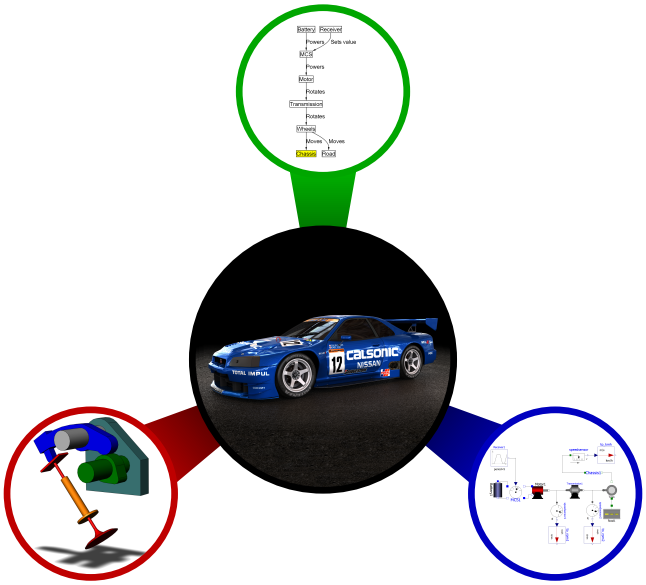
University of Twente

Faculty of Engineering Technology
Design, Production and Management

Improving software assistance for design engineers by integrating mathematical and function modeling



H.M. Bakker

Master of Science

July 2012

Exam Committee:

Chairman:	Prof.Dr.Ir. F.J.A.M. van Houten
Supervisor:	Dr.Ir. W.W. Wits
Mentor:	Prof.Dr. L.S. Chechurin
External member:	Dr.Ir. J.J. Homminga

Exam date: July 31, 2012

Report number: OPM-1119

The cover image symbolizes the various model types coming together in ModelEvolution, or how a technical system from the real world can be modeled by various model types. The green circle shows a function model; the blue circle shows a Modelica model and the red circle shows a CAD model.

3D model of the Nissan Skyline GT-R R34 JGTC CALSONIC courtesy of keshon83 on deviantART.

Personal introduction

This report is the conclusion of a 1.5-year lasting cooperation between the University of Twente and Saint Petersburg State Polytechnical University – consisting of my internship and master thesis.

One of the results is a software prototype that integrates function modeling, CAD software, parts of TRIZ and Modelica modeling. Besides developing the software, we wrote a few papers of which the writing process and the presentation of one were a new experience for me.

The software development was a process that had learning points like confrontation with the consequences of earlier choices. Because the goals of the software became clearer during the research process, sometimes the architecture had to be changed to suit the new goals.

Besides the academic work, this project was also one with a lot of personal development – living in Russia with all its surprises and studying its language were experiences that I wouldn't want to have missed.

Acknowledgments

Firstly, I would like to thank Leonid Chechurin for all efforts he did to make this project possible, but also for the good care he took for me during my time in Russia. He really was both a good supervisor and a good host.

Secondly, I'm grateful to Wessel Wits for his supervision and feedback during this project.

I would also like to thank Sergei and Nina Chechurin for their great hospitality.

I would like to thank Leonid's colleagues for accepting me as a member of their team, for their jokes and for them making me taste me so many different kinds of traditional food.

Furthermore, I would like to thank the Saint Petersburg State Polytechnical University for allowing me to work on my internship and master thesis there.

I also would like to thank my friends for their friendship and for the wonderful things we did together.

I would like to thank Valeri Souchkov and Sergei Ikovenko for introducing me to Leonid and thereby making this project possible.

Thanks go to Ivo Stammis for his hints and for proofreading my report.

Finally, I would like to thank my parents and sister for their support before and during this project.

Summary

TRIZ *function modeling* is an analysis method applied by engineers to describe the component relations in a technical system in terms of *functions*. Such function models are a basis for the product improvement process based on the TRIZ theory of innovation. Since product improvement implies that a product already exists, usually CAD models of the product are already available. In ongoing research, software has been developed to automate the creation of function models from SolidWorks assemblies and to assist engineers in working with them, to make use of this situation.

By means of using functions, it is expressed *what* components do and not *how* they do it. This abstraction removes *mental inertia*, thereby enabling engineers to think of new solutions. Because the functions are qualitative and because they are expressed in natural language, they are relatively imprecise. On the other hand, mathematical models can describe technical systems in a much more detailed way, but it is inconvenient to use separate software for this. Furthermore, using static function models to describe a dynamic system with changing functions or a changing structure is quite difficult.

To solve these inconveniences, a workflow was developed to integrate CAD software, TRIZ, the existing function modeling software and Modelica. Modelica is a mathematical modeling language that can be used in a similar way to Simulink. Modelica was chosen because of its advantages over Simulink such as better support for modeling of physical systems and support for acausality. To simplify working with function models of changing systems, the concept *Dynamic Function Modeling* was introduced. This concept consists of function models that can change their structure depending on time.

During this project, the integration between the existing function modeling software and Modelica was implemented. For the software user this integration resulted in the possibility of editing Modelica code from the function model diagram and being assisted in writing this code, performing simulation of the internal Modelica model and having visualization of the simulation results visually linked to the components of the function model diagram. The Dynamic Function Modeling concept is implemented by automatically changing the structure of the function model depending on the values of the variables, resulting from the simulations.

The integration possibilities are demonstrated by means of a case study, in which a function model and Modelica model of an RC-car are made and simulated using the developed software. This case study shows the strengths of the integration, but also the points that need improvement during following projects.

Besides the software integration of function modeling and Modelica, the function modeling software was extended with more assistance features.

Samenvatting

TRIZ *function modeling* is een analysemethode, toegepast door engineers, om de relaties tussen componenten van een technisch systeem te omschrijven m.b.v. *functies*. Zulke *function models* vormen een basis voor het productverbeteringsproces gebaseerd op de innovatietheorie *TRIZ*. Aangezien productverbetering impliceert dat er al een bestaand product is, zijn er meestal al CAD modellen beschikbaar. Om van dit gegeven gebruik te maken is er tijdens een lopend onderzoek software ontwikkeld, die het maken van function models op basis van SolidWorks assemblies automatiseert en engineers assisteert in het werken met deze function models.

Door gebruik te maken van functies, wordt de nadruk gelegd op *wat* componenten doen en niet *hoe* ze dat doen. Deze abstractie voorkomt *mental inertia*, waardoor engineers makkelijker aan nieuwe oplossingen kunnen denken. Omdat functies kwalitatief zijn en omdat ze in natuurlijke taal uitgedrukt worden, zijn ze niet heel exact. Aan de andere kant kunnen technische systemen veel gedetailleerder beschreven worden met wiskundige modellen, maar is het ongemakkelijk om hiervoor aparte software te gebruiken. Verder is het lastig om statische function models te gebruiken om dynamische systemen met veranderende functies of een veranderende structuur te beschrijven.

Om deze tekortkomingen aan te pakken is een workflow ontwikkeld om CAD software, TRIZ, de bestaande function modeling software en Modelica te integreren. Modelica is een wiskundige modelleertaal die op een soortgelijke manier als Simulink gebruikt kan worden. Modelica is gekozen vanwege zijn voordelen t.o.v. Simulink zoals betere ondersteuning voor het modelleren van fysieke systemen en ondersteuning voor acausaliteit. Om het werken met function models van veranderende systemen makkelijker te maken is het concept *Dynamic Function Modeling* geïntroduceerd. Dit concept beschrijft function models waarvan de structuur afhankelijk van de tijd kan veranderen.

Tijdens dit project is de integratie van de bestaande function modeling-software en Modelica geïmplementeerd. Voor de gebruiker van de software betekent deze integratie de mogelijkheid tot het (geassisteerd) bewerken van Modelica-code vanuit het function model-diagram, het simuleren van deze code en het tonen van de simulatieresultaten, visueel gelinkt aan de componenten van het function model-diagram. Het Dynamic Function Modeling-concept is geïmplementeerd door het automatisch veranderen van de function model-structuur afhankelijk van de simulatieresultaten.

De mogelijkheden die de integratie biedt worden gedemonstreerd door middel van een case study, waarin een function model en een Modelica-model van een RC-auto gemaakt en gesimuleerd worden met de ontwikkelde software. Deze case study laat de sterke kanten van de integratie zien, maar ook de verbeterpunten die aangepakt moeten worden tijdens volgende projecten.

Naast de software-integratie van function modeling en Modelica is de function modeling-software uitgebreid met meer assistentiemogelijkheden.

Contents

List of Figures

Part I

Theory and background

1 Introduction

This chapter explains the situation at the beginning of this project, the goals for this project, how the research will be performed and how this report is structured.

1.1 Project background

The daily job of product developers and engineers consists of various tasks. Strategy decisions, cooperation tasks, product design, product analysis, simulation are some examples of such tasks. When the computer programs – that are supporting the engineer for these tasks – are not integrated with each other the work of the engineer will be harder and more prone to mistakes. This is because, when there is no integration, the engineer has to create separate models of the same product. Besides more needed effort to perform the same work, mistakes can be made in keeping the different models the same.

When working in a team with people from different fields of profession (multi-disciplinary development), they all have their preferred tools, methods, their own knowledge and their own software. Also, they have their own thinking inertia based on their professional knowledge and experience. To make these people cooperate well and to remove their thinking inertia, a common innovation platform that removes thinking inertia is needed. As explained by Wits et al. in [48], *TRIZ function modeling* (FM) is a good method for this. TRIZ is, in short, a group of methods to improve creativity in a systematic way. FM is one of these methods. It is used to analyze technical systems by expressing the relations between components of these systems in terms of *functions*. This way, it is emphasized *what* components do and not *how* they do this.

As reasoned by Chechurin et al. in [12], FM can sometimes lack precision in its description of technical systems. After all, it is a qualitative modeling technique based on natural language. Mathematical modeling, on the other hand, can be very precise but also requires expertise. Also, currently there exists no FM software that provides support for mathematical modeling or that provides integration with mathematical modeling software like Simulink or Modelica.

This project is part of an ongoing research striving to support engineers in working with FMs. Two papers have been published about software that automates the creation of FMs from SolidWorks assemblies [6] and performs *function ranking* and *trimming* [13]. The prototype FM software that was developed for this papers will be called ModelEvolution. This software will be taken as a basis for this project.

1.2 Research goal

In the situation described in the previous section, there are several tools available to the engineer: FM, other TRIZ methods, CAD software and mathematical modeling software. Mathematical modeling support could be a useful addition to FM software. However, there is currently no FM software that provides integration with mathematical software. Also there is no literature that describes how to implement such an integration.

Therefore, the goal of this project is to find out the interaction possibilities between the mentioned tools first. When these possibilities are known, they need to be implemented. This is of course needed to be able to use them, but also to verify whether or not they actually ease the engineers' work.

1.3 Research approach

First the components of the product development ecosystem, that were to be included in the research, are listed. These components are *CAD software*, *Function Modeling*, *Modelica*, *Optimization* and *TRIZ*. Modelica is the software that is used to provide mathematical support. Optimization of the mathematical models is presented as a separate step in the workflow.

Secondly, a thorough literature study into the tools, into similar software and the into existing interactions between the tools is done. After that, ideas for new interactions are generated. This is done by looking at the data that each tool requires, by looking at the data that each tool provides and by matching these data afterwards.

The possible new interactions and the existing interactions between the tools are drawn together in a workflow diagram (figure 1.1) – a scheme that shows the information flows between the tools.

During this project, the interaction between FM and mathematical is chosen to be implemented in the ModelEvolution prototype.

This project focuses on software development. A case study consisting of the modeling and simulation of an RC-car is done to demonstrate the implemented functionality and to verify the software's capabilities of working with larger and more detailed models. A user survey amongst a large group of users can be done in a future project.

1.4 Research outline

First a literature study into the most important topics for this project is presented as a broad introduction to the project's background in chapters 2, 3 and 4. These chapters respectively discuss TRIZ, Mathematical modeling and Software integration. Function Modeling is discussed as a part of the chapter on TRIZ in section 2.1. Details on Modelica are given as a part of chapter 3 in section 3.4.

In chapter 5 the workflow of figure 1.1 is discussed. The literature study into the existing principles, integrations and similar software is presented here together with the new integration ideas.

The outlines of the software development, done for this project, are discussed in chapter 6. This chapter also demonstrates the workings of the software with a discussion of its (new) features. More technical details on the implementation are provided in appendix B.

The screenshots describing the working of the software are bundled with the screenshots of the case study in chapter 7. This is done to have them all in a central place, clearly showing the workflow steps.

The case study concerning the RC-car to demonstrate the software's capabilities is presented in chapter 8.

The main part of the report ends with the conclusions and recommendations in chapter 9.

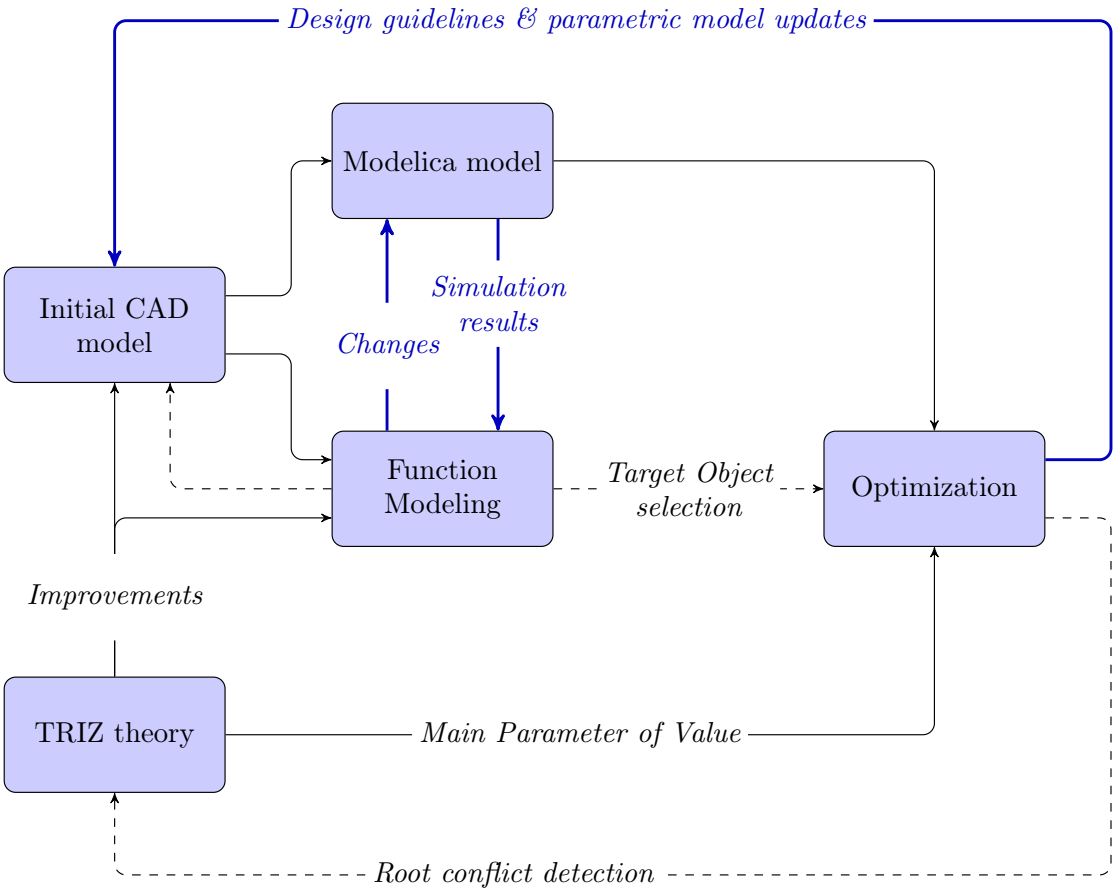


Figure 1.1: Overall workflow

2 TRIZ – Theory of Inventive Problem Solving

TRIZ (Russian abbreviation of *Theory of Inventive Problem Solving*) is a group of methods that offer a systematic approach for the improvement of technical systems. It was developed by Genrich Altshuller [1] and a group of scientists around him. Their first ideas were presented in a Russian journal on psychology [2] and afterwards these ideas were developed by a growing group of people.

Salamatov provides a book [39] about TRIZ that gives a lot of practical examples from history, besides a broad discussion of the theory.

The goal of TRIZ is to make a technical system achieve *ideality* [39, p. 112-114]. The better the main function of a system is performed and the less mass, volume and energy are used to perform this function, the more ideal a system is. Souchkov defines the ideality of a system in [41] as its net performance divided by its costs:

$$\text{ideality} = \frac{\text{useful effects} - \text{negative effects}}{\text{costs}} \tag{2.1}$$

Generally spoken, the problem solving using TRIZ is done as shown in figure 2.1. First the existing technical system is analyzed. In this phase, one searches for technical contradictions [39, p. 69]. One of the contradictions is selected to work on and an abstract version of the contradiction is composed. TRIZ provides a collection of abstract principles to solve such abstract contradictions. These principles are derived from other inventions providing solutions for similar problems. One way of selecting suitable principles for solving the selected contradiction, is by using the Contradiction Matrix [3]. Using the selected principles, solution candidates can be generated for the initial problem.

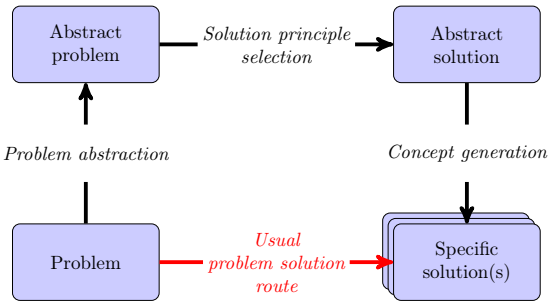


Figure 2.1: General TRIZ problem solving steps

There are two reasons for following this abstraction route instead of the usual problem solving route. The first one is to distract the user from his own *solution space*. Probably

he has experience in his own field, and probably he has already thought about the usual solutions from this field. The inclination to stick to the usual solutions is called *mental inertia* and the goal is to remove this. The other reason is to make similarities with problems and solutions from other fields visible to the user.

2.1 Function modeling in TRIZ

Function modeling [14] [9], also known as function analysis [41, p. 69-74], is one of the methods from TRIZ to describe the relations between a system's components. It is not part of the 'original' TRIZ, but it was added later. It can be used to analyze an existing system with a focus on the functions that the components perform and the parameters that are changed. These functions are usually described as *object-verb-subject*-triples. Gerasimov et al. [19] give a definition of a *function* and related terms. Litvin et al. [32] restate a part of these definitions in English and propose to extend Function Modeling with spatio-temporal parameters.

The goal of this method is to make an abstraction of the system: the actual implementation of these interactions is less important. For example, in function modeling it does not matter if a component is rotated by an electric motor, a combustion motor or a human hand. The important part is that the component is rotated: the goal of this abstraction is to remove mental inertia.

According to Souchkov [41, p. 73-74], the problems found during function analysis should first be ranked according to their importance using comparative ranking. The problems can then be solved using the Inventive Standards (after transforming the problem to a Substance-Field model). Souchkov also mentions ARIZ or Trends of System Evolution as ways to solve the problems.

Litvin et al. [32] mention "Trimming, Feature Transfer, Super Effect Analysis, and Function Oriented Search" as methods that can be performed on Function Models to increase the ideality of the described systems. This is not done by solving problems / contradictions per se, but they can increase the amount of positive effects performed by the system (Super-Effect Analysis [24] [18]) or reduce the amount of components to perform the same functionality (Trimming [13] [25] [19]).

Because functions are qualitative (they say *what* a component does and not *how* a component does it) and because they are defined using natural language, ambiguity or invalid functions can slip into function models [9]. This is a shortcoming of FM that can be solved by training of the users or by software assistance helping them to formulate their functions better.

2.1.1 Function model creation

The process of creating a FM usually starts with creating an *interaction matrix* – a table in which all system components are listed and where interacting components are marked. After the creation of the interaction matrix, the interactions are listed and extended with function descriptions. Additional details, like whether or not a function is harmful, can be added too. Formally spoken, this list of components and their functions is a FM

already. However, this data is usually presented in diagram form to be more insightful. Figures of this process are shown in appendix A.

Engineers can select one of the components of the system to be a *target component*. This component is the component that the system performs its useful function on. For example, the target component of the system "knife and bread" would be bread since the knife cuts the bread. In FMs the target component is usually marked yellow (see figure 2.2).

2.1.2 Multi-layer Function Modeling

As products become more complex, engineers usually group their components in sub-assemblies that together form the total assembly (even larger products can have sub-sub-assemblies, etc). This way, the components form a hierarchical tree. To reflect this hierarchy, every (sub-)sub-assembly can be represented in its own FM and can be put as a *black box* in the FM of the level above it (see figure 2.2).

When working with such multi-layer FMs without sophisticated tools, it is quite hard to keep an overview of the system, especially when there are interactions across the various hierarchy levels. Figure 2.3 is an example of how working with multi-layer FMs can be made easier: the system hierarchy is presented as a tree in the left-hand part of the program window. Each hierarchy level can be clicked to show the FM of that level. Furthermore, all FM components can be double-clicked (like in Simulink) to show the FM below them. Components outside the currently shown FM (like in different branches of the component tree or in different hierarchy levels), that are connected with a function to a component of the currently shown FM, are shown dashed. For the sake of legibility, this type of functions is not shown in figure 2.2.

2.1.3 Dynamic Function Modeling

Technical systems are usually dynamic systems - systems that change over time. Because of this, the system can be in different states. Between these states, the structure of the system can change or functions can change.

When the differences between the states are small, the engineer can try to combine the various states in one FM. However, as the differences become larger, it becomes harder to combine the FMs of different states and consequently separate FMs are required. This makes it harder to keep an overview of the system in all states. The various approaches to handle this problem that are currently used are discussed in [12].

These approaches each have their shortcomings and therefore Dynamic FM was introduced – a transient FM that is defined as a function of time. One could perceive this as a movie where the frames are FMs of the system's states.

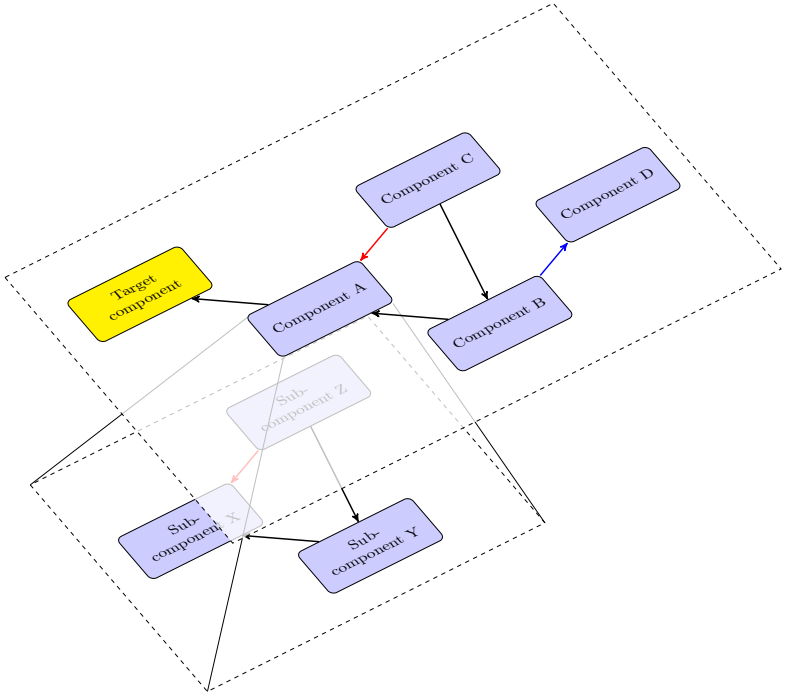


Figure 2.2: Multi-layer function model

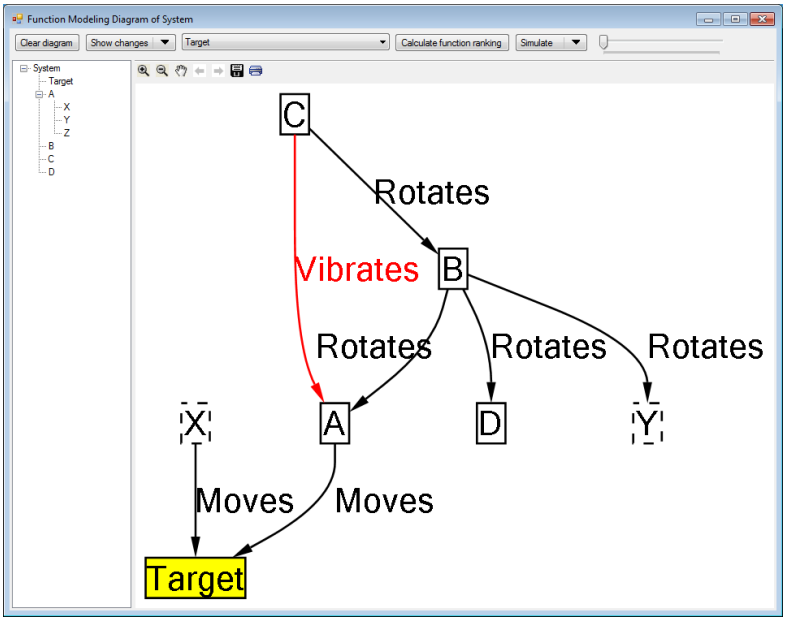


Figure 2.3: Multi-layer function model of figure 2.2 in ModelEvolution

2.2 TRIZ and Computer Aided Innovation software

Specialized software has been developed to assist engineers in applying TRIZ. The concept *TRIZ software* can be divided in two groups: software that helps engineers to apply a principle developed using TRIZ and software that guides in the process of applying TRIZ.

The software presented by Cardillo et al. in [10] is an example of the first group. Their software performs Topology Optimization, based on TRIZ principles, to solve geometrical contradictions in CAD software.

The second group is also referred to as Computer Aided Innovation (CAI) software (although not all CAI software is always related to TRIZ). The software assists engineers in their innovation-related tasks. In [23], Hüsig and Kohn distinguish three groups of CAI software: Strategy Management, Idea Management and Patent Management. Software covering more than two categories are labeled as Holistic Solutions. Two examples of such software are Invention Machine Goldfire [20] and TRIZacquisition [46].

In [23] a broad overview of CAI software, its advantages and a market overview are given. Two publications more specifically targeted at TRIZ are written by Cascini [11] and Ikovenko [25].

The software developed during the research for this thesis is based on the software resulting from the research for [6] and [13]. These publications describe the integration of CAD software and FM. Furthermore they explain how engineers can be assisted by software when performing innovation tasks using FM diagrams.

3 Mathematical modeling and its support by software

3.1 Introduction

This chapter will discuss mathematical modeling. First there will be explained what it is and what it can be used for. After that, different software types that can help engineers with mathematical modeling are discussed and some examples of each of these software types are given. Finally, the Modelica language and its implementations are discussed because this software will be used for the research of this thesis.

3.2 Mathematical modeling (of dynamic systems)

Mathematical modeling is a technique to describe (some part of) the reality using a system of equations. Bender [7] defines it as "an abstract, simplified, mathematical construct related to a part of reality and created for a particular purpose". The mathematical nature yields that the results will be exact, but not per se precise - the validity of the outcomes depend on the assumptions made during the modeling and the input data.

The purposes of such modeling can be the analysis of a current situation, simulation (for example to predict experiment outcomes without spending money on real experiments) and optimization. Optimization can be either done analytically or by performing a set of simulations using different parameters and selecting the best parameters afterwards.

A special type of mathematical models are models of *dynamic systems*, where their state can described as a function of time. Such systems usually consist of differential equations which need to be integrated over time to simulate them. Mathematical software can perform this function.

3.3 Mathematical modeling software

Various software packages exist that provide support for mathematical modeling. Generally spoken, there are two types of mathematical software (besides spreadsheet software). First, there is software using a procedural language: it performs calculations step by step following the input of the user or following a script. Examples of this software type are Matlab, Octave, Maple and Mathematica.

The other type of software is software where the user can make a model of a situation, and have the software consider the equations of the model as a whole during the simu-

lation phase. Examples of this software type are Simulink [40] and various Modelica [5] implementations. For this software type, the order of equations does in the model files does not matter.

When comparing these two types of software, the second type is usually of a higher abstraction level. One reason for this is that for the first software type users have to reorder equations themselves. Another reason is that they can also use commands between the equations - for example to change some values between the calculations.

Simulink and Modelica both provide a graphical interface for building models as block diagrams. Using these diagrams, the user does not even need to directly deal with equations anymore – the software can translate the graphical block representation to a system of equations.

The simulation of the models is done by numerical integration of the differential equations that make up the model. Both Simulink and Modelica implementations provide several solvers that perform this task.

3.4 Modelica

Modelica [5] is a standardized mathematical modeling language that can describe the dynamics of technical systems. Although the models themselves are text-based, they can be represented by block diagrams like Simulink [40]. Another similarity is that both systems can have models that are built up from multiple layers: a model of a DC-motor can be built from electrical components. This motor model can then be re-used as a *black box* in other models where the user does not need to worry much about the workings of this black box (except for setting some parameters).

Although there are similarities between Simulink and Modelica, there are also differences. First, Modelica is not a program, but a language. Several implementations (computer programs) are available that provide support for this language, like Dymola, OpenModelica, MathModelica, CATIA and others. Each of them provide access to the Modelica Standard Library [5, p. 213], consisting of basic models that are provided by the Modelica Association. This makes sure that not too advanced models will work across the various implementations. Besides this, some of the implementations also provide their own additions to add value for their users – for example Dymola provides advanced car modeling libraries like used in [16].

The second difference is that – as a part of the Modelica Standard Library – Modelica provides support for physical units: variables not only have a value, but can also have a unit. This information can be used for, amongst others, unit checking [4] (to correct mistakes) and unit conversion.

Another difference is the support for acausality provided by Modelica. In Simulink, connections between models are visualized by arrows. These arrows have a direction, clearly showing the input and output of the blocks that the arrows connect. In Modelica, most models have connections *without* such a direction, making the relations acausal. This principle makes it easier to model physical systems. For example, a motor does not only deliver torque to other components, but the delivered torque also works upon the

motor as a reaction torque. In Modelica this is possible using only one connection, while modeling it in Simulink would require a feedback loop.

Because of the advantages over Simulink and other mathematical software (support for modeling of physical systems, acausality and block diagrams), Modelica is chosen to be used for the mathematical modeling support. Out of the various available Modelica implementations, OpenModelica was chosen because of its free availability (in contrast to commercial implementations, for which licenses could not easily be obtained) and because of its possibility to be integrated via CORBA. CORBA is a technology that allows different computer programs to communicate with each other. The usage of this technology is explained in section 6.3.3.

4 Advantages of software integration

The goal of this project is to achieve integration between several software components. There are several reasons for applying software integration. Hüsig and Kohn explain that it can enhance efficiency [23, p. 553] by removing the need for duplicated work and by preventing mistakes. They also state that software that aggregates, structures and visualizes data enhances effectiveness by lowering complexity and increasing clarity. This can be achieved by using the data from the various integration components.

Effectiveness can also be increased by streamlining the users' workflow. When people have to switch less often between programs, or when they do not have to switch at all between programs anymore, they will lose less time.

Integration can be a starting point for automation tasks. By combining data or reusing data, some tasks can be automated. Examples of this are given in [6] (like interaction detection and function proposing) and [13] (function ranking).

Integration can also stimulate the cooperation between various people. People that used different programs before, targeted at their own job, can now use the same software (collection) and can always have access to the latest version of the data.

An well-known example of software integration, that is used in product development, is Product Lifecycle Management (PLM) software. It is a term for software that engineering teams can use to create and share all sorts of data, related to the product and its parts, in a central place. Such software can be Computer Aided Design (CAD) software to design the parts, but also Computer Aided Engineering (CAE) software to perform analyses and Computer Aided Manufacturing (CAM) software to control the machines are examples of PLM software. All these types of software work together to perform the tasks needed during product development.

Part II

Research results

5 Workflow of integrated components

5.1 Overview

In figure 5.1 the workflow can be seen. The CAD software is placed on the left-hand side since the product improvement process is assumed to start with a CAD model. From here a FM and a Modelica model are generated. This is discussed in sections 5.2 and 5.3.

Some literature and ideas exist about going from FM to CAD. They are presented in section 5.4, but since there is no implementation available the arrow is drawn dashed in the workflow diagram.

The work done for this thesis focuses on the interaction between FM and Modelica, which is described in sections 5.5 and 5.6. Changes made in the FM structure are reflected in the Modelica model. Besides that, Modelica code can be edited through the components of the FM. The results from the Modelica model simulation can be used and shown in several ways in the FM diagram. The possibility of generating FMs from Modelica is mentioned in section 5.6, but is not investigated further.

The Modelica model can be optimized to find the design parameters that make the system perform best. This process is described in section 5.7. The users can be assisted in composing an optimization objective by information from FM and TRIZ, as explained in section 5.7.1.

TRIZ theory can help improving CAD models and FMs, both direct (through software assistance) or via a thinking process performed by engineers. This is discussed in chapter 2 and this will not be repeated here.

The optimization results can be used for improving the initial CAD model, as shown in section 5.7.2. The optimization process can also be used to find conflicting optimization interests, called *contradictions*, which can be solved by TRIZ (see section 5.7.3).

5.2 CAD to Function Modeling

Solutions for the CAD to Function Modeling part have been published in [6] and [13]. The research done for these publications led the development of ModelEvolution. The software first extracts the component hierarchy in a CAD assembly to create the component tree for a multi-layer FM (as explained before in section 2.1.2). After that it checks the interactions between components and pre-fills an Interaction Matrix that the user can complete. Then the user is guided through the process of selecting the right function for each interaction that was checked in the Interaction Matrix based on component names, interaction type and component metadata. Furthermore ModelEvolution is able to respond to events in the CAD assembly and in the FM, and it can provide the user

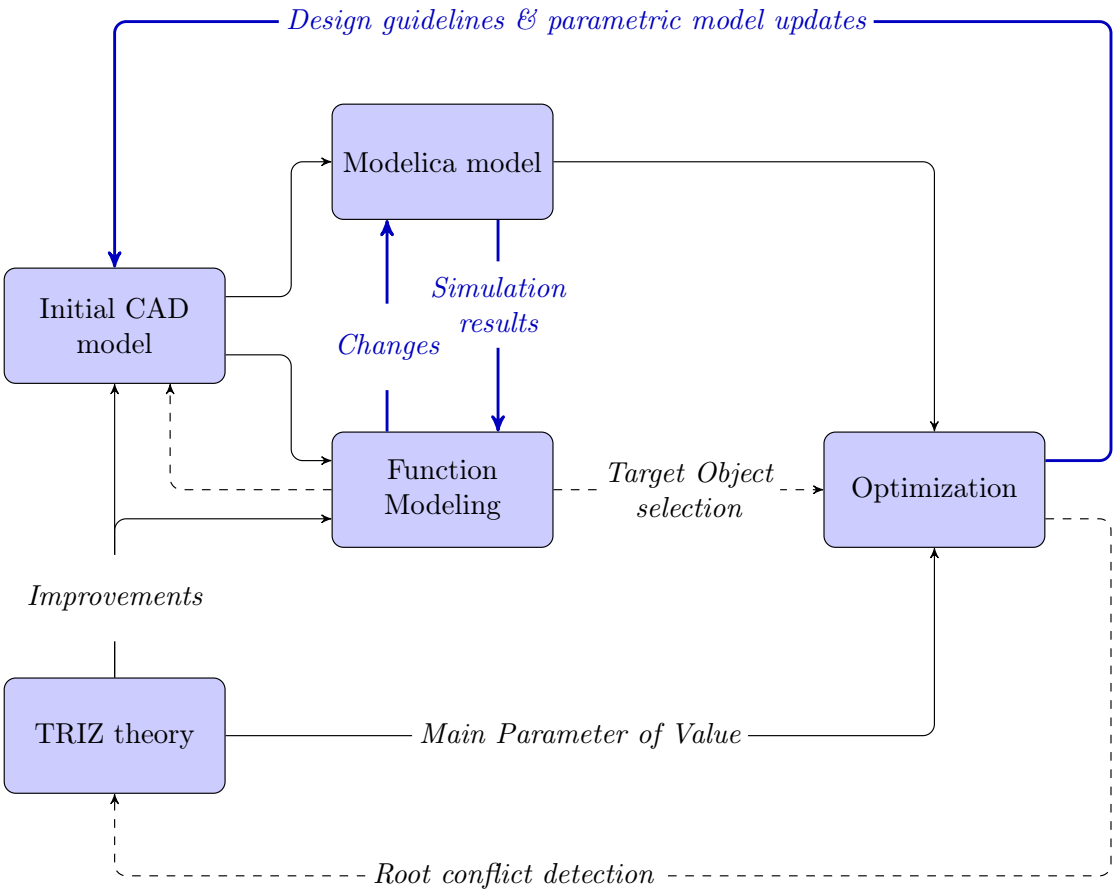


Figure 5.1: Overall workflow *(figure 1.1 repeated)*. The **thick blue arrows** show the interactions that are worked on during this project. The dashed black arrows stand for interactions that are not implemented yet and that are not worked on during this project. Available interactions are shown using normal black arrows.

with more information about functions and principles that can be used to perform these functions.

ModelEvolution is used as a basis for this thesis research.

5.3 CAD to Modelica

The conversion of CAD models to Modelica models has been researched by multiple people. One of the earliest examples is the master thesis [28] by Larsson. Since then, both the Modelica language and the translation software from CAD to Modelica advanced a lot. One of the most recent publications in this field is the Ph.D. thesis [26] by Juhász. Besides giving a broad overview of the state of the art, he describes the software he developed that enables a workflow to convert CAD assemblies in Pro/Engineer to Modelica models that can be simulated using collision handling and accurate visualization.

Conversion from CAD data to Modelica models comprises the extraction of component data from CAD and the extraction of relations between the components (constraints, also known as mates) like joints. This data forms the basis for what is needed for a mathematical model. For collision handling, the shapes of the components are needed too. These shapes can also be used for better visualization of the system during the simulation, because by default physical components in Modelica are visualized as primitive shapes.

Importing of CAD models is currently possible in commercial software like Dymola and SimulationX.

5.4 Function Modeling to CAD

At the moment, no software is available that implements this part of the roadmap depicted in figure 5.1. However, there are some publications that present parts of the solution for this case. One of the first articles about the route from FM to CAD was published by Léon-Rovira [29]. He proposed to create change proposal rules in CAD assemblies, depending on "typical situations of insufficient or harmful actions or results" in FMs. He also proposed to have the CAD program automatically generate alternative designs based on these rules to give the user a quick overview of the consequences.

Bluntzer et al. [8], Gomes et al. [21] and Kratzer et al. [27] describe systems that can assist in generating CAD models based on certain *functional requirements*. These requirements are more specific than the notion of a function in TRIZ, because in TRIZ there is only dealt with quality (what does a component do) whereas functional requirements also deal with quantities. These requirements are similar to the ones used in the House of Quality (Hauser and Clausing [22]). This means that function data from FM cannot be directly linked to the functional requirements that are the input for the systems presented by Bluntzer et al., Gomes et al. and Kratzer et al.

Terninko [44] shows how QFD, TRIZ and Taguchi can be combined to find a robust composition of functional requirements. However, this is a manual process and it needs more information than FM alone can provide.

Cascini [11] argues that functional modeling is still a good candidate for linking "the conceptual design stage and detailed design". He gives arguments for this and also mentions the possibility of linking geometrical features to functions in digital CAD libraries, assisting the user to perform routine design tasks.

5.5 Function Modeling to Modelica

From a FM diagram, users can be assisted in creating a Modelica model. First, the FM system hierarchy can be extracted from the FM component tree to create a structure of Modelica models. Second, users can edit the Modelica models from within the FM program. Furthermore, they can be assisted in their choice of connector components. Also, Modelica example models can be shown for principles like *radiation absorption*.

This part of the workflow is developed as part of the research for this thesis. The user interaction with this workflow part is described in a general way in section 6.1 and as part of a case study in section 8.3. Its implementation is discussed in section 6.3.3 and more technical details are given in section B.2.

Since FM is a qualitative modeling approach, no data about quantities is available from pure FMs only. There are two possible solutions for this. The FM software could be extended to support *quantitative functions* (how much does component A heat component B?) and component properties like mass etc. Furthermore CAD data could be used to supply information about component properties (see section 5.3). These solutions are no part of this research.

5.6 Modelica to Function Modeling

The results of Modelica simulation can be used in three ways. The results can be plotted in a graph that is linked to a FM component, but more interesting are showing the results over time using tooltips linked to components and having simulation results affecting the visibility of components – thereby enabling the concept of *Dynamic FM*.

This part of the workflow is developed as part of the research for this thesis. The user interaction with this workflow part is described in a general way in section 6.1 and as part of a case study in section 8.4. Its implementation is discussed in section 6.3.3 and more technical details are given in section B.2.

SimulationX can visualize the results of a Modelica simulation over time by modifying the color of the connection between models. Users have to manually select a variable to visualize. Similarly to this, functions in the FM could be used to visualize selected simulation results by varying their color or the line thickness of their arrows.

Besides using Modelica simulation results in FMs, FMs could be generated out of Modelica models since these models also have a hierarchy and, more important, connections between submodels. These two last ideas are not implemented during this project, but they are listed in section 9.2.

5.7 Modelica optimization

The next step, after composing a mathematical model of a system (using analysis), is optimizing it to have it perform as well as possible. For simple models this can usually be done analytically, but when systems become more complex or nonlinear, numerical optimization is a possible solution. [45] gives various examples of numerical optimization types and present software that can be used to automatically optimize the parameters of Modelica models. This software is now bundled with OpenModelica.

Besides OpenModelica, other Modelica software like Dymola can also perform optimization.

5.7.1 Optimization objective selection

Each optimization process needs a goal to optimize to, an optimization *objective*. Usually this objective is expressed in the form of a mathematical function that needs to be minimized or maximized.

TRIZ provides two possible ways for assisting in formulating this function. The first one is Target Object selection. During FM, the object upon which the system performs its most important useful function(s) is called the *target component* (usually marked yellow). Then it is also very likely that the useful function(s) performed upon this component should perform as well as possible. By definition, every function changes some parameter of the component it works on. Therefore, the software could show a list of variables of the target component to choose from when composing the objective function for the optimization process. The arrow in the workflow diagram is marked dashed because this idea is not implemented yet.

The second, but less automated, way is through composition of a *Main Parameter of Value*. This method can help in deciding the properties that a product needs to have in order to be innovative and / or outstanding ([31]). These properties, for their part, can then help in formulating an objective function.

5.7.2 Modelica optimization to CAD

The optimization results, consisting of a (set of) parameter(s), could be used in two ways. The first use is for design guidelines, which serve as hints to the designer. The second use is with parametric CAD models.

Design guidelines

If the input for the optimization program would be an optimization objective, optionally together with a maximum tolerable error, the optimization program could generate a set of feasible solution candidates. Each parameter could then have upper or lower limit, needed to reach the objective, which could be presented as design guidelines to the designer. Some experiments with showing design guidelines in SolidWorks were done as a part of this project. They are presented in section 6.5.1.

Parametric CAD models

Parametric CAD models are models of which (some of) the dimensions are not directly set by numeric values, but by parameters or by mathematical functions depending on such parameters. This way valid variants of components can be quickly generated. Almost any of the commercial CAD packages provides support for such models.

Ansys Workbench provides such a link already. It can load parametric CAD models, optimize their parameters for a certain goal (e.g. example mass reduction while ensuring that stresses are not too high) and generate optimal geometries. [36] gives examples of this workflow, but [17] shows that CAD models as used in industry might need preparation before they can be used this way.

In [21] a workflow for generating optimized CAD models is published.

If Modelica parameters would be coupled to the parameters of a CAD model, the CAD model could be automatically updated using the results of the optimization process.

5.7.3 Modelica optimization results and TRIZ

When having more than one optimization objective, it can happen that a parameter needs to be increased for one of the objectives and that the same parameter needs to be decreased for the other objective. It can also happen that a parameter needs to be changed (increased or decreased), but that this is prevented by given boundary conditions. Such situations are called contradictions, which can be solved by TRIZ theory.

The user would greatly benefit from having the optimization software detect such situations, because the software could then propose to start a TRIZ project to solve this problem. It could also assist with the contradiction formulation and for the user it is convenient to have the software taking initiative. Since there is no software implementation of this idea, yet, the arrow in the workflow diagram is drawn dashed.

5.8 Conclusions

A workflow linking CAD software, TRIZ, FM, Modelica and optimization is developed. Existing, possible and developed interactions between the tools are described.

The steps going from CAD to FM and to Modelica already exist, as well as the steps going from TRIZ to CAD, to FM and to optimization. Furthermore, the step going from Modelica to optimization already exists.

The steps going from FM to optimization, from FM to CAD and from optimization to Modelica do not exist yet.

The interaction steps between FM and Modelica as well as the step from optimization to CAD are worked on during this project.

6 Implementation

This chapter will discuss the implementation of ModelEvolution. First its usage is explained. After that, the architecture of software components and their interactions are discussed. Furthermore, some general additions to the FM features (not per se related to Modelica) and some experiments are presented.

6.1 Usage overview

This section will describe ModelEvolution from the user's perspective. Since the general FM functionality of the software is discussed already in [6] and [13], the usage of Modelica features will be presented in more detail in this section.

In figure 7.1 the main window of ModelEvolution is shown. It shows the function model diagram (A) and its component hierarchy tree (B).

Along the top edge of the window, a list of elements is shown to interact with the FM. The leftmost button (C) allows the user to clear the FM (including its accompanying Modelica model) and to start from scratch. The change tracking functionality offered by button D is discussed in section 6.4. Using dropdown-list E the target component (marked yellow) of the FM can be selected. The next button (F) offers *function ranking* as explained in [13].

The last two interface elements deal with the functionality for Modelica support. Button *Simulate* allows the engineer to simulate the Modelica model embedded in his FM. The simulation settings (start and stop time, as well as the amount of timesteps) can be customized, as shown in section 8.4.

While dragging the time slider H the simulation results are shown in tooltips next to the FM components they belong to (see figure). Components are also made (in)visible based on the value of their *visibility* variable. This allows to have the Dynamic FM functionality, as published in [12]. Components are not hidden completely, but they are made light gray. This was chosen to give a visual cue to the user that the component is invisible now, but will be visible on some other timestep.

Every FM component has a Modelica model. The code for each of these models can be accessed through the context menus of functions (figure 7.3a) or of components (figure 7.10a). Through the context menu of components it is also possible to see the simulation results visualized in a plot graph (figure 7.13).

The dialog window to edit the code (figure 7.10b) shows multiple textboxes where code can be filled in: the code is split over the functions that the component performs, and there is one extra textbox for miscellaneous code. Every code edit section related to a function also has a connector proposal box (figure 7.9b) and an *Examples* button.

The connector proposals are a list of Modelica connectors that might be suitable to the physical domain that was selected (figure 7.9a) when adding the function. The *Examples* button links to a dialog (figure 7.3b) that shows phenomena that can perform the function, provided they are available in ModelEvolution's database.

6.2 Modelica entities needing integration

In this section the entities of Modelica that need integration into ModelEvolution will be discussed. Their implementation will be discussed in section 6.3.3.

6.2.1 Model parts

Modelica models usually exist from two parts. In the first part the components of which the model is composed are listed. *Variables*, *parameters* and other models are all added to this section as *objects*, since Modelica is an object-oriented language.

Variables and parameters contain the properties of a model, for example the temperature of some fluid or the electrical resistance of a resistor. The difference between variables and parameters is that variables are calculated depending on other variables or parameters and that parameters are directly set by the user.

The second part of models consists of *equations*. Equations define the relations between variables and parameters within a model. As opposed to most other computer programs, in Modelica it does not matter which variable is on the left-hand side of the equation. This is because in Modelica, equations are no *assignations* but real equations.

A special type of *equations* are *connections*. Connections are used to link the variables or parameters of two separate models. These connections connect the connectors of two models. Connectors are part of these models, like other components. The interesting concept of these connectors is that they can resemble properties of real connectors and that there can be multiple variables in one connector. For example, a standard *Modelica.Electrical.Analog.Interfaces.Pin* has both a *current* flowing through it and a *voltage* potential at it. Similarly, a connection connecting two of such pins resembles a wire. In terms of an equation, a connection sets the variables of its connectors to be equal.

Modelica models can contain *annotations* to store additional data. Not all of this data is standardized (they can differ per Modelica implementation), but the Modelica Language Specification does provide standards for, amongst others, annotations storing the model appearance and the simulation settings.

6.2.2 Simulation

When using OpenModelica, the simulations are performed by the OpenModelica Compiler (OMC). The simulations can be performed according to settings given to OMC, like the start and stop time, the precision, the solver type, the output format and others.

OMC can either output the simulation results as a file containing comma-separated values (CSV), as an AutoCAD plot file or as a Matlab file. These results consist of the values of all model variables for all simulation timesteps.

6.3 Architecture

This section will deal with the software architecture from a bird's-eye view. A more detailed discussion of the software architecture, including UML diagrams (diagrams showing the source code structure), can be found in appendix B.

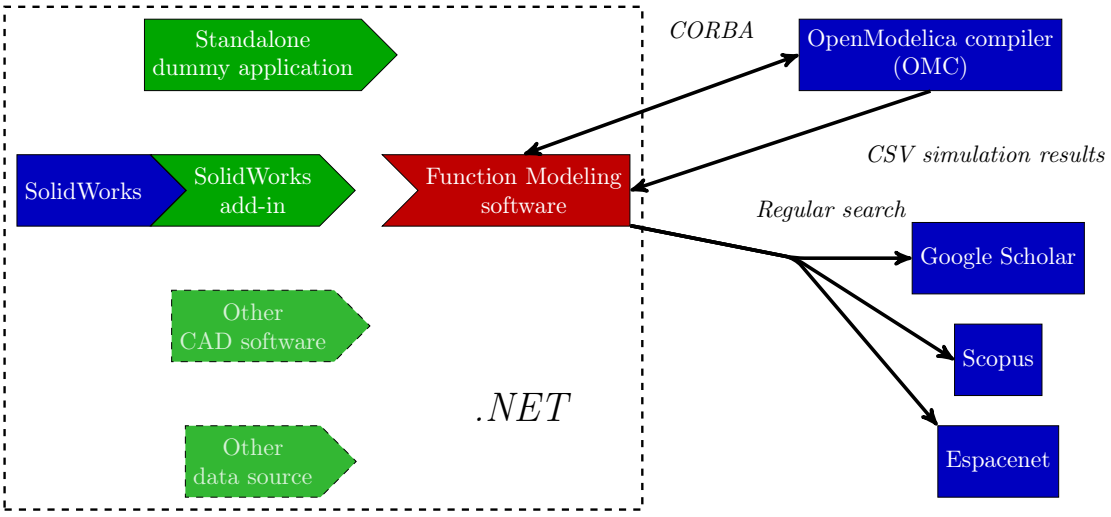


Figure 6.1: Architecture

6.3.1 Restructuring

At the beginning of this project ModelEvolution was directly integrated with SolidWorks. For the research of this thesis, the Function Modeling module was made independent of SolidWorks for three reasons. The first reason was having cleaner code because general code could be clearly separated from CAD-specific code. The second reason was that directly depending on SolidWorks required a restart of the software every time a code change was made, which caused long developing and debugging cycle times. To start the Function Modeling module a dummy CAD plugin was developed. Using this dummy CAD plugin The third reason was that separating the FM code from the SolidWorks add-in code allowed for possible development of plugins for other CAD software.

The separation of the Function Modeling module from the SolidWorks add-in was done by introducing an *interface class* (see [30, ch. 13]). It is visualized in figure 6.1: the green modules, that inherit from the *ICadPlugin*-interface class, can plug into the red Function Modeling module. They are interchangeable.

6.3.2 Function Modeling module

After restructuring, the Function Modeling module contained all code for interacting with the FM like displaying, creating, editing, saving, loading FMs. Also it contained

the code for function proposing, etc. This code is not new and will therefore not be discussed in detail.

Two groups of code were added to this: the code for the Modelica support (see section 6.3.3) and the code for extra FM features (see section 6.4).

6.3.3 Modelica support

The Modelica support is implemented as follows. The Function Modeling module connects to the OpenModelica Compiler (OMC) using CORBA ([15]) via the OMCCorba module (see section B.3). Every action, that is performed on the internal Modelica model (see section 6.1), is performed by OMC:

- adding, removing and changing code
- loading and saving code
- simulating the Modelica model

The commands to make OMC perform the right actions are encapsulated in functions that can be called by other code in the Function Modeling module. These commands are sent to OMC through OMCCorba. These functions are packaged in the *ModelicaHandler* class of the Function Modeling module. The functions are for example called when changing the FM (for every new FM component, a Modelica model is added to the internal Modelica model of the system), when entering the code in a code editing dialog or when clicking the *Simulate* button.

The FM software automatically creates an internal Modelica model of the system based on the hierarchy of the FM components. Also, the Modelica structure is automatically updated upon changes in the FM. After the Modelica structure is generated, it can be filled through the Modelica code editor dialog of each component. There are two reasons for this automatic Modelica model structure generation. The first reason is, that this kind of automation makes it easier for the user because it solves a task for him. Guessing by the software is not needed here (like with function proposing) – the FM structure and Modelica structure match one to one. The second reason is that, by taking control over the Modelica model structure creation, the software can decide the names for the models inside it. This is needed for the extraction and visualization of the simulation results, because they are linked to these model names.

Some simulation settings are fixed because they are either required for the program's workings (like the simulation results format which needs to be processed by the FM software) or they are less often used (like the solver choice). The simulation settings that are set by the user (start and stop time, as well as the amount of timesteps) are combined with the fixed simulation settings when they are sent to OMC to start the simulation.

When the simulation is completed successfully, a file containing the simulation results as comma-separated values (CSV) is produced. These results are read by the CSV reader module published on [33] and stored in a *DataTable* in the *SimulationResults* class of the Function Modeling module. It is needed to store these results in such a DataTable in

memory because a computer cannot easily search for specific values in a CSV file. Re-reading the CSV every time from the hard drive would take too much time resulting in a slow experience for the user, while a DataTable is specifically meant for fast searching of the data stored in it.

An example where specific data has to be found and loaded quickly is the time slider. Every time the time slider is moved a bit, the simulation results corresponding to the new position of the slider have to be loaded. Since 500 timesteps are a normal count for a simulation and since the slider can be moved fast from left to right, it can happen that 500 data queries have to be performed in less than a second (the time it takes to move the slider from left to right). Therefore, querying the data must be as fast as possible to ensure a good performance when working with these results.

The simulation results can also be shown in a plot graph. Here the data in memory can be used again without reloading it from the hard drive.

6.4 Function Modeling improvements

Besides work on the support for Modelica, work was done on improving the general FM workflow. This work consists of the following new features:

- Change tracking. The differences between the initial FM (for example gained from importing a CAD model) and an improved FM (for example the result of trimming) are recorded and can be presented in a report
- More supported search engines. Espacenet and Scopus were added to support respectively easy access to patents searches and literature searches
- Improved phenomena proposing. The user interface combines information about the phenomenon, the supported search engines, required items to make the phenomenon possible and a Modelica example

These features will be presented by a small demonstration about the heating of a cooking pot.

First a FM of a cooking pot, that is heated by burning gas, is drawn. This is set as the initial FM, to start the change recording (see figure 7.2a). When right-clicking a function, a context menu for interaction with this function is shown (figure 7.3a). From here the user can ask ModelEvolution to suggest phenomena that can perform the clicked function.

Each phenomenon shows some information along with some examples (figure 7.3b). Furthermore there are buttons to search for patents on Espacenet (figure 7.6) or to search for literature on Scopus and Google Scholar. Also, a list of requirements can be coupled to the phenomenon information (figure 7.4). In a future version, this list could be used to propose the engineer to add the components on the list, when he decides to use the respective phenomenon.

Modelica also has its place amongst the improvements. Every phenomenon has a button to show a Modelica code example – provided that an example is available in

the database. A button to easily copy-paste the code is available in the code dialog. In figure 7.5 an example model is given for *Radiation*. These phenomena examples can also be shown by clicking the *Examples* button from the Modelica code editor (see figure 7.10b).

In figure 7.2b the updated FM is shown, where the burning gas is replaced by electromagnetic induction to heat the cooking pot. When clicking the *Show changes*-button (*D* in figure 7.1) the initial FM (figure 7.2a) and the new FM are compared. If no initial FM would have been set, an error message would have been shown.

The changes are shown in a change report (figure 7.7). It can be seen that the *Electromagnetic induction* was added and that the *Burning gas* was removed. The removed and added functions are also shown in the report.

6.5 Experiments

Besides the features that ended up in the ModelEvolution prototype, some experiments are done to put some more thought in some ideas or to try whether certain features are useful or not. The most notable ones are discussed in this section.

6.5.1 Design guidelines in SolidWorks

Although Modelica optimization was not the focus area of this research, some effort was put in the part of showing design guidelines in SolidWorks. This was done, because this part of the workflow had not been researched before and it was interesting to visualize this idea to widen the scope of this project.

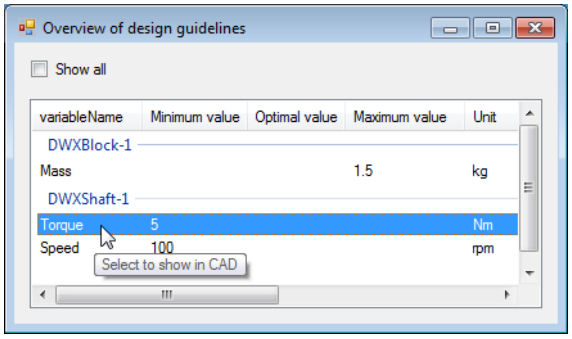
A *mockup* is made of a software component where all available design guidelines are presented to the user (see figure 6.2a). A *mockup* in this context is software that does not contain all program logic to be fully functional, because its purpose is to visualize certain principles. In this case the presented design guidelines are manually entered – they are not the result of an actual optimization process because this would require extra development of the OpenModelica optimization program.

The design guidelines are grouped per component and they are divided in minimum values, optimal values and maximum values. The unit of the guidelines is shown as extra information.

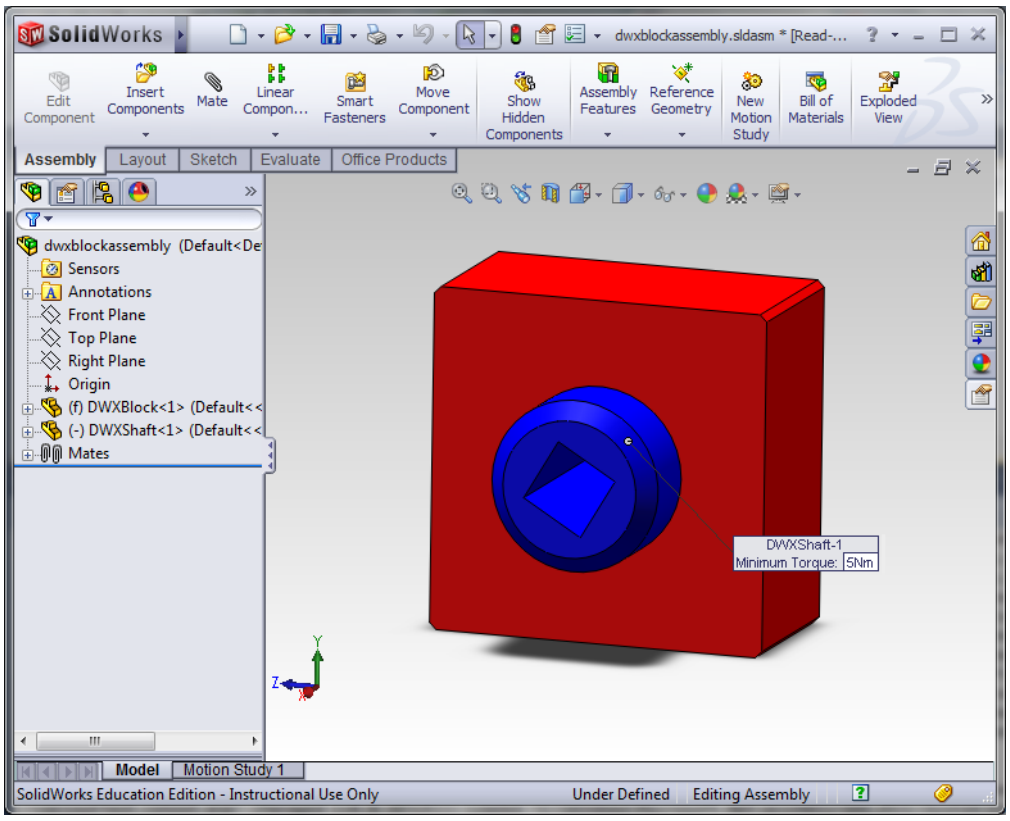
Upon selection of a design guideline in the guidelines overview of figure 6.2a, the guideline is shown in SolidWorks in a *callout* that presents the information to the user. This dialog type was chosen to present the guidelines because it can be visually connected to the parts they belong to in SolidWorks.

6.5.2 Automated model connecting

While pursuing the goal of assisting engineers as much as possible, it was tried to automate connecting the Modelica models that share functions (the model of the component that performs a function and the model of the component upon which this function is performed).



(a)



(b)

Figure 6.2: In the overview of design guidelines (a), the user can select the ones that are visible in SolidWorks (b). The *callout* in this example shows that the motor shaft should at least be able to provide a torque of $5Nm$

This was done by letting the user select a connector from the list of proposed connectors (see section 6.1) or by letting the user enter a connector type manually. After giving the connector a name, the connector would be added to both models of the components that share the function and a *connect* statement would be added to the model of their parent component.

There are two requirements for this feature to be helpful. First, the user should have a clear overview of all connections in the system. A lacking overview might cause confusion and the user might forget models parts are connected already and which models are not. The second requirement is that the user still needs the possibility to manually add connections. Otherwise the user might feel constrained to what the software limits him to do.

7 Software screenshots

This chapter contains screenshots of various workflows within ModelEvolution. They are grouped here to offer a good overview of its usage steps in a central place.

7.1 Main window

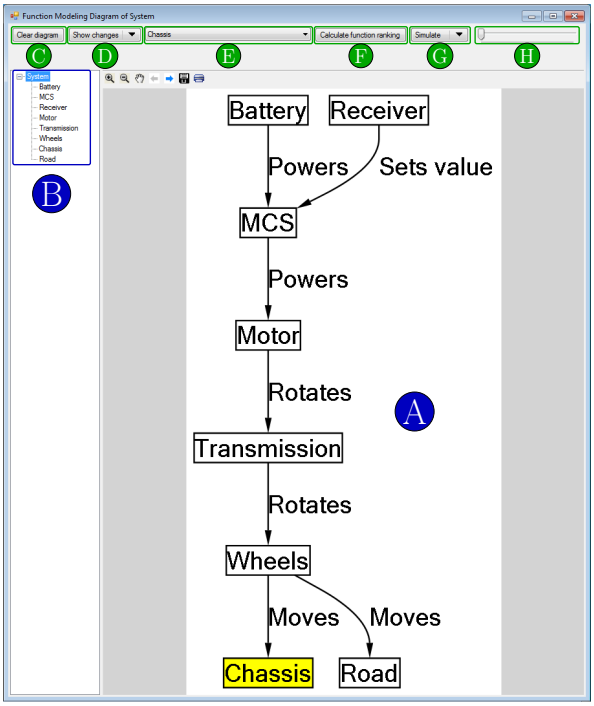
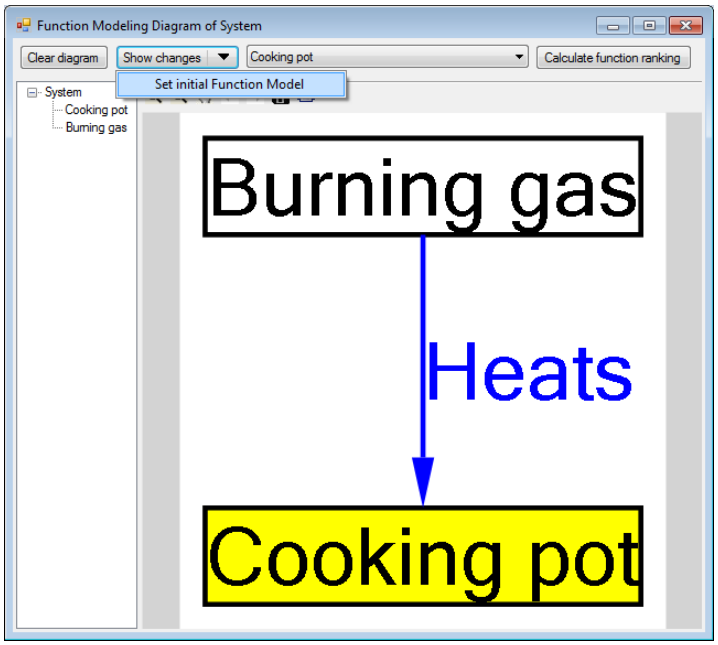
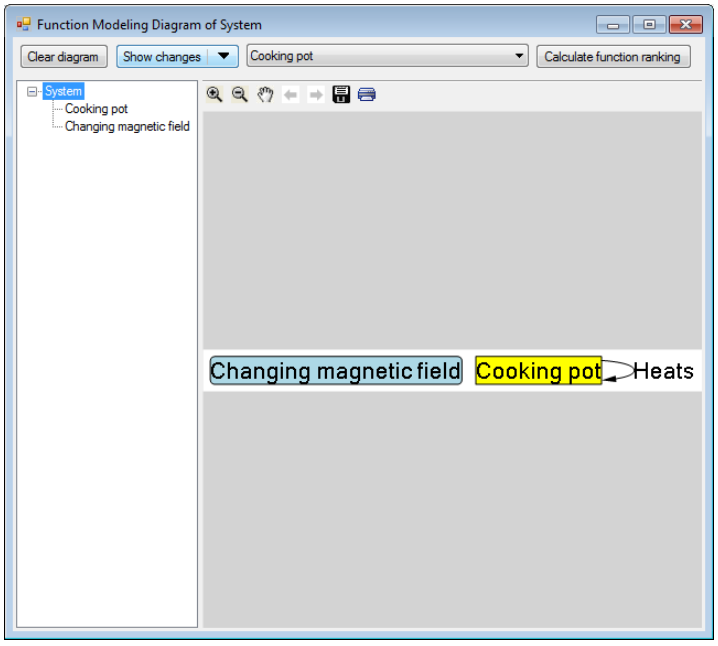


Figure 7.1: Main window

7.2 Function modeling improvements

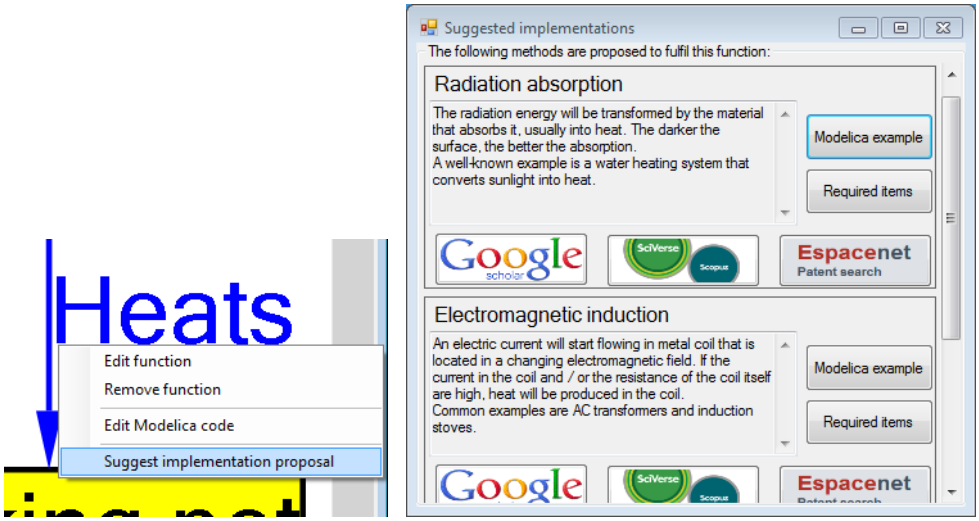


(a)



(b)

Figure 7.2: The initial FM when starting change tracking (a) and the updated FM using a different working principle (b)



(a) (b)

Figure 7.3: Context menu (a) and phenomena proposals (b) for the function *Heats*

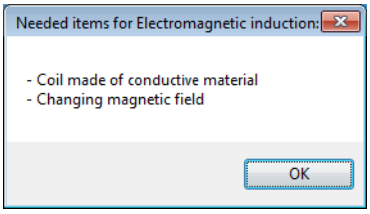


Figure 7.4: Requirements for *Electromagnetic induction*

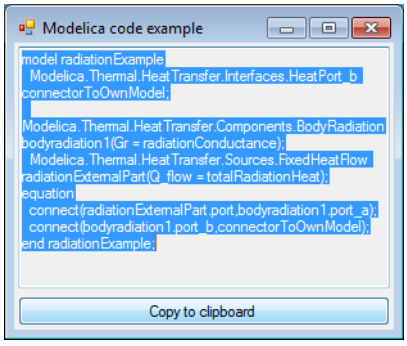


Figure 7.5: Modelica code example for the phenomenon *Radiation*

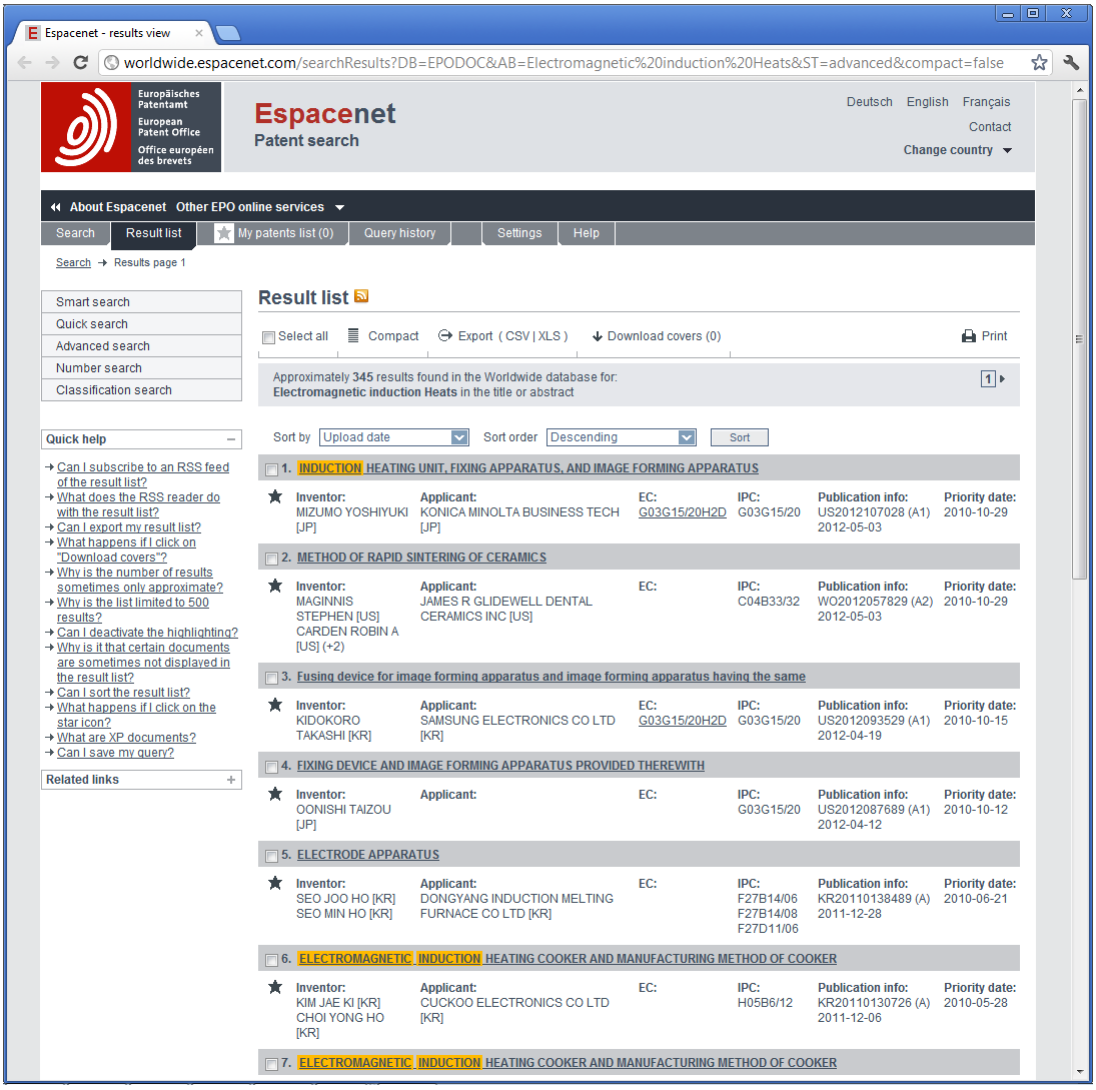


Figure 7.6: Espacenet patent search results for *Electromagnetic induction* combined with *Heats*

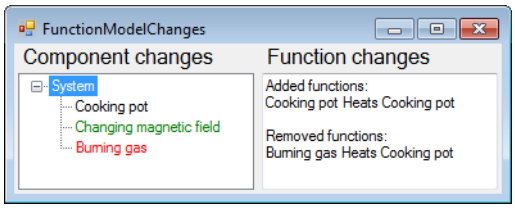


Figure 7.7: Change report

7.3 Case study

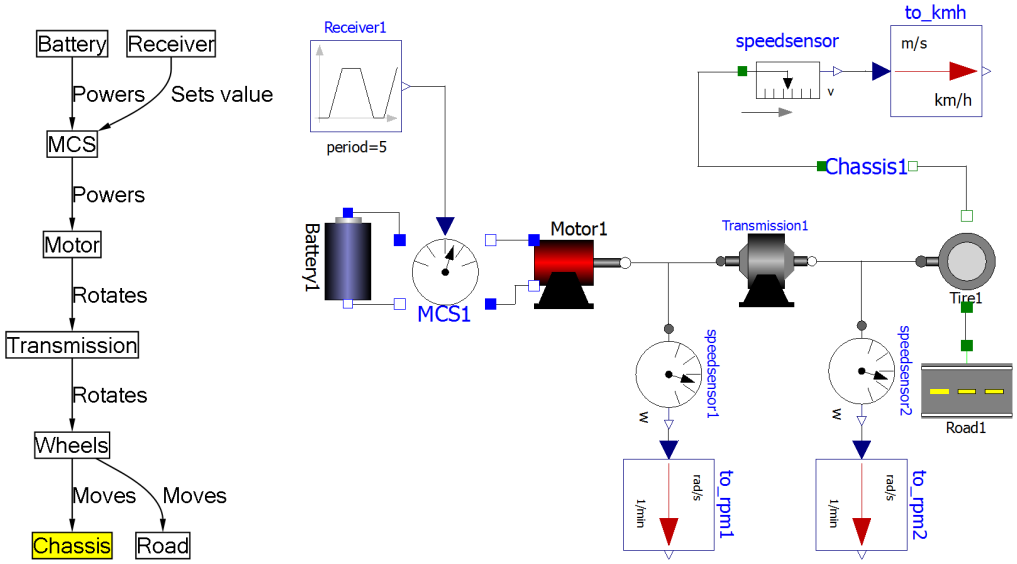
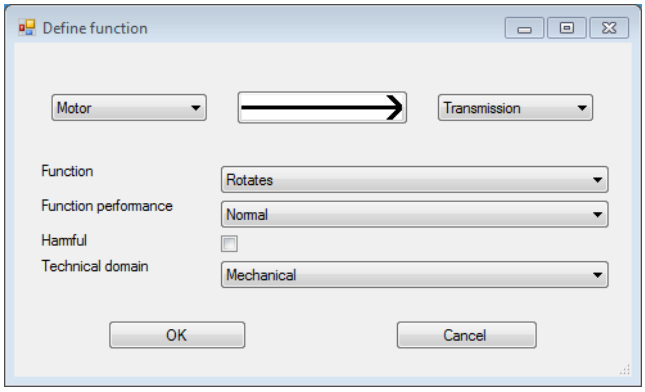
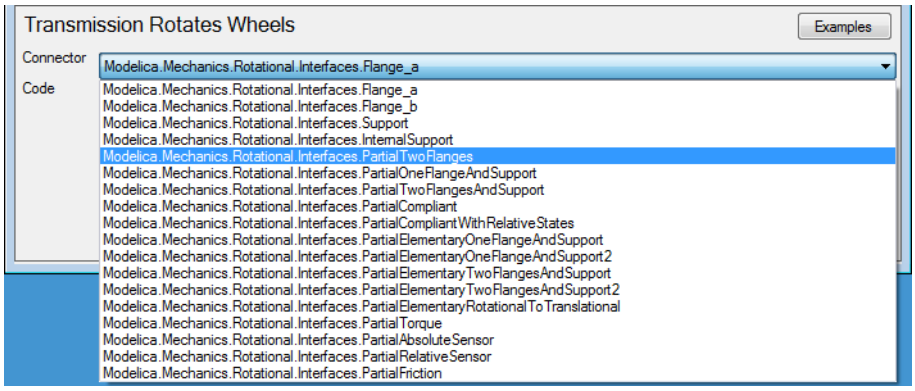


Figure 7.8: Function model diagram (a) of the RC-car and corresponding Modelica model (b)

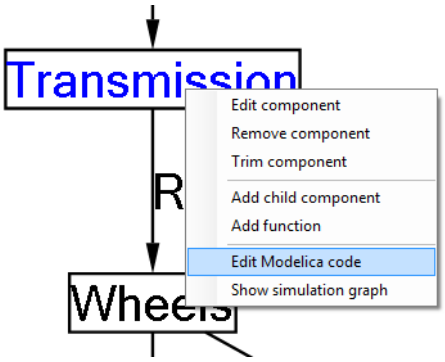


(a)

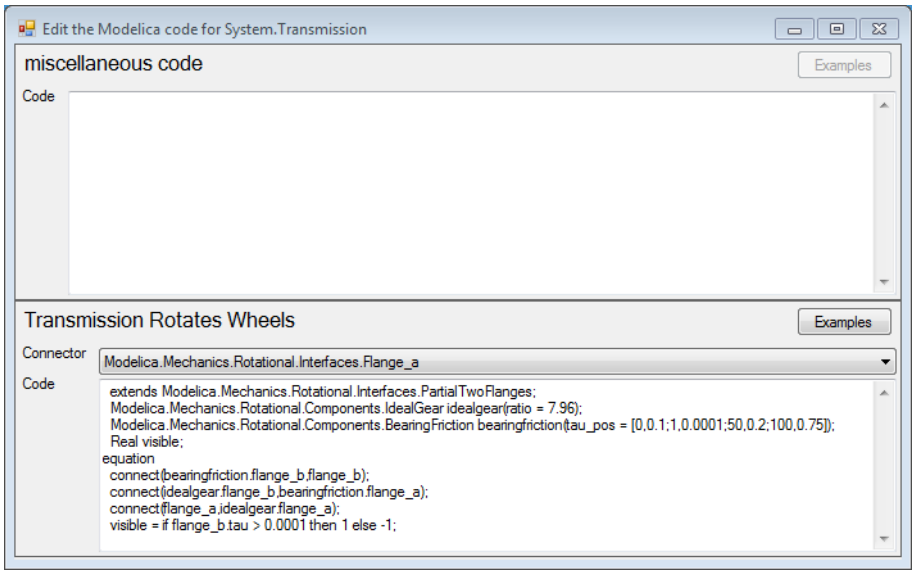


(b)

Figure 7.9: ModelEvolution proposes the function *Rotates* and the domain *Mechanical* when the component *Motor* is selected (a). The selected domain is used to propose a list of possible Modelica connector models (b)

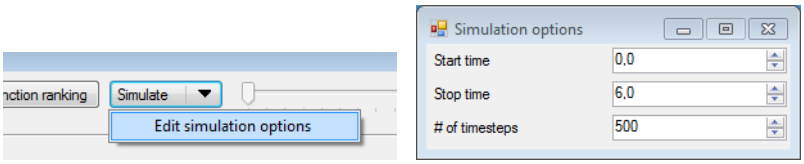


(a)



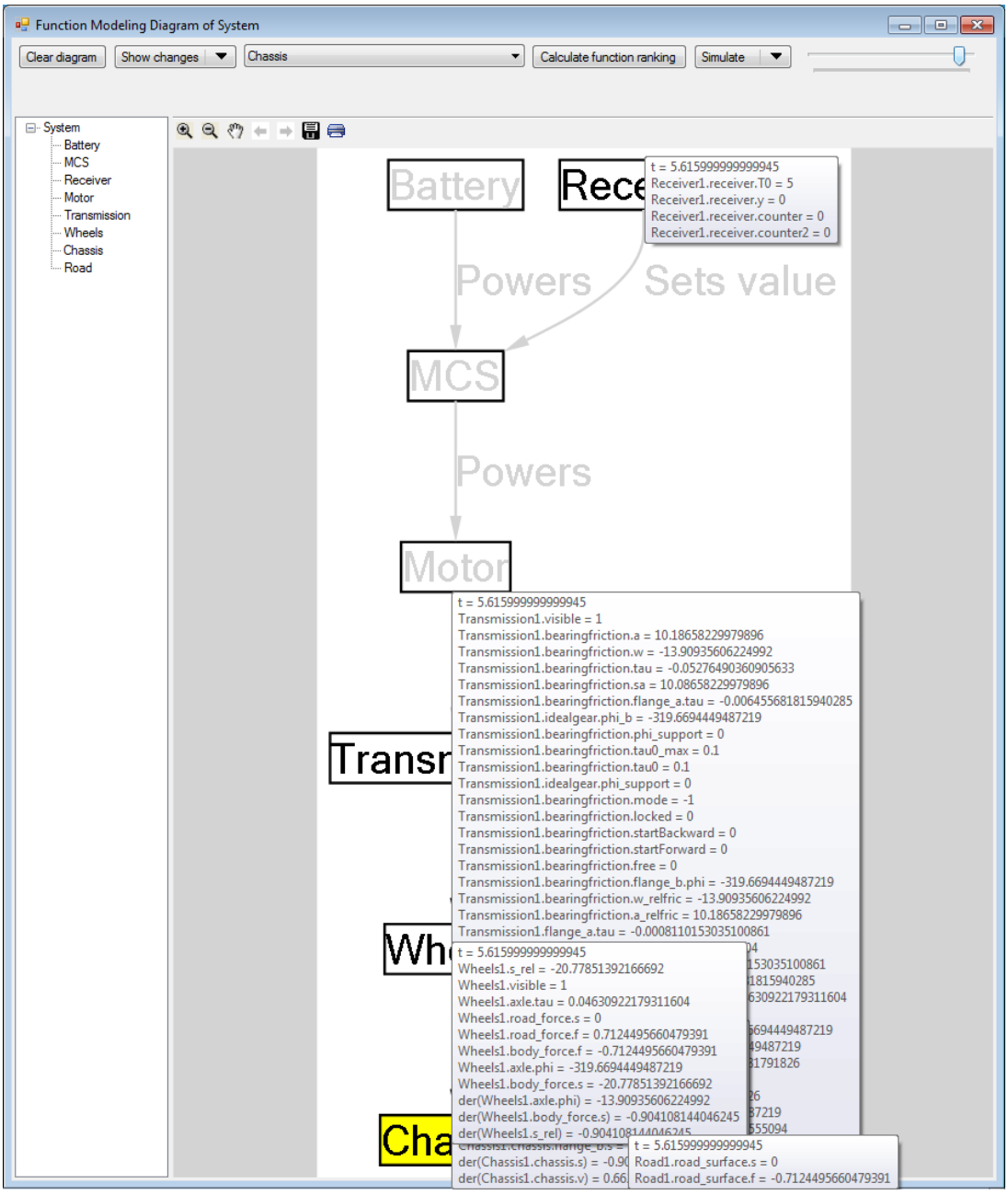
(b)

Figure 7.10: Right-clicking on a component allows the user to enter the Modelica code for this component (a). The code for the *Transmission* component is entered (b)



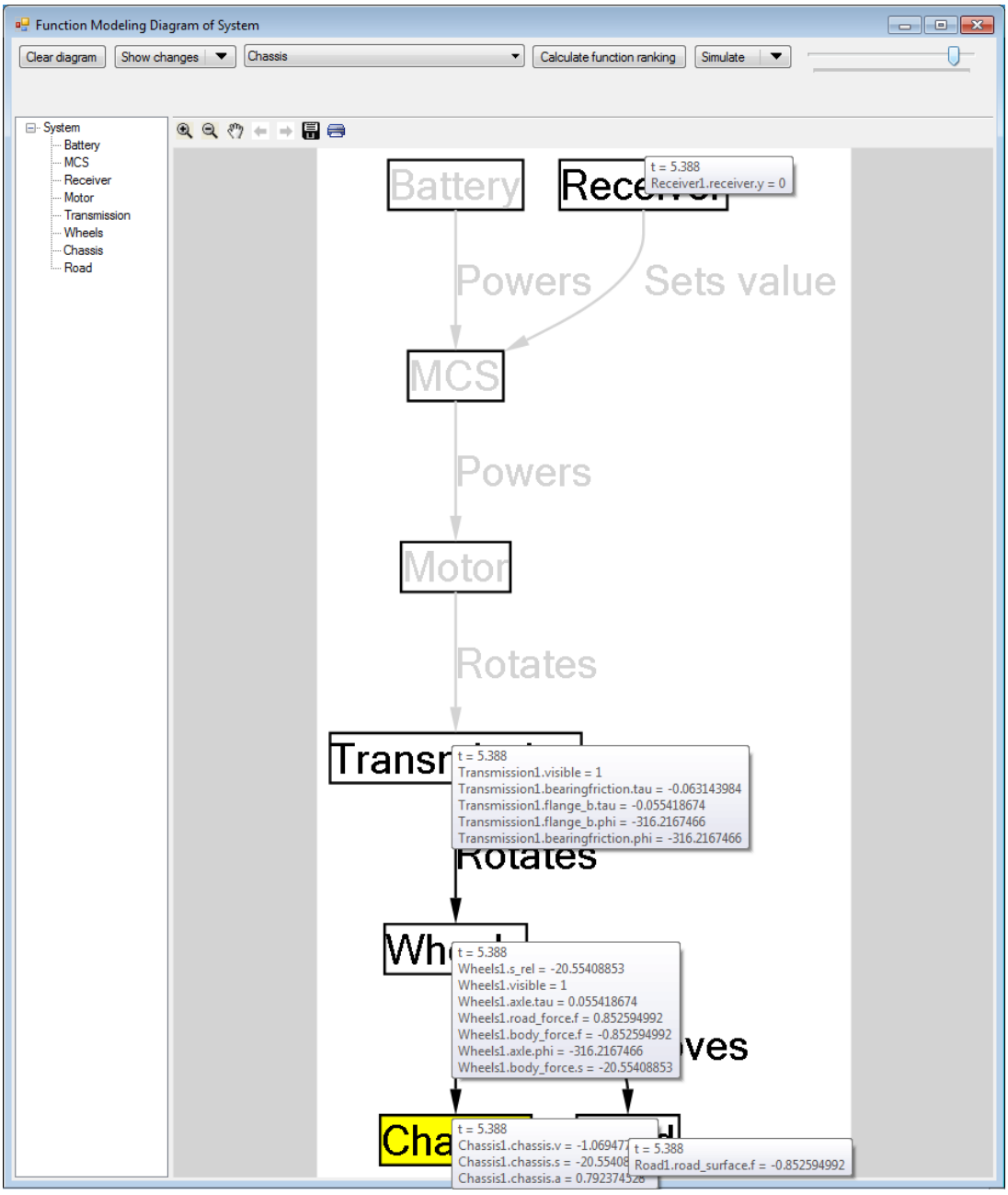
(a) (b)

Figure 7.11: The simulation options dialog (b) can be reached by the menu built into the *Simulate*-button (a)



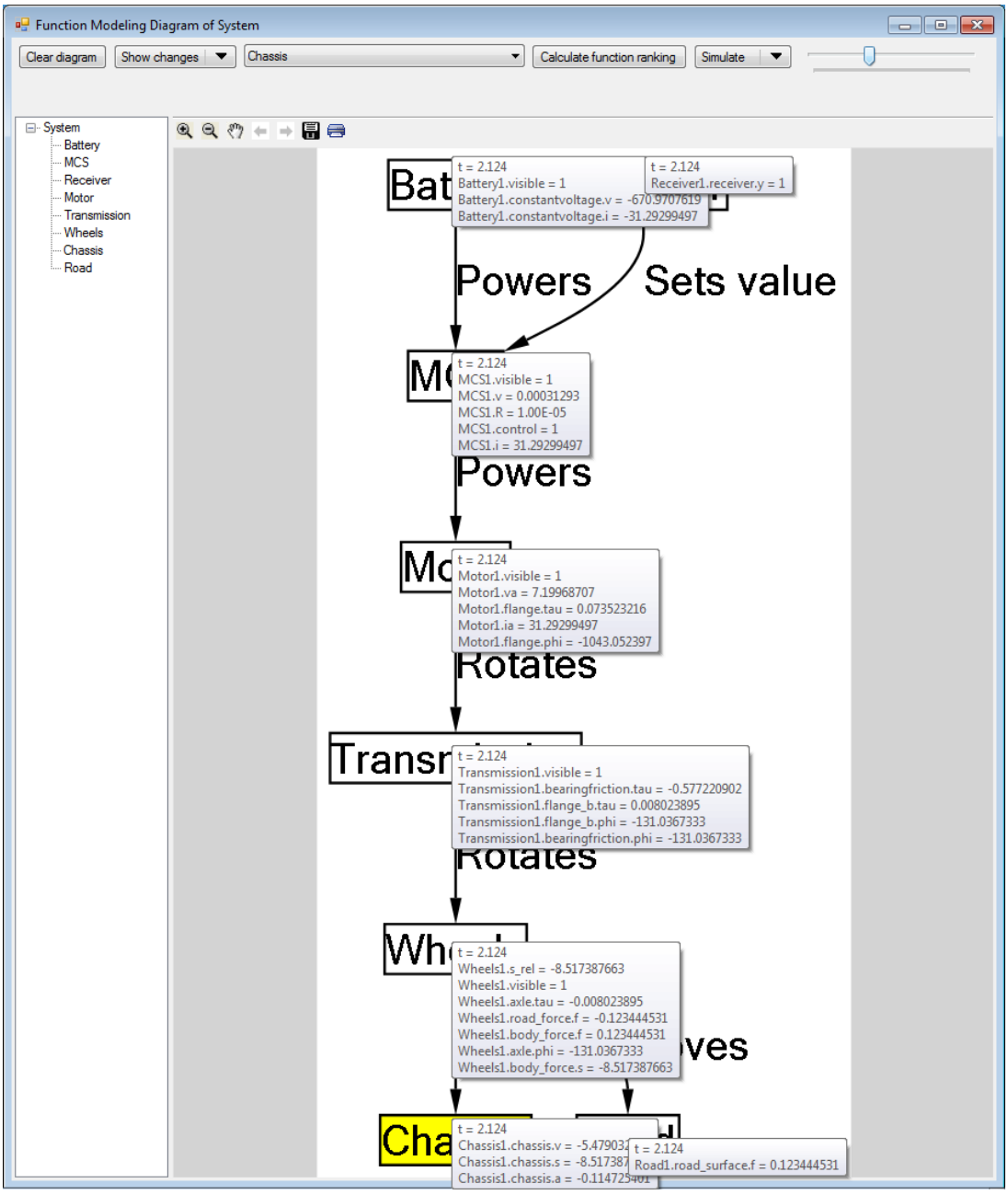
(a)

Figure 7.12: While dragging the time slider, the simulation results are shown in tooltips and the components are visible depending on the value of their *visibility* variable. The difference between not filtering the results (a) and filtering the results (b) is clearly visible. The difference between invisible (b) and visible (c) components is also shown



(b)

Figure 7.12: While dragging the time slider, the simulation results are shown in tooltips and the components are visible depending on the value of their *visibility* variable. The difference between not filtering the results (a) and filtering the results (b) is clearly visible. The difference between invisible (b) and visible (c) components is also shown



(c)

Figure 7.12: While dragging the time slider, the simulation results are shown in tooltips and the components are visible depending on the value of their *visibility* variable. The difference between not filtering the results (a) and filtering the results (b) is clearly visible. The difference between invisible (b) and visible (c) components is also shown

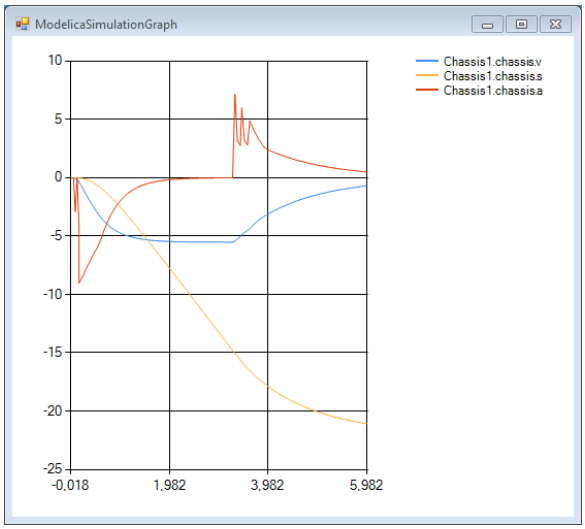


Figure 7.13: Graph of simulation results for the *Chassis* component

8 Case study

8.1 Introduction

To demonstrate the workings of the software, a case study concerning an RC-car (a toy) is presented (shown in figure 8.1). This project focuses on the integration of FM and Modelica. Therefore, the case study concerns of a function model, that is extended with a Modelica model.

Figure 8.1: Tamiya item #58255

The model will cover the acceleration and deceleration of the RC-car over a relatively short time. First the FM was created in ModelEvolution and the Modelica model was created in OMEdit (part of OpenModelica) for easy editing of the details. After that the Modelica model was merged into the FM. This will be discussed in section 8.3. The resulting model can be seen in figure 7.8b. The assumptions done to construct this model are discussed in section 8.2. Several speed sensors and unit conversion blocks are included in the Modelica model, but these will not be discussed further.

Finally the model was simulated, which will be discussed in section 8.3.

8.2 Modeling assumptions

During the modeling, some assumptions were made to make an abstraction of the real car. These assumptions are discussed below. The data comes from three different sources: the specifications [43] of a RC-car kit from Tamiya, an extensive review [38] of a RC-car with a comparable frame and the specifications [37] of the RS-540SH motor that is used in these cars by default. Because this case study is more aimed at showing the integration of

Modelica modeling and function modeling, the actual values are not of vital importance. However, some effort is done to acquire realistic values.

The mechanical speed controller (shown as MCS1 in the Modelica model) is modeled as a resistor with a variable resistance. The model can take a value between 0.0 and 1.0 as input where the first value marks a stop and the second value marks full speed. For $0.2 <= control <= 0.5$ the resistance is 0.4Ω. For $0.55 <= control <= 0.75$ the resistance is 0.2Ω. For $0.8 <= control <= 1.0$ the resistance is $1 \cdot 10^{-5}\Omega$. For all other values the resistance is $1 \cdot 10^{5}\Omega$. This approach of using a variable resistor is chosen because more realistic models using switches (to open the circuit like the real controller does) did not work in OpenModelica.

The receiver, modeled as a linear profile, is used to control the mechanical speed controller. It is chosen to take $0.1s$ to rise from 0.0 to 1.0, then to have its value remain 1.0 for $3.0s$ and after that it takes $0.3s$ for its value to return to 0.0 again.

Because the model only covers a short time, decharging effects of the battery are not taken into account. Therefore, the battery can be modeled as a constant voltage source. According to [38] the voltage is $7.2V$. To make it a valid Modelica model, the Battery1 model also contains a "Ground" component.

The motor is modeled in a similar way to the approach explained in [35]. However, the influence of the inductor is neglected since it is assumed to be small compared to the mechanical (rotational and translational) inertia of the rest of the system. The motor properties are calculated from the motor specifications [37]. The motor constant $k_m = 2.35 \cdot 10^{-3} Nm/A$ is calculated using data from the most efficient torque point ($I = 13.0A$ and $T = 30.6 \cdot 10^{-3} Nm$ at $197400rpm$). The internal resistance $R_{internal} = 1.796 \cdot 10^{-1}\Omega$ is calculated using the same data as follows: the voltage-drop over the motor is assumed to be equal to the sum of the back-emf voltage and the voltage drop over the internal resistance. The back-emf voltage of $4.87V$ is given by $U_{emf} = k * \omega$ (ω in rad/s). The total voltage drop is $7.2V$, resulting in a voltage drop of $2.33V$ over the internal resistance. Using the given current, the internal resistance can be calculated by $R_{internal} = U_{res}/I$.

The Transmission1 model is composed of an IdealGear component and a BearingFriction component. According to [43] the transmission ratio is 7,96:1, so $i = 7.96$. The friction in the transmission is lumped together with the other friction components.

The BearingFriction component in Transmission1 is added to account for the friction of the transmission, the bearings, the wheels and the air drag. This friction is empirically determined by setting the friction torque in such a way that the maximum speed of the car is around 20km/h. It was found that the needed friction torque to limit the maximum speed is around $0.6Nm$. Since the friction is non-linear (at a standstill the friction of the rotating parts is a bit higher than when it just starts running and after that the friction is assumed to increase quadratically) a table is used to friction torque values for various angular velocities.

According to [43] the mass of the total car is $m = 1.076kg$. This accounts for the translational inertia of the car.

The rotational inertia of all components (of the motor, the transmission and the wheels) together is lumped as one inertia component, but the value of this inertia component turned out to be negligible compared to the translational inertia of the car. Therefore it

is omitted from the model.

All wheels together are modeled as one wheel (Tire1 in the Modelica model). This is allowed because it only transforms the torque and rotational angle of the transmission to a force and displacement. The tire diameter is set to $d = 65 \cdot 10^{-3} m$ according to [43]. The wheel in the FM moves both the chassis and the road. This is because the axle of the wheel moves the chassis forward and because the wheel pushes the road backward.

8.3 Modeling

First the FM is modeled in ModelEvolution. The software assists in selecting the function and domain by pre-selecting probable candidates. An example of this can be seen in figure 7.9a. The selected domain is used later on to give hints for some possible Modelica connector models (figure 7.9b).

In the FM some functions are omitted because they do not influence the acceleration / deceleration behavior of the car and because they would clutter the model. Examples of these functions are all the *holds*-functions like *Chassis holds Battery*, *Chassis holds motor*, etc. Also, *Battery powers Receiver* is not included.

The implemented way of entering Modelica code (only text-based modeling separated over components and functions) works well enough for small, simple models, but it turned out that it does not work so well for larger, more complex models. The splitting of Modelica code over functions was meant to make the entering of models more structured, but actually it makes it confusing to enter models for components performing functions that are interdependent. Furthermore, it is hard to keep an overview of the total model and its connections due the lack of graphical editing results.

Therefore the basic Modelica model structure, automatically generated by the FM software (based on the FM structure), was filled in the OpenModelica Editor (figure 7.8b) for convenience reasons and entered back in the FM (figure 7.10).

8.4 Simulation

After completion of the modeling process, the model was simulated. The simulation options were set using figure 7.11 to a time-frame of six seconds. 500 timesteps were used to ensure a detailed simulation. After clicking the *Simulate*-button, the time slider becomes available. When sliding, tooltips above each component showed the calculated values of each of the components' variables. This can be seen in figure 7.12.

Here a problem showed up: The amount of internal variables in the more complex Modelica models causes the tooltips to fill the screen. This happens when models exist of multiple components, of which some of them have subcomponents. Modelica exports all internal variables, not only the top-level ones.

To show how it looks when only useful variables are shown, the results file was manually filtered before allowing ModelEvolution to read the results. This is shown in figure 7.12b. A possible way of solving this could be showing only the variables of the connectors,

thereby hiding all internal variables. However, the consequences and results of this possible solutions are not investigated.

While moving the time slider, the visibility of the components is set depending on their *visibility* variable. This variable has to be defined manually depending on what the engineer thinks is necessary for the component to perform its function(s). In figure 7.12 can be seen that the *Battery*, *MCS* and *Motor* do not perform their function anymore because the *Receiver* already set the *control* variable of the *MCS* to 0.0. To visualise that, the visibility of the *Battery*, *MCS* and *Motor* was made dependent on the electrical current in their models. However, the *Transmission* still performs a function, because there is a friction component inside the model which exerts a decelerating torque upon the *Wheels*.

Besides the dynamic FM showing the simulation results, the results can also be plotted in graphs (accessible through the context menu of each component, see figure 7.10a). In figure 7.13 the graph for the *Chassis* component is shown. When hovering over the results, the value at the location of the mouse cursor is shown using a tooltip. The reason for the distance being negative is that the *Tire1* model is connected to the right-hand connector of the *Chassis1* model. This was done for layout reasons.

8.5 Case study conclusions

The modeling and simulation of an RC-car, a larger and more complex system than the small test models used during development, showed that it is possible to use the developed workflow for combining a FM and Modelica model for a real-life system. However, during the case study some shortcomings appeared that need to be taken care of before the software is suitable for general use. These shortcomings mainly consist of the way how Modelica models have to be edited and how the simulation results are displayed. It would be easier if the components' Modelica code would not be separated over functions and if simulation results could be filtered before displaying them.

The reason that the overflow of simulation results during visualization did not show up during development, was that simple models with a small amount of variables were used for quick purposeful testing. Every test was created with a special aim in mind - for example one test for the storage of code in a component, one test for the dynamic hiding and showing of components, etc.

As described before, two points were tailored for this case study. The first one is that, instead of filling the Modelica model details using the text editing dialog of the FM software, the OpenModelica Editor was used to edit the generated model structure in a graphical way. However, the basic Modelica model structure is generated automatically from the FM by ModelEvolution. The second tailored point is the manual filtering of simulation results, to show the difference between the filtered and non-filtered situation.

It is convenient to have access to the Modelica code linked to FM components. However, the current implementation of the code editing itself lacks a lot of conveniences that more mature editors offer, like drag-and-drop support and visual connecting of components. This visual way of editing is not implemented because it would take a lot of time

to reimplement the already existing features from other software. Instead there is chosen to focus on other integration points between FM and Modelica, like Dynamic FM and the visualization of the simulation results.

It would require usability research to investigate whether the proposing of connectors is a valuable addition. It is something that sets it apart from other editors, but more experienced people might not need it.

9 Conclusions and recommendations

9.1 Conclusions

This research has three outcomes:

- a workflow for integration possibilities of TRIZ, CAD software, FM software and Modelica (Optimization). This workflow has been commented on and is backed by a thorough literature study
- a new way of working with FMs of changing system structures (Dynamic FM)
- a software prototype demonstrating the interaction between FM and Modelica

The easy access to the Modelica code and simulation graphs through the context menu of components in the FM are nice to have, but the input method for the code itself needs some work. Besides lack of support for graphical editing of the Modelica models, it is hard to maintain a clear overview during connecting the models and when working with Modelica models that have more layers than the FM. However, the structure of the internal Modelica model that is updated upon changes in the FM is practical. Graphical editing of Modelica models is not implemented because it would take too much time to reimplement functionality that other programs already offer.

Dynamic FM and the visualization of results as tooltips linked to the FM components are nice to have. They give the user more insight into the transient behavior of his model. Besides that, they set the software apart since there is no other software at the moment that does this. However, the filtering of results still needs work to prevent cluttering of the interface (regarding the tooltips and the plot graphs). Having too many visible results at the same time literally fills the screen.

To achieve integration of FM and Modelica, it would be a simpler solution to generate the Modelica model from the FM structure. This model could then be opened, edited and simulated in a dedicated Modelica editor. Finally the results could then be imported in the FM software. This way, Dynamic FM and the visualization of simulation results are still supported, but editing of Modelica code through the FM components is lost then. Connector proposing based on the functions' domains is lost then, too (unless it would be supplied as a comment in the exported Modelica model).

Other reasons for not choosing this easier approach, are that there was strived for as much independence of external programs as possible and that it was interesting to experiment with new ways of editing Modelica code (like splitting code over functions).

9.2 Recommendations

Although a lot of ideas were worked out, not all ideas for new features and not all experiment results ended up in the software prototype. This is because they were not either not within the scope of this project or were not achievable given the situation. However, they could be implemented in other projects to improve the software even more. This section lists the ideas for future improvements.

The workflow shown in figure 5.1 shows some dashed arrows – interactions that have not been implemented yet. To complete the interactions between the components listed in this workflow diagram, research should be done into creation of CAD models from FMs, contradiction detection during Modelica optimization and usage of the FM target component to assist in formulating an optimization objective.

During this project an experiment was done regarding the presentation of design guidelines and their visualization in SolidWorks. However, more research is needed to automate the derivation of design guidelines from the optimization results. Also, the visualization of design guidelines like done in the experiment might not be the only or best way to do this. More research is needed here.

The automatic connection of models, that belong to components sharing a function, needs a good insight of the user into the connections. Therefore it is recommended to implement this functionality after at least implementing graphical displaying (or even better, editing) of the Modelica models.

Filtering of the simulation results is a point that needs to be implemented to make the software usable when working with larger models. A simple way of doing this would be letting the user select which variables he wants to display. This is already done in other software. However, it is recommended to investigate whether the software can assist in filtering these results.

At the moment, the interaction matrix is only used while generating the initial function model. It was implemented like this, because it was assumed that the interaction matrix is only used at this working stage. However, it could be useful to be able to switch back and forth between the function model and the interaction matrix. For example going back to the interaction matrix would allow for having a quick overview of the interactions between components, especially when working with a multi-layer function model.

Another improvement would be the implementation of a client / server model. This way multiple users could work on the same function model at the same time, not needing to send their models back and forth, not needing to wait for each other's updates and always having access to the latest version. In the past, several projects like [42] and [34] aimed to deliver a client-server model to allow multiple users to work simultaneous with one OpenModelica Compiler instance. A similar goal could be set for the FM part of this software.

Dynamic FM was introduced to better combine the various states of a system in one FM. Modelica provides support for modeling of physical dynamic systems, including support for thermodynamic models and models containing fluids. The software could be improved to detect whether there would be phase changes in the fluids in the Modelica models, and show these phase changes in the FM. If this turns out to be impossible due

to lack of Modelica support for phase change detection, the software could approach the problem in a different way by detecting the usage of fluid models in Modelica and hinting the user to model the phase changes manually.

One of the shortcomings of FM is, that users might be inclined to use ambiguous or invalid function descriptions since they are formulated using natural language [9]. An example of an *invalid function* is *to protect*, since it does not change a parameter of the object it works on. The software could help the users by notifying them when they try to define such a function and by giving them information on how functions can be formulated better.

In section 6.4 an idea is given of having the software automatically add the items, listed as requirements for a certain phenomenon, to the FM upon choosing of that phenomenon. This could help a user to faster update his FM after choosing for a certain implementation.

With regard to Modelica, two improvements could be named (as discussed in section 5.6). The first one is using the arrow thickness or color in FMs to visualize the value of a Modelica variable from the simulation results. It is already implemented in SimulationX, but would be a valuable addition to have it in FM as well. The second one is generating FMs from Modelica models by using information from their model hierarchy and interactions between submodels. Information about the technical domain of interactions could be derived from their interaction connector types. This automated generation could help engineers, who already have a Modelica model, with quickly creating a FM from it. However, it also needs to be checked whether there is a demand for this workflow.

During this project, a case study is done to demonstrate the implemented functionality and to verify the software's capabilities of working with larger and more detailed models. However, the interaction with the software by a large group of users needs to be done researched to verify the usability of the developed workflow steps and their implementation. Another research goal can be the difference in task completion time when comparing the non-integrated and integrated software.

Bibliography

[1] *Altshuller*. 2006. URL: http://www.triz.org/triz/altshuller.shtml (visited on 04/14/2011).

[2] Genrich Saulovich Altshuller and Rafael Borisovich Shapiro. "About the Psychology of Technical Creativity (О психологии изобретательского творчества)". Russian. In: *Questions of psychology (Вопросы психологии)* 6 (1956), pp. 37–49. URL: http://www.altshuller.ru/triz/triz0.asp (visited on 07/14/2012).

[3] Genrich Saulovich Altshuller, Lev Shulyak, and Steven Rodman. *40 Principles: TRIZ Keys to Technical Innovation (Triztools, V. 1)*. English. 1st ed. Technical Innovation Center, INC., 1998. ISBN: 9780964074033.

[4] Peter Aronsson and David Broman. "Extendable Physical Unit Checking with Understandable Error Reporting". In: *Proceedings of the 7th Modelica Conference*. Como, Oct. 2009, pp. 890–897. DOI: 10.3384/ecp09430027.

[5] Peter Aronsson et al. *Modelica® - A Unified Object-Oriented Language for Physical Systems Modeling // Language Specification*. Tech. rep. Linköping, Sweden, Feb. 29, 2012. URL: https://www.modelica.org/documents/ModelicaSpec32Revision1.pdf (visited on 07/14/2012).

[6] Hans M. Bakker, Leonid S. Chechurin, and Wessel W. Wits. "Integrating TRIZ function modeling in CAD software". In: *Proceedings of TRIZfest-2011*. Ed. by Leonid S. Chechurin. The International TRIZ Association (MATRIZ). Saint Petersburg: Издательство Политехнического университета, 2011, p. 18. URL: www.matriz.org.

[7] Edward A. Bender. *An Introduction to Mathematical Modeling*. Dover Books on Computer Science Series. Dover Publications, 2000. ISBN: 9780486411804.

[8] Jean-Bernard Bluntzer, Samuel Gomes, and Jean-Claude Sagot. "From Functional Analysis to Specific Parameters: Description of Knowledge Based CAD Model". In: *2008 IEEE International Conference on Signal Image Technology and Internet Based Systems*. IEEE, Nov. 2008, pp. 665–671. ISBN: 978-0-7695-3493-0. DOI: 10.1109/SITIS.2008.41.

[9] Bohuslav Busov, Darrell Mann, and Pavel Jirman. "TRIZ and Invention Machine®: Methods and Systems for Creative Engineering and Education in the 21st century". Boston, Sept. 1999.

[10] Alessandro Cardillo et al. "A Novel Paradigm for Computer-Aided Design: TRIZ-Based Hybridization of Topologically Optimized Density Distributions". In: *Growth and Development of Computer-Aided Innovation*. Ed. by Runhua Tan, Guozhong Cao, and Noel León-Rovira. Vol. 304. IFIP Advances in Information and Communication Technology. Springer Boston, 2009, pp. 38–50. ISBN: 978-36-420-3345-2. DOI: 10.1007/978-3-642-03346-9_5.

[11] Gaetano Cascini. "State-of-the-Art and Trends of Computer-Aided Innovation Tools". In: *Building the Information Society*. Ed. by René Jacquart. Vol. 156. IFIP International Federation for Information Processing. Springer Boston, 2004, pp. 461–470. ISBN: 978-1-4020-8156-9. DOI: 10.1007/978-1-4020-8157-6_40.

[12] Leonid S. Chechurin, Hans M. Bakker, and Wessel W. Wits. "Dynamic Function Modeling Concept And Its Software Realization". In: *Proceedings of TRIZfest-2012*. The International TRIZ Association (MATRIZ). Lappeenranta, 2012. In press.

[13] Leonid S. Chechurin et al. "Introducing trimming and function ranking to SolidWorks based on function analysis". In: *Proceedings of Triz Future Conference 2011*. Ed. by Gaetano Cascini and Tom H J Vaneker. ETRIA. Dublin: Institute of Technology Tallaght, 2011, pp. 215–225. ISBN: 978-0-9551218-2-1.

[14] Ives De Saeger. "Function Modeling Issues". Berchem. URL: http://www.p41.be/wp-content/uploads/Microsoft-Word-DeSaeger_functionmodelingv2.pdf (visited on 07/14/2012).

[15] *Documents Associated With CORBA, 3.2*. Tech. rep. Object Management Group, Inc., Nov. 2011. URL: http://www.omg.org/spec/CORBA/3.2/ (visited on 06/25/2012).

[16] Johannes Edrén et al. "Modelica and Dymola for education in vehicle dynamics at KTH". In: *Proceedings of the 7th Modelica Conference*. Como, 2009, pp. 775–783. DOI: 10.3384/ecp09430112.

[17] Markus Egerland, Dirk Roos, and Johannes Will. "Optimization of a fan shroud by ANSYS / DesignModeler and optiSLang". In: *Ansys Conference & 25th CADFEM Users' Meeting*. Dresden, 2007.

[18] V.M. Gerasimov and S.S. Litvin. *Construction of Functionally Ideal Models when Conducting VEA*. Russian. Chelyabinsk: Фонд материалов по ТРИЗ ЧОУНБ, 1989.

[19] V.M. Gerasimov et al. *Basic statements of methodology for performing function-cost analysis: Methodological recommendations*. Russian. Ed. by Kh.R. Parksheyan and V.V. Sysun. Moscow: Информ-ФСА, 1991. URL: http://www.triz-summit.ru/ru/section.php?docId=3952 (visited on 07/14/2012).

[20] *Goldfire*. Computer software. Invention Machine. URL: http://inventionmachine.com/products-and-services/innovation-software/goldfire-Innovation-Tools/ (visited on 07/14/2012).

[21] Samuel Gomes et al. "Functional design and optimisation of parametric CAD models in a knowledge-based PLM environment". In: *International Journal of Product Development* 9.1/2/3 (2009), pp. 60–77. ISSN: 1477-9056. DOI: 10.1504/IJPD.2009.026174.

[22] John R. Hauser and Don Clausing. "The House of Quality". In: *Harvard Business Review* 66.3 (1988), pp. 63–73. ISSN: 0017-8012.

[23] Stefan Hüsig and Stefan Kohn. "Computer aided innovation—State of the art from a new product development perspective". In: *Computers in Industry* 60.8 (Oct. 2009), pp. 551–562. ISSN: 0166-3615. DOI: 10.1016/j.compind.2009.05.011.

[24] Sergei Ikovenko. "Super-Effect Analysis". Presentation. Saint Petersburg, July 21, 2011.

[25] Sergei Ikovenko. "TRIZ and Computer Aided Inventing". In: *Building the Information Society*. Ed. by Renè Jacquart. Vol. 156. IFIP International Federation for Information Processing. Springer Boston, 2004, pp. 475–485. ISBN: 978-1-4020-8156-9. DOI: 10.1007/978-1-4020-8157-6_42.

[26] Tamás Juhász. "Advanced Solutions in Object-Oriented Mechatronic Simulation". Ph.D. Thesis. Budapest University of Technology and Economics, 2008. URL: http://www.omikk.bme.hu/collections/phd/Villamosmernoki_es_Informatikai_Kar/2009/Juhasz_Tamas/ertekezes.pdf (visited on 07/14/2012).

[27] Martin Kratzer et al. "An agent-based system for supporting design engineers in the embodiment design phase". In: *ICED 11 - 18th International Conference on Engineering Design - Impacting Society Through Engineering Design*. Ed. by S.J. Culley et al. Vol. 10. August. Copenhagen, 2011, pp. 178–189.

[28] Håkan Larsson. "Translation of 3D CAD Models to Modelica". M.Sc. Thesis. Linköping University, 1999. URL: http://www.ida.liu.se/ext/pur/reports/2000-pelab.html.

[29] Noel León-Rovira. "A proposal to integrate TRIZ and CAD (Computer Aided TRIZ-based Design)". In: *Proceedings of TRIZCON 2001*. 2001. URL: http://www.triz-journal.com/archives/2001/07/g/index.htm (visited on 07/14/2012).

[30] Jesse Liberty and Brian MacDonald. *Learning C# 3.0*. O'Reilly Media, Inc., Nov. 2008. ISBN: 9780596521066.

[31] Simon S. Litvin. "Main Parameters of Value: TRIZ-based Tool Connecting Business Challenges to Technical Problems in Product/Process Innovation". In: *7th Japan TRIZ Symposium 2011*. Yokohama: GEN3 Partners, Inc., 2011. URL: http://www.triz-japan.org/sympo2011/EI01eS-Litvin_(Keynote)-110817.pdf (visited on 07/14/2012).

[32] Simon S. Litvin, Naum Feygenson, and Oleg Feygenson. "Advanced function approach". In: *Procedia Engineering* 9 (Jan. 2011), pp. 92–102. ISSN: 1877-7058. DOI: 10.1016/j.proeng.2011.03.103.

[33] Sebastien Lorion. *A Fast CSV Reader*. Nov. 10, 2011. URL: http://www.codeproject.com/Articles/9258/A-Fast-CSV-Reader (visited on 04/18/2012).

[34] Tari Mohsen-Torabzadeh et al. "OMWeb – Virtual Web-based Remote Laboratory for Modelica in Engineering Courses". In: *Proceedings of the 8th International Modelica Conference*. Ed. by Christoph Clauß. Linköping: Linköping University Electronic Press, June 2011, pp. 153–159. DOI: 10.3384/ecp11063153.

[35] Adrian Pop. "Contributions to Meta-Modeling Tools and Methods". PhD thesis. Linköping University, 2005. ISBN: 9185299413. URL: http://www.ida.liu.se/~adrpo/lic/adrpo-LicentiateThesis-Bookmarks.pdf (visited on 07/14/2012).

[36] Stefan Reh et al. "Probabilistic finite element analysis using ANSYS". In: *Structural Safety* 28.1-2 (Jan. 2006), pp. 17–43. ISSN: 0167-4730. DOI: 10.1016/j.strusafe.2005.03.010.

[37] *RS-540SH specifications*. Mabuchi Motor Co., Ltd. URL: http://www.mabuchi-motor.co.jp/cgi-bin/catalog/e_catalog.cgi?CAT_ID=rs_540sh (visited on 06/11/2012).

[38] Daniel Rutter. *Review: Tamiya Ferrari 360 Modena Challenge*. Dec. 2011. URL: http://www.dansdata.com/ta04.htm (visited on 06/11/2012).

[39] Yuri P. Salamatov. *TRIZ: the Right Solution at the Right Time: A Guide to Innovative Problem Solving*. English. Trans. by Oleg Kraev. 2nd ed. Krasnoyarsk: Institute of Innovative Design, Jan. 2005. ISBN: 90-804680-1-0.

[40] *Simulink®*. Computer software. The MathWorks, Inc. URL: www.mathworks.com/products/simulink (visited on 02/16/2012).

[41] Valeri V. Souchkov. *TRIZ & SYSTEMATIC INNOVATION - Guides to TRIZ and xTRIZ - Techniques and References*. English. Enschede: ICG Training & Consulting, June 2011. URL: www.xtriz.com (visited on 07/14/2012).

[42] Martin Suchan et al. *Might Modelica Editor*. Oct. 2009. URL: http://trac.assembla.com/ModelicaEditor/ (visited on 05/25/2012).

[43] *Tamiya America Item #58255 | RC Calsonic Skyline GT-R (R34) - TL01*. Tamiya America, Inc. URL: http://www.tamiyausa.com/product/item.php?product-id=58255 (visited on 06/11/2012).

[44] John Terninko. "The QFD, TRIZ and Taguchi Connection: Customer-Driven Robust Innovation". In: *Transactions from the Ninth Symposium on Quality Function Deployment*. Novi, MI: QFD Institute, 1997, pp. 441–445. ISBN: 9781889477091.

[45] Hubert Thieriot et al. "Towards Design Optimization with OpenModelica Emphasizing Parameter Optimization with Genetic Algorithms". In: *Proceedings of the 8th International Modelica Conference, 2011*. June 2011, pp. 756–762. DOI: 10.3384/ecp11063756.

[46] *TRIZacquisition*. Computer software. TRIZ Consortium, 2010. URL: http://lgeco.insa-strasbourg.fr/trizacquisition/presentation/ (visited on 07/14/2012).

[47] Dominic Ullmann and Others. *IIOP.NET*. Computer software. Oct. 2008. URL: `http://iiop-net.sourceforge.net/` (visited on 07/14/2012).

[48] Wessel W. Wits, Hans M. Bakker, and Leonid S. Chechurin. "Towards multidisciplinary support tools for innovation tasks". In: *Procedia CIRP* 2 (Proceedings of the 1st CIRP Global Web Conference: Interdisciplinary Research in Production Engineering (CIRPE2012) 2012). Ed. by Wessel W. Wits, Juan M. Jauregui-Becker, and Hans-Christian Mohring. ISSN: 2212-8271. In press.

The first TRIZ publications were made in Russian, because TRIZ was originally developed in Russia. Some Russian publications are added to the bibliography for historical reasons or because the English publications on a certain topic were scarce.

A Function model creation

This chapter shows the creation steps that lead to a FM. This can done in an automated way, like ModelEvolution provides, or by hand. In this example the valve-cam assembly (shown in figure A.1) from the SolidWorks 2011 Motion Studies tutorial is used.

First an interaction matrix (see figure A.2) is composed. This is a symmetric matrix where all components of the system are listed along both axes. A check is placed for each interaction. ModelEvolution can extract the interactions from the CAD model and thereby automatically fill the interaction matrix. The user can make changes afterwards.

For each interaction in the matrix, a function needs to be defined. This can be a list of functions on paper, but ModelEvolution helps the user by proposing probable functions (see figure A.3) that the user can change to his liking. Additional information like whether or not the function is harmful can be added as extra information.

When all interactions are defined as functions, a function model diagram (figure A.4) can be drawn to visualize them. The components are drawn as blocks and the arrows connecting them represent the functions.

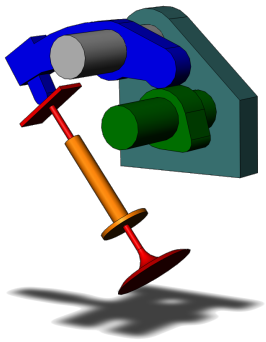


Figure A.1: Valve-cam assembly from the SolidWorks 2011 Motion Studies tutorial

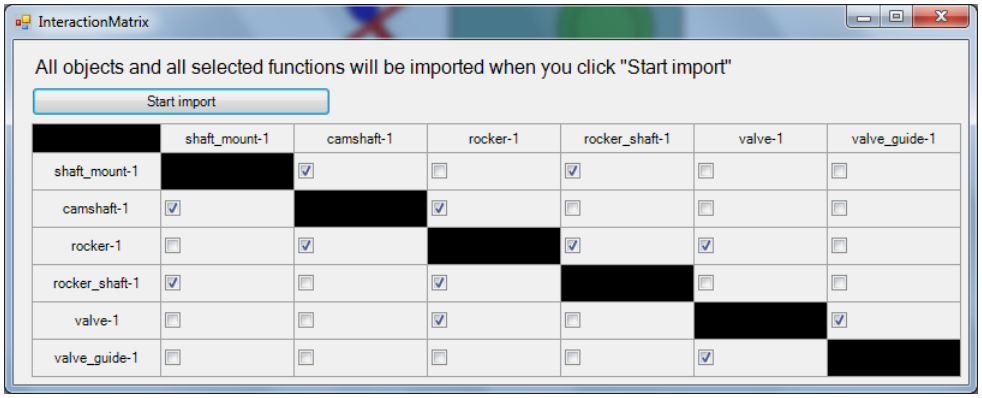


Figure A.2: Interaction matrix

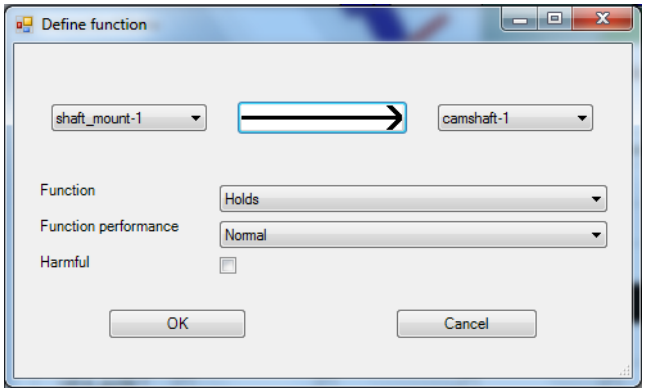


Figure A.3: Function definition

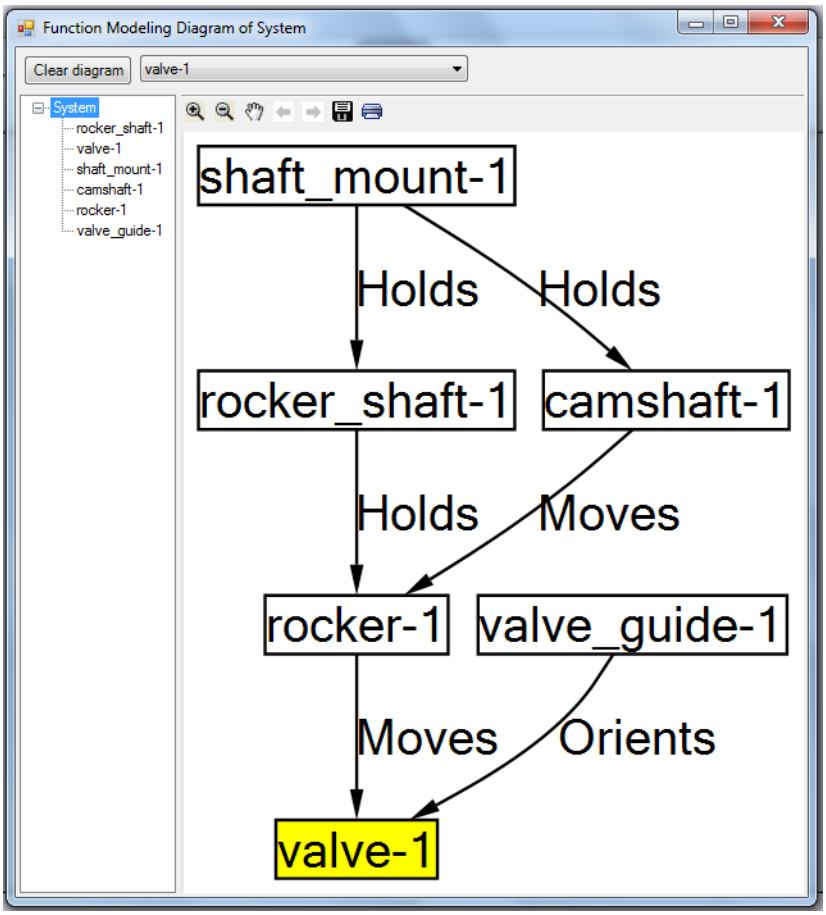


Figure A.4: Function model diagram

B Software architecture

B.1 Restructuring

Before the start of this research, the research only focused on integration with SolidWorks (see [6] and [13]). Therefore, ModelEvolution was only integrated with SolidWorks at the beginning of this research. In order to allow for well-structured integration with several CAD programs, including a *dummy cad plugin* for testing reasons, an abstraction layer was built between SolidWorks and the software. This was done by introducing an *interface class* (see [30, ch. 13]) called *ICadPlugin*. The principle of using an *interface class* was chosen because CAD plugins need multiple inheritance – they need to inherit of both *ICadPlugin* and the interface class provided by the CAD software they are meant for, in the case of SolidWorks this interface is called *ISwAddin* (see section B.4).

The abstraction process resulted in having separate software components: a generic FM module and a CAD plugin written specifically for SolidWorks. This SolidWorks add-in is discussed in section B.4.

The FM module calls functions of this *ICadPlugin* interface class and without knowing about the actual implementation in the CAD plugins. Plugins for every CAD suite can be developed by inheriting from the interface class and implementing its virtual functions. This is visualized in figure 6.1 by means of the green software components that can fit into the red FM module.

Before the abstraction, the FM software depended on SolidWorks to be started, because it was developed as a SolidWorks plugin. However, SolidWorks was not needed during this project. Therefore, SolidWorks became a burden which needed to be restarted every time a code change was made. The abstraction allowed to develop a *dummy cad plugin* – a standalone program of which the only purpose is to start the FM software. Using this approach drastically sped up the development and debugging.

B.2 Function Modeling module

After the restructuring discussed in section B.1, the work of adding support for Modelica to the Function Modeling module was started.

Figure B.1 shows the code structure of this module - it is the module that is responsible for all operations on the FM, that holds a database needed for the proposing functionality. For the sake of legibility, only the most important relations between classes are drawn.

The plugin code is divided over several parts. *AddinCore* contains the code neccessary to interface with CAD add-ins and other programs. Furthermore it contains code to start the module.

The FM logic is located in *FMCore*. This part contains all code for drawing the FM and interacting with it. The proposing features, the change tracking and function ranking code are also located here.

CustomControls is not directly related to FM functionality, but contains some custom user interface elements, like the special button type that is used for the *Simulate*-button. This button has a special clickable section to allow editing of the simulation settings (figure 7.11).

B.2.1 Modelica integration in the Function Modeling module

The most important newly added code of this module resides in the *Modelica* part. The *ModelicaHandler*-class contains all functions to perform actions using OMC. These functions exist of predefined commands that can be sent to OMC via the OMCCorba module. It has a *SimulationResults* object into which simulation results are loaded via the CSVReader module. The *SimulationResults* class has a *DataTable* object where the results are stored. This *DataTable* allows fast querying of the data, which is not directly possible with a CSV file.

The user can edit components' Modelica code through the *ModelicaClassEditor* dialog. This dialog can contain multiple *ModelicaFunctionCodeControl* sections – one code editing section per function. This dialog can be seen in figure 7.10b.

EditSimulationOptions is the dialog that allows the user to make settings for the simulation (shown in figure 7.11b). These settings are sent to the *ModelicaHandler* class using an object of the *ModelicaSimulationOptions* class.

The *ModelicaSimulationGraph* class contains the dialog to present the simulation results of a component in the FM as a graph (shown in figure 7.13). Besides the graph dialog, the simulation results are shown in the FM diagram using *tooltips* above the FM components they belong to. These tooltips are drawn by the *FunctionDiagram* class. It reads the data from the *DataTable* in the *SimulationOptions* object of *ModelicaHandler*. Everytime the time slider of the FM diagram window is moved, the tooltips are updated and shown for a few seconds.

B.3 Connection to OpenModelica

To perform all Modelica-related tasks, the Function Modeling module of ModelEvolution connects to the OpenModelica compiler (OMC). This connection is done using CORBA technology ([15]), which allows computer processes to communicate with each other using network connections (this means that it is possible to have processes of separate computers communicating with each other, but this is not necessary).

The OMC can be started as a CORBA server, after which other programs can connect to it. The communication logic for connecting to OMC was packaged in a separate DLL file called *OMCCorba* for easy testing of its functionality. The UML-diagram of the OMCCorba module is shown in figure B.2. OMCCorba uses IIOP.NET ([47]), a CORBA implementation for the .NET programming platform. The OMCCorba DLL provides a

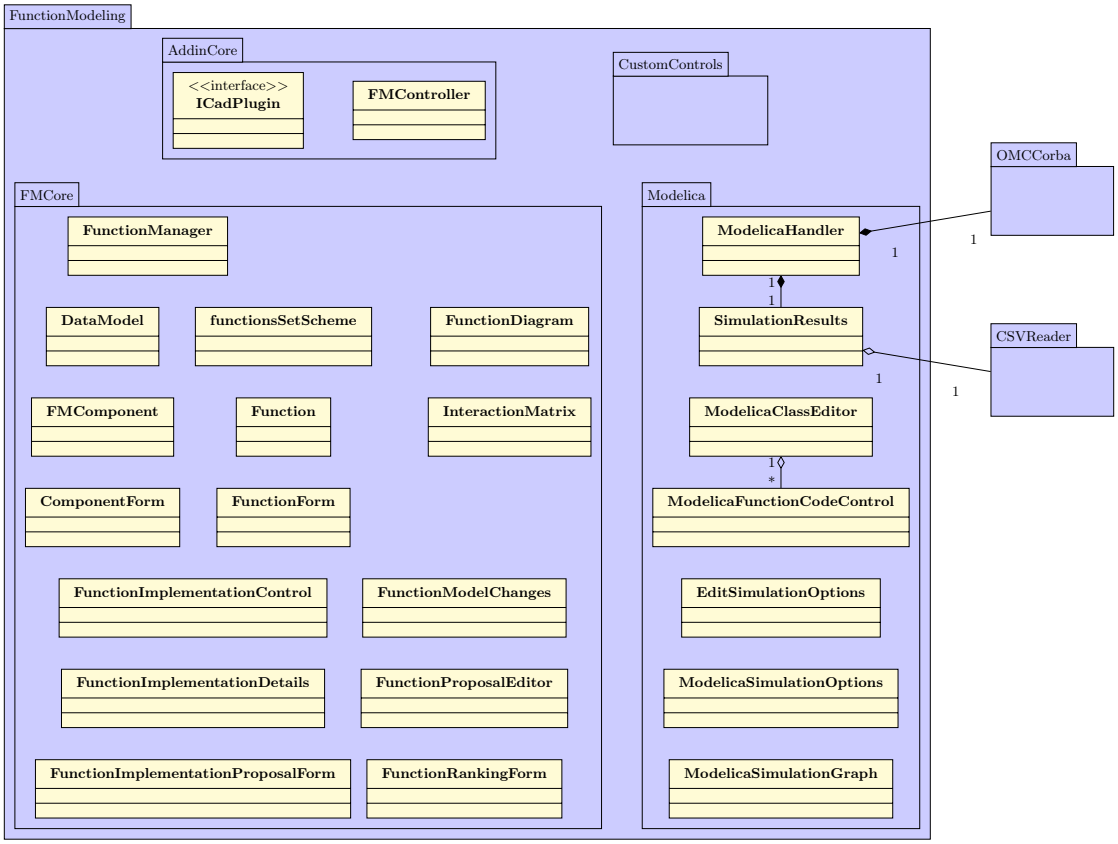


Figure B.1: UML-diagram of FunctionModeling module

function that other programs can use to send commands to OMC and read the response to these commands. This function is *SendExpression(string expression)*.

Besides the transmission of commands and data, the DLL takes care of starting OMC when it is not running yet and closing it when the main program is shut down.

The software project by [42] was taken as an example for the implementation of a CORBA connection with OMC.

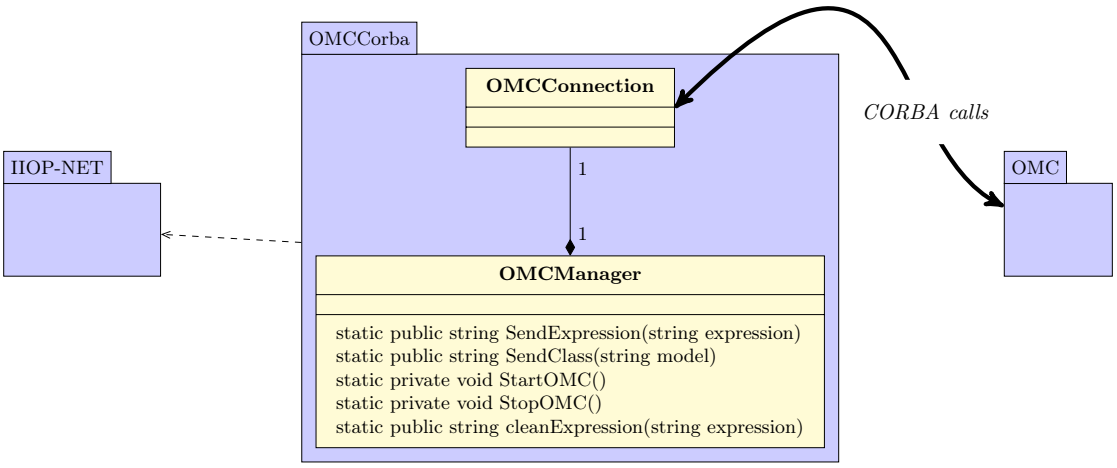


Figure B.2: UML-diagram of OMCCorba module

B.4 SolidWorks add-in

The SolidWorks add-in can start the FM module and provide it with the necessary information about the loaded assembly and interactions within this assembly. Its architecture is shown in figure B.3.

In order to work as a SolidWorks add-in, it needs to inherit from the *ISwAddin* interface which is provided by the SolidWorks API. For the interaction with the FM module it inherits from the ICadPlugin interface.

The SolidWorks add-in has a few classes that assist in providing the needed functionality. The logic for providing information about the assembly resides in the *AddinToolbox*. The code for responding to events in SolidWorks (like adding and removing of a part, adding a new mate, etc) is located in *CADEventHandler*. *bitmaphandler* provides functionality for converting bitmap icons to a format that SolidWorks can use to draw icons on buttons.

Since the interaction with SolidWorks is out of the scope of this research, there will not be elaborated on more details of this component.

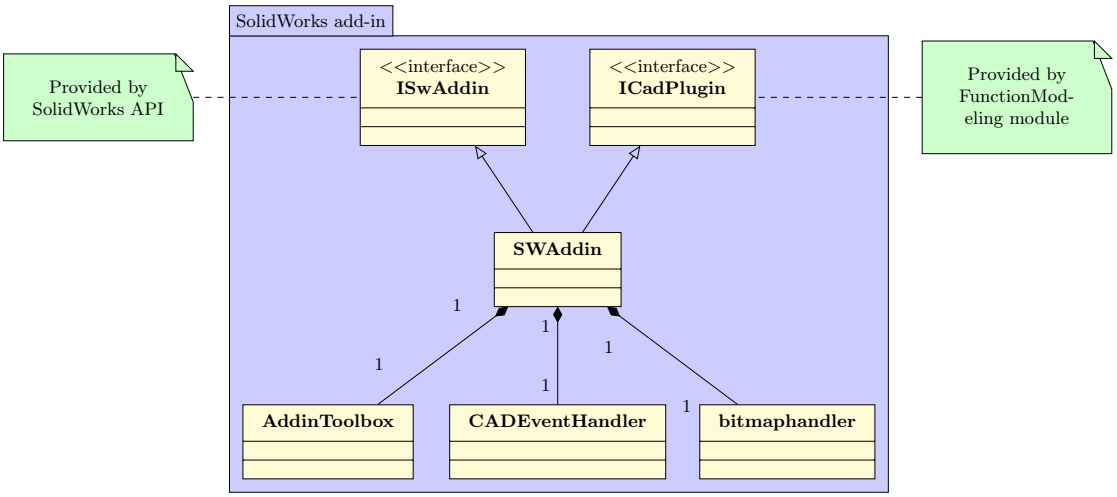


Figure B.3: UML-diagram of SolidWorks add-in