A DSL supporting both Graphical and Textual Editing.

1 Introduction

In Model driven engineering DSL stands for Domain Specific Language. It refers to a custom language that is designed to be used to develop programs in a specific domain. DSL can either be of textual or graphical notation. The creation of a DSL first needs the creation of an abstract syntax which is known as a Metamodel. From the Metamodel, one can then create graphical concrete syntax as well as textual concrete syntax.

Graphical concrete syntax consists of graphical symbols, compositional rules and mapping of the graphical symbols to the abstract syntax [7]. Textual concrete syntax on the other hands needs a grammar that can determine which character sequences are valid in the language [7].

Currently, Ericsson uses an in-house developed DSL to create applications for its Baseband Switches. The DSL uses graphical notations and is known as HIVE to Ericsson. This means that developers create models of the applications using graphical notations and later use Model transformation tools to transform the models into C code which can be compiled to be working applications The transformations are done in three steps. The first step is a Model to Model Transformation that transforms an instance model created using HIVE to DIVE which is another metamodel that is completely in text. From the DIVE, another model to model transformation is done that transforms the DIVE model into a C instance model that conforms to the C abstract syntax tree. From the C abstract syntax tree a model to text transformation is performed to get .c and .h text files.

From a traditional point of view, in order to create programs, developers write code (which is in text) in one of the many programming languages available. Introducing model driven development means that developers no longer write code in languages they know, they have to switch to a DSL instead. Shifting to a DSL already has its issues as many developers would prefer to code in a language that they already know and have experience in. Introducing model driven development with only a graphical DSL makes it even harder given that creating programs using graphical notation is not a usual practice for most developers.

Also when modelling, some use cases are easier to specify when using graphical notations while others can be specified easier using textual notations. Both the graphical and textual DSLs have their advantages and it would be best if all these advantages can be harnessed. So a solution that will support the use of both graphical and textual DSL in an easy and cost effective way is needed.

2 Statement of the problem

Currently, Ericsson has only the graphical version of the DSL and as stated in the introduction some programmers would prefer to use text when modelling and in some cases, the use cases can better be understood when modelled using text. So the first problem is the lack of a textual version of the DSL. To address this, a textual version of the existing graphical DSL will be created. This will be done using Xtext which is a tool for creating textual DSLs based on the Eclipse Modelling Framework (EMF). Currently the company uses RSA from IBM which is also built on EMF.
Having both the textual and graphical version being used simultaneously raises several more general problems to be addressed. One of the main problem is how to maintain the two DSLs without adding a lot of maintenance costs to the company. This means that in case the DSLs needs to be updated one should not have to do a lot of manual work. The study aims to provide a solution (an automation) that will make it easy to maintain both the languages. There are several proposed solutions on how to have both textual and graphical DSLs and this thesis will look into three of the them and come up with the best solution.
Another problem that arises is how the end users can be able to switch between the graphical and textual views in an easy way. This thesis will also provide a solution of switching between the two DSLs in an easy and effective way. The idea here is that a person will only need to press a button and switch between the views and can edit the models in any of the views.
The format in which the models will be stored is also another problem that the thesis will look into. Since there will be both textual and graphical models, the study will explore the options of storing the models either as models or as text files.

3 Purpose of the study

The purpose of this study is to create an Xtext version of an existing DSL, investigate advantages and disadvantages of introducing a textual DSL to complement the existing graphical DSL. The study will also report findings on how best and in what situations Textual and Graphical DSLs can be used together.

4 Review of the literature

1. Derivation and Refinement of Textual Syntax for Models

Authors: Florian Heidenreich, Jendrik Johannes, Sven Karol, Mirko Seifert, and Christian Wende
Summary: This paper describes how EMFText tool can be used to generate Textual Syntax from models. The tool takes a Model as input and generates Textual Syntax. The syntax generated uses Human Usable Textual Notation (HUTN) and is a generic syntax. The paper also shows how the generated syntax can be customized to become more specific for its purpose.

Significance: This paper can offer great help when it comes to the creation of the Textual DSL as it offers users with a skeleton to begin with that can be customized therefore saving time.

2. Harmonizing Textual and Graphical Visualizations of Domain Specific Models

Authors: Colin Atkinson, Ralph Gerbig

Summary: This paper defines a prototype platform in which Textual and Graphical version of the DSL can be used together by programmers. It is based on the concept of separating the Notation part of the language from the Abstract syntax. This will result into a language which can have either graphical, textual or even tabular view.

Significance: This concept can be looked into when it comes to finding proper ways for the Graphical and Textual DSL to be used together.

3. Domain specific languages with graphical and textual views

Authors: Francisco Perez Andres, Juan de Lara1, and Esther Guerra

Summary: This paper describes an approach of creating DSL with both Textual and graphical view. The approach used is to first generate the whole meta-model for the Language and then break it down to various viewpoints that are relevant to the domain being modelled. The viewpoints are the graphical concrete syntax and from these Textual Concrete syntax can automatically generated using triple graph grammar rules.

Significance: This paper can provide further insights on the possibility of generating Textual concrete syntax from Graphical concrete syntax. At least the skeleton of the textual syntax.

4. Integrating Textual and Graphical Modelling Languages

Authors: Luc Engelen and Mark van den Brand

Summary: This paper acknowledges the fact that both graphical and textual DSLs have their advantages and tries to find a way to use both languages so as to leverage these advantages. The paper describes two approaches of switching from graphical to textual views and vice versa. The first approach is called Grammware which uses the grammar rules for switching and the second approach is Modelware which uses rules conforming to models and Metamodels for switching.

Significance: This paper can help the research after the Textual language is in place. It can help us to decide the best way to be able to use both the Textual and Graphical DSLs together.

5. Text-based Modeling

Authors: Hans Grönniger, Holger Krahn, Bernhard Rumpe, Martin Schindler and Steven Völkel

Summary: This paper discusses the advantages of using Textual based modelling over graphical modelling. It uses the implementation of a Textual version of the UML state machine to explain the advantages.

Significance: By knowing the advantages of Textual DSLs this paper can help in the DSL creation process. We can determine which parts will be important to implement first so as to harness these advantages.

6. Textual Modelling Embedded into Graphical Modelling

Authors: Markus Scheidgen

Summary: This paper provides a way that textual editors can be embedded into graphical editors. In this way when creating a model using the graphical editor, designers can also add text to the graphical model using a text editor that is embedded in the Graphical editor. The paper shows how to embed EMF textual editors into graphical editors created using GMF.

Significance: The paper provides more possibilities of using both Textual and Graphical DSLs together. This can also be something whose feasibility can be researched at Ericsson.

7. Model Driven Analysis and Synthesis of Concrete Syntax

Authors: Muller Pierre-Alain, Fleurey Franck, Fondement Frédéric, Hassenforder Michel, Schneckenburger Rémi, Gérard Sébastien, Jézéquel Jean-Marc

Summary: This paper notices that formally defining concrete syntax for a language is a big challenge. It describes an alternative way that can map abstract syntax to concrete syntax using bi-directional mapping tools to facilitate model to text and text to model transformation

Significance: This paper can be used to help in the creation of the Textual DSL in this research.

8. Supporting the DSL Spectrum

Authors: David S. Wile

Summary: This paper discusses several approaches that can be used when designing DSLs. It also describes various issues that may be related to each approach.

Significance: Content from this paper can be used when designing the Textual DSL.

9. DSL Engineering - Designing, Implementing and Using Domain Specific Languages

Authors: Markus Voelter

Summary: This book discusses how one can design and implement domain specific languages and focuses mainly on Textual domain specific languages.

Significance: Content from this book will be used when creating the Textual DSL.

5 Research question and/or Hypotheses

- What is the best way to have both textual and graphical DSL without doubling the maintenance costs?
- How can we switch between Textual and Graphical DSL without any loss of information?
- How can syntax errors be handled when switching between them?

6 The Design – Methods and Procedures

This section describes the methods that will be used to answer the research questions mentioned in the previous section. Each research question has been listed together with the methods to be used.

- **What is the best way to have both textual and graphical DSL without doubling the maintenance costs?**

As it has been mentioned before, the company need to add a textual version of the DSL but the addition of this DSL should not double the costs when it comes to updating the language. There are three alternatives that can be used to have both the graphical and textual DSLs.
The first one is to have one core meta model and use a common transformation tool to automatically generate the textual and graphical meta model (See fig 1.). In case the language evolves, changes will be made in the core meta model and the corresponding graphical and textual metamodel will be generated.

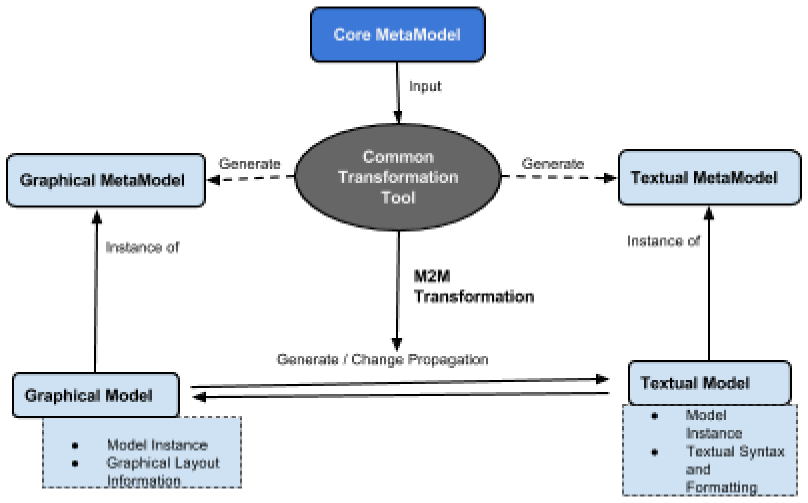
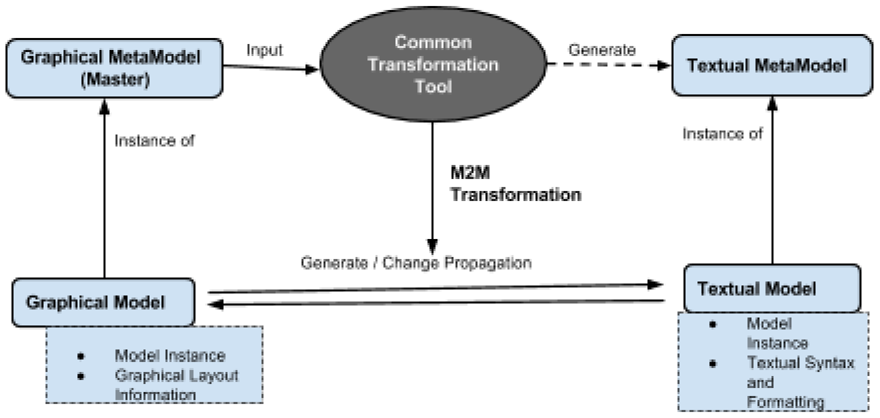


Fig 1:Core MetaModel used to generate Graphical and Textual MetaModels

The second alternative is to have the existing graphical metamodel as the master meta model and automatically generate the textual meta model from the graphical meta model using a transformation tool. In this case updates made on the graphical metamodel which now acts as the master meta model will be propagated to the generated textual meta model through the transformation tool (See fig 2.).



Graphical MetaModel (Master)
Input
Common Transformation Tool
Generate
Textual MetaModel
Instance of
Instance of
M2M Transformation
Generate / Change Propagation
Graphical Model
Textual Model
Model Instance
Graphical Layout Information
Model Instance
Textual Syntax and Formatting

Fig 2 :Graphical MetaModel used as a master/core to generate the Textual MetaModel

The third alternative will be to have one metamodel for the DSL and then from this metamodel get both the textual and graphical views. This can be achieved by completely separating the concept of notation (which is the concrete syntax) from the abstract syntax. Having a visualiser (See fig 3) that determines how the model will be displayed will make it possible to switch between graphical and textual views. When a model instance has been created using the textual editor, it can be opened using the graphical editor as well and the visualiser will make this possible. Therefore one can edit an instance model in either graphical or textual notation and save it. With this kind of setting there is no need of using model to model transformation to switch between views. This concept has been discussed in details in [9].

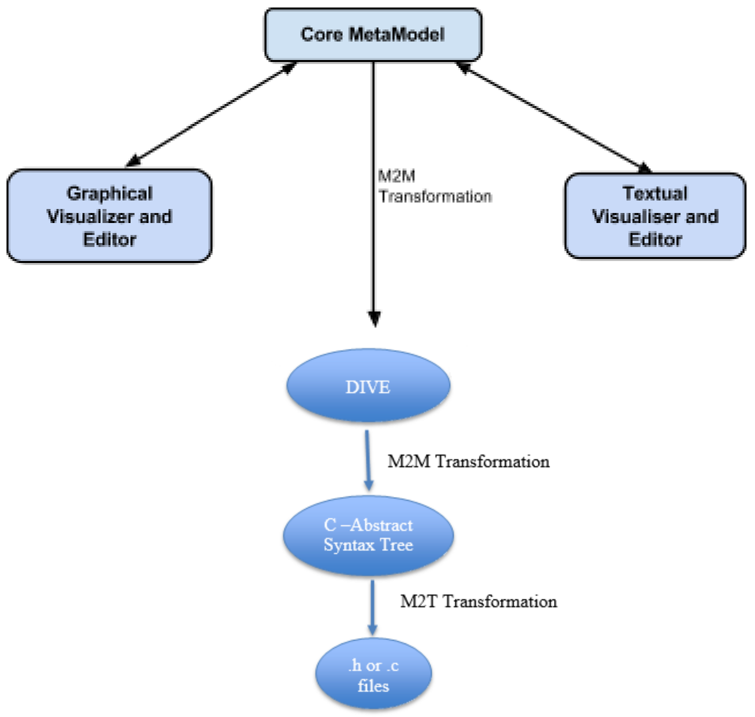


Fig 3: One core metamodel and a visualiser

All the three alternatives will be looked into.Prototypes of the three alternatives will be created. Using the prototypes an experiment will be conducted where updates will be done to the language and the effort of getting the updates in both the textual and graphical versions will be measured.The measurements will be in terms of time needed to make the update and the complexity of making the updates. From the experiments, the alternative that promises least maintenance effort will be determined.

With the textual DSL present, models created using this version need to be converted into models conforming to the DIVE metamodel (an Ericsson developed metamodel). Therefore another model to model transformation needs to be in place to transform the Textual Instance model into a model corresponding to the DIVE which is also pure text. Transformation from DIVE to C abstract syntax and to .c and .h files has already been implemented by Ericsson. Also the Model to Model Transformation from the Graphical instance model (known as HIVE to Ericsson) to DIVE has already been implemented by the company.(See Fig.4)

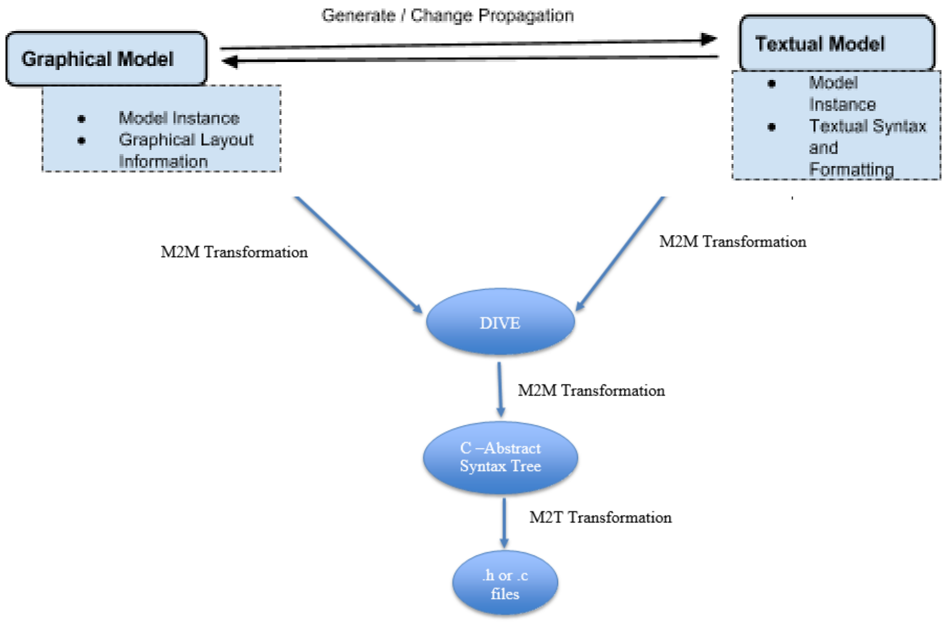


Fig 4: Transformations from model instances to .c and .h files

- **How can we switch between Textual and Graphical DSL without any loss of information?**

Since there will be both graphical and textual versions of the DSL, there needs to be a good and easy way for programmers to switch between them. The aim here is to make the switching between the DSLs as automatic as possible. So eclipse tools will be investigated to create an automation of switching between textual and graphical views. The concept of model to model transformation will be used to convert the graphical models to textual models and vice versa.

An experiment will then be conducted where a designer modelling using the graphical notation will switch to the textual notation. An analysis will be done to measure if all data present in the Graphical model is also present in the Textual Model. Also a designer modelling in using the textual notation will switch to the graphical notation and the same analysis will be performed to look for any loss of information.
This will focus on bottom part of the figures above to see if there will be any problems when switching between the models.

- **How can syntax errors be handled when switching between them?**

Models created by both graphical and Textual DSLs can be inconsistent. Inconsistent models are models that do not comply with the metamodel constraints. To answer this question, inconsistent models will be created using the Textual version of the language and then the programmer will switch to the graphical view. The errors that are shown (if any) will be recorded. The model in graphical view will be looked at in the point where the inconsistency is supposed to be. This way we can know how syntax errors are handled for inconsistent models when switching between views.

7 Limitations and Delimitations

Given that we will not have access to conduct surveys or observations with programmers or designers at Ericsson, it will not be possible to measure the effectiveness of the Textual DSL compared to the Graphical DSL. Also due to the same reasons, limitations associated with the new Textual DSL may not be observed during the time given for the thesis. To avoid adding extra tool maintenance costs to the company, the development will be based on the Eclipse Modelling Framework, this means that the study will not report the possibility of using other modelling frameworks.

Furthermore, since the study will be conducted in one company, the described challenges may not be challenges that could arise if the study was conducted elsewhere. The process described may also differ depending on company and situations as well.

8 Significance of the study

In research this study will help to identify a process or processes that can be used to create Textual DSLs. It will provide a solution for using both graphical and textual DSLs for modelling and being able to switch between the two views in an easy and effective way. Also the study will provide a solution for having both graphical and textual DSLs without doubling the maintenance costs. Moreover, various challenges of creating and using DSLs will be provided as an addition to the little literature that exists on this area.
The practitioner which Ericsson in this case will benefit by getting a Textual version of the DSL so that developers can have a choice on which one to use. Also having the Textual version of the DSL will enable the company to conduct further research such as that of comparing which of the two is more efficient or adds more value to the company.

9 References

1. Andrés, F., Lara, J. de, & Guerra, E. (2008). Domain specific languages with graphical and textual views. *Applications of Graph Transformations* ..., 82–97. Retrieved from http://link.springer.com/chapter/10.1007/978-3-540-89020-1_7
2. Engelen, L., & van den Brand, M. (2010). Integrating Textual and Graphical Modelling Languages. *Electronic Notes in Theoretical Computer Science*. doi:10.1016/j.entcs.2010.08.035
3. Grönniger, H., Krahn, H. Rumpe, B., Schindler, M and Völkel, S. Text-based Modelling.
4. Muller, P.-A., Muller, P.-A., Fleurey, F., Fleurey, F., Fondement, F., Fondement, F., … Jézéquel, J.-M. (2006). *Model-Driven Analysis and Synthesis of Concrete Syntax*. *Model Driven Engineering Languages and Systems* (pp. 98 – 110). doi:10.1007/11880240_8
5. Scheidgen, M. (2008). Textual modelling embedded into graphical modelling. In *Model Driven Architecture–Foundations and Applications* (pp. 153–168). doi:http://dx.doi.org/10.1007/978-3-540-69100-6_11
6. Wile, D. S. (2001). Supporting the DSL Spectrum. *Journal of Computing and Information Technology*. doi:10.2498/cit.2001.04.01
7. Brambilla, M., Cabot, J. and Wimmer, M. 2012. *Model-driven software engineering in practice*. [San Rafael, Calif.]: Morgan & Claypool.
8. Heidenreich, F., Johannes, J., Karol, S., Seifert, M., & Wende, C. (2009). Derivation and Refinement of Textual Syntax for Models. In *Proceedings of ECMDA-FA* (pp. 114–129). doi:http://dx.doi.org/10.1007/978-3-642-02674-4_9
9. Atkinson, C., & Gerbig, R. (2013). Harmonizing textual and graphical visualizations of domain specific models. In *Proceedings of the 2nd Workshop on Graphical Modeling Language Development, GMLD 2013 - In Conjunction with European Conference on Modelling Foundations and Applications, ECMFA 2013* (pp. 32–41). doi:10.1145/2489820.2489823
10. Voelter, M. and Benz, S. 2013. *DSL engineering*. [S.l.: s.n.].