation of Jorvik, Jorvik has no influence in what is being typed in to the Emfatic editor (the Emfatic static analysis tool). Hence, we need to implement EVL constraints to do this check before the transformation to UML profile happens.

- "For each reference (ref or val) in the metamodel [...]" <- should not it be "attr" instead of "val"? – according to Emfatic, 'ref' and 'val' are equivalent to EReference, where 'attr' is equivalent to 'EAttribute', in here we mean EReference.

=== Section 4.3 ===

- Figure 5: how is the "isRequired" attribute handled? – Papyrus does not expose isRequired attribute to the users, hence it is not discussed.
- Subsections 4.3.1 and 4.3.2 are unnecessarily long. The OCL snippets could be included directly in the bullet list just before Section 4.3.1 – We feel there is a need to describe the OCL as UML metamodel is rather complicated, with these examples, the users may derive their own OCL constraints
- "one of the elements a "familiarWith" association connects to, has [...]" <- comma - addressed
- "[...] while the other has the "Tool" stereotypes applied to it" <- stereotype - addressed
- Subsection 4.3.1 has malformed sentences - addressed

=== Section 4.4 ===

- Figures and examples are needed to understand this section – we have explained the purpose of elementTypesConfiguration, and provided an example in the original text. We added a footnote to refer the readers to the official Papyrus document which explains the ElementTypesConfiguration framework in depth.

=== Section 4.5 ===

- "Finally, each ToolConfiguration, [...]" <- comma - addressed

=== Section 4.7 ===

- "the standard UML profile and the user-defined UML profile need to be applied in order to initialize the diagram" <- which diagram? – we have added an description to state that the diagram is the UML diagram with the user-defined UML profile applied to it
- How is the model initialized using Jorvik? – the model is initialised the same way that Papyrus does, which is not in the scope of the discussion in this paper.

=== Section 4.8 ===

- This section contains information about the architecture model which drove the way the entire Papyrs 3.0+ works for creating profile-based graphical editors. To me, this is clearly what should motivate the need to refactor Jorvik and should be discussed directly in the introduction. – we have addressed this in the introduction

=== Section 4.9 ===

- This entire section is unnecessary and should be removed. – with respects, we think differently. The content in this section is based on our own discovery in implementing Papyrus editors that support UML profiles, the information is not available online. Hence, this may be the only reference, for people who wish to create Papyrus editor with their UML profiles. We have shorten this section nonetheless.

=== Section 4.10 ===

- The feature presented in this section is not motivated. – we added a description for this section
- The discussion about the "::=" syntax of the ETL engine is unnecessarily long – the discussion has been reduced
- "denoted by ct" <- should not it be "t" instead of "ct"? – ct and sc are notations to denote intermediate transformation results during the resolution operation triggered by the "::=" notation.
- "denoted by sc" <- should not it be "s" instead of "sc"? – same as above

=== Section 4.11 ===

- Vertical space is required before the title of Section 4.11 - addressed
- Table 2: it is unclear why this table starts with id #2. I suppose that the first transformation rule is the "Ecore 2 UML Profile Transformation (M2M)" presented in Figure 4 on page 12, but I am not sure about it. – the ids should be #2b - #5b (shown in figure 4), this is addressed now
- Also, after a first reading, it was unclear to me that transformations #2--4 are M2M transformation rules (using the Epsilon Transformation Language) while transformation #5 is model-to-text (using the Epsilon Generation Language), which explains the difference between the transformation rules' extensions in Table 2. I had to look deeper into the previous pages (especially Figure 4 on page 12) to find this information. – thanks, we have added a description to state the types of the transformations
- I think Table 2 with the required files (with their extensions) adds unnecessary details that confuse the reader more than they help him/her to understand. The only useful information is that polishing transformations refine the model rather than overwriting the original transformation rules. Examples of polishing transformation rules would be more of interest. The only example given is illustrated in Listing 2 (transformation #5). The motivation behind, e.g., transformation #1 (probably the most interesting since it changes the way an annotated Ecore metamodel is transformed into a UML profile) or transformation #3 is unclear to me. Can I write a polishing transformation rule #3 to, e.g., overcome the limitation of nested relations as discussed in Section 4.12? – we have stated that in order for the polishing transformation to be executed (atop of the default transformation), the users of Jorvik need to create transformations that match the names listed in Table 2 in order for Jorvik to pick them up. The limitation mentioned in Section 4.12 is a limitation by the implementation of Papyrus, therefore there is nothing that Jorvik can do to overcome

this limitation.

=== Section 5.3 ===

- "In this experiment, we compare the time needed to develop an editor using Papyrus [...]" <- the time unit is missing. It is only given (implicitly) in the enumeration on page 28 (minutes). It is confusing since, on page 27, the time spent for creating the initial example editor is given in months. – To make the units clear, we have now added unites in the captions of the tables when times are presented.
- Simple past, present, and present perfect tenses are mixed in the first paragraph of Section 5.3 - addressed
- "[...] again the Papyrus apporach" <- approach - addressed
- "for both the case" <- for both cases - addressed
- Task 8 (OCL Constraints): 2 weeks for experienced OCL experts for creating the OCL constraints described in Section 4.3 is overestimated. Besides, there are only three references annotated with @Edge(base="Association") in the Website metamodel and one class annotated with @Edge(base="Association") in the FTA metamodel. Therefore, creating 4 times 2 OCL constraints does not take 2 weeks. – this time is estimated based on the fact we needed to study the complex UML metamodel in order to write the OCL constraints, it actually did take us 2 weeks to complete the OCL constraints – and we consider ourselves rather familiar with UML and OCL.
- "[...] one participant highlighted that felt completely lost before receiving the Essential information" <- that s(h)e felt. Which participant (#1 or #2) should be said to be consistent with the rest of the evaluation section. - addressed
- Section 5.3.2: cannot be an enumerated list - addressed
- "Finally, both mentioned that the time give was enough" <- given - addressed
- It is said on page 33 that "the second participant had no EMF experience in the past and had never created an EMF metamodel" while it is said on page 34 that "Participant #2 only used Ecore occasionally". This is quite confusing. - addressed

=== Section 6 (Related work) ===

- Section 6.2: the sentence starting with "The methodology requires the manual definition [...]" is too long and should be split. - addressed
- "Another relevant research work is JUMP [2] that support the automatic generate profile" <- supports; generation - addressed
- "Also, the transformation of models from UML [...] a distributable custom graphical editor as en Eclipse plugin" <- as an Eclipse plugin? - addressed

[1] https://wiki.eclipse.org/Papyrus/Oxygen_Work_Description/NewFeature/PapyrusAFViewpointSwitch

Reviewer: 2

Public Comments (these will be made available to the author)
SUMMARY

The paper presents an approach for generating Papyrus editors that are capable of handling profiled UML models. Instead of the built-in Papyrus approach, in which such editors are generated from a set of interrelated artifacts, the paper proposes to define UML profiles using a single artifact, namely annotated Ecore models. The approach has been implemented in a tool called Jorvik, which includes a model transformation pipeline to generate the Papyrus profile definition artifacts from such an annotated Ecore model. The feasibility of the approach is demonstrated using examples, and its benefits over the traditional Papyrus workflow are evaluated in a user study.

- This precisely summarises the work described in the paper.

EVALUATION

The paper is an extension of the author's ECMFA'18 paper on the same topic. Novelties are improved tooling facilities and the empirical evaluation including user experiments. The latter one is a significant contribution from a scientific point of view, which justifies the paper for publication in SoSyM.

I generally acknowledge the author's effort to implement the Jorvik tool suite. In particular, I like the consequent usage of MDE technologies in the implementation which, from a broader perspective, is a general form of validation of the MDE paradigm and its supporting tools/techniques. The evaluation is sound and, to me, it is convincing. Threats to validity are discussed.

However, I also have a couple of issues, particularly with the technical parts, which the authors should consider to further improve the paper.

From a high-level perspective, I am missing a clear differentiation of whether the problem addressed in the paper is just a tooling problem related to Papyrus, or whether it is a general problem that concerns the specification of the UML profile mechanism as an OMG standard. Taking a software engineer's view of object-oriented analysis, design and implementation, the OMG standards provide object-oriented analysis/design (meta-)models which, as a matter of fact, need to be refined and translated to proper implementations by tool vendors. For example, the work presented in [B] stresses the fact that there is not only one meta-model for a given modeling language, but that there are several meta-models in different phases of object-oriented analysis, design and implementation. The following additional examples from the literature are meant to be positive examples that clearly differentiate between the above mentioned aspects:
* The work presented in [A] clarifies the meaning of subset and union properties that have been introduced with MOF 2.0. This is an example that identifies problems directly within the OMG standards which are solved here through additional clarifications. These clarifications, in turn, will help the tool vendors to come up with behavioral consistent implementations of subset and union properties handling.

* Another work which addresses design flaws within the UML Superstructure Specification has been presented in [D]. It discusses the implications of these flaws on certain model management tasks. In other words, some of the model management problems result from an inappropriate design of the UML metamodel and can be solved by switching to a more appropriate metamodel (or refinement of the UML metamodel).

- We agree with the comment of the reviewer on that aspect. Indeed, OMG proposes standards that need to be translated in implementations by tool vendors. In addition, tool vendors are able to use the specific metamodels that fits for the purpose of their tools. Papyrus follows the OMG standard when it comes to the definition of UML profiles which, according to our opinion, fits for purpose. However, this UML profile is a single artefact required by the tool to have a working graphical editor for it. The problem we are addressing with this work is the fact that Papyrus also asks from the developers to define a number of other interconnected artefacts, on top of the UML profile, which include repetitive tasks, require significant effort and knowledge on the internals of the tool. These artefacts are not related to the UML standard and guidelines proposed by OMG. This is something verified by our evaluation as well, as in all the cases, participants were able to define the UML profile correctly (or in one case partially correctly) within the time limits using Papyrus. However, they mostly failed in the creation of all the other Papyrus related artefacts. As Papyrus is the de-facto open-source tool for defining UML profiles and editors, we believe that making it easier for developers to make use of its functionalities, is desirable. We agree that this should be better clarified in the paper thus we added the following lines in Section 2:

"As mentioned in [B], it could be the case that in different phases of the system enginering process, different metamodels of the same modelling language could be used based on the current phase's needs. Papyrus opted for the option that allows the definition of UML profiles based on a metamodel that follows the specifications defined by the OMG standard. As a result, it might be the case that problems that arise when using tools that implement such metamodels are due to the use of an inappropriate version of the metamodel by the tool vendors. Examples of solutions in tackling such problems that are a result of potential design flaws in OMG specifications are presented in [A] and [D]. However, our personal experience, which is verified by the evaluation results, shows that the problems with the creation of UML profiles and graphical editors in Papyrus, arise by the labour-intensive and repetitive tasks related mostly in the definition of all the other artefacts (which are a result of a design decision by the tool vendor) rather than the UML profile itself. We argue that this labour-intensive, repetitive and error-prone tasks could be automated."

The lack of such a differentiation in the paper at hand mostly pertains the general motivation and the background section 2. However, readers also may get lost in some of the technical discussions in sections 3 and 4:

This applies in particular to Section 4.3.2 (navigability). The sentence "We need to highlight, that currently, opposite references are not supported" really puzzles me. I think we are talking about UML Associations here, which are conceptual model elements that can be navigable in one direction, both directions or not navigable at all. On the contrary, opposite references are just a low-level design decision in EMF, serving as a workaround to express bidirectional relationships (which are not natively supported by EMF references).

- We understand the confusion related to this sentence. Let's assume that a user has in the annotated ECore metamodel a reference going from Class A to Class B named A2B and another reference from Class B to Class A named B2A. Jorvik will generate 2 tools in the palette under the edges compartment named A2B and B2A. If in the annotated ECore metamodel, the users have used the opposite mechanism between these specific references of A and B (i.e., A2B is the opposite of B2A) then the generated editor should produce just one edge tool in the compartment which would allow the creation of a bi-directional association between A and B automatically. This is not supported right now and, unfortunately, even if users have opposite references set in the annotated ECore metamodel, two edge tools are generated.

I also wonder about the encoding of navigability in Listing 6: I did not know that Properties serving as member ends provide a convenience function isNavigable(). More generally, this brings me to the point that some of the details are hard to understand without having the UML metamodel in mind. I am not asking for a general introduction into UML, since this can be expected from the SoSyM readers. However, code snippets like the one in Listing 6 would be much easier to understand if the relevant excerpt of the UML metamodel is shown along with the listing. The same applies to code snippets which rely on an in-depth knowledge of the Ecore metamodel.

- We agree that some of the concepts used in the code snippets require deep knowledge of the UML and Ecore metamodels. We believe that the better approach to this would be to include a reference to the appropriate parts of the UML metamodel (not the whole standard but the specific pages/diagrams) as a footnote so readers who are not familiar can quickly navigate to. We also added as a footnote a reference to the Ecore metamodel which is relatively small in size.

I also have some problems with Listing 7. If I got it correctly, in your approach, a profiled class (here: Person) is represented as a single object in the abstract syntax of your profiled model, right? This is different from the OMG standard, where a conceptual element in a profiled model (like Person) would be represented by two objects, the instance of UML::Class and the instance of the stereotype (Person) attached to that class. There is nothing to say against your design decision to render sterotyped elements as a single object only, but it has some implications. For example, according to the OMG standard, a profile can be applied to a model which is an instance of the base meta-model and later be revoked. This is not possible in

your approach. Likewise, a base element can have multiple attached stereotypes, which also seems to be impossible with your approach. If I am not completely wrong here, I would expect this to be discussed in the paper.

- There is a misunderstanding here. In our approach, when a new element is drag-and-dropped from the palette (e.g., Person) is of type UML::Class and has the stereotype (e.g., <<Person>>) attached to this class. In other words, the role of the palette is to create the UML class and attach the stereotype that belongs to this node/edge tool to the UML class automatically. Thus, it is done in the way OMG standard describes and actually is the one Papyrus supports. As a result, in fact users are able at any point to attach more custom stereotypes if they like to any element appearing on the canvas and also remove the stereotype that the palette tool has automatically assigned (e.g., <<Person>>).

We believe that the misunderstanding is due to the code written in line 2 of Listing 7 where the transformation source element is of type "Person". This is a handy feature of the Epsilon driver that parses UML models. Stereotypes are treated also as types so this will pick all the elements that have the stereotype <<Person>> attached to them. This does not mean though that the element's type is Person. If some change this line to transform all UML::Classes which have the stereotype <<Person>> attached to them, it will produce the same results. Also, in general if someone queries the profiled model to get all the elements of type UML::Class, she will get back all the nodes on the canvas, no matter which stereotype(s) have assigned on them.

We hope that this solves the misunderstanding.

As for broadening the scope of related work: You might want to mention that editors are only one particular aspect of an MDE environment supporting UML profiles. There are other building blocks such as model transformators, validators, code generators, diff/merge tools and other model management tools that need to work with profiled UML models. For example, a work which addresses the generation of basic change operations over profiled UML models has been presented in [C]. Although generating a different kind of artifact, the approach has in common with yours that these artifacts are generated from a conceptually unified representation of the metamodel and profile definition, and it also has to address issues such as preventing manual adaptations when re-generating the artifacts (e.g. in response to profile evolution).

- Thank you, we have added a short sub section in related work to mention these.

Minors:

run against model conforming to - addressed
=> models

Line 19 & 22 - addressed
=> Line 19 & 22:

NB Line 8: - addressed
=> NB ?

by the polishing transformation is shown in - addressed
=> which is shown

while tests the rule only - addressed
=> while it tests

We do not considered this as a - addressed
=> consider

we performed manual review of the models/artefacts - addressed
=> we performed a manual review

Another relevant research work is JUMP [2] that support - addressed
=> that supports

editor as en Eclipse plugin. => as an - addressed

[A] Alanen, M., & Porres, I. (2005). Subset and union properties in modeling languages. Technical Report 731, TUCS.
[B] Kehrer, T., & Kelter, U. (2014). Versioning of ordered model element sets. Softwaretechnik-Trends, 34(2).
[C] Kehrer, T., Rindt, M., Pietsch, P., & Kelter, U. (2013, October). Generating Edit Operations for Profiled UML Models. In ME@ MoDELS (pp. 30-39).
[D] Kelter, Udo, and Maik Schmidt. "Comparing state machines." Proceedings of the 2008 international workshop on Comparison and versioning of software models. ACM, 2008.

Reviewer: 3

Public Comments (these will be made available to the author)
This paper is based on a previous publication [ECMFA18] where a tool for automating the generation of UML profile graphical editors for Papyrus is presented. Papyrus is one of the most used UML-based tool also supporting the UML profiles definition. Papyrus also offers the possibility to define profile specific graphical editors but the process often results difficult, time-consuming, error-prone and the learning curve can be very substantial. For this reason, the authors proposed this tool, called Jorvik, enabling the automatic generation of all the artefacts needed based on Ecore annotated metamodel. The annotated meta models are automatically transformed into UML profiles and the process also generates all the artifacts needed to run the graphical editor. The tool is evaluated on a case study

named Archimate, and they evaluated the approach also for the completeness of supporting various domains and metamodels.

- This summarises precisely the work presented in the paper.

Compared with the conference version, all the sections have been extended in length and content, the validation has been deeply improved and a complete brand new experiment is part of it. The tool has been adapted to support Papyrus 3.0 (from 2.0) and a new section to explain a validation script supported now in the tool has been added. Also the CSS styling has been improved and demonstrated for making the editor more customizable.

The paper is generally well written, it is understandable in the way demonstrates the tool applied to examples. The contribution section is explained in all the details, without omitting a single transformation or code generator included in the tool. Sometimes the technical details are too much and it could be omitted without affecting the general flow of the paper. The evaluation offers different experiments, but I think the main positive point is that at the end the tool reduced by 90% the handwritten code compared to the normal process.

- We would like to thank the reviewer for these kind words.

I have an observation that I would suggest to address: the annotations look similar to the ones used in EUGENIA for automating the GMF tooling generation. If the annotations used are the same, or an extension or the mechanism resembles the one used for EUGENIA, this should be stated clearly, with citation and explanation.

- Indeed, the annotation and the inspiration for this work come directly from EUGENIA. In fact, in our initial prototype we used different annotations but we decided that we should change them to fit those used in EUGENIA to by as consistent as possible, as these two tools are performing similar operations but in a different domain. We agree that this should be stated clearly in the paper, thus we added the following in Section 3.1 in the paper.

"Due to the relevance of our work with Eugenia [28], a tool that uses the same principles for the generation of GMF editors and inspired our work, and to be as consistent as possible, the syntax of our annotations match those in Eugenia, where possible."

In some of the sentences in the contribution section, the explanations do not report that the screenshots or the reported listings are results of the application for the running example, for instance:
- last sentence before sec. 3.2 "…the produced papyrus editor is presented in Fig.3"—> the graphical representation of the model refers to the case study, and the generated editor, the palette, and everything supporting the modeler to use it.
- Also when listing 3 is mentioned I would explicitly specify that the generated CSS is

relative to the application example.
- Same for the last sentence in sec. 4.11

- We agree with the observation for the first two points. We made the following changes in the Section 3.2.

Section 3.2: “The automated M2M and M2T transformations are then executed on the Ecore file of the running example presented in Listing 1. The produced SDPL Papyrus editor is shown in Figure 3.”

Listing 3: “The resulted output for the running example that is amended automatically in the original CSS file (by the polishing transformation) is shown in Listing 3.”

Regarding the third point on the last sentence of Section 4.11, in fact these names are not specific for the running example, but these exact names must be used by developers of polishing transformation irrespective of the domain. We added the following in Section 4.2 to clarify this.

“Polishing transformation developers have to use these specific filenames, in any case where Jorvik is used and polishing transformations are needed, in order for those to be picked by Jorvik and executed. Each transformation that has a different filename than the one shown in Table 2 (or it is not present at all as each polishing transformation is optional), then Jorvik will not be able to find it and execute it.”

The transformation workflow reported in Fig.4 presents horizontal lines considered as input/output but I would include the vertical lines since some of the transformations need to be executed before or after the others. For instance, the Architecture Model Generation needs a lot of previously explained artifacts. This is again evident in sec.4.9.

- The reviewer is correct in pointing that there are actually some dependencies in the process. In fact, one step (that of architecture model generation) should be executed after the creation of two models: the element types configuration (step 3) and the palette (step 4). Our wording in Section 4.8 is misleading. We mistakenly write that the architecture model needs as inputs the 6 different artefacts. In fact, it only takes as input the 4 four of them (i.e., the annotated ECore, the palette, the element types configuration and the UML Metamodel which ships with Papyrus and is not generated by our tool). It also refers to four of them (i.e., the palette, the element types configuration, the creation command class and the CSS) as described at the last list point of this subsection. Any other transformation described requires as input only the annotated ECore metamodel given as input (and some of them the ECore meta-metamodel and the UML ECore metamodel which ship with Papyrus). We’ve replaced the misleading text in Secion 4.8 with the following. As we believe that Figure 4 is

already a little bit crowded we decided to explicitly mention in the text instead that this is the only execution order dependency.

“The program needs as input 1) the annotated Ecore metamodel for the DSL, 2) the Element Types Configuration model and 3) the Palette Configuration model. This makes this transformation the only one of those listed in Steps #1-#9 in Figure 4 that has a dependency on other transformations (i.e., Steps #3 and #4). All the other transformations, take as input the annotated Ecore metamodel and some of them other metamodels the ship with Papyrus (e.g., UML Ecore metamodel, etc.)”

We also replaced the last list point text with the following.

“The ArchitectureDescriptionLanguage points to the Element Types Configuration and the Creation Command class. The PapyrusDiagrams point to the Palette Configuration model and the CSS file.”

Finally, we need to mention that the names of all the artefacts generated are known in advance: this is because they either have default names which are hardcoded in the generator (e.g., diagramshapes.elementtypesconfigurations) or are constructed dynamically based on the name of the root package in the annotated ECore (e.g., process.architecture in the case of the SPDL example shown in the paper where the root package name is “Process”. This way although some generated artefacts point/refer to other generated artefacts, there is no need to run in a specific order (except the architecture one as explained above), as we now in advance the names of the models to which they refer to.

When the authors say that “it is user responsibility to avoid names collision” I was wondering whether this can be formalised an additional validation rule.

- Yes, it is possible to have this checked by an additional validation rule. Thank you for this idea. We have implemented the rule in EVL. The rule is also added in Table 1 (#11) and the footnote pointing that it is users’ responsibility to check for name classes is removed.

In section 4.10 at a certain point ct and sc are used in the text but I had difficulty in finding the meaning of those variables.

- We have in the text the following sentence: “the expression “s.familiarWith" returns a collection of UMLProcess!Tools (denoted by ct).” What we would like to say is that from now on when we refer to “ct” we mean that it is the collection of UMLProcess!Tools returned from the expression s.familiarWith. The same for “sc”; it is the name that we will use to refer to the sub-collections containing the transformed elements from the process described in the text. We use this convention, as we need to refer to the collection and sub-collections later in the paragraph and it would make the text difficult

to follow if we instead used the complete description of what these collection/sub-collections are. We hope this clarifies the issue.

In sec.4.12 the authors stated that the indentation of the elements in the case of containment is not possible and it is listed as a limitation: I was wondering if this can be addressed with additional CSS rules generation.

- We are afraid that this is not possible. This issue is related to the features of the elements of the visual editor that Papyrus allows custom editors to change. To the best of our knowledge, Papyrus CSS does not offer any rule to override this.

All the artifacts are stored in the linked repository, but I couldn't find a link for metamodels tested in section 5.2. It would be nice to have in the git repository a screenshot of the generated tool with an example model reported in the diagram, e.g., the example diagram for Wordpress metamodel, etc.
The experiment reports the result in table 5 and the unit of measure should be reported in the text explaining the table, e.g., Total (m).

- We added the following in section 5.3.1 to address this: “Table 5 lists an overview of the tasks. It also includes the times (in minutes) for the total time given (i.e., Total (m)) and the deadlines (in minutes) for the task with the default (i.e., Default (m)) and the essential (i.e., Essential (m)) knowledge. The task descriptions are as follows:”

The experimental evaluation is sound and I only have a clarification that maybe should be added. The experts involved in the experiment, if are the same, working with traditional Papyrus and Jorvik, could have understood the metamodel after the first experiment, and then this could reduce the time for understanding the domain in the second experiment. If I’m wrong it should be better clarified.

- Your understanding is correct. We added the following in Section 5.3.5 (Threats to Validity).

“The Jorvik experiment was run in both cases after the Papyrus one. Participants might become familiar with the domain described in the metamodel after finishing the Papyrus experiment and this could reduce the time for understanding the domain in the Jorvik experiment. However, the knowledge of the domain described in the metamodel is mostly useful when constructing the UML profile. This was the task that the participants performed better in the Papyrus experiment, thus we do not believe that this has any (significant) impact in the results presented.”

I conclude saying that Title and abstract are appropriate, the introduction states clearly the objectives, the contribution is quite relevant for the journal. The contribution is technically sound even if the research contribution is "limited" to the level of automation of the proposed tool respect to the manual process. The evaluation clearly confirms this. The related section presents an appropriate number of references and discussions.

Minor comments:
- The tool name should be emphasised in some way, e.g., \emph or UPPERCASE - addressed
- page 4 “(e.g.,for “A_to_B” —>missing closing parenthesis - addressed
- footnote 1 is spliced in two pages, try to reduce or redistribute to a single page if possible – this would be handled by the editor when paper is subjected to publishing
- Fig.1 caption has a final “.” - addressed
- page 8 “e.g., ATL [22]” —> please add Acceleo for completeness - addressed
- I would move the sentence at page 9 explaining the SDPL metamodel on top when the metamodel is introduced for the first time - addressed
- At the end of page 13 I would replace “.” With “:” - addressed
- When ref or val from EMFATIC are mentioned should be at least briefly clarified in the meaning, for readers that are not expert of that tool. - addressed
- Second line of sec. 4.6, add “default” to CSS style. - addressed
- spacing before sec 4.11 – this would be fixed by the editor when paper is subjected to publishing
- In section 5 Archimate example should be briefly introduced. - addressed
- Multiple footnote reporting the git repository can be replaced by a citation to the website - addressed
- multiple occurrences of “approach” with typos, e.g., approach - addressed
- page 31 footnote 20 presents “.” Before the footnote index. - addressed
- footnote 21 “creatin”—>”creating” - addressed