Sirius modeling tool use in electrical system applications

Arkadiusz Rys

University of Antwerp, Belgium

Abstract

Sirius[1] is an Eclipse project which is built on top of the Eclipse modeling technologies to aid in designing a graphical modeling workbench. A user can create meta-models and models on top of these meta-models which can then be used to validate whether constraints are met or even generate code. **Sirius** has the added benefit of allowing users to work directly within the graphical representation of the generated models. [1] The goal of this paper is to create an overview of the functionalities of **Sirius** and compare it to other known modeling tools. We will do this and show how **Sirius** is applicable to the design of electrical systems.

Keywords: Model-Driven Engineering, Sirius, Eclipse modeling Framework, Electrical Systems

1. Introduction

Sirius is a Model Driven Engineering tool developed by Obeo and Thales with the help of the community. It is a graphical tool where the user can edit the properties of diagrams and other visualizations within the visualization itself. As Model Driven Engineering can be used to develop domain specific applications where the representation can be used by a domain expert, it lends itself greatly for the case of designing complex electrical systems.

Email address: Arkadiusz.Rys@student.uantwerpen.be (Arkadiusz Rys)

[1] Siriuscanbefoundathttp://www.eclipse.org/sirius/

Section 2 will elaborate more on the details of the architecture on top of which **Sirius** is based. The many features and capabilities of **Sirius** will be presented in section 3 with a comparison between **Sirius** and **AToMPM** following in section 4. Section 5 explains how we will apply **Sirius** to our problem. Section 6 finally concludes.

2. Architecture

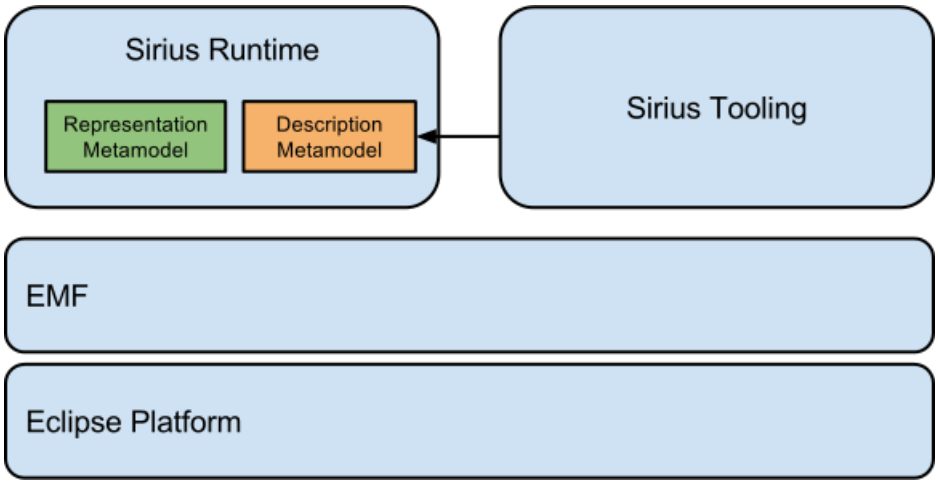

Figure 1: Sirius architecture model overview.

Eclipse. **Sirius** is built on top of the eclipse platform as seen in Figure 1.[2] Eclipse is rather extendable and **Sirius** acts like a plugin in this system. This allows us to extend the functionality of **Sirius** by installing more Eclipse plugins which could aid in Model Transformations or code generation.

Eclipse modeling framework. **Sirius** is not built directly on top of Eclipse, EMF or the Eclipse modeling framework connects the two. EMF is used to design the ecore meta-models. Editors can be generated to edit stored data textually

[2] `https://www.eclipse.org/sirius/doc/developer/Architecture_Overview.html`

and **Sirius** extends these capabilities by allowing to edit the data within the diagrams themselves. The EMF layer is where model transformations happen.

Split. At the highest level you can see how the **Sirius** Tooling is split from the **Sirius** Runtime which interprets the models. This has the advantage of a smaller package for the end users, which will not have any of the tools needed to edit the underlying structure. The **Sirius** runtime is where the end-user would interact with the models.

This is not the only way **Sirius** splits its architecture. The core is also split from any dialect specific extensions like diagrams or trees. This way, more dialects can be developed by third parties just by accessing **Sirius**' API.

Another optimization happens when models are updated. **Sirius** uses a refresh algorithm which is incremental and therefore only the changes are propagated to the model, this results in them being available to be viewed immediately.

Graphical modeling Framework. **Sirius** uses the GMF or Graphical modeling Framework notation and runtime. The internal model is computed from the designed domain- and specification model. Then the **Sirius** internal diagram model is used as the semantic model for the notation model. GMF tooling was used to initialize the GMF code to manipulate the internal **Sirius** diagram model but now the generated code and GMF tooling are not used anymore.[2, 3]

3. Capabilities

Sirius support five representations out of the box:

- Diagrams
- Sequence Diagrams
- Tables
- Trees

- Properties view

The difference with AToMPM is how **Sirius** has more than only diagrams. **Sirius** allows us to have a combination of these representations in a single project, you can even have multiple represenatations of the same type.

3.1. Diagrams

Diagrams are very versatile. In **Sirius** they have quite a lot of options so we will cover a few.

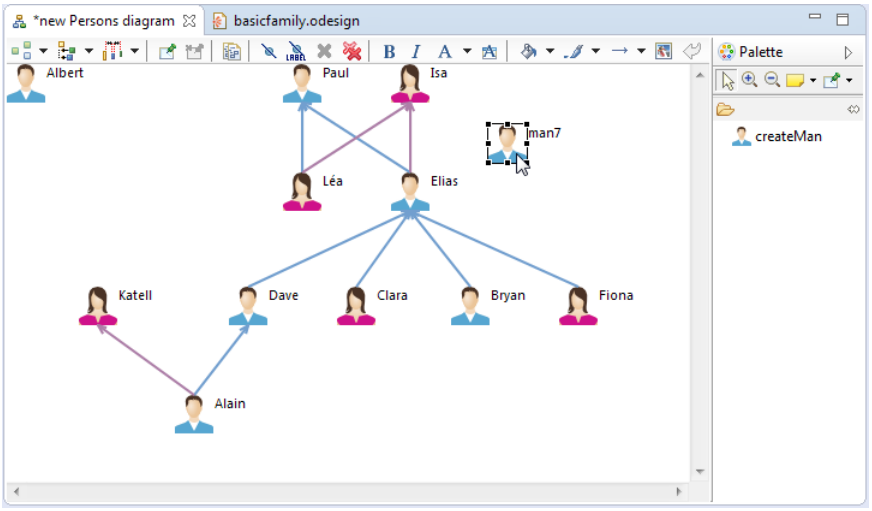

Figure 2: Example of a diagram in **Sirius**.[3]

Layers. Diagrams can have one or more layers which can be independently shown or hidden. In these layers we can define graphical representation which will be mapped onto elements.

Styling. Every aspect of the diagram can be styled. Styles can be conditional. For example: weighted edges with a weight higher than 5 can be turned red.

[3] `https://www.eclipse.org/sirius/doc/specifier/Sirius%20Specifier%20Manual.html`

Tools. We can also define tools which will be available to the user. These can be used on the representation or be defined to happen on a specific event like the reconnecting of an edge.

Filters. Defining filters, which will hide or show elements matching specific conditions is also possible. This gives the designer more choice than disabling whole layers of elements.

Validation. The model can be validated when required. Rules have to be set before the validation can take place which is comparable to the way global rules are defined within **AToMPM**.

More options are available within the framework as it is meant to be able to encompass any design compatible with the EMF core.

3.2. Tables

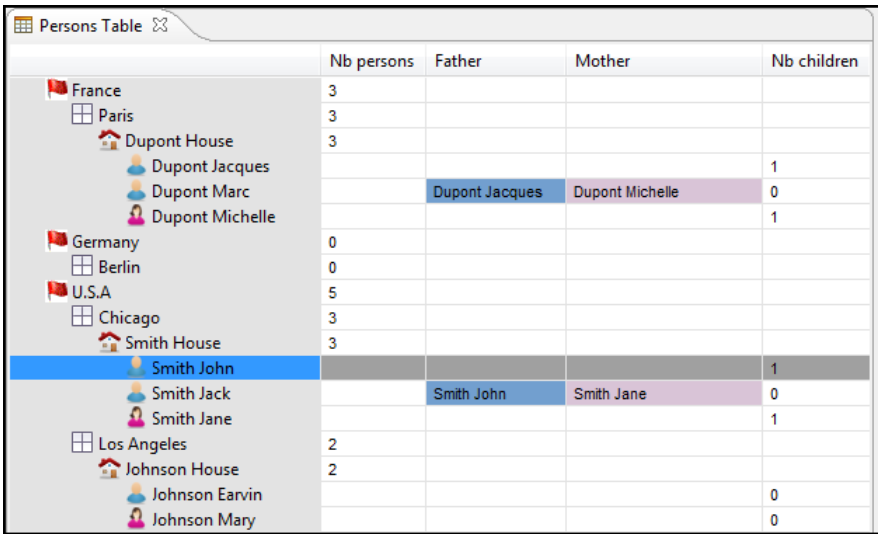

Figure 3: Example of an edition table in **Sirius**.[4]

[4] https://www.eclipse.org/sirius/doc/specifier/Sirius%20Specifier%20Manual.html

Sirius allows to define tables. These give us the option of editing the data within a table which at times will be faster than fiddling with a diagram. We have two types of tables within **Sirius**.

(1) The Edition Tables behave just like any old regular table would, the column header mappings will be some (computed) attribute.

(2) The Cross Tables are a special kind of tables which are optimized to represent relationships between elements. Both the columns and row headers will represents elements with the corresponding cell checked when a relationship between them exists.

3.3. Trees

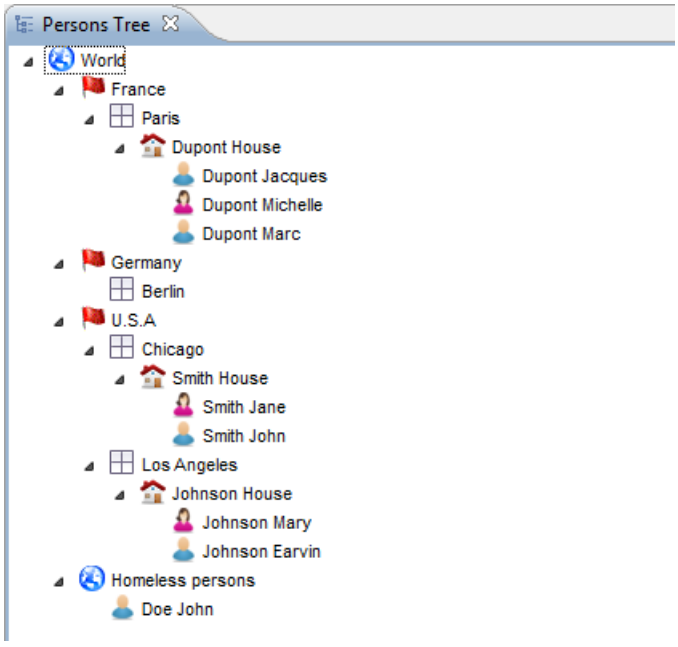

Figure 4: Example of a tree view in **Sirius**.[5]

Tree views are the hierarchical views you can see all throughout **Sirius** within its own editing windows. The items within these are created lazily however they

[5] `https://www.eclipse.org/sirius/doc/specifier/Sirius%20Specifier%20Manual.html`

are not deleted implicitly.

3.4. Overview

Users familiar with Eclipse will recognize the layout of the Sirius workbench. As **Sirius** allows many views or representations of the same data we can edit the data in any them and the changes will propagate. This allows the designer to open both views at the same time and monitor whether the changes in one view have the desired effect on the others.

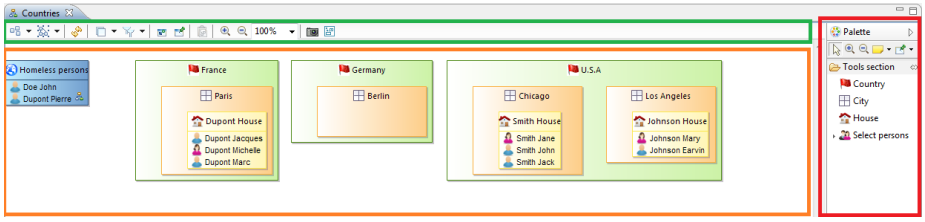

Figure 5: The interface the end-user would see for diagram editing.[6]

Whenever the end-user manipulates the models, he does so in a simpler view. This view is the one the end user would get for a diagram. The canvas in the center (orange) is where they would create and edit their model. The palette (red) shows what they have at their disposal (the tools and elements we defined) and at the top in the menu (green) we have some general options. The behavior when they add, delete or perform any other operation is also defined by the person who designed the model.

4. Comparison

As **Sirius** is a graphical modeling tool its best to compare it to other tools in the same category. This is why we have chosen to compare it with **AToMPM**. The distributed nature of **AToMPM**[4] versus the Eclipse based **Sirius** is hard to compare as both have their qualities.

[6] `https://www.eclipse.org/sirius/doc/specifier/Sirius%20Specifier%20Manual.html`

Sirius has the advantage of a well established platform with a huge user base. Extending **Sirius**' functionality can be done by installing additional Eclipse plugins or by creating new types of representations using the exposed API. A different approach has been taken in **AToMPM** where extending functionality can be done by defining additional models.

When you know your way around the many properties and menus of **Sirius**, it is quite easy to edit models in a fast way thanks to the table representations while the singular diagram view can make this harder in **AToMPM**.

The way **Sirius** is designed makes it a little bit harder to get your first model. This only applies when you don't have experience using the Eclipse Runtime Configurations.

While the tools differ in the previously mentioned properties they both make a distinction between abstract and concrete syntax. A lot happens behind the scenes in both tools, **AToMPM** will the the RAMification for you and **Sirius** can automate other properties like layouts. So there is definately a level of automation in both tools. The quality of visual notation is highly dependable on the user or creator of the models rather than relying solely on the tool itself.[5]

5. Case study

Applying the power of modeling to electrical systems. More specifically home installations as they lend themselves to be modeled quite easily.

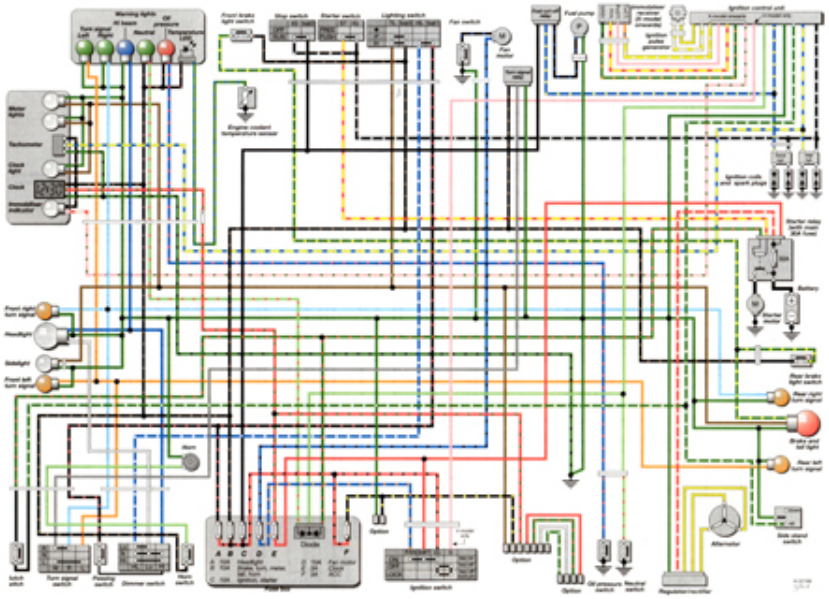

Figure 6: A simple representation for an electrical wiring system.

Electrical installations consist of many connected components which could all be modeled within **Sirius**. These include, but are not limited to: cables, switches and outlets.[6] **Sirius** is the appropriate tool for the job because of the ease of multiple representations. This allows us to have a view which corresponds to the official notation and one which is more life-like. Tables can be used to summarize the system in questions and view its properties element-wise.

Formal standards exists to validate whether a system is up to specifications, the A.R.E.I[4]. Requirements could be collected from these standards and implemented directly into the model checking utilities.

Examples of possible things to check.

- Cable colors: Certain cable colors are prohibited while others are reserved for a special application only.

[6] `Imagefromhttp://riwatt.be/wp-content/uploads/2016/06/scheme1.jpg`

- Short circuits.
- Allocated distribition points per wire.
- Fuse ameprage.

Here we also have quite a lot more options.

The last part of the case study would involve model transformations and/or code generation. We could create a simulator to simulate actions and reactions, convert the system to a different formalism so it can be analyzed, convert it to different wiring systems (Centralized, decentralized) or perform cost optimization.

6. Conclusion

In conclusion we can see how Model Driven Engineering tools can be used to grahically model complex systems. We have explored some aspects of **Sirius** and are now more awre of the choices available to us when in need of a graphical modeling tool. We also have explored how **Sirius** can be used in the specific case of modeling electrical systems.

References

[1] Sirius.
URL `http://www.eclipse.org/sirius/`

[2] M. Porhel, Sirius Forum.
URL `https://www.eclipse.org/forums/index.php/t/1070145/`

[3] M. Porhel, Sirius Documentation.
URL `https://www.eclipse.org/sirius/doc/developer/Architecture_Overview.html`

[4] E. Syriani, H. Vangheluwe, R. Mannadiar, C. Hansen, S. V. Mierlo, H. Ergin, Atompm: A web-based modeling environment.

[5] A.-V. vzw, A.R.E.I Vincotte.
URL http://www.epc-platform.be/files/arei-beknopt-vincotte.pdf

[6] B. . Decker, The complete guide to wiring, sixth Edition, Cool Springs Press, 2014.