Reviewer: 1  
  
Public Comments (these will be made available to the author)  
1. Are the title, abstract, and keywords appropriate?  
  
Title and abstract are appropriate. No keyword is associated with the submission.  
  
2. Does the introduction state the objectives of the submission in terms that encourage the reader to read on?  
  
The objectives are clearly stated in the introduction.  
  
3. How relevant is this submission to the readers of this journal? The target audience of the journal are practitioners and researchers from industry and academia with a vested interest in high quality modeling practices and research. Indicate the extent that the paper will be relevant to this target audience.  
  
The submission is relevant to researchers from academia but I doubt it is for practitioners and researchers from industry (see detailed review below).  
  
4. How does this submission advance the field of software and system modeling research and practice? Comment on any novel contributions or significant insights gained. The journal aims to publish papers that deepen understanding of modeling practices and techniques or contribute significant new ideas that revolutionize or incrementally advance the field.  
  
This submission is an extension of the paper accepted and presented at ECMFA 2018. It presents an extension of the tool (Jorvik) and a detailed evaluation of the tool and process. However, the tooling part of the contribution is only incremental. It does not significantly advance the field of software and system modeling research and practice. Besides, I have several concerns about the contribution and the implementation (see detailed review).  
  
5. Is the submission technically sound? For example, comment on (1) adherence to standards if standard notations/techniques/methods are used, (2) soundness of mathematical expressions, and (3) soundness of conclusions drawn from objective premises.  
  
An important number of concerns weaken the soundness of the contribution and of the evaluation that has been conducted (see detailed review).  
  
6. Does the submission contain sufficient and most appropriate references? Journal versions of work are preferred over conference versions. Indicate important missing references, if any.  
  
The paper misses an important part of the literature with respect to tools that are already used by practitioners for building graphical editors based on DSLs. On that aspect, Jorvik presents clear limitations compared to other tools (see detailed review).  
  
7. Comment on the organization of the submission. Is it focused? Is the length appropriate for the topic?  
  
Section 4 (implementation) is unnecessarily long compared to Section 3 (Proposed Approach). It could be considerably reduced by removing unnecessary details about the implementation.  
  
8. Please comment on the readability of this submission. Please comment on the degree of effort required to read and understand this paper.  
  
An excessive number of details provided in Section 4 burdens the reading and forces the reader to go back to the beginning of the section.  
  
  
=== Detailed review ===  
  
  
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.  
  
- 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](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](http://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](http://d.name/) = 'source'))  
          , isTargetExist : Boolean = dependencies->exists(d | (d.client->includes(stereotype)).\_and([d.name](http://d.name/) = 'target')) in  
          isSourceExist.\_and(isTargetExist) implies (  
              let depS : Dependency = dependencies->select(d | (d.client->includes(stereotype)).\_and([d.name](http://d.name/) = 'source'))->asOrderedSet()->first()  
                , depT : Dependency = dependencies->select(d | (d.client->includes(stereotype)).\_and([d.name](http://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(sourceS))  
          )  
    ))  
  
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. Whilst we appreciate the detailed example provided to support your argument, 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 involved in typical UML profile creation. 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 at all.

- 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. We provide the OCL constraints for navigability and end types just to demonstrate what can be achieved. We also would like to point out that Jorvik as its current shape, is not a complete product, there are a lot of space for new features to be developed. However, we would like to get more feedback for Jorvik before we add 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, e.g. 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”. We would like to invite you to 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 presentation, 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 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. 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. Jorvik is not a complete product, we provide, as much as possible, facilities that help users of Jorvik towards their goal, there is still huge 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 Associations (we will change the annotations to ownedEnd and memberEnd instead of source and target). To support all notions of edges, it requires a vast amount of effort. For example, in UML, Association extends Relationship, which in turn extends Element. On the other hand, Transition extends RedefinableElement, which extends NamedElement. We now include the plans to support all UML elements in the future work. Towards that direction, it also implies that in the Ecore metamodel, the keywords in the annotations will change. We will make the change in our implementation which supports Association only.

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.  
  
- 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.   
  
- 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.  
  
- 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.  
  
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  
- 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;  
  
=== Section 3 ===  
  
- The motivation behind the backward transformation from a UML model to an EMF one is missing. Why do we need this feature?  
- 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  
- Fig. 3 is never referenced.  
  
=== Section 4.2 ===  
  
- The footnote 6 seems to me to be an important limitation of the tool that could be easily checked by Jorvik.  
- "For each reference (ref or val) in the metamodel [...]" <- should not it be "attr" instead of "val"?  
  
=== Section 4.3 ===  
  
- Figure 5: how is the "isRequired" attribute handled?  
- 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  
- "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  
  
=== Section 4.4 ===  
  
- Figures and examples are needed to understand this section  
  
=== 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.  
  
=== Section 4.9 ===  
  
- This entire section is unnecessary and should be removed.  
  
=== Section 4.10 ===  
  
- The feature presented in this section is not motivated.   
- 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.  
  
  
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).  
  
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).  
  
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.  
  
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.  
  
  
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).  
  
  
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. - addressed  
    => as an  
  
  
---  
  
[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.   
  
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.   
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.   
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   
  
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.  
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.  
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.   
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.   
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).  
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.   
  
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   
- I would move the sentence at page 9 explaining the SDPL metamodel on top when the metamodel is introduced for the first time  
- 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.  
- 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.   
- Multiple footnote reporting the git repository can be replaced by a citation to the website  
- 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