# Model Driven Approach for Electrical Circuit Modeling

MoDELCiM

Hanifi Güneş
Department of Electrical and Electronics Engineering
Bilkent University
Ankara, Turkey
gunesi@ug.bilkent.edu.tr

***Abstract*— 'Schematic Capture' programs have been offering very divergent set of capabilities and thus, mostly being chosen by electrical engineers in accordance with the provided capabilities. They, however, are not interoperable with each other and focus purely on the visual aspects of the electrical components rather than their behavioral features leading a distortion from the original domain.**

**This paper aims to create a DSL for electrical circuits. In so doing, interoperability between the tools will be assured and the behavioral aspects of them will be fostered. Two separate approaches, based on MOF and UML2 Profiling are employed to define the DSL. Then, model-to-text and model-to-model transformations are applied to demonstrate the interoperability and the consolidated behavioral aspects.**

***Keywords-Model-driven approach; electrical circuit modeling; automated circuit modeling; electrical circuit simplification; UML; MOF; model-to-text transformation; model-to-model transformation; openArchitectureWare; ATL; XPand***

## I. INTRODUCTION

Integrated circuit technologies have been playing a crucial role for people's everyday lives. While increasing demand for electrical instruments has been making electrical circuits an inevitable part of today's world, so have they been making the design and implementation of electrical circuits more complex than ever before. It is not for electrical engineers have a vendetta against computer-aided design (CAD) tools but since the tools are not interoperable and intelligible with each other thus, forcing engineers to put a separate effort to grasp them first and then to model exactly the same circuit using these tools. This syndrome is mainly because there is no concrete electrical circuit modeling standard that has been out-of-the-shelf yet. To this end, modeling of electrical circuits seems inevitable from a systemic point of view.

Likewise, schematic capture tools, today, focus on the visual aspects of the electrical components -that is concrete syntax- rather than their behavioral features. Instead of taking, for instance, a resistor as an electrical component with its own special attributes, today's CAD tools simply call it as a drawing or a rectangular block shown on the screen. This leads a serious shift from the problem domain and signifies the absence of high level of abstraction and thus, of high level of modeling, which multiplies complexity and slashes down the productivity and reusability.

For resolving the aforementioned problems, a Domain-Specific Language will be realized in this project as DSL's are quite beneficial as they help cover the original problem domain concepts better [1]. Electrical circuits, however, meet a very wide range of separate needs. This widely varying nature of electrical circuits makes it virtually impossible to define a meta-model that fits every single need well. This emphasizes the point that the language engaged in must be domain-specific as much as possible. Because of this reason, this study simply works on defining a domain specific language for electrical circuits in general, yet not especially for filter or antenna circuits, to name a few, that necessitate comprehensive knowledge to be employed and should have their own enhanced domain specific languages defined.

To increase readability of the circuit models and possibly to provide a common ground for interoperability purposes, model-to-text transformation will be applied to existent models. Given any model, the transformation will map the model to an XML file. The XML file then can be used by various schematic capture tools to produce exactly the same circuit model. Next, model-to-model transformation will be used to simplify a given circuit in order to *minimize the number of components used* in the circuit. Here plenty of different strategies can be utilized while simplifying the circuit such as component based reduction, cost based reduction, overall power consumption based reduction and so forth. We however, opted to work with component based reduction as it is the simplest to demonstrate and to be captured.

The following sections will be reserved to address these issues in detail. The next section solely focuses on domain analysis while the latter ones will mention mapping of concepts to the grammar, the definition of meta-model including abstract syntax, static semantics and concrete syntax respectively. Then, we will go into model-to-text, model-to-model transformations and finalize the paper with the conclusions taken out of this study.

## II. ELECTRICAL CIRCUITS

Model-Driven Software Engineering brings in the notion that models should capture domain-specific knowledge, which yields an increase in productivity. The idea behind is quite simple: instead of forcing programmers to translate their domain-specific knowledge every time needed, build a language/model for the domain then using translators to end up with artifacts [2]. However, not everything is as easy as it sounds. Defining a meta-model from scratch requires extensive domain knowledge. As separation of concerns lying at different levels are of more than crucial in model-driven architecture.

Likewise, a meta-model, i.e. model of models must form a basis for any model in its domain seamlessly. This factor actually increases the significance and the complexity of domain analysis process especially for loosely defined domains, whose models always have the risk of nonconformance with its meta-model at any time during its design or even worse while the meta-model is somehow in operation.

Fortunately, any electrical circuit incorporates the notion of separation of levels in its body very well. In other words, the elements at M1 and M2 seem no confusing.

To better span circuits as much as possible, we checked some electronic circuit design books and googled some key words regarding to electrical circuit design basics. Standing on my background in electrical engineering, after doing some meta-thinking over the data observed at the previous stages, we came up with a primitive meta-model and tried to validate it against all the circuits we found earlier and iteratively improved the model. The very latest version of the meta-model and the concepts extracted from the domain will be covered in the following chapter.

Electrical circuits, today, are widely used in different areas to meet the demands of very divergent customers. Although they have a well-defined modeling schema and concrete syntax, there is no interoperability between electrical circuit modeling tools. Even electrical circuits are used in different applications as separate entities of the system. Hence, constructing a meta-model for electrical circuits is necessary to determine precisely what an electrical circuit means in different application domains. To extract the domain concepts from the domain, a recursive approach is followed as below:

- An Electronic circuit is by definition a closed path formed by the interconnection of electronic components through which an electric current can flow [3].
- An electrical network is an interconnection of electrical elements such as resistors, inductors, capacitors, transmission lines, voltage sources, current sources, and switches [4].
- An electrical circuit is a network that has a closed loop, giving a return path for the current. A network is a connection of two or more components, and may not necessarily be a circuit [4].
- A network that also contains active electronic components is known as an electronic circuit [4].

The concepts can be re-defined as follows:

1. *Circuit*: A closed loop combined by electronic components.
2. *Network*: Connection of two or more electronic elements not having an active component!
3. *Electronic component*: Basic electronic element with two or more connecting leads.
4. *Active component*: Have gain and/or directionality e.g. semiconductor devices [5].
5. *Passive component*: Have neither of gain nor directionality [5]. e.g. ordinary electrical elements such as resistor and lamp

The obvious model elements from definitions above are circuit, network and component. After a commonality analysis, one can say that the model will contain base *Component* and *Port* elements.

A *network* is a container *component* consisting of *passive components* and just like networks a *circuit* is a set of *components* containing at least one *active component*. *DataFlow* in the model stands for the connection in between two components. In typical electrical circuits, it simply corresponds to wire. For more complicated examples, however, we cannot directly say that.

## III. DOMAIN SPECIFIC LANGUAGE

Among alternate ways of expressing domain model, BNF and EBNF, EBNF is opted to textually visualize the resultant domain model.

*1)* Circuit ::= Components Connection {Feature}*
*2)* Components ::= {Component}[1..*]
*3)* Connection ::= {DataFlow}[1..*]
*4)* Component ::= Network | ActiveComp | PassiveComp {Feature}*
*5)* Network::= PassiveComponents Connection
*6)* PassiveComponents ::= {PassiveComp}[2..*]
*7)* DataFlow ::= SourcePort DestinationPort
*8)* SourcePort ::= Port
*9)* DestinationPort ::= Port
*10)* Port ::= Markers| InputPort | OutputPort

To better explain the grammar, we can say that a Circuit is a combination of 'Component's interconnected via DataFlow connections. A Component can either be of type Network, ActiveComponent or PassiveComponent. To extensibly describe a Circuit or a Component, we use 'Feature's that are simple name-value pairs. A Network is a Circuit like container except that it cannot have any ActiveComponent whereas a Circuit might have. A PassiveComponent complying with the earlier definition is the one having no gain or directionality while an ActiveComponent stands for the vice-versa.

The above grammer checked against sample models chosen manually. Unfortunately there is no easy way to that yet one can still manually plug any model and monitor its conformance to the grammar.

Apparently graphical representations when compared to the textual ones are much more powerful in that they are easy to read and understand, meaning that their expressiveness are much better than that of the textual ones.

## IV. META-MODELS

### A. *Meta-Model From Stratch/MOF*

For graphical modeling, we prefered to work with TopCased [6], all in one modeling platform built upon Eclipse [7]. There, one can easily define a meta-model provided that the whole set of existing meta-model elements conform to definitions in ECore package. For our case, elements like Circuit, Network, Component, DataFlow, Port and their children conform to ECore::EClass. The meta-model built upon MOF can be found in Appendix A.

Below meta-model from MOF is shown together with an example model. In the example, all elements rounded by red circle have an instance-of relation with Component element from meta-model. Eminently the whole example is instance of a Network since no ActiveComponent instance is contained in the model. All components are connected via solid-lines that symbolize the DataFlow.

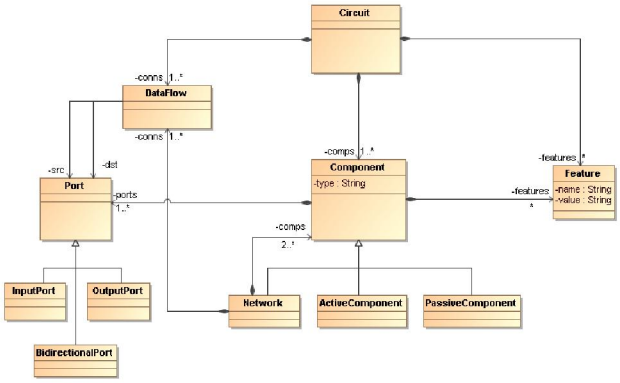


Figure 1. Meta-model from-scracth

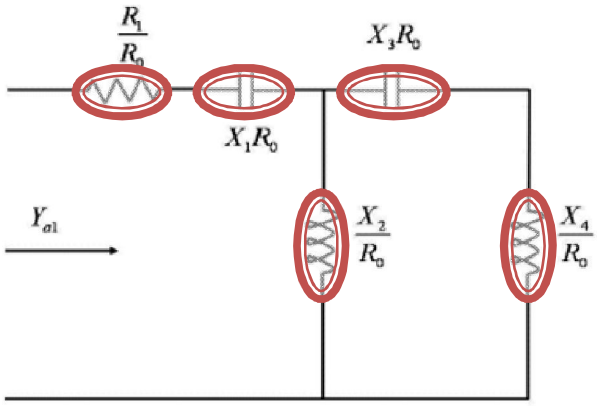


Figure 2. An example model demonstrating meta-model conformance

### B. *Meta-Model Using UML Profiling*

Alternate way of illustrating a meta-model is using UML Profiling mechanism. UML Profiling enables us to customize the way we define our models in regard to the any application domain using stereotypes, tagged values and constraints effecting specific elements in a model [8].

The UML profiling is performed using Lightweight extension of UML 2. We created profiles/stereotypes and applied them to the appropriate model elements. Stereotypes are noteworthy here, because they let us to assign labels, constraints, icons and properties to our model elements. Please refer to Annexes Figure II for the completed model. To complete our study, the proceeding chapters will deal with static semantics and concrete syntax.

An example illustrating the usage of UML stereotypes and circuit model corresponding to the UML model can be found below. The circuit has three components and three connecting wires in between these components. The overall picture constitutes a circuit because the voltage source seen in the figure is an instance of an active component.

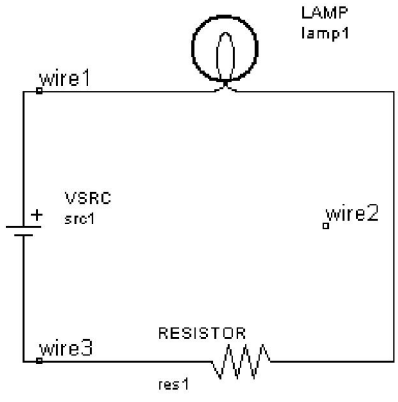


Figure 3. Circuit model for the example demostrated

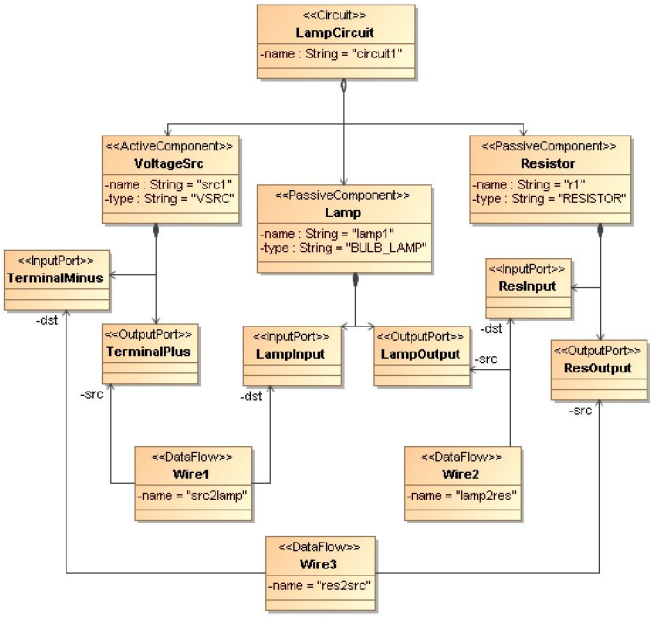


Figure 4. The same circuit using UML2 Profiling

The example above has a circuit element, the root container containing three electrical components whose are stereotyped as either Active or Passive Component. Each element is a two-terminal device meaning that they only have an input and output ports and these ports are bound together via wire instances stereotyped as DataFlow. For simplicity reason, two other stereotypes: Feature and Network are not demonstrated in this example. Feature is an extensibility window to capture the behavioral aspects of any element in an electrical model precisely. The next stereotype, Network as defined previously is a component container consisting of all kinds of components but active component instances.

## V. STATIC SEMANTICS

Abstract syntax itself may not be sufficient to express the nature of a model in all aspects. In this case, we use static semantics in form of OCL [9] to improve and better the expressiveness of abstract syntax. The OCL is a textual, descriptive language used to define constraints. The below short table contains OCL statements used to well-define our meta-model. No further explanation on the statements will be given since they are self-expressive.

1. **Context Circuit:**
   inv: self.comps->select
   (c|c.isKindOf(ActiveComponent))->size()>0

2. **Context DataFlow:**
   inv: self.src->forAll(s|self.dst->forAll(d|s<>d))
   inv: self.src->forAll(s|s.isKindOf(OutpurPort))
   inv: self.dst->forAll(d|d.isKindOf(InputPort))

3. **Context Network:**
   inv: self.comps->select
   (c|c.isKindOf(ActiveComponent))->size()=0

## VI. CONCRETE SYNTAX

Electronic circuits definitely had been existed much far before the notion of meta-modeling was argued. Within these years, electrical engineers shaped how to illustrate a circuit model and well-defined the basic rules and icons for either circuit element. As of late 50s most of the circuit element standards have been put by IEEE and some other institutions to assure a seamless communication between parties.

In this study we will also be using this enhanced circuit modeling standards as to be our concrete syntax. To our understanding, no further study is necessary at the point of concrete syntax first because of fact that since it is well-defined, no need to do this and second as it is widely used in practical world today, the switching cost would be incredibly much, which is not feasible.

A quick list of graphical representation of basic circuit elements are given below. All these components are connected through wires. A wire has presumed to have two terminals that bind two separate 'Port's each other.

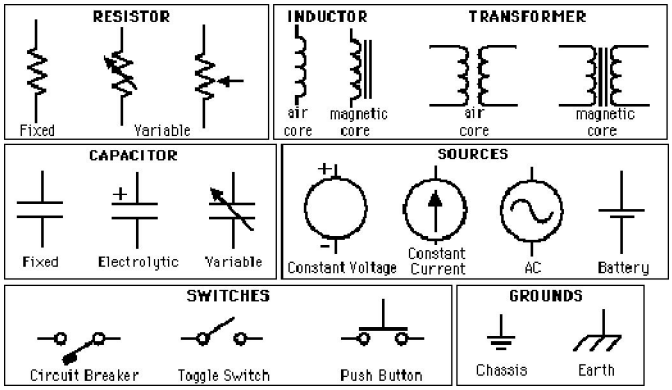


Figure 5. Concrete syntax of basic electrical components

Some examples illustrating the use of concrete syntax will be given soon in this section. Although some slight changes might seen in graphical representations, more or less, all basic electrical components are used in the same way regardless of the tool used.

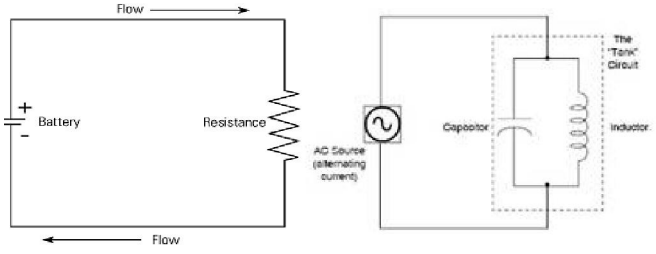


Figure 6. Two basic examples using the concrete syntax

## VII. MODEL TRANSFORMATIONS

### A. *Model to Text Transformation*

Model-Driven approach enables automatic generation of code using transformations at any level. In this case, we aim to create an XML file that can be executed by any CAD tool to produce the same circuit schema. This way interoperability, portability and reusability of circuit models could be fostered across other CAD tools.

The target XML schema is defined far before defining the transformation rules. Since no existent schematic capture tool supports an XML file format for storing the schema info, a new extensible schema for this purpose is determined and defined from scratch. However, we could not visualize the models due to the absence of CAD tools with a mature open format.

The transformation engine used in the study is openArchitectureWare. oAW requires a meta-model defined and a model instance conforming to the meta-model. It first validates the model against check rules that are defined by an OCL-like language and then using a template file generates the target XML file. All these operations are stored in a workflow file that directs oAW engine by instructing which tasks will be executed under which configurations.

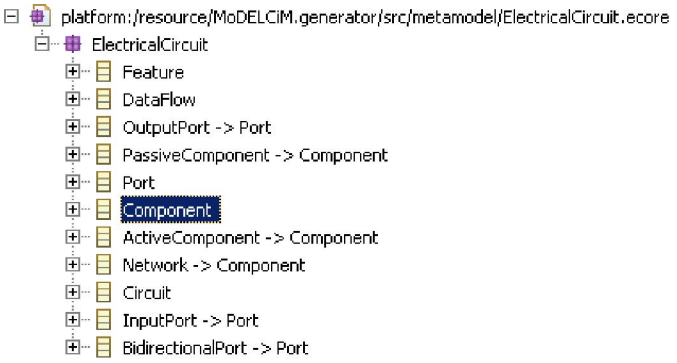


Figure 7. Meta-model used in oAW

oAW's workflow supports a smart model conversion bridge named uml2ecore. Using the adapter one can convert a meta-model in UML2 to ecore. It enables users to use pure UML to design their meta-models. The conversion works seamlessly and helps increase productivity by means of keeping models clear and light. Even more, it automatically generates check files used to validate models at lower levels. A tiny part of the check file is depicted in the figure 8. There are two levels of checks: errors and warnings. Errors stops execution of workflow immediately while warnings reports the message and retain execution thread alive.

```
import ElectricalCircuit;                                    MoDELCiM.generator/src/metamodel/ElectricalCircuitConstrai

extension org::openarchitectureware::util::stdlib::naming;

context Circuit WARNING loc()+" name not specified: ["+metaType.name+"]":
    name != null;
context Circuit ERROR loc()+" comps cannot be empty: "+
                (metaType.getProperty("name") != null ?
                metaType.getProperty("name").get(this): "") +" ["+metaType.name+"]":
    !comps.isEmpty;
context Circuit ERROR loc()+" conns cannot be empty: "+
                (metaType.getProperty("name") != null ?
                metaType.getProperty("name").get(this): "") +" ["+metaType.name+"]":
    !conns.isEmpty;

context Network ERROR loc()+" network must have 2 or more connections: "+
                (metaType.getProperty("name") != null ?
                metaType.getProperty("name").get(this): "") +" ["+metaType.name+"]":
    !conns.isEmpty;
context Network ERROR loc()+" network must contain 'PassiveComponent's only: "+
                (metaType.getProperty("name") != null ?
                metaType.getProperty("name").get(this): "") +" ["+metaType.name+"]":
    comps.forAll(c|c.metaType == PassiveComponent);


context Component ERROR loc()+" components must have unique names: "+name:
    ((Circuit)eContainer).comps.select( c | c.name == name ).size == 1;
context Component ERROR loc()+" components must have unique names: "+name:
    ((Network)eContainer).comps.select( c | c.name == name ).size == 1;
```

Figure 8. A small part of the check file used by oAW

Finally a template file is defined in order for mapping model elements to text segments (XML nodes). A root entry point is mentioned in the workflow file so that transformation engine can initiate transformation properly. As part of the XML template, figure 9 can be given. Xtend is a very handy standard of oAW that betters users' hands. It enables in-line coding, reduces code repetitions so increases reusability and helps separate template from underlying logic.

```
«IMPORT ElectricalCircuit»

«EXTENSION template::XmlGeneratorExtensions»

«DEFINE circuit FOR Circuit»
    «FILE name+".xml" xml»
        <?xml version="1.0" encoding="utf-8"?>
        <circuit name="«name»">
            <properties>
                «EXPAND property FOREACH features»
            </properties>
            <components>
                «EXPAND component FOREACH nonNetworkComponents()»
                «EXPAND network FOREACH networks()»
            </components>
            <connections>
                «EXPAND conn FOREACH conns»
            </connections>
        </circuit>
    «ENDFILE»
«ENDDEFINE»
```

Figure 9. Small segment from XML template

The define block above is central to template declaration. Here an outlet so-called xml is used to beautify the generated output. The outlet definition is located in the workflow file concerning the intended needs. The model on which transformation will work can be shown as follows. The model, as easily seen, is nothing but the one in the previous pretty simple example stated out in figure 3. We have only three components: one active and two passive elements. No network is incorporated but some features are enclosed.

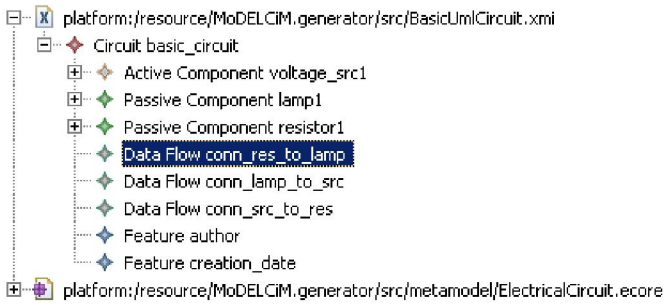


Figure 10. The model of the simple circuit mentioned in figure 3.

The resultant XML is an aggregation of the elements above to the file. It fully uses the extensibility of xml and makes the circuit model human-readable and portable across the schematic capture tools (not intelligible as much as the original concrete syntax but still seems fine comparing to the model above). As part of the output file, figure 11 can be addressed.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <circuit name="basic_circuit">
  + <properties>
  - <components>
    - <component name="voltage_src1">
      + <properties>
      - <ports>
          <port type="out" name="plus" />
          <port type="in" name="minus" />
        </ports>
      </component>
    - <component name="lamp1">
      + <properties>
      - <ports>
          <port type="out" name="lamp1_out" />
          <port type="in" name="lamp1_in" />
        </ports>
      </component>
```

Figure 11. Part of the output XML file.

## B. *Model to Model Transformation*

A circuit containing many components might perform exactly the same way as another one having fewer elements. This is because of the fact that many equivalent circuits may satisfy the same input-output relation. Since it is a naive and vast resource consuming exercise to utilize several circuit elements when fewer elements are applicable, it may be highly cost and resource effective to deduce the number elements used in a network. At that point, a model-to-model transformation mapping electrical circuit domain itself can be applied to reduce the number of elements contained thus resulting with a simplified circuit.

Given a circuit at the outset, the model-to-model transformation simplifies the circuit and yields the production of a simplified circuit. Sometimes, however, a network of elements is intentionally placed instead of an equivalent network having fewer elements. This might possibly be for fulfilling some technical constraints like power consumption, net impedance and so on. In this project the strategy used is far apart from complex processes that require the execution of a series of algorithms and necessitate some calculations to be done.

The tool used for model-to-model transformations in this study is ATL. The transformation in this study basically cancels the unnecessary short-connections and thus, reduces the size and complexity of any circuit. The algorithm used is self-explanatory. If a DataFlow object has a src and dst bound to the any two terminals of the same component then simply cancel the flow and simplify the circuit.

ATL has a refining execution mode that is quite beneficial especially for transformation mapping a meta-model itself as in our case. In this mode the input model is modified if necessary. And the rest of the model is retained in the output. Thanks to the refining mode, excessive coding is eliminated. So the code is kept light and clean. The final version of the ATL code is in Appendix D.

The model demonstrated below in has a short connection. The resistor in the circuit is shorted leading the component to be cancelled out. The transformation is fed by the input model in figure 12 and produces the resultant circuit in figure 13.

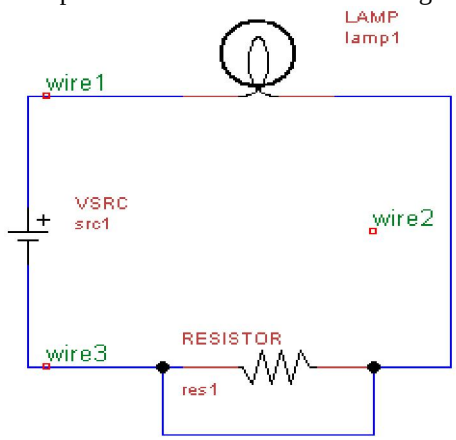


Figure 12. The input model with the resistor shorted.

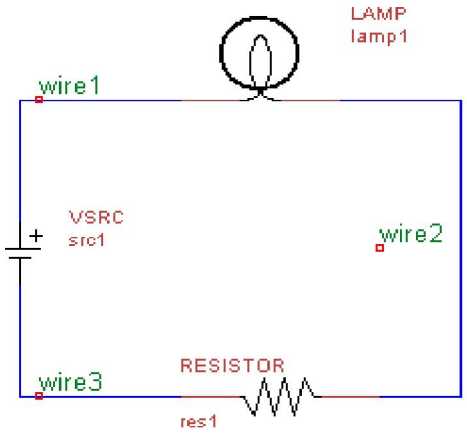


Figure 13. The resultant model with no short.

## VIII. CONCLUSIONS

This project helped me grasp many key aspects of model-driven software development. First of all, the central role of a DSL: DSL's are quite beneficial in that they better cover the problem domain and save a great effort while developing solutions. They, however, require an upfront investment and in many aspects are not easy to construct. I have faced with many difficulties while striving to build the meta-model. Mainly, the confusion of M2 and M1 was a tricky problem. To get over this problem, I visited domain analysis process again and again till its completion. Many kinds of resources including textbooks, published articles and web were gone into because expertise and a perfect domain analysis will be handy to come up with a precise meta-model defined.

Not only that, immature tools were also annoying. During the project, Eclipse Modeling Framework (EMF) [10] threw several exceptions and unexpectedly halted many times. It is even incapable of drawing and placing associations correctly and rendering the graph properly. These are all indicating that EMF still needs to be improved. At the first milestone the meta-modeling tools that I used were mostly Eclipse plug-ins that have still been incubating. After suffering from immature and instable environments, I seized MagicDraw UML, a stable modeling tool, workaround that supports model exportation in several formats including XMI. Then, using UML2Ecore bridge and XMI model exported from MagicDraw, I managed to refine the meta-model quite easily with automatically generated constraint check file in pocket.

As a consequence derived, I can say that meta-modeling is a quite time consuming process and requires a thorough attention employed with right tools chosen. There is always possibility of level confusion yielding the creation of an ill-behaved meta-model, which will be eliminated by a seamless domain analysis and expertise in the target domain.

Other than those, it was captivating to use modeling in a field other than computer science, it is widely used in computer science though. As models and modeling are to an extent assets of systemic approach; it was of paramount significance to apply this systemic approach in electrical engineering, it was exciting to witness the possible use of model-driven approach in electrical engineering.

As far as the two methodologies used to create DSL are concerned, meta-modeling from scratch is interestingly easier than that of UML2 Profiling. The main reason making UML Profiling harder to understand is eminently the lack of sufficient UML background. In spite of the fact that I am good at modeling on UML, I still needed to spend some note-worth time to grasp the notion of profiling, stereotyping and other UML basics. The other reason that makes UML Profiling harder is associated to the tools used for the creation of models. Many tools have better and enhanced support for MOF when compared to UML2 Profiling. And the final reason of going for meta-model from scratch is its higher readability. Models created from scratch is much simpler and reader-friendly than UML profiling does.

Another lesson taken out of this project is concerning the expressiveness of textual and graphical representation of models. The comparison of textual and graphical representations can be easily made through model-to-text transformation described earlier in this report. XML decreases the level of understandability, readability and traceability whiling bettering portability and providing a means of interoperability. The outcome here is that textual representation (XML) can be used across divergent tools while using graphical one for the end-users. For textual representation, it is since the interoperability is of incredible consideration and again computers have no readability problem as in the case with human-beings. Likewise, as graphical representation in this case is more user-friendly and expressive, readable and intelligible, it is better to serve them as long as users come into play.

For model transformations play a key role in MDSD, it was vital to understand the idea behind them. Starting with model-to-text transformation I tried to comprehend them all. An M2T transformation is used to output an XML file storing the circuit info. OpenArchitectureWare is used for this purpose. Its extended list of samples and hands-on practices made it simpler to learn the tool. Since there is no common ground for the schematic capture tools or a consensus on an open file format, the output XML schema is created by my own with a separate effort. The schema is designed to be as extensive as possible through the Feature meta-class. I had no difficulty while generating the file. However, the visualization of the generated XML file unfortunately is not possible due to the lack of open standards in the industry and of course to the deficiency of tools built upon such open standards. As for model-to-text, I couldn't realize the transformed models generated by model-to-model transformation simply because of the same fact: lack of standards and tools supporting them. M2M transformation is made using Atlas Transformation Language (ATL). It was a pretty nice experience to evolve a given circuit using a transformation definition that is defined within a specific scope. Since the scope is highly restricted in this case, programmers are not forced to know any other concept that is beyond the scope. Rather, they are forced to be to the point. Eminently, this way MDSD increases the productivity and quality of the product, which is one of the most important observations that I inferred from the project.

The bottom line is that MDSD seems a promising field of study that will definitely update the face of software engineering in the near future. I am really glad to meet this revolutionary approach through this project by means of MDSD course.

## REFERENCES


[1] J. White, Douglas C. Schmidt, A. Nechypurenko and E. Wuchner, "Domain-Specific Intelligence Frameworks for Assisting Modelers in Combinatorically Challenging Domains"

[2] Karsai, G., A. Agrawal, and A. Ledeczi, "A Metamodel-Driven MDA Process and its Tools", WISME, UML 2003 Conference, San Francisco, CA, 2003

[3] http://en.wikipedia.org/wiki/Electronic_circuit

[4] http://en.wikipedia.org/wiki/Electrical_network

[5] Khalil, Hassan (2001). Nonlinear Systems (3rd Edition). Prentice Hall. ISBN 0130673897.

[6] http://www.topcased.org/

[7] http://www.eclipse.org/

[8] Unified Modeling Language (UML), Version 1.5, Object Management Group (2003), http://www.omg.org/cgi-bin/doc?formal/03-03-01.

[9] OCL 2.0 Specification: http://www.omg.org/cgi-bin/apps/doc?ptc/2003-10-14.

[10] http://www.eclipse.org/modeling/emf


## APPENDIX A

Plot of the meta-model from scratch.

## APPENDIX B

Plot of the meta-model using UML profiling.

## APPENDIX C

Textual representation of the UML profiling.

## APPENDIX D

Complete oAW code for model-to-text transformation

## APPENDIX E

Complete XML output of model-to-text transformation

## APPENDIX F

ATL transformation code.