A generative Approach for creating Eclipse Sirius Editors for generic Systems

Francesco Bedini
Systems & Software Engineering Group
Technische Universität Ilmenau
Ilmenau, Germany
0000-0002-8354-1835

Ralph Maschotta
Systems & Software Engineering Group
Technische Universität Ilmenau
Ilmenau, Germany
0000-0001-8447-3996

Armin Zimmermann
Systems & Software Engineering Group
Technische Universität Ilmenau
Ilmenau, Germany
0000-0001-7439-7686

***Abstract*—Model-Driven Engineering (MDE) is getting more and more important for modeling, analyzing, and simulating complicated systems. It can also be used for both documenting and generating source code, which is less error-prone than a manually written one. For defining a model, it is common to have a graphical representation that can be edited through an editor. Creating such an editor for a given domain may be a difficult task for first-time users and a tedious, repetitive, and error-prone task for experienced ones. This paper introduces a new automated flow to ease the creation of ready-to-use Sirius editors based on a model, graphically defined by the domain experts, which describe their domains' structure. We provide different model transformations to generate the required artifacts to obtain a fully-fledged Sirius editor based on a generated domain metamodel. The generated editor can then be distributed as an Eclipse application or as a collaborative web application. Thanks to this generative approach, it is possible to reduce the cost of refactoring the domain's model in successive iterations, as only the final models need to be updated to conform to the latest format. At the same time, the editor gets generated and hence updated automatically at practically no cost.**

***Index Terms*—sirius, editor, metamodeling, eclipse, emf, ecore**

I. INTRODUCTION

Model-Driven Engineering (MDE) is gaining more importance for modeling, analyzing, and simulating complicated systems. It can also be used to document and generate source code, which is less error-prone than manually written one [1]. For defining a model, it is common to have a graphical representation of the model, which can be edited through a drag and drop editor.

Sirius is an Eclipse project that allows creating such graphical editors for the *Eclipse Modeling Framework* (EMF)-models [2]. Thanks to it, users have a reusable way of defining editors, for the IDE Eclipse or the web, by specifying a *Viewpoint Specification Project*, without needing to be an expert of programming languages, the Eclipse environment, and its plug-in system. It allows to quickly define an editor to graphically define models of different natures, either using an existing metamodel (which is a model describing the structure of another model) or after specifying one for a certain domain [3]. It would be possible to specify representations of viewpoints of various natures with Sirius, such as generic diagrams, edition tables, cross tables, trees, and sequence diagrams [4]. In this paper, we are putting our focus on the diagrams specification.

By providing one or multiple diagrams to edit a model, the *user experience* of the final modelers can be improved and simplified with additional layers of abstraction on top of the data model. Moreover, it is possible to add validation rules to the different kinds of diagrams to validate its semantic and provide visual feedback. In this way, the user can be confident about its model and diagrams' structural and semantic validity.

However, specifying a Sirius Viewpoint Specification is not always a simple assignment. It requires specific knowledge about the Sirius available tools and the Eclipse system [5]. Even experienced users have to repeatedly go through error-prone tasks for each element of the domain.

As Model-Driven Engineering (MDE) aims to provide more dependable programs by generating their source code, we would like to provide more robust and reliable Sirius editors by generating them with an automated process. Generating an editor for any unconstrained metamodel would not be an easy task. The generator needs a way to infer how the metamodel elements need to behave inside the editor.

This paper's remainder is structured as follows: the following subsections introduces the nomenclature (I-A) used in this paper and the motivation (I-B). Section II briefly explains the current flow for defining a Sirius editor based on a given metamodel and discusses the existing alternatives. Section III described our proposed generative flow. Section IV shows the definition and generation of an example realized with a proof-of-concept implementation of our method, whereas in Section V the preliminary results discussing the advantages and disadvantages of our approach. Section VI concludes this paper and gives possible hints for future works.

A. Nomenclature

First, we would like to introduce the terms that are used in this paper, with the help of a *use case* diagram (shown in Figure 1).

In the illustration, the actors represent:

This work has received funding from the *Carl Zeiss Foundation* as part of the project "Engineering for Smart Manufacturing" (E4SM) under grant agreement no. P2017-01-005.

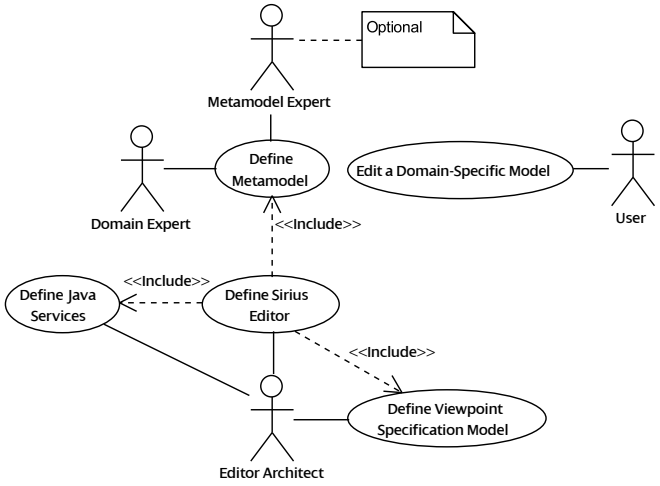

Fig. 1. A UML *use case* diagram explaining the nomenclature used in this paper.

Domain Expert:
: has deep knowledge about its system's domain;

Metamodel Expert:
: knows how to represent the system as a metamodel or as a *Domain Specific Language* (DSL, whose grammar is described as a metamodel). Its tasks also include gathering and analyzing information from various domain experts in order to realize a compliant metamodel;

Editor Architect:
: has the task of designing and realizing a Sirius editor for a given metamodel;

User:
: uses the Sirius editor for creating its models, which can be used for different purposes, such as generating documentation or the whole source code, simulation, or analysis.

For a more detailed explanation of all Eclipse Sirius-related terms, please refer to the Sirius Glossary [6].

B. Motivation

This work was realized as part of the *Engineering for Smart Manufacturing* (E4SM) project, which aims to simplify the application and adoption of machine learning and software engineering methods for the small and medium industries. The goal consists of encouraging the adoption of model-based systems engineering (MBSE) and MDE between these categories by providing tools to automate tedious tasks that require specific knowledge, and simplify the onboarding process for beginners.

Often projects require the collaboration of heterogeneous departments and expertise, which can all be tackled separately with domain-specific models and then gathered together in a comprehensive global model. The domain experts of each branch need efficient tools to describe their domain models, and then we simplify and speed up the remaining process by generating the required editors and validation rules.

In this paper, we are going to propose a generative approach for realising this, with the help of *Operational Query/View/Transformation* (*QVTo*) and *Acceleo* model-to-model/model-to-text transformations for generating the required artifacts. We are showing, with a preliminary implementation of the generation process, if it is possible to realize a usable Sirius editor for domain-specific-models without the help of an editor architect.

This generative approach would enable, for example, to instruct the generators to add standardized properties inside all (or a subset) of the elements for performing specific kinds of analysis. For example, it is possible to insert a latency property in all connectors to compute the average latency between any given pair of items. In this way, it is possible to have a reusable set of tools for analysis and simulations that can be reused consistently on different models when generated through our method.

II. THE CURRENT FLOW FOR DEFINING A GRAPHICAL EDITOR

Defining an efficient Sirius editor is not a simple task. Sirius can work without visible issues with an under-specified viewpoint specification, which results in higher consumption of resources, for example, by using the default filters to find the elements which should be rendered on the current diagram. To assure a better performance, Obeo, the company that created Sirius, published in [7] a collection of best practices to follow to obtain a robust Sirius editor, which can efficiently handle larger non-trivial models.

Even for experienced users, it is easy to oversee these best-practices, as there are no warning systems similar to those for programming languages suggesting refactoring an element with a better implementation. Many fields have to be manually filled-in multiple times for each element-mapping inside the Sirius *Viewpoint Specification* project.

Figure 2 shows a conventional flow for defining a Sirius editor, as described for example in [8]. In this case, we suppose that the metamodel is user-specific, and it is not already existing. Most probably, after the DSL definition, the user would create a model containing all possible metamodel elements and connection/containment for testing both that the metamodel specification is reasonable and that the Sirius editor behaves as expected. The most time-intensive set of tasks (which is highlighted on the diagram) can begin: the definition of the Sirius *Viewpoint Specification* project with an iterative approach. Luckily, Sirius works without requiring any compilation; this means that the Sirius specifications may be changed, and the edits are visible in real-time.

The *editor architect* needs to define all the different mappings and styling for all desired elements and diagrams until the testing model is getting rendered as desired, and the users are satisfied with the outcome and can edit the model without any issue.

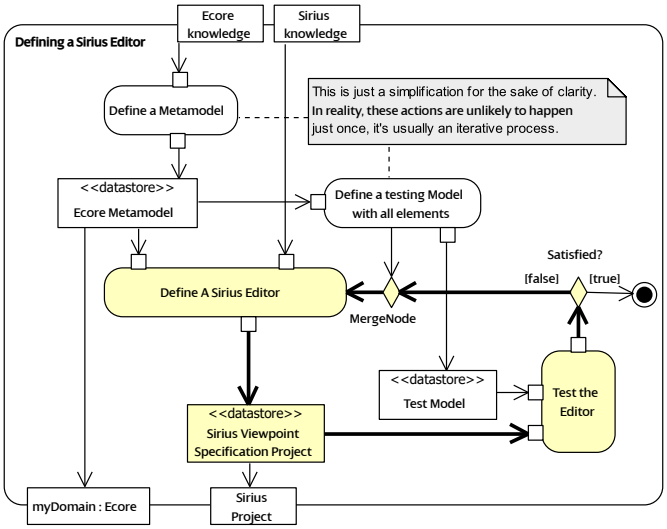

Fig. 2. A UML Activity Diagram showing a common flow for currently defining a Sirius Editor. The highlighted path forms a loop where most time gets wasted with tedious tasks requiring specific knowledge about the Sirius tool.

In this flow, plenty of repetitive tasks could be automated for a more accurate and robust result. Unfortunately, this is not so feasible as input any metamodel (i.e., without explicitly knowing what each element of the metamodel represents in reality). Here comes the editor architect's experience to the rescue. He/she knows how to define one or more graphical representations for all or a subset of the domain's elements based on each element's meaning.

The editor architect can infer, either from its role in the domain metamodel or by gathering additional information from the domain experts, how each diagram element should behave in the editor.

Similar generative approaches based on the Eclipse EMF/GMF toolkit include the one from "*EuGENia*" [9] and the "*CINCO SCCE Meta Tooling Framework*" [10].

Since October 2020, Sirius viewpoint specification projects can be used to create web versions, which allow live multi-users collaborative editors. Lately, similar generative approaches for web editors instead of Eclipse plug-ins have been [11] for EuGENia, and Pyro [12], which is based on Cinco.

III. METHOD

This cumbersome and repetitive manual definition process can be avoided or simplified by introducing a new generative approach. Our suggested flow is possible, thanks to a proposed metamodel that must be used to define the user's system *DomainDescription* model. In this way, the transformation tools all know how each element behaves and how they should be represented to the final user.

Of course, the domain expert should still have the possibility of changing the elements' styling to make the editor more appealing and understandable for the users, but we also want to keep it simple to avoid overwhelming the domain-experts with irrelevant details.

TABLE I
METHOD'S STEPS, ACTORS AND OUTPUTS

1	2	3
Domain Experts	Automatic Generation	Users
define	**produces**	**use**
Sirius Domain Specification Editor	- Domain metamodel - Sirius Viewpoint Specification Project	Sirius Domain Model Editor

Our approach can be divided in three main steps, which are shown in Table I. First, the domain experts need to describe their domain with the help of a graphical Sirius meta editor, which we provide. This editor allows defining a model that contains both the semantic of the domain and the styling information used by the generated editor. This model will then be generated in the second step. This process can be automated entirely in Eclipse in order to be easily accessible for the domain experts. The generated Sirius editor can then be distributed to the final users as an eclipse or a web project, as currently allowed by Sirius.

Our method is similar to the approach of the "*CINCO SCCE Meta Tooling Framework*" [13]. Apart from the different tools used in the generation step (Acceleo[1] in our case versus xtend[2]), the main differences lay at the beginning and the end of the generative process. We provide a graphical editor to define the domain model, whereas the Cinco group allows defining the language textually. Both approaches have pros and cons. A graphical language can be clearer to be visualized and edited by non-programmers but does not scale well for more extensive and complicated models. In that case, a textual language may be more suitable.

The other main difference is the output of the generation process. With our approach, a Sirius editor gets generated as a single Viewpoint Specification Project, while Cinco generates a Graphiti Editor. A comparison between these two kinds of editors can be found in [14]. Another difference is that with our metamodel (and thanks to Sirius), we can support more kinds of elements natively, like ports at the border of nodes and containers, while Cinco supports nodes, containers, and edges. On the other hand, Cinco has many other useful functionalities, and, unlike our preliminary proof-of-concept, it is fully implemented and keeps getting improved.

The generation process is shown in the activity diagram in Figure 3. The domain expert needs to provide his domain description as an Ecore model, which can be realized using the graphical Sirius editor we provide. If the model is valid, the model is first converted with an endogenous QVTo M2M transformation to a *DomainImpl* Ecore model, which contains the semantic structure defined in the *DomainDescription*

[1]https://www.eclipse.org/acceleo
[2]https://www.eclipse.org/xtend

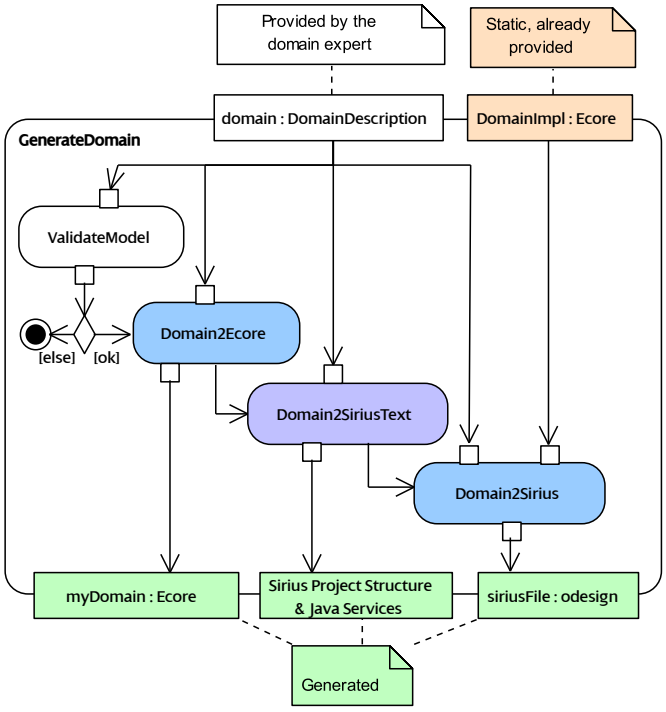

Fig. 3. A UML Activity Diagram showing the necessary steps required to generate a Sirius editor using our method.

model. The Sirius project is then generated by a M2T transformation realized with Acceleo, which takes care of generating the Sirius Viewpoint specification project and the required Java services, and an exogenous QVTo M2M transformation for generating the Sirius' *odesign* file, which contains the styling information from the model and the definition of the graphical editor.

A. UML vs a new Metamodel

Our proposed approach consists of a flow that transforms a model, an instance of our *DomainDescription* Ecore metamodel, to a ready-to-use Sirius editor. Before describing our *DomainDescription* metamodel in details, one question that may arise would be *why did you choose to define another metamodel instead of using UML directly?*

We wanted to simplify the definition of the editor for inexperienced users. Although UML would probably be more comfortable for modelers to start with, these categories would probably have enough experience to define a Sirius editor on their own.

Moreover, it would be possible to specify in the UML model plenty of attributes or elements that our generators would not consider, as they would be out of scope. In this way, it would not be clear for the users which attributes would cause an effect and which does not. Using our metamodel, it should be clearer what possible customizations are available and what each attribute does.

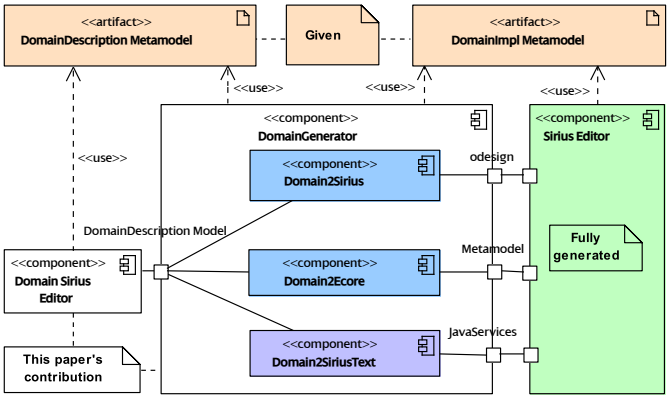

Fig. 4. A UML Component Diagram showing the components involved in the pipeline.

B. Our generative Flow

Figure 4 shows a component diagram representing the components and the artifacts of our proposed flow. We provide two metamodels (in orange): the *DomainDescription* metamodel, which describes what the elements of the domain are and how they can be connected, and the *DomainImpl* metamodel, which is used at runtime by the generated Sirius editor to know which role and behavior the elements have in the diagram (such as nodes, containers, and edges).

With our Sirius editor, the domain-expert can define one *DomainDescription* model, which contains both the semantic and some essential styling attributes of the domain elements. This model will be the starting point and only input for all our transformations.

The M2M *Domain2Ecore* transformation converts the *DomainDescription* model to a pure Ecore model, by filtering out all styling and editor-related information. At this point, the *Domain2SiriusText* M2T transformation generates a Sirius *Viewpoint Specification* project and the Java services needed to control the edge connections, which will be explained in more details in Section III-D. The Domain2Sirius M2M transformation completes the *DomainGenerator* component by creating the `odesign` file which contain the graphical definition (mapping) of all non-abstract elements of the *DomainDescription* model.

The following section explains in detail the content of our provided metamodels.

C. The Domain Metamodels

The first metamodel we had to define was the *DomainDescription* metamodel, which is shown in the class diagram in Figure 5. This metamodel allows specifying the domain's elements, how they can be connected, and under which cardinality conditions.

Figure 6 summarizes the semantic of the instances of the *DomainDescription* metamodel in a graphical way. *Nodes* and *Containers* may have *Ports*, *Containers* may contain other *Nodes* and *Containers*. *Nodes*, *Containers*, and *Ports*, can be generalised as *Items*. *Edges* can connect one *Item* to one and

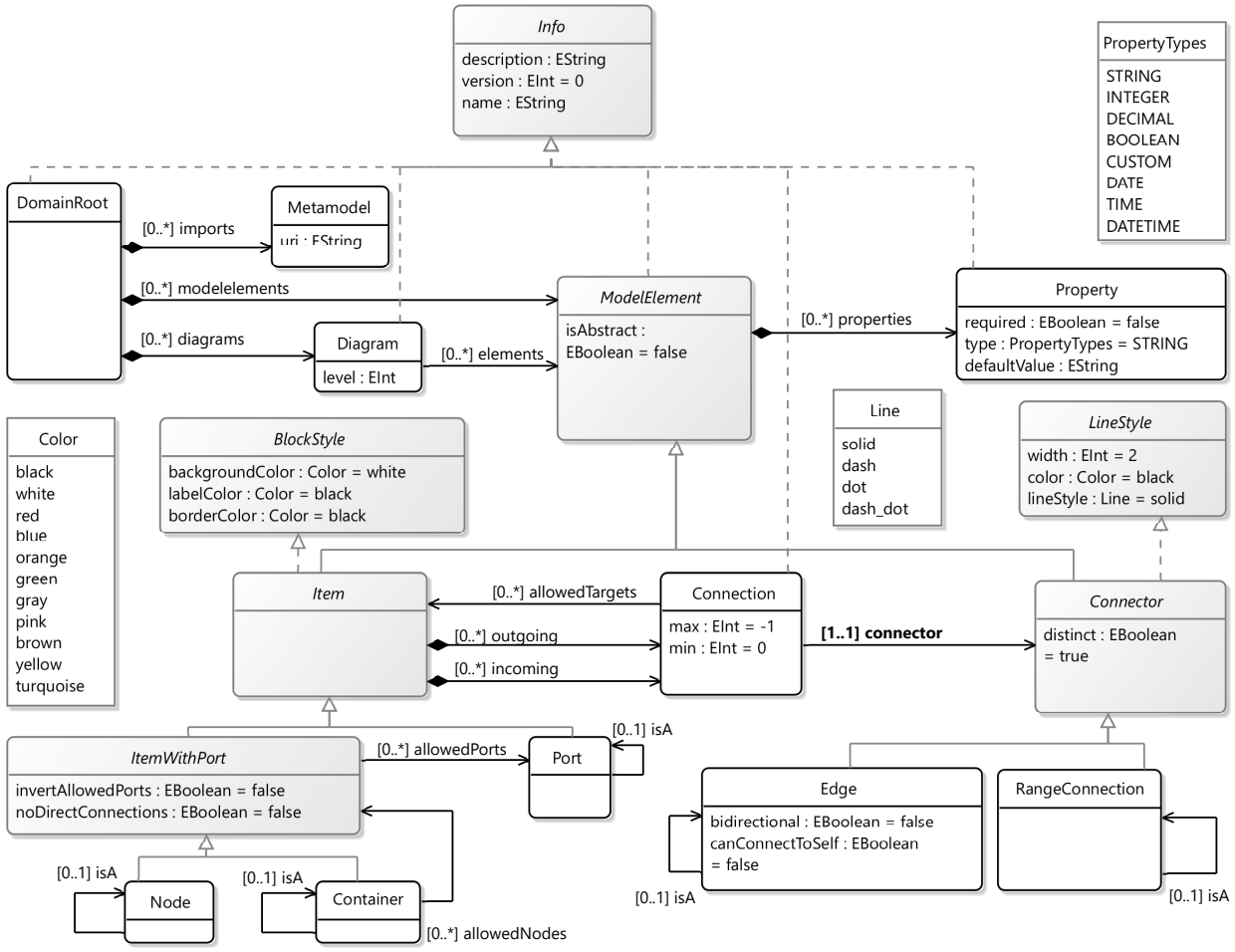

Fig. 5. The *DomainDescription* metamodel class diagram. Abstract classes, which cannot be directly instantiated, are shown in gray. The root element is the *DomainRoot* on the top left. It holds references to external metamodels, and it contains the definition of the domain elements and the available diagrams. Diagrams may contain all or a subset of elements. All elements may have additional properties, and all items and connectors can be styled. Each element has standard information that the user can fill in, like a name, a description, and a version, which may be used to generate automated model transformations.

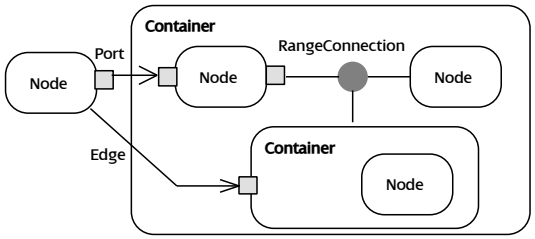

Fig. 6. An example showing all possible connections and allocations of the *DomainDescription* elements instances.

only one *Item*. *RangeConnections* can connect n *Items* to one *Item*.

It is possible to specify what kind of diagrams will be made available to the users and what model elements can be drawn on each of them. The specification of the different views is done by creating elements of type *diagram* and assigning the *ModelElements* that should be available in each diagram.

The following list describes all available model elements, with an example which will then be realized in Section IV.

Node:
: A simple atomic[3] element, like a sensor;

Container:
: An element which can contain other nodes or other containers, such as a sensor box;

Port:
: Interface element that can be placed on the border of nodes or containers, like an Ethernet port;

Edge:
: A 1 to 1 connector between elements, for example, an Ethernet cable;

RangeConnection:
: a 1 to n connection between elements, like a router providing wireless connectivity.

Abstract model elements will not get any creation or editing tool on the Sirius editor and will be generated as abstract in the

[3]The concept of *atomic* is domain-specific and depends on the desired modeling abstraction level. What is considered atomic in one domain may be a container in another.

Ecore model. Each *ModelElement* may have properties, which have a type and a default value. Additional metamodels can be referenced in order to reuse existing models.

Our generators make it possible to produce different editors (or views for the domain model), which only contain a subset of the domain elements.

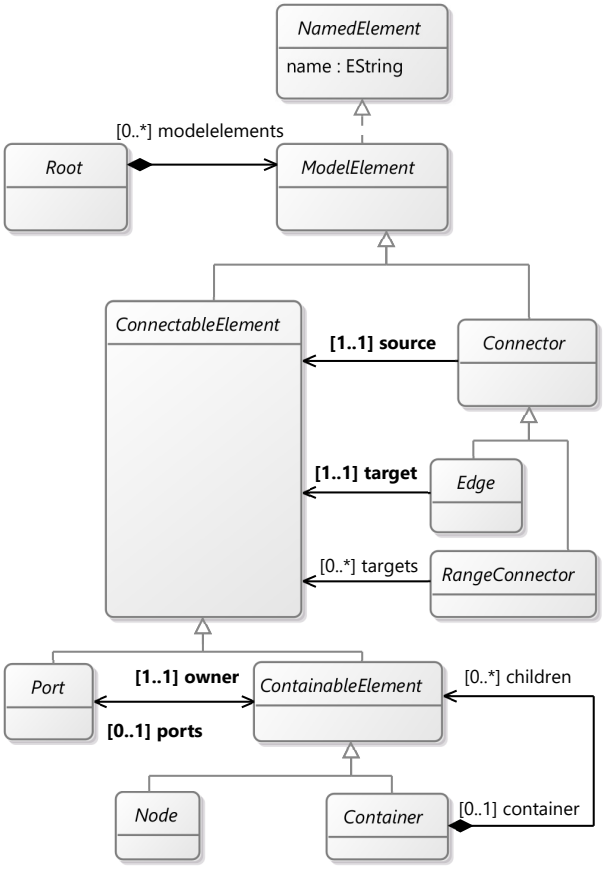

Fig. 7. The *DomainImpl* metamodel.

Figure 7 shows a diagram introducing the *DomainImpl* metamodel. All its metamodel classes are abstract, which means no elements and no models can be defined directly using just this metamodel. In order to use it, it is necessary to define another metamodel and define generalizations (or *Is-a*-relationships) from concrete classes to one of the abstract *DomainImpl* elements. This metamodel is used to assign a type to each of our generated metamodel elements so that the generated Sirius editor will know how each element should behave.

D. Generating the Edge Connections Constraints

As the allowed edge connection semantic and constraints would be too intricate to be expressed using the *Acceleo Query Language* (AQL), which is a declarative language, we opted for adding another M2T transformation for generating Java Services, which can programmatically verify if a connection is allowed or not. Moreover, with this transformation, we can generate the Sirius project folder structure.

For each kind of edge, a service method gets generated. These methods get called in the *Connection Complete Precondition* of the corresponding *Edge Creation* tools. These function decides whether the selected `source` and `target` may be connected together with the chosen *Connector*. An example of the general structure of these generated functions is shown in the pseudocode in Figure 8.

```
bool canConnectEdge(root, source, target){
//optional, if canConnectToSelf is false
if (source == target)
    return false;

//check the source and target types
if (connection between types is not allowed)
    return false;

//optional, if distinct was true
if (root has edge btw. source & target)
    return false;

//optional, for each connection with a max
//multiplicity set to lower than *
//check outgoing multiplicity
if (source instanceof <Type>) {
    if(outgoingEdges >= <multiplicityValue>) {
      return false; //Outgoing mult. overflow
    }
}
//check incoming multiplicity
if (target instanceof <Type>) {
    if(incomingEdges >= <multiplicityValue>) {
      return false; //Incoming mult. overflow
    }
}
return true; //Nothing against connecting
}
```

Fig. 8. An example of generated code for checking the *Connectors* constraints (as pseudocode).

IV. Example

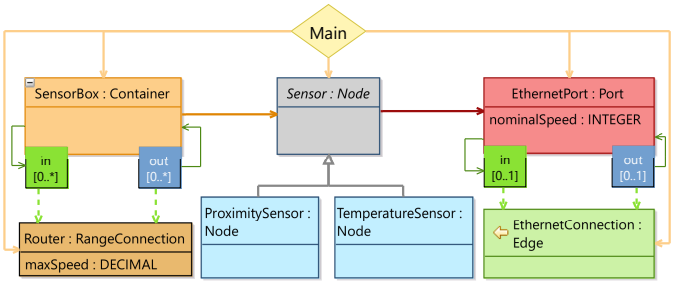

Fig. 9. A *DomainDescription* example model with all available element types. The yellow rhombus represent a diagram. Its connections show which elements it will contain. All other element types are contained in the first row of the block after the colons. Abstract elements are shown in gray.

Figure 9 shows the *DomainDescription* model used as an example in Section III-C. There are two kinds of sensors: proximity sensors and temperature sensors, which can be contained by a sensor box that enables wireless connectivity

through routers. Moreover, the single Sensors may have an Ethernet Port to communicate through an Ethernet connection.

In this model, we generate a *Main* diagram, which will be an editor that gives the possibility to edit all domain elements.

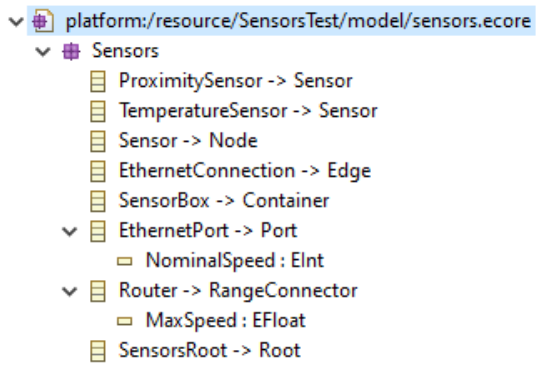

Fig. 10. The generated Ecore model from the sample *DomainDescription* model.

The generated metamodel is shown in Figure 10. All elements which are not children of a is-A relationship are generated with a generalization relationship to one of the *DomainImpl* metamodel element. In this way, this metamodel only contains the elements of the domain. How they can be connected is already defined in the abstract *DomainImpl* model, which remains consistent for all our generated editors.

Due to technical reasons, the model needs to have one root element. That is why a *SensorsRoot* element is included automatically, even if it is not present in the *DomainDescription* model.

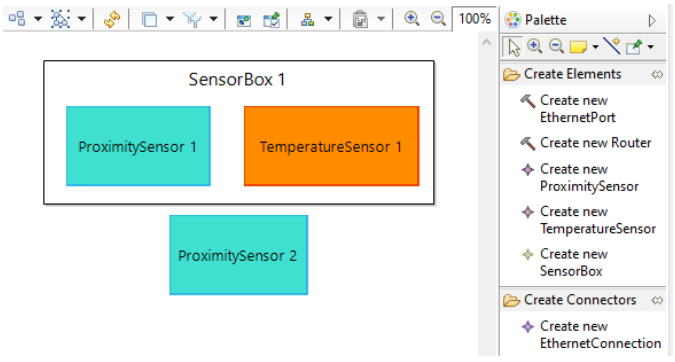

Fig. 11. The generated Sirius Editor from the generation of the sample *DomainDescription* model.

Figure 11 shows the final transformation result: a Sirius editor with creation and editing tools that are ready to use. The tool's palette is available on the right-hand side. It contains creation tools for all non-abstract elements of the *DomainDescription* model. The left-hand side shows the main editor area. The elements' colors are customizable by specifying the required value in the *DomainDescription* model.

The following section discusses the main advantages and limitations of our approach.

V. Discussion

A. Advantages

Simplified co-evolution of metamodel and editor:
: our approach is more robust to changes to the domain, as only the existing user models must be upgraded, either manually or with an automated migration [15], to keep working. This approach is a possible solution to the resilience problems which arise when a metamodel get changed [16];

Optional styling:
: unlike in the Sirius viewpoint specification project, where each mapping must have a manually defined styling to work, our default values inside the metamodel and the generation step take care of generating a valid `.odesign` file with a default black and white styling, even when the user did not set any;

Faster *go live*:
: with the specification of just one *DomainDescription* model, it is possible to start editing a domain-model in few minutes.

B. Limitations

Generalisations:
: the current implementation only supports a one-level hierarchy;

Different views:
: it is currently possible to create different diagrams and define which kind of items will be shown in each (without filtering), but it is not possible to have different graphical representations for the same object in different diagrams. However, it would be possible to extend the current M2M Sirius generator to generate other kinds of viewpoints additionally;

Customisation:
: all elements of the same kind (*Node*, *Container*, etc.) are currently generated with a fixed shape.

No manual changes:
: if the user decides to extend or change the generated Sirius project directly, these changes will be overwritten by a subsequent generation.

VI. Conclusions

In this paper, we have shown how it is possible to simplify the creation of Sirius editors for inexperienced users thanks to two M2M transformations, which take care of generating both the Ecore metamodel and the Sirius *Viewpoint Specification* `odesign` file, and a M2T transformation, which takes care of setting up the Sirius project structure and the Java services.

Our proof of concept is promising and shows that it is possible to generate editors for static models with a limited number of different elements. When this number increases, our Sirius editor's current implementation makes it complicated to define the domain concisely and understandably. Nevertheless, with the realization of a different (for example, textual) editor, the generation of more challenging editors could become possible.

The main challenge was maintaining a balance between expressibility and usability. We wanted to avoid that the user, in the end, needs to set in the metamodel all properties that he would need to set in the Sirius viewpoint specification project. To achieve this, we had to limit and simplify the kind of diagrams that the user can obtain using our flow.

All our final model constraints, defined in the *Domain-Description* model, are not enforced by the model's structure; the generated editor takes care of enforcing them. If the users edit their final model directly, they could bring their models in an inconsistent or prohibited state. We assume that the users will only use the generated editor to edit their models. If this is the case, a consistent state is guaranteed.

The tool's robustness, ease of use, and the number of useful functionalities for the users deeply influence the acceptance of the proposed flow. Our proof-of-concept still has to be completed for all elements, and the flow is currently not automatized. We are releasing all artifacts and transformations as open-source under the license *GNU GPLv3*. Contributions are welcome: https://github.com/tuiSSE/sirius-meta-editor

A. Future Works

The amount of elements of the *DomainDescription* model could be extended to include, for example, *Add-ons* (which allow to dynamically extend one *Item* with a fixed set of additional properties) or *Function* definitions to describe a dynamic behavior.

As the transformations are modular and do not depend on each other, it would be possible to adapt this approach to other tools rather than Sirius by defining a different transformation. The opposite is also possible: extending other generative tools, such as Cinco, to apply our generation process (with some required changes to remove the non-existing elements such as ports) on their metamodel.

ACKNOWLEDGMENT

We want to thank Dr. Stefan Naujokat and his team at the *Teschnische Universität Dortmund*, Germany, for their support in solving our few issues and doubts regarding the Cinco framework.

REFERENCES

[1] J. A. Hurtado, M. C. Bastarrica, S. F. Ochoa, and J. Simmonds, "Mde software process lines in small companies," *Journal of Systems and Software*, vol. 86, no. 5, pp. 1153 – 1171, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121212002749

[2] Eclipse Sirius. (2020) Eclipse Sirius Documentation. [Online]. Available: https://www.eclipse.org/sirius/doc/

[3] S. Jäger, R. Maschotta, T. Jungebloud, A. Wichmann, and A. Zimmermann, "Creation of domain-specific languages for executable system models with the eclipse modeling project," in *2016 Annual IEEE Systems Conference (SysCon)*, 2016, pp. 1–8.

[4] V. Viyović, M. Maksimović, and B. Perisić, "Sirius: A rapid development of DSM graphical editor," in *IEEE 18th International Conference on Intelligent Engineering Systems INES 2014*. IEEE, 2014, pp. 233–238.

[5] A. Iung, J. Carbonell, L. Marchezan, E. Rodrigues, M. Bernardino, F. P. Basso, and B. Medeiros, "Systematic mapping study on domain-specific language development tools," *Empirical Software Engineering*, pp. 1–45, 2020.

[6] Eclipse Sirius. (2020) Eclipse Sirius Glossary. [Online]. Available: https://www.eclipse.org/sirius/doc/Glossary.html

[7] Obeo. (2017) Eclipse Sirius Best Practices: Version 5.1. [Online]. Available: https://www.obeodesigner.com/en/best-practices

[8] R. Maschotta, S. Jäger, and A. Zimmermann, "Teaching model driven architecture approach with the sirius project," in *European Conference of Software Engineering Education (ECSEE 2016)*, 2016.

[9] D. S. Kolovos, A. García-Domínguez, L. M. Rose, and R. F. Paige, "Eugenia: towards disciplined and automated development of gmf-based graphical model editors," *Software & Systems Modeling*, vol. 16, no. 1, pp. 229–255, 2017.

[10] S. Naujokat, M. Lybecait, D. Kopetzki, and B. Steffen, "Cinco: a simplicity-driven approach to full generation of domain-specific graphical modeling tools," *International Journal on Software Tools for Technology Transfer*, vol. 20, no. 3, pp. 327–354, 2018.

[11] F. Rani, P. Diez, E. Chavarriaga, E. Guerra, and J. de Lara, "Automated migration of eugenia graphical editors to the web," in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, ser. MODELS '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3417990.3420205

[12] P. Zweihoff, S. Naujokat, and B. Steffen, "Pyro: Generating domain-specific collaborative online modeling environments," in *Fundamental Approaches to Software Engineering*, R. Hähnle and W. van der Aalst, Eds. Cham: Springer International Publishing, 2019, pp. 101–115.

[13] TU Dortmund. Cinco scce meta tooling framework. [Online]. Available: https://cinco.scce.info

[14] V. Vujović, M. Maksimović, and B. Perišić, "Comparative analysis of dsm graphical editor frameworks: Graphiti vs. sirius," in *Proceedings of the 23rd International Electrotechnical and Computer Science Conference (ERK'14)*, 2014.

[15] M. Eysholdt, S. Frey, and W. Hasselbring, "EMF Ecore Based Meta Model Evolution and Model Co-Evolution," *Softwaretechnik-Trends*, vol. 29, no. 2, pp. 20–21, May 2009, (Proceedings of the 11th Workshop Software-Reengineering (WSR 2009)). [Online]. Available: http://eprints.uni-kiel.de/14464/

[16] J. Di Rocco, D. Di Ruscio, H. Narayanankutty, and A. Pierantonio, "Resilience in Sirius Editors: Understanding the Impact of Metamodel Changes," in *MODELS Workshops*, 2018, pp. 620–630.