Modeling with Sirius; a language for home automation systems

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 which are based on these meta-models. Once a model is created we can validate whether constraints are met or even generate code. An added benefit of **Sirius** is the ability for 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 show how it can be applied with the use of other technologies to design home automation systems.

Keywords: Model Driven Engineering, Sirius, Eclipse modeling Framework, IoT, Home automation

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 home automation systems. We will discuss how, where and why such application is viable.

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

[1] Sirius can be found at `http://www.eclipse.org/sirius/`

Preprint submitted to University of Antwerp *December 15, 2017*

The rest of the publication will be split in the following sections. 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 section *6* explaining how we will apply **Sirius** to our problem. The choice of candidate for this problem is described in section *4*. A few things were done in order to help the project start off in the right direction, these can be found in section *5*. Section *7* finally concludes.

2. Architecture

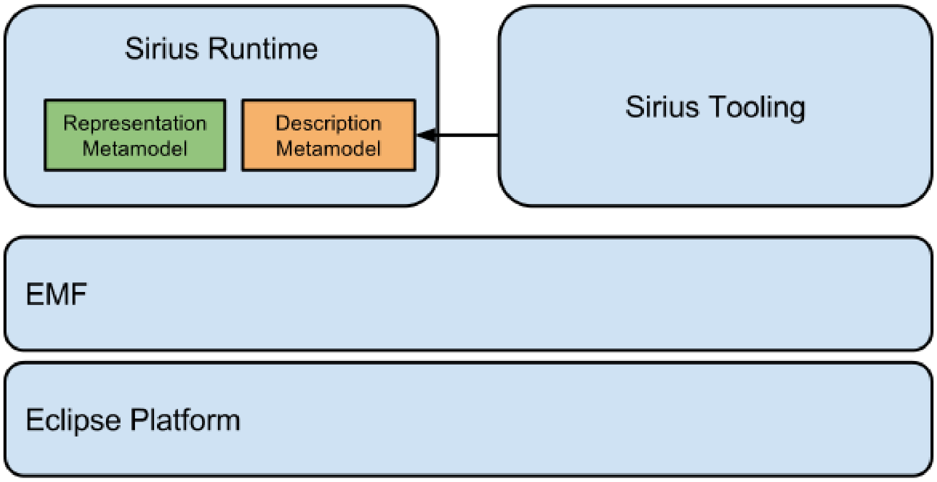


Figure 1: Sirius architecture model overview.

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**.

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

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 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 but now the generated code and GMF tooling are not used anymore.[2, 3]

3. Capabilities

Sirius supports five representations out of the box:

[3] `https://www.eclipse.org/viatra/`

- Diagrams
- Sequence Diagrams
- Tables
- Trees
- Properties view

Sirius allows us to have a combination of these representations in a single project, you can even have multiple representations of the same type or create a completely new representation.

3.1. Diagrams

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

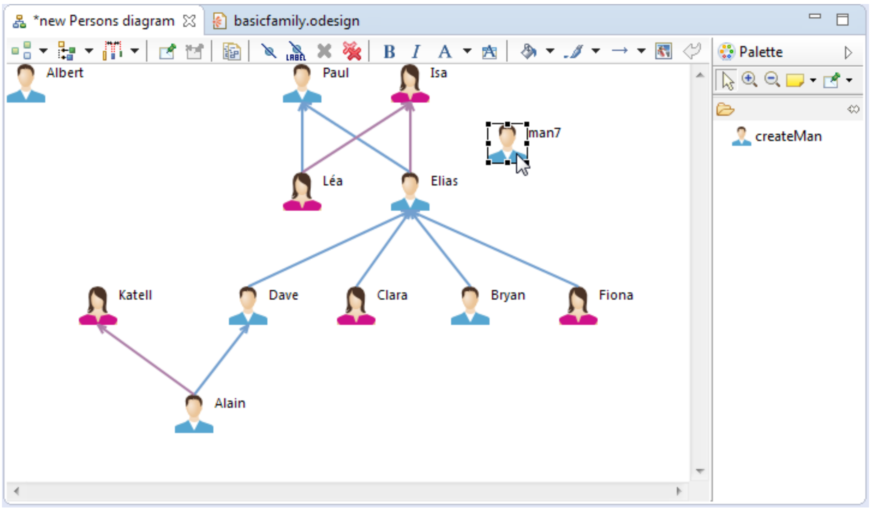


Figure 2: Example of a diagram in **Sirius**.[4]

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

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.

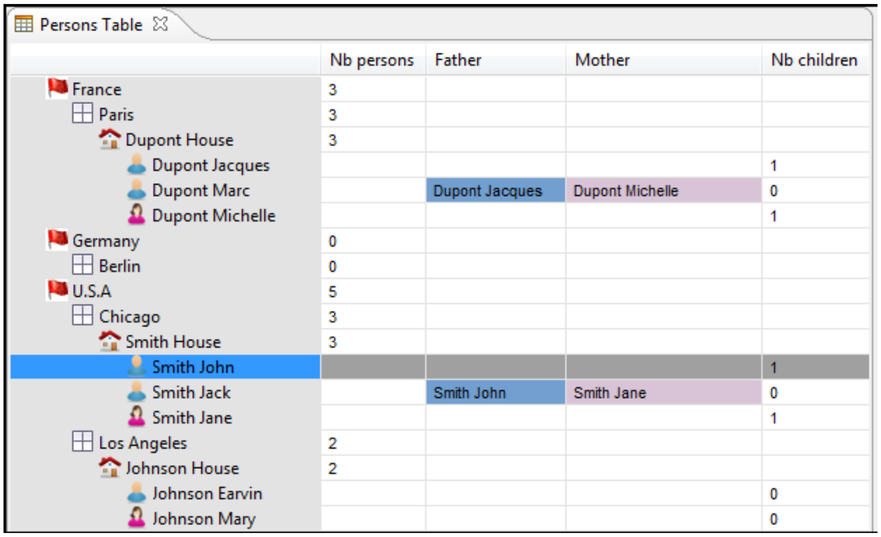
Styling. Every aspect of the diagram can be styled. Styles can be conditional. For example: weighted edges with a weight higher than 5 can be turned red.

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 just disabling whole layers of elements.

Validation. The model can be validated when required. Rules have to be set before the validation can take place.

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



	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]

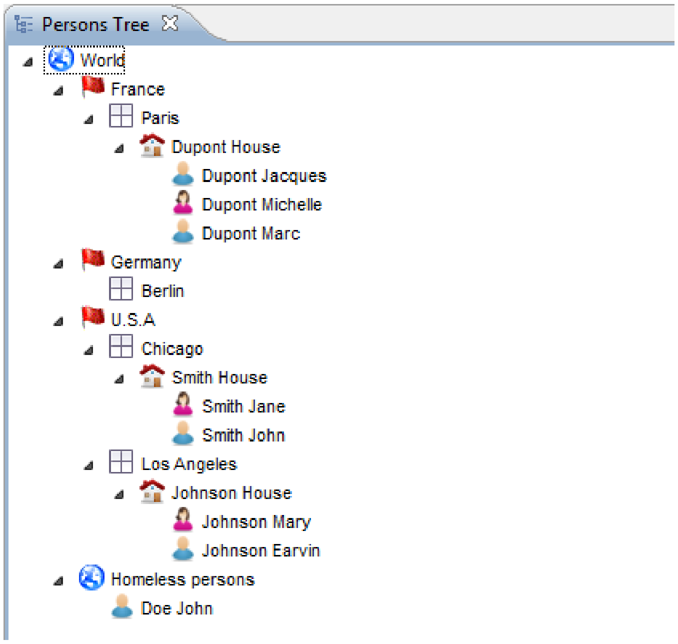
Sirius allows to define tables. These give us the option of editing the data within a table which at times will be faster than fiddling with a diagram. We have two types of tables within **Sirius**.

(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.

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

3.3. Trees



- World
  - France
    - Paris
      - Dupont House
        - Dupont Jacques
        - Dupont Michelle
        - Dupont Marc
  - Germany
    - Berlin
  - U.S.A
    - Chicago
      - Smith House
        - Smith Jane
        - Smith John
    - Los Angeles
      - Johnson House
        - Johnson Mary
        - Johnson Earvin
  - Homeless persons
    - Doe John

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

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** workbench. As **Sirius** allows many views or representations of the same data we can edit the data in any of 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.

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

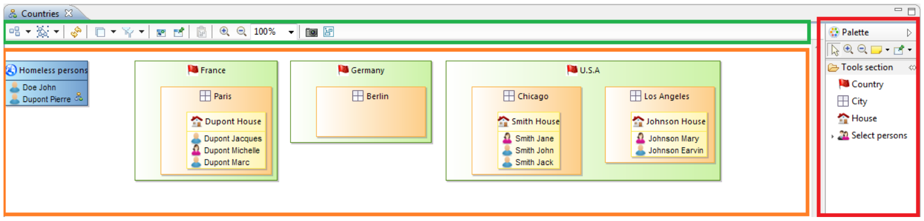


Figure 5: The interface the end-user would see for diagram editing.[7]

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. Candidates

In order to create a model we need to research some home automation platforms first. The ones covered in this publication are:

- OpenHAB[4]
- If this then that[5]
- Home Control Assistant[6]

The first draft included ***Allen*** *patterns*[7], this is why we'll also mention their support by any specific platform. This is rather easy as only **OpenHAB** allows us to define such patterns. Although not in a very user friendly manner. Furthermore it is also the only open source alternative which supports a concise textual definition of rules.

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

5. Preparation

In order to get a general idea on how to create a good model and the workings of event processing, two resources were consulted. The first is a publication about complex event processing[8]. The second is a work package showcasing the design decisions and tools used for problems solved by complex event processing[9].

6. Case study

For the case study a project consisting of multiple parts will be designed and realized. A formalism dedicated to modeling the textual rule and action input of **OpenHAB** will be created. This will include the creation of user friendly tools which will provide any required operation on the model. These tools and models will be designed with *D. L. Moody* his work on notations in mind[10]. Then a model transformation will be defined which will be a able to generate code for both the *application condition* and *action*. This will be done using **Viatra**. The application condition should also be importable from the **Xtend** textual definition which **OpenHAB** utilizes and perhaps even allow analysis.

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 which are not familiar with the **Xtend** language.

7. Conclusion

In conclusion we can see how *model driven engineering tools* can be used to graphically model complex systems. We have explored some aspects of **Sirius** and are now more aware 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 home automation systems for use in OpenHAB.

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] OpenHAB.
URL https://www.openhab.org/

[5] IFTTT.
URL https://ifttt.com/

[6] Home Control Assistant.
URL http://www.hcatech.com/

[7] T. A. Alspaugh, Allen's Interval Algebra.
URL https://www.ics.uci.edu/%7Ealspaugh/cls/shr/allen.html

[8] I. Dávid, I. Ráth, D. Varró, Foundations for streaming model transformations by complex event processingdoi:10.1007/s10270-016-0533-1.

[9] SocEDA Work packages and deliverables, work package 4.
URL https://research.linagora.com/display/soceda/Work+packages+and+deliverables#wp4

[10] D. L. Moody, The physics of notations: Toward a scientific basis for constructing visual notations in software engineering, IEEE Transactions on Software Engineering 35 (6) (2009) 756–779.