

Sirius modeling tool use in electrical system applications

Arkadiusz Ryś

University of Antwerp, Belgium

Abstract

Sirius¹ 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. Additionally, model transformations and code generation can be achieved using plugins. I will apply **Sirius** to a use-case modeling electrical systems. Consequently, I will compare **Sirius** against other graphical modeling tools. **Sirius** has the added benefit of allowing users to work directly within the graphical representation of the generated models [4]. Therefore, the goal of this paper is to create an overview of the functionalities of **Sirius** and compare it to other known modeling tools.

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 visualizations themselves. 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. I will discuss how, where and why such an application is viable.

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

¹Sirius can be found at <http://www.eclipse.org/sirius/>

The rest of the publication will be split into the following sections. Section
10 2 will elaborate more on the details of the architecture on top of which **Sirius**
is designed. The many features and capabilities of **Sirius** will be presented
in section 3. Section 4 follows, comparing **Sirius** to other tools with section
5 explaining how I will apply **Sirius** to my problem. I give application
examples in section 6. Section 7 finally concludes.

15 2. Architecture

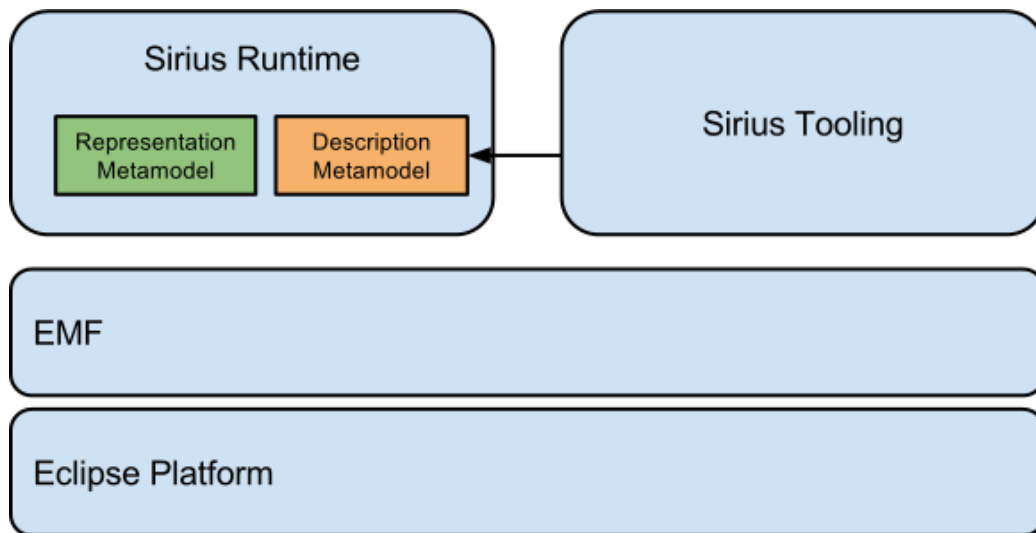


Figure 1: Sirius architecture model overview.

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 the user to extend the functionality of **Sirius** by installing
more Eclipse plugins which could aid in **model transformations** or **code**
20 **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

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

design the Ecore meta-models. Editors can be generated to edit stored data textually and **Sirius** extends these capabilities by allowing to edit the data within the diagrams themselves. The EMF layer is where model transformations happen. One of such plugins is **Viatra**³.

Composition. At the highest level you can see how the *Sirius tooling* is split from the *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 can interact with the models.

This is not the only way **Sirius** separates 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. GMF tooling was used to initialize the GMF code to manipulate the internal **Sirius** diagram model. However, currently the generated code and GMF tooling are no longer in use [5, 6].

3. Capabilities

Sirius supports five representations out of the box:

- Diagrams
- Sequence Diagrams
- Tables
- Trees

³<https://www.eclipse.org/viatra/>

- Properties view

The difference with AToMPM is how **Sirius** has more than only diagrams.
55 **Sirius** allows having a combination of these representations in a single project.
More specifically, multiple representations of the same type can be created to
show different aspects of the model. Users who want to create a completely
new representation have that option as well.

3.1. Diagrams

60 Diagrams are very versatile.

Sirius has a vast amount of diagram options. I will therefore, cover the
most interesting options for the electrical use-case.

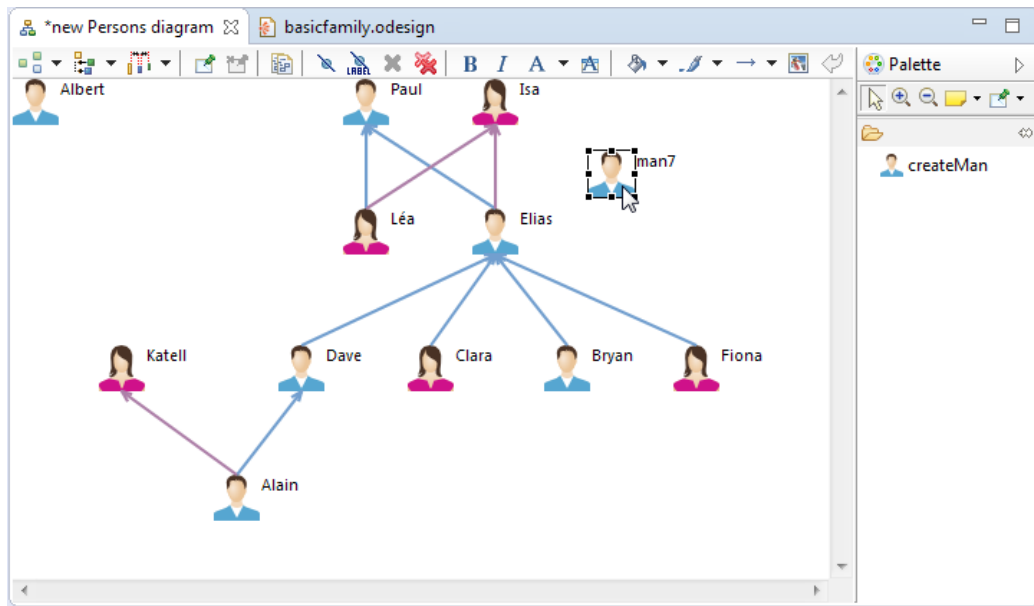


Figure 2: Example of a diagram in **Sirius**.⁴

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

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

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.

Tools. I 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
 70 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 just disabling whole layers of elements.

Validation. The model can be validated when required. Rules have to be set
 75 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

	Nb persons	Father	Mother	Nb children
France	3			
Paris	3			
Dupont House	3			
Dupont Jacques				1
Dupont Marc		Dupont Jacques	Dupont Michelle	0
Dupont Michelle				1
Germany	0			
Berlin	0			
U.S.A	5			
Chicago	3			
Smith House	3			
Smith John				1
Smith Jack		Smith John	Smith Jane	0
Smith Jane				1
Los Angeles	2			
Johnson House	2			
Johnson Earvin				0
Johnson Mary				0

Figure 3: Example of an edition table in **Sirius**.⁵

80 **Sirius** allows to define tables. These tables create the option to edit the data within a compact overview which at times will be faster than fiddling with a diagram. There are 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.
- 85 (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 represent elements with the corresponding cell checked when a relationship between them exists.

3.3. Trees

90 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 are not deleted implicitly.

3.4. Overview

Users familiar with Eclipse will recognize the layout of the **Sirius** work-
95 bench. As **Sirius** allows many views or representations of the same data. Consequently, editing the data in one view will propagate the change to all other views. 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.

100 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 the creator defined). At the top in the menu (green) there are some general
105 options. The behavior when they add, delete or perform any other operation is also defined by the person who designed the model.

⁵<https://www.eclipse.org/sirius/doc/specifier/Sirius%20Specifier%20Manual.html>

⁶<https://www.eclipse.org/sirius/doc/specifier/Sirius%20Specifier%20Manual.html>

⁷<https://www.eclipse.org/sirius/doc/specifier/Sirius%20Specifier%20Manual.html>

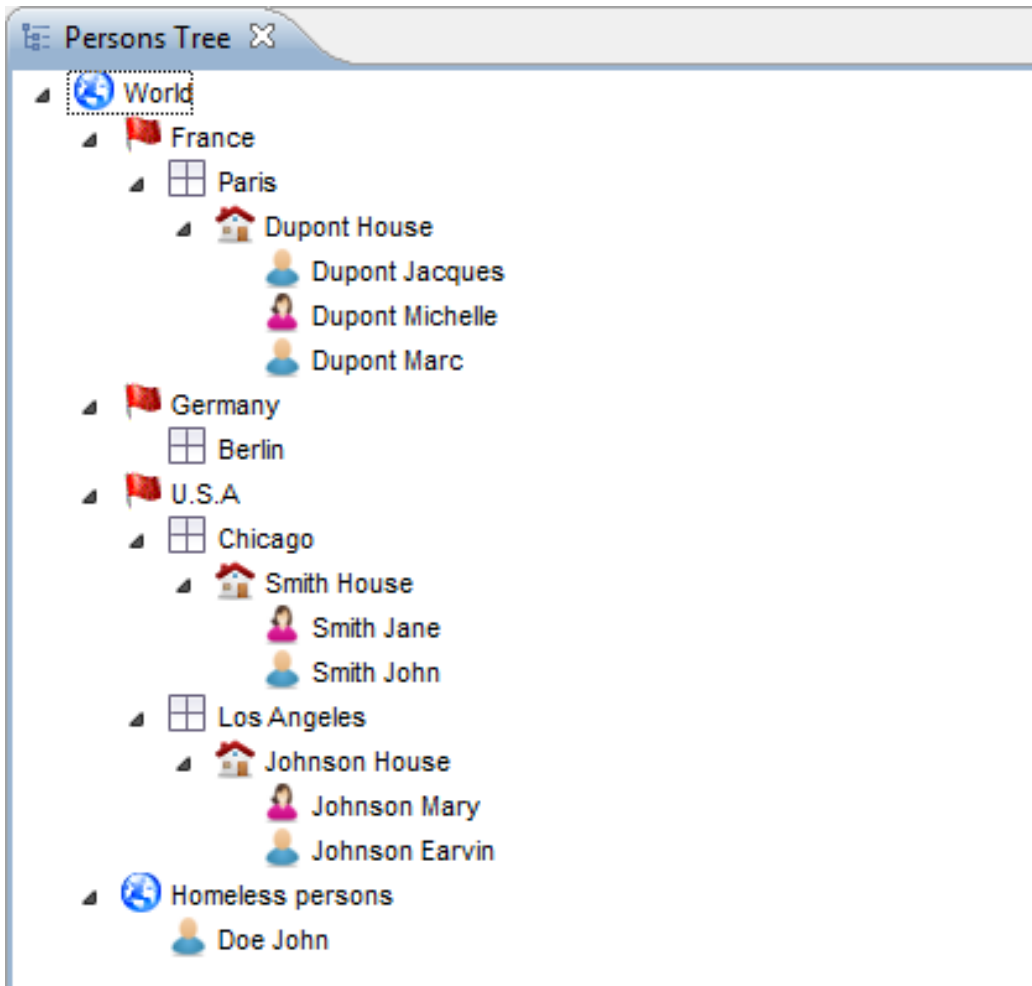


Figure 4: Example of a tree view in **Sirius**.⁶

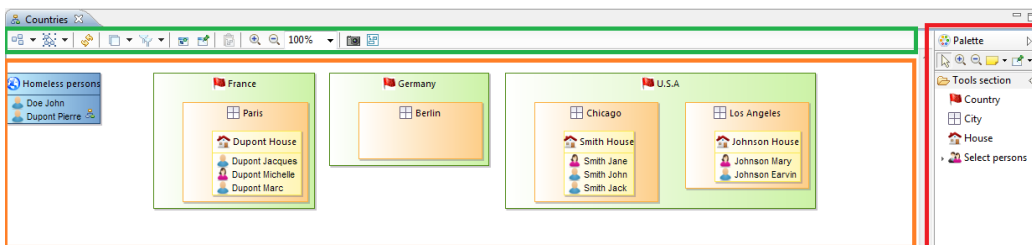


Figure 5: The interface the end-user would see for diagram editing.⁷

4. Comparison

As **Sirius** is a graphical modeling tool it is best to compare it to other tools in the same category. This is why I have chosen to compare it with
110 **AToMPM**. The distributed nature of **AToMPM** [7] versus the Eclipse based **Sirius** is hard to compare as both have their specific qualities.

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
115 AP.. 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
120 **AToMPM**.

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

While the tools differ in the previously mentioned properties they both
125 make a distinction between abstract and concrete syntax. A lot happens behind the scenes in both tools, **AToMPM** will do the RAMification for you and **Sirius** can automate other properties like layouts. So there is definitely 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
130 the tool itself [1].

5. Case study

For the case study a project consisting of multiple parts will be designed and realized. This will include the creation of user friendly tools which will provide any required operation on the model. These tools and models will
135 be designed with *D. L. Moody* his work on notations in mind [3].

The reason for defining a visual rule language is so the user can have a better grasp and overview of what is being done. This will also help users who are not familiar with the **Xtend** language.

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

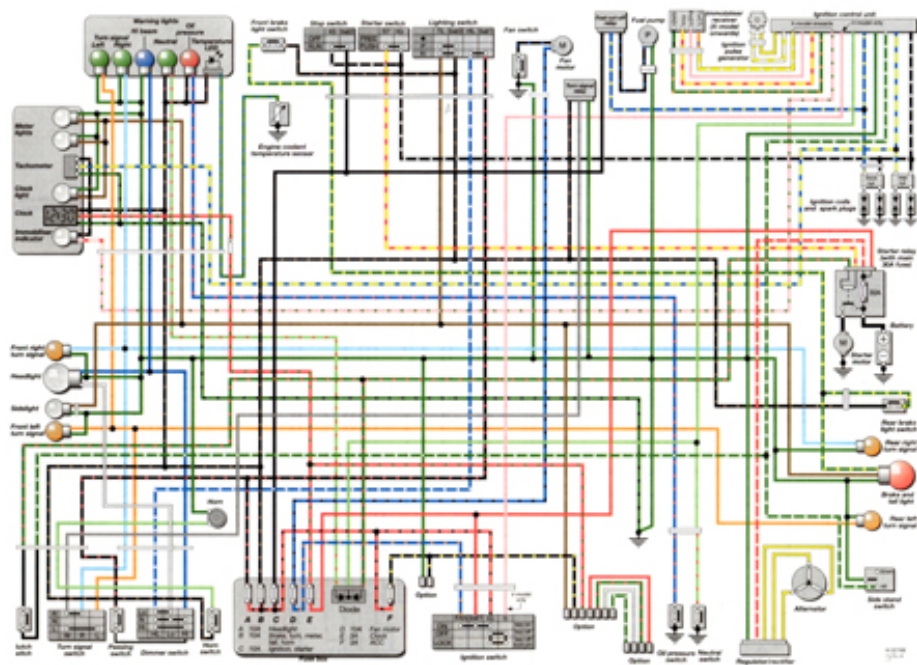


Figure 6: A simple representation for an electrical wiring system.

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 [2]. **Sirius** is the appropriate tool for the job because of the ease of multiple representations. It is possible to have a view which
145 corresponds to the official notation and one which is more life-like. Tables can be used to summarize the system in question and view its properties element-wise.

Guidelines, including the A.R.E.I [7], exist to validate whether a system is up to specifications. Requirements could be collected from these standards
150 and implemented directly into the model checking utilities.

Examples of possible things to check. There are many possible options which can be checked for validity. I will give a non-exhaustive list of examples to illustrate a few cases.

- Cable colors: certain cable colors are prohibited while others are reserved for special applications only.
155
- Short circuits.
- Allocated distribution points per wire.
- Fuse amperage.

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

6. Application

The current meta-model, as seen in figure 7, is rather simple. This is
165 because creating all the required validation and tooling needs to be done by hand. The main classes in the meta-model are:

- Plan: holds the entirety of the electrical circuit.
- Wire: used to connect components.

⁷Imagefrom<http://riwatt.be/wp-content/uploads/2016/06/scheme1.jpg>

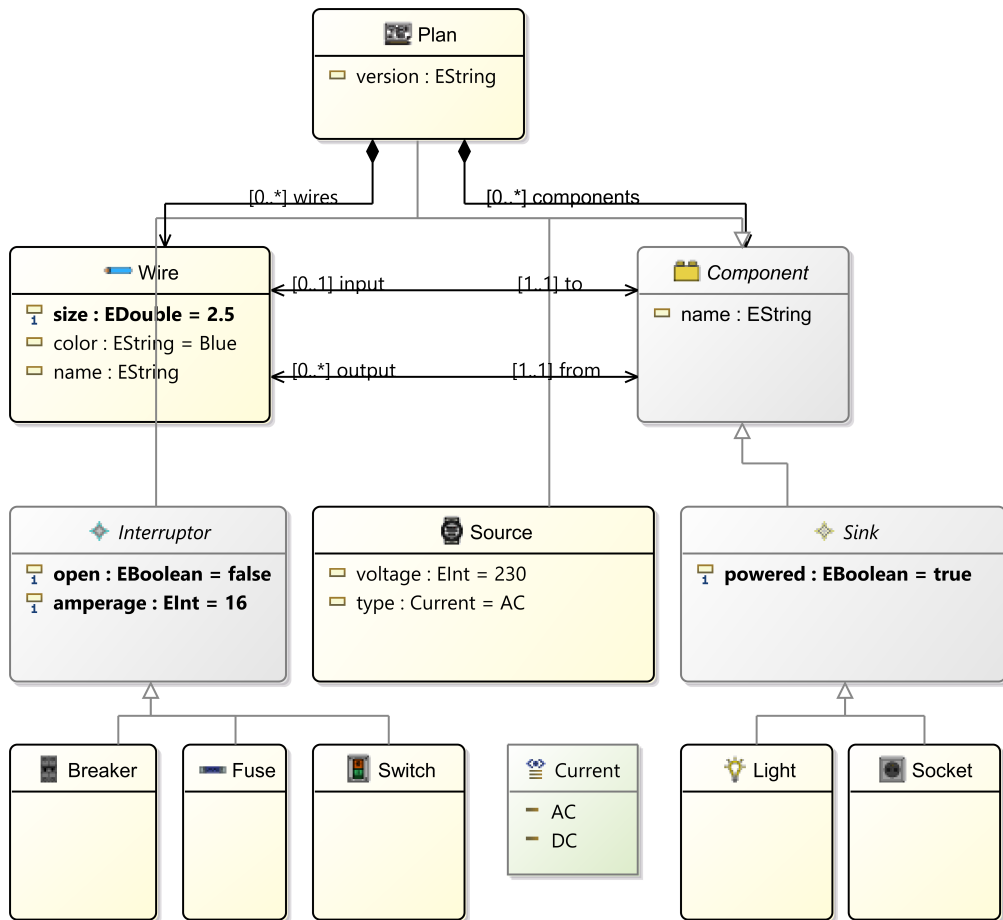


Figure 7: Ecore meta-model of the electrical installation.

- 170
- Component: a superclass encompassing any possible electrical components.

A model can be built using these three classes. To allow more options the Component class is divided into six specialized classes. The current model

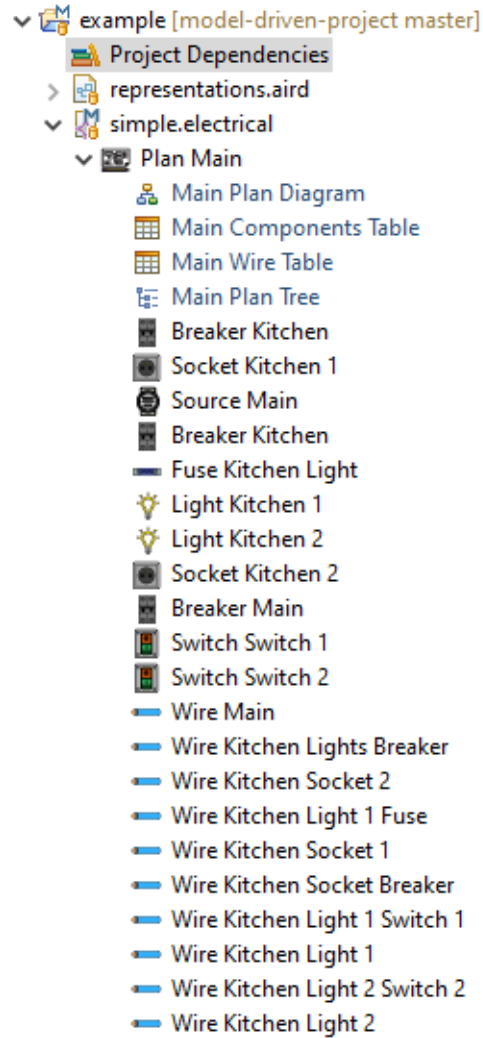


Figure 8: Items used for the examples which will be displayed as diagrams, trees and tables.

175 already showcases many Eclipse modeling framework and Sirius features. The ability to create, validate, filter and define tools for each instance of the

meta-model is implemented. Examples for diagrams, trees and tables are shown in figures 9, 10, 12 and 11. Not only are all CRUD operations for each class defined, but any error which does not pass validation is given an appropriate action which will automatically fix mistakes. When an operation is too hard to implement using the AQL language Sirius uses for accessing model information, plain Java can be used. Figure 13 showcases the use of Java services for harder operations. A lot more can be achieved and

Figure 9: Electrical plan diagram representation.












	Name	Amperage	Open	Powered
 Kitchen	Kitchen	20	<input type="checkbox"/> false	
 Kitchen 1	Kitchen 1			<input checked="" type="checkbox"/> true
 Main	Main			
 Kitchen	Kitchen	16	<input type="checkbox"/> false	
 Kitchen Light	Kitchen Light	2	<input type="checkbox"/> false	
 Kitchen 1	Kitchen 1			<input checked="" type="checkbox"/> true
 Kitchen 2	Kitchen 2			<input checked="" type="checkbox"/> true
 Kitchen 2	Kitchen 2			<input checked="" type="checkbox"/> true
 Main	Main	40	<input type="checkbox"/> false	
 Switch 1	Switch 1	16	<input type="checkbox"/> false	
 Switch 2	Switch 2	16	<input type="checkbox"/> false	

Figure 10: A table containing an example of components and calculated attributes.

this model is just a proof of concept. Model transformation can be used to create more complex and closer to real life version of the diagrams. The transformations could be achieved by using another Eclipse plugin called Viatra. If the model needs to simulated **GEMOC**⁸ can be used.

⁸<https://projects.eclipse.org/proposals/eclipse-gemoc-studio>

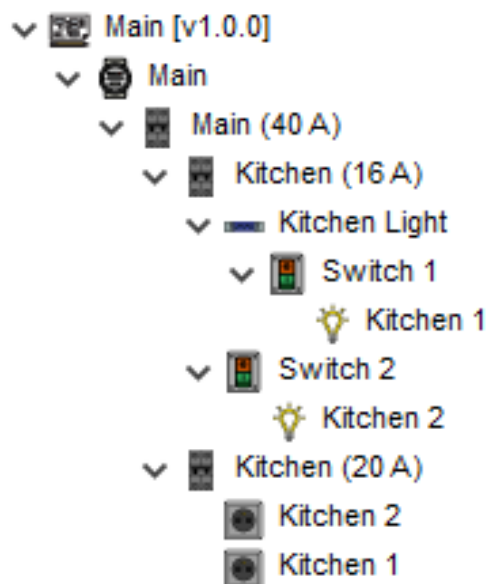


Figure 11: A tree-like view of the electrical plan.

	Name	Color	Size	Max Amps	From	To	Voltage	Voltage Type
— Main	Main	Blue	4.0	40.0	Main	Main	230	AC
— Kitchen Lights Breaker	Kitchen Lights Breaker	Blue	4.0	40.0	Main	Kitchen	230	AC
— Kitchen Socket 2	Kitchen Socket 2	Blue	2.5	25.0	Kitchen	Kitchen 2	230	AC
— Kitchen Light 1 Fuse	Kitchen Light 1 Fuse	Blue	1.5	15.0	Kitchen	Kitchen Light	230	AC
— Kitchen Socket 1	Kitchen Socket 1	Blue	2.5	25.0	Kitchen	Kitchen 1	230	AC
— Kitchen Socket Breaker	Kitchen Socket Breaker	Blue	4.0	40.0	Main	Kitchen	230	AC
— Kitchen Light 1 Switch	Kitchen Light 1 Switch 1	Blue	1.5	15.0	Kitchen Light	Switch 1	230	AC
— Kitchen Light 1	Kitchen Light 1	Blue	1.5	15.0	Switch 1	Kitchen 1	230	AC
— Kitchen Light 2 Switch	Kitchen Light 2 Switch 2	Blue	1.5	15.0	Kitchen	Switch 2	230	AC
— Kitchen Light 2	Kitchen Light 2	Blue	1.5	15.0	Switch 2	Kitchen 2	230	AC

Figure 12: A second table, this time for wires.

```

public List<Component> getDisconnected(Plan plan) {
    List<Component> disconnected = new ArrayList<Component>();
    List<Component> components = plan.getComponents();
    for (Component component: components) {
        if (!this.connectedToSource(component)) {
            disconnected.add(component);
        }
    }
    return disconnected;
}

public Source getConnectedSource(Component component) {
    if (component instanceof Source) {
        return (Source)component;
    } else if (component.getInput() == null) {
        return null;
    }
    Component next_component = component.getInput().getFrom();
    return getConnectedSource(next_component);
}

```

Figure 13: Example of Java based services.

7. Conclusion

In conclusion I can see how *model driven engineering tools* can be used to graphically model complex systems. The ease of use and pace of creations allows for rapid prototyping. I have explored some aspects of **Sirius** and can confirm that its application to domain specific problems is helpful. I also have explored how **Sirius** can be used in the specific case of modeling electrical systems by designing a prototype wiring system.

References

- [1] Aib-Vincotte, . A.R.E.I Vincotte. URL: <http://www.epc-platform.be/files/arei-beknopt-vincotte.pdf>.
- [2] Decker, B., 2014. The complete guide to wiring. sixth ed., Cool Springs Press.
- [3] Moody, D.L., 2009. The physics of notations: Toward a scientific basis for constructing visual notations in software engineering. IEEE Transactions on Software Engineering 35, 756–779.
- [4] Obeo, . Sirius - The easiest way to get your own Modeling Tool. URL: <http://www.eclipse.org/sirius/>.
- [5] Porhel, M., a. Eclipse Community Forums: Sirius > Is Sirius based on GMF? URL: <https://www.eclipse.org/forums/index.php/t/1070145/>.
- [6] Porhel, M., b. Sirius Documentation Architecture Overview<. URL: https://www.eclipse.org/sirius/doc/developer/Architecture_Overview.html.
- [7] Syriani, E., Vangheluwe, H., Mannadiar, R., Hansen, C., Mierlo, S.V., Ergin, H., . Atompm: A web-based modeling environment .