

ISSN 0280-5316
ISRN LUTFD2/TFRT--5893--SE

A Graphical User Interface for Polyethylene Production Grade Changes

Max Stenmark

Lund University
Department of Automatic Control
December 2011

Lund University Department of Automatic Control Box 118 SE-221 00 Lund Sweden		<i>Document name</i> MASTER THESIS	
		<i>Date of issue</i> December 2011	
		<i>Document Number</i> ISRN LUTFD2/TFRT--5893--SE	
<i>Author(s)</i> Max Stenmark		<i>Supervisor</i> Per-Ola Larsson, Dept. of Automatic Control, Lund University, Sweden Johan Åkesson, Dept. of Automatic Control, Lund University, Sweden Tore Hägglund, Dept. of Automatic Control, Lund University, Sweden (examiner)	
		<i>Sponsoring organization</i>	
<i>Title and subtitle</i> A Graphical User Interface for Polyethylene Production Grade Changes (Ett grafiskt användargränssnitt för kvalitetsomställningar i polyeterproduktion).			
<i>Abstract</i> <p>To compete globally, the European process industry needs to increase its cost effectiveness by minimizing waste and maximizing revenue. This can be achieved by adapting to the market demand and always produce the most profitable products. To minimize the cost, the production changes needs to be as effective as possible. By optimizing the product or grade change, the time spent producing inexpensive products outside of the specifications is limited. As part of a PIC-LU project, optimization methods for polyethylene production grade changes have been developed at Lund University. The optimization methods use a calibrated Modelica model of a polyethylene production plant and the optimization problems are solved with JModelica.org, an open source software platform for modeling, simulation and optimization. The Master's thesis was carried out within the PIC-LU project and in coordination with an industrial partner, Borealis. This Master's thesis investigates the possibility of combining calibration and optimization methods into a graphical user interface. The goal of the graphical user interface is to make the technology and research results more available. By creating a graphical user interface the engineers at Borealis can test and evaluate optimal strategies for grade changes. The objective is that the graphical user interface can be used as an aid in the planning and decision making. The result of the Master's thesis is a graphical user interface capable of solving grade change optimization problems, and allowing the user to evaluate the calculations by comparing the optimization results with measurement data.</p>			
<i>Keywords</i>			
<i>Classification system and/ or index terms (if any)</i>			
<i>Supplementary bibliographical information</i>			
<i>ISSN and key title</i> 0280-5316			<i>ISBN</i>
<i>Language</i> English	<i>Number of pages</i> 1-54	<i>Recipient's notes</i>	
<i>Security classification</i>			

Abstract

To compete globally, the European process industry needs to increase its cost effectiveness by minimizing waste and maximizing revenue. This can be achieved by adapting to the market demand and always produce the most profitable products. To minimize the cost, the production changes needs to be as effective as possible. By optimizing the product or grade change, the time spent producing inexpensive products outside of the specifications is limited.

As part of a PIC-LU project, optimization methods for polyethylene production grade changes have been developed at Lund University. The optimization methods use a calibrated Modelica model of a polyethylene production plant and the optimization problems are solved with JModelica.org, an open source software platform for modeling, simulation and optimization. The Master's thesis was carried out within the PIC-LU project and in coordination with an industrial partner, Borealis.

This Master's thesis investigates the possibility of combining calibration and optimization methods into a graphical user interface. The goal of the graphical user interface is to make the technology and research results more available. By creating a graphical user interface the engineers at Borealis can test and evaluate optimal strategies for grade changes. The objective is that the graphical user interface can be used as an aid in the planning and decision making.

The result of the Master's thesis is a graphical user interface capable of solving grade change optimization problems, and allowing the user to evaluate the calculations by comparing the optimization results with measurement data.

Acknowledgment

I am grateful for all the support I have received during my work on my Master's thesis. I want to thank my supervisors, Johan Åkesson and Per-Ola Larsson for their endless patience. I also want to thank Niklas Andersson for all the help I have received. I am also grateful for all the valuable feedback from people working for Borealis, especially Staffan Skålen.

Special thanks to friends and family members who remained supportive and pushed me to complete my Master's thesis despite my best efforts to avoid finishing it.

Contents

1	Introduction	1
1.1	PIC-LU and Borealis	1
1.2	Scope	2
1.2.1	Problem Formulation	3
1.2.2	Limitations	3
1.2.3	Interaction and Feedback	3
1.3	Structure of the Report	4
2	Background	5
2.1	Plant Model	5
2.1.1	JModelica.org	6
2.2	Calibration	6
2.3	Optimization	7
2.4	Python	9
2.4.1	Graphical User Interfaces in Python	9
2.4.2	wxPython	11
2.4.3	Matplotlib and NumPy	12
3	Work Flow and Specifications	13
3.1	Work Flow	13
3.2	Specifications	13
4	Graphical User Interface	17
4.1	Main Window	17
4.1.1	Grade Selection	18
4.1.2	Calibration	18
4.1.3	Optimization Settings	19
4.1.4	Optimization	20
4.1.5	Plotting	21
4.2	Features and Settings	24
4.2.1	Edit Grades	24
4.2.2	Menus	24
4.2.3	Settings	25

5	Implementation	29
5.1	Model Modifications	29
	5.1.1 Constraints	29
	5.1.2 Initial Trajectory	30
	5.1.3 Changing Model Values	31
5.2	Compiling	32
5.3	Threads and Processes	33
	5.3.1 Threads	33
	5.3.2 Processes	34
5.4	IP21 Communication	36
5.5	Saved Files	36
5.6	Design of the Graphical User Interface	37
	5.6.1 Restricting User Input	37
	5.6.2 Increasing flexibility	37
5.7	Program Structure	38
6	Epilogue	40
6.1	Discussion	40
	6.1.1 Code Structure	41
6.2	Conclusions	41
6.3	Future Work	42
	6.3.1 Dynamic Calibration	42
	6.3.2 Expanding the Model	42
	6.3.3 Output to Operators	42
	6.3.4 Economic Cost Function	42

Chapter 1

Introduction

During the past decade European companies have moved more and more of the production to countries with lower cost. The increased competition from low wage countries has forced the European industry to specialize and to produce more advanced products. Other countries are rapidly catching up and moving from bulk items to more advanced products. In a few years even specialized products can be made abroad.

To compete with cheap labor and lower raw material cost, Sweden and other European countries have to keep innovating. One attempt to increase the competitiveness of Swedish industry is a research program called **Process Industry Centre** at **Lund University** or **PIC-LU**. The collaboration is funded by the Swedish Foundation for Strategic Research with the objective to increase the competence level both in academia and industry.

One PIC-LU project studies optimal grades changes in the chemical process industry. In order to ensure that the most profitable product is produced the manufacturing process is flexible and it is possible to produce different products or grades in the same plant. This makes it possible to produce specialized products and to adjust the production after the customer demand.

During the transitions between different products material is produced that does not fulfill the specifications. This material is sold at a lower price or thrown away. By optimizing the grade change the cost can be minimized and the revenue increased. The goal of the thesis is to develop a graphical user interface for optimizing polyethylene production grade changes.

1.1 PIC-LU and Borealis

The goal of PIC-LU is to create an interdisciplinary center focusing on research as well as education together with the Swedish process industry. In Lund the departments of Chemical Engineering and Automatic Control participate. In addition, several companies take part as well. One of the companies is Borealis.

Borealis is an international company headquartered in Vienna (Borealis AG,

2011). An investment fund from Abu Dhabi holds a controlling interest in the company. Borealis has a plant in Stenungsund producing the widely used plastic polyethylene.

Borealis is currently expanding both in Sweden but, maybe more importantly, in Abu Dhabi. The raw material cost in Abu Dhabi is negligible because unused byproducts from the oil industry can be used to make plastic. In combination with low labor cost, investments in Abu Dhabi are very profitable. Despite the competition from abroad, the Swedish plants have continued to remain profitable. This has been accomplished by producing more advanced products of higher quality.

The increased flexibility and higher quality has been achieved by having years of experience and knowledgeable employees. This is also the reason why an investment company from Abu Dhabi owns a Swedish plant. Abu Dhabi has cheap raw material but lacks the know-how. The continued expansion of the Swedish plant is explained by the proximity to an experienced workforce. After a Swedish factory has been built and is operational key personnel is moved to Abu Dhabi to help launch identical factories.

To keep the edge, and to ensure that Sweden remains competitive in the future, several research projects exist. One of them is together with Lund University within the PIC-LU framework. The focus of this project is to study optimal grade changes.

The goal of the collaboration between PIC-LU and Borealis is to develop a tool to aid Borealis' process engineers in the planning and decision-making. By having a tool, new, unknown transitions can be explored and it becomes easier to test new products and concepts.

Furthermore, the tool should also have the ability to put a monetary cost on a transition. Today only back of the envelope calculations exist over the monetary efficiency of the transition. A tool makes it possible to investigate how changed raw material cost or product prices effect the optimal grade change. By having a better understanding of the price of a transition a more informed decision can be made and the planning becomes more straightforward. Even the environmental impact can be studied. By putting a cost on emissions and minimizing waste the manufacturing process can be made more environmentally friendly.

1.2 Scope

As a part of the PIC-LU project researchers in Lund are developing methods for model calibration and grade change optimization in collaboration with Borealis. The collaboration between industry and academia has many advantages. The project can work as a forum for knowledge transfers while academia get access to challenging real world research problems. In order to make the results of the research more accessible, the result can be packaged in a graphical user interface. The purpose of this thesis is to develop a graphical user interface that combines the calibration and optimization methods into a tool that can be used in the production planning process by Borealis.

1.2.1 Problem Formulation

The aim of the thesis is to develop a graphical user interface for grade change optimization. The goal is to design an interface that works as a bridge between the research results produced in the project and the engineers at Borealis. The core functionality of the graphical user interface is:

- Define and edit grade data.
- Provide a graphical interface for model calibration of a polyethylene plant.
- Incorporate functionality to optimize grade changes in a polyethylene plant.
- View, plot, compare and save the results from the optimization.
- Import, save and plot measurement data from the IP21 database.
- Provide functionality to validate the results from the optimization by comparing it to real measurement data.
- Provide a graphical user interface for changing optimization settings, constraints, raw material prices and product prices.

1.2.2 Limitations

The goal of this thesis is not to develop a commercial product. The graphical user interface is only one part of a larger project, a project that is still ongoing. Therefore a lot of work still remains. The graphical user interface developed as a part of the thesis lacks several key components. A simplified optimization algorithm is used instead of an economic optimization method that relies on product prices, grade specifications and raw material cost to calculate the optimal grade change. Additionally only a basic calibration method is used.

The goal of the thesis is not to develop a flawless application; the goal is to show the potential of the methods developed in the project.

1.2.3 Interaction and Feedback

The graphical user interface has been developed in close collaboration with engineers at Borealis. The objective was to create a useful graphical interface and therefore the input from Borealis has been influential and the specifications for the user interface have been produced together with engineers at Borealis. The objective is that the end product should be determined by what Borealis requires. To ensure that Borealis needs are met, engineers at Borealis have been continuously testing the interface and feedback was used throughout the project to update requirements and amend the features list.

1.3 Structure of the Report

In Chapter 2 the tools, methods and software used in this thesis will be described. The thesis will continue with a brief description of the design process and the requirements of the graphical user interface in Chapter 3. Chapter 4 describes the graphical user interface and Chapter 5 the implementation. The thesis will end with Chapter 6 containing a discussion and conclusions. In addition Chapter 6 contains a brief description of future work.

Chapter 2

Background

In this chapter the methods and tools used to formulate and solve the grade change optimization and model calibration problems are discussed. The reasons why these tools are used are motivated. In addition there is a thorough description of the available technologies for making graphical user interfaces. Out of several toolboxes a few were chosen. The intention and motivation behind the selections are also reflected upon.

2.1 Plant Model

The Borstar[®] process consists of three reactors, one pre-polymerization reactor, a loop reactor and a gas phase reactor as shown in Figure 2.1. Polymers with different molecular weight distributions can be achieved by changing the operating conditions of the reactors. The weight distribution of the molecules determines the material properties of the produced plastic. The flexibility of the Borstar[®] process ensures that a wide range of different products or grades can be made.

The model of the plant is based on model currently used by Borealis in a model predictive controller. The model is written in Modelica, a high level modeling language developed to encode large complex physical systems. The model is not completely finished and an older version lacking the recycling system is therefore used in the graphical user interface. This means that optimization results are not comparable with real measurement data, although the procedure for computing them is realistic.

Modelica is a suitable language for modeling but it does not provide a good platform for integration with numerical algorithms used for solving optimization problems. In order to solve optimization problems the model has to be transformed in a multistep process. For a description of how Modelica models can be used for solving optimization problems, see (Åkesson et al., 2009).

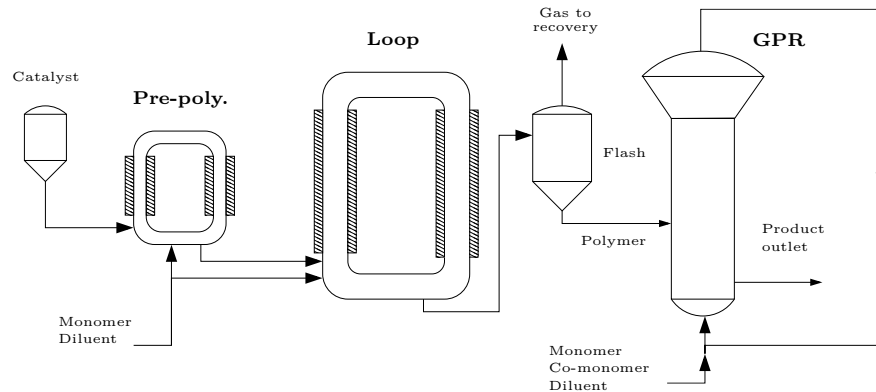


Figure 2.1: Pre polymerization reactor, loop reactor and gas phase reactor.

2.1.1 JModelica.org

One set of tools able to transform and connect a Modelica model with state of the art optimizations algorithms is gathered in a toolbox labeled JModelica.org (The Modelica Association, 2011). JModelica.org is an open source platform based on the Modelica language. To be able to define optimization problems an extension to the Modelica language called Optimica is used. The goal of the JModelica.org project is to create platform capable of being used both in industry and in academic research. JModelica.org provides a Python interface to run simulations and optimizations. The Python interface makes it easy to script the optimization process and to visualize the results. In this project, JModelica.org is used to solve both the model calibration and the grade change optimization.

2.2 Calibration

A model calibration is used to minimize discrepancies between model output and real data when known outflows are used. Originally the model has been calibrated with a combination of process know-how, empirical data and trial and error. By using measurement data to calibrate the model the model accuracy could increase significantly.

The used calibration method is a steady state calibration algorithm that tries to minimize the deviation to measurement data. For a more thorough description of the calibration algorithm see (Andersson et al., 2011).

If \mathbf{u} denotes the inputs, \mathbf{y} the outputs, \mathbf{x} the dynamic variables and \mathbf{w} the algebraic variables, the model in its differential algebraic equation (DAE) form can be formulated as

$$\begin{aligned}
\mathbf{0} &= F(\mathbf{z}, \mathbf{u}, \mathbf{p}) \\
\mathbf{y} &= g(\mathbf{z}, \mathbf{u}, \mathbf{p}) \\
\mathbf{z}^T &= [\dot{\mathbf{x}}^T \quad \mathbf{x}^T \quad \mathbf{w}^T],
\end{aligned} \tag{2.1}$$

where \mathbf{p} denotes the calibrated model parameters. The calibration can then be formulated as an optimization problem in the following way

$$\begin{aligned}
&\underset{\mathbf{p}, \mathbf{u}}{\text{minimize}} && (\hat{\mathbf{y}}_z - \mathbf{y}_z)^T \mathbf{W}_z (\hat{\mathbf{y}}_z - \mathbf{y}_z) + (\hat{\mathbf{y}}_u - \mathbf{y}_u)^T \mathbf{W}_u (\hat{\mathbf{y}}_u - \mathbf{y}_u) \\
&\text{subject to} && F_0(\mathbf{z}, \mathbf{u}, \mathbf{p}) = \mathbf{0}, \\
&&& \dot{\mathbf{x}} = \mathbf{0}, \\
&&& \mathbf{y}_z = g_z(\mathbf{z}), \\
&&& \mathbf{y}_u = g_u(\mathbf{u}),
\end{aligned} \tag{2.2}$$

where $\hat{\mathbf{y}}_z$ and $\hat{\mathbf{y}}_u$ are state and input measurements and \mathbf{W}_z and \mathbf{W}_u are diagonal weight matrices. The calibrated model parameter are later be used in the optimization.

2.3 Optimization

The optimization can be divided into three steps. In the first step, two stationary solutions are calculated, one for the start grade and one for the end grade. In the steady state the grade has to fulfill certain measurement benchmarks to ensure that the produced plastic is within specifications. Normally this means that the melt flow rates (MFR), the split factors as well as the density are within predefined limits. The split factors determine how much of the plastic is produced in the different reactors and the MFR describes the properties of the plastic. In the plant model used in this thesis the MFR models where not completed and the hydrogen-ethylene ratio where used instead of MFR.

If dynamic variables are denoted \mathbf{x} , inputs \mathbf{u} , outputs \mathbf{y} and the algebraic variables \mathbf{w} , the model can be written as

$$\begin{aligned}
\mathbf{0} &= F(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{w}, \mathbf{u}) \\
\mathbf{y} &= g(\mathbf{x}, \mathbf{w}, \mathbf{u}).
\end{aligned} \tag{2.3}$$

In the steady state solution the input \mathbf{u} is an algebraic variable and an extended algebraic vector $\mathbf{z} = [\mathbf{w} \quad \mathbf{u}]$ can be constructed, see (Larsson et al., 2011). The steady state problem can then be formulated as

$$\begin{aligned}
\mathbf{0} &= \tilde{F}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{z}) \\
\mathbf{0} &= \dot{\mathbf{x}} \\
\mathbf{0} &= \tilde{g}(\mathbf{x}, \mathbf{z}) - \mathbf{y}_{spec},
\end{aligned} \tag{2.4}$$

where \mathbf{y}_{spec} contains grade definitions and $\tilde{\mathbf{F}}$ and $\tilde{\mathbf{g}}$ corresponds to \mathbf{F} and \mathbf{g} without the input \mathbf{u} . The initialization problem can be solved by JModelica.org and the stationary solutions can be used in the next step.

The second step is to simulate a trajectory between the two steady states. The trajectory is used to initialize the optimization algorithm. A crude initial trajectory is created by creating a ramp of the inputs between the start and end values and simulating the plant model with the inputs as seen in Figure 2.2.

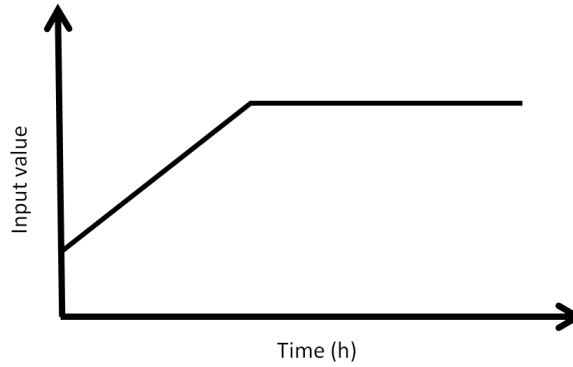


Figure 2.2: Ramp between the start and end values.

The last step is the optimization. The trajectory is the initial guess to the optimization algorithm and the values from the steady state solution for the start grade are used as a starting point. The optimization then tries to minimize a cost function. The cost function used here is quadratic. The cost function as well as the problem formulation is described in greater detail in (Larsson et al., 2011).

The deviation vectors are

$$\Delta \mathbf{y} = \mathbf{y} - \mathbf{y}_{\mathbf{B}}, \Delta \mathbf{u} = \mathbf{u} - \mathbf{u}_{\mathbf{B}}, \Delta \mathbf{w} = \mathbf{w} - \mathbf{w}_{\mathbf{B}}, \quad (2.5)$$

where the steady state solution for the end grade provides the end values $\mathbf{y}_{\mathbf{B}}$, $\mathbf{u}_{\mathbf{B}}$ and $\mathbf{w}_{\mathbf{B}}$.

Between the start time $t = t_1$ and end time $t = t_2$ the optimization problem can be formulated as

$$\begin{aligned}
& \underset{\mathbf{u}}{\text{minimize}} && \int_{t_1}^{t_2} \begin{bmatrix} \Delta \mathbf{y} \\ \Delta \mathbf{u} \\ \Delta \mathbf{w} \\ \dot{\mathbf{u}} \end{bmatrix}^T \begin{bmatrix} \mathbf{Q}_{\Delta \mathbf{y}} & 0 & 0 & 0 \\ 0 & \mathbf{Q}_{\Delta \mathbf{u}} & 0 & 0 \\ 0 & 0 & \mathbf{Q}_{\Delta \mathbf{w}} & 0 \\ 0 & 0 & 0 & \mathbf{Q}_{\Delta \dot{\mathbf{u}}} \end{bmatrix} \begin{bmatrix} \Delta \mathbf{y} \\ \Delta \mathbf{u} \\ \Delta \mathbf{w} \\ \dot{\mathbf{u}} \end{bmatrix} dt \\
& \text{subject to} && \mathbf{0} = \mathbf{F}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{w}, \mathbf{u}), \\
& && \mathbf{y} = \mathbf{g}(\mathbf{x}, \mathbf{w}, \mathbf{u}), \\
& && \mathbf{y}_{min} \leq \mathbf{y} \leq \mathbf{y}_{max}, \mathbf{u}_{min} \leq \mathbf{u} \leq \mathbf{u}_{max}, \\
& && \mathbf{w}_{min} \leq \mathbf{w} \leq \mathbf{w}_{max}, \dot{\mathbf{u}}_{min} \leq \dot{\mathbf{u}} \leq \dot{\mathbf{u}}_{max}, \\
& && \mathbf{x}_{min} \leq \mathbf{x} \leq \mathbf{x}_{max},
\end{aligned} \tag{2.6}$$

where the weight matrices $\mathbf{Q}_{\Delta \mathbf{y}}$, $\mathbf{Q}_{\Delta \mathbf{u}}$, $\mathbf{Q}_{\Delta \mathbf{w}}$, and $\mathbf{Q}_{\dot{\mathbf{u}}}$, are diagonal. By assigning different values to the weights it is possible to change the importance of different variables.

The minimization problem is translated to a nonlinear program (NLP) by JModelica.org. The resulting NLP problem is solved by the interior point solver IPOPT (Wächter and Biegler, 2006).

2.4 Python

The graphical user interface is written in Python (Python Software Foundation, 2011). In the early stages of the project different approaches were discussed mainly because the integration between IP21 database and other technology from Borealis lack Python interfaces. In the end Python was selected. The main reason for selecting Python is that the JModelica.org framework has a Python interface.

Python is a high level, general purpose, objects oriented programming language. One of the main goals of Python is to create an easy to read syntax. The result is a programming language very similar to pseudo code and the syntax is closer to human needs instead of computer instructions. The syntax is not only easy to read, it is generally fast to develop in and easy to use and learn.

The usability has a cost. The language is slower than compiled languages and not suited for demanding computations. To counteract this drawback Python makes it easy to integrate code written in other languages. Generally anything in need of high execution speed is written in other languages and later integrated into Python. The main reason to still use Python is that processor cycles are cheap while developer salaries are expensive.

2.4.1 Graphical User Interfaces in Python

Python has several toolkits for creating graphical user interfaces. The de-facto standard toolkit is Tkinter. Tkinter is often regarded as old, boring and feature

poor. To fulfill the role as the standard toolkit Tkinter focuses on multiplatform support as well as stability. The requirement to work flawlessly on multiple platforms might be a reason for the slow progress.

The main advantage of Tkinter is that it works. It works on multiple platforms and the few things it is capable of it does very well. Because of the limits of Tkinter several other toolkits have emerged. The main ones are wxPython, PyQt and PyGTK.

The first step of creating a user interface in Python is always to choose which toolkit to use. They all have strengths and weakness. All four major contenders have an impressive list of features, everything from HTML-rendering to widgets. The amount is large enough not to limit the choice. In fact, all of the mentioned toolkits can be used to create the graphical user interface used in this project. The final decision was made by elimination.

Tkinter is eliminated because of the look and feel. The older versions of Tkinter have a particular none native look. In newer versions it has improved but there is no compelling reason to still use Tkinter.

PyQt has some impressive development tools like Qt Designer. Qt Designer provides a drag and drop interface for creating applications. This is not something that is useful in this project since the flexibility of the graphical user interface require that a considerable amount of the code it is written manually. The drawback of PyQt is the restrictive license. To develop closed source application a license fee has to be paid by each developer both for PyQt and the underlining framework Qt. The added bureaucracy is the reason for eliminating PyQt.

The creator of PyGTK is a developer for GNOME so naturally the Linux integration is great. If the interface was meant to be used on Linux PyGTK would have been the natural choice. However this is not the case. Borealis uses Windows and while it is possible to make PyGTK work on Windows it is not worth the added trouble.

The last toolkit is wxPython. wxPython has an active community resulting in plenty of available support and tutorials. The active community is a great reason to choose wxPython. As a consequence of the large community wxPython is feature-rich. The community also guarantees that most of the problems a developer might face have already been solved before by someone else and the solutions are easily found online. wxPython is by many regarded as the toolkit that should replace Tkinter as the standard toolkit for creating graphical applications in Python. To ensure that the graphical user interface works on multiple platforms wxPython uses local graphic engines on different operative systems. On Linux wxPython uses the GTK+ engine. The engine is also used by PyGTK and the results are similar application on Linux. PyGTK has a more Python like interface resulting in pleasant Python integration. wxPython on the other hand is closer to the source written in C++. The preference is arbitrary and in the end the preferred syntax is a very subjective choice.

The main reason to select wxPython is in the end the Windows implementation because the application written as part of this thesis is in the end going to be used mostly on Windows. By using different graphical engines on different

Toolkit	Look and feel	License	Windows support
Tkinter	✗	✓	✓
PyQt	✓	✗	✓
PyGTK	✓	✓	✗
wxPython	✓	✓	✓

Table 2.1: Toolkit elimination

operative systems wxPython creates applications with a native look and feel. Applications look and behave as other applications and the result is a more intuitive behavior. A summary of the reason for choosing wxPython is shown in Table 2.1.

A more thorough comparison of the main graphical toolkits can be found in (Polo, 2008).

2.4.2 wxPython

wxPython is a wrapper around wxWidgets which is a graphical user interface toolkit written in C++ (wxPython, 2011). Graphical user interfaces are generally very dependent on performance to respond quickly to user input. This is why wxPython as well as the majority of other toolkits are written in other languages and not in Python. Instead a wrapper is provided to incorporate graphical elements or widgets into Python applications.

wxPython consists of a collection of widgets. A widget is a basic building block such as a button, slider, panel or a text field. By placing the widgets on panels or in windows the graphical user interface is created. wxPython has a large library of widgets covering everything from basic functionality to custom made widgets.

wxPython also supports several ways to order and position widgets in a window. Often it is not desirable to determine the position of widgets by hand. Instead several different tools can be used to place objects and items in grids or along the sides of the window. When creating an advanced user interface it is important to not only place the widgets in the preferable place but also make them behave properly when something is changed or a window is resized.

wxPython is used to provide the link between widgets and the Python code of the application. The core functionality of the application, created as a part of the thesis, is written in Python. While wxPython provides the tools to create plots it is often easier to use already existing toolkits specializing in the area. Matplotlib is such a library and is specially made for plotting.

2.4.3 Matplotlib and NumPy

Matplotlib is a Python library used to make plots. The goal of matplotlib is to provide functionality similar to the plotting capabilities of Matlab. Matplotlib supports several backends, one of them is wxPython. This means that plots in matplotlib can easily be integrated into graphical application made in wxPython or other GUI toolkits. Matplotlib is not the only plotting library in Python but it is mature and provides most of the required functionality. Matplotlib is used extensively in the application to compare and visualize data.

NumPy is an extension to Python that provides support for high level mathematical functions. NumPy supports multidimensional arrays and matrices as well as a wide collection of mathematical functions. NumPy is an essential package needed for scientific computing and is used extensively in the application.

Chapter 3

Work Flow and Specifications

3.1 Work Flow

The functionality of the application can be divided into a hierarchal structure as seen in Figure 3.1. The different modules contain related functionality. Depending on the user choices different modules needs to be accessed. The user can for example select between comparing and display old data or running a new optimization. The reason for not using the structure in Figure 3.1 as model for the graphical user interface is that a natural order between actions exists. Changes to optimization settings or model calibrations should be performed before the grade change optimization is started.

To increase the usability of the user interface a typical work flow has been used as a model for the application. The interface restricts the user from performing actions in the incorrect order by limiting the choices presented to the user. This has resulted in a linear work flow. The work flow is described in Figure 3.2.

From the start display the user can edit a grade or bypass the optimization process and instead directly plot results. This is useful if the user wants to plot old optimization results. After the initial selection of either plotting old data or making a new optimization the process is linear. First the start and end grade is selected to define the grade change. When the optimization is finished the results can be plotted or the process can be repeated for a new optimization.

3.2 Specifications

The intension with the graphical user interface is to connect process data with optimization and calibration algorithms. The goal is to create a graphical user interface able to configure optimization settings, selecting a grade transition

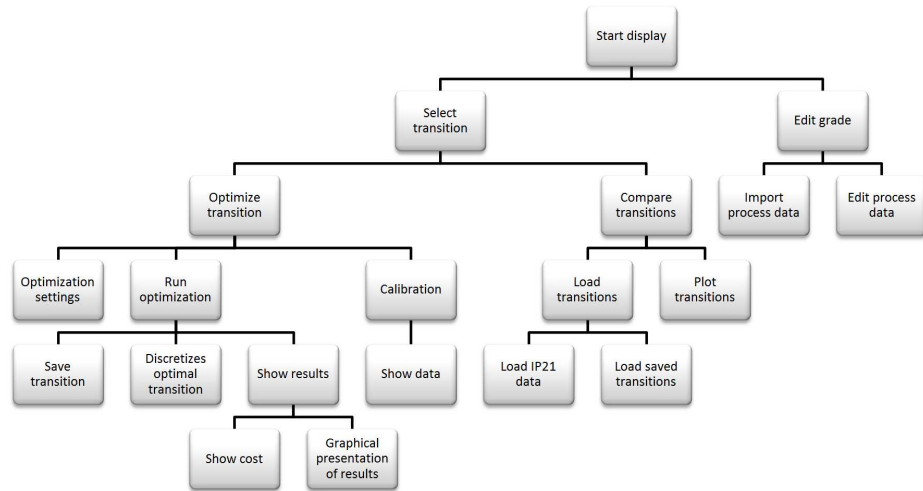


Figure 3.1: Schematic structure of the graphical user interface.

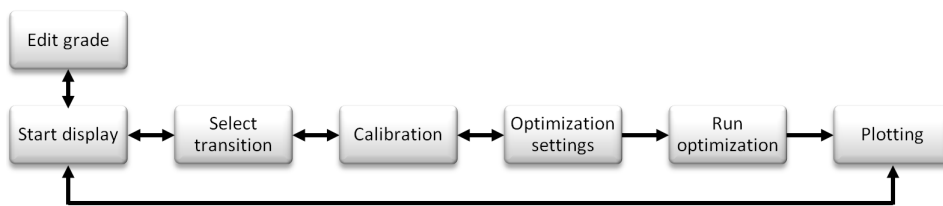


Figure 3.2: Schematic structure of the work flow.

as well as plotting the results from optimizations. The following specifications were used when creating the graphical user interface.

Start Display

The graphical user interface should have a start display welcoming the user. From the start display the user should be able to select the transition by selecting the start and end grade used in the optimization. The user should also be able to define new products or edit existing ones. The start display and the functionality to edit grades are shown on the top and top left side of Figure 3.1.

Grades

The grade has to fulfill certain specifications to ensure that the produced plastic is within tolerance limits as described in Section 2.3. The grade specifications as well as necessary flows, operating conditions and measurement values needed

in the calibration are used to define a grade. Each product should be storable as a text file and when needed loaded in to the graphical user interface where the parameters defining the grade can be edited.

Calibration

In the case of the static model calibration, measurement data stored in the grade specifications are used to calibrate the model. The graphical user interface should have the ability to import measurement data from the IP21 database and display the data to the user. As the static calibration only calibrates against steady state measurement values it only needs the average value of the measurements over a time period. The user interface should therefore calculate and display the average measurement values.

The graphical user interface should also have the ability to perform calibrations for different predefined parameter sets and the interface should have the ability to display the calibrated parameters.

Optimization Settings

The goal of the user interface is to move all the necessary optimization settings from script files to the application. Settings such as number of elements and maximum number of iterations should be changeable from the user interface. In addition it should be possible to change the diagonal elements in the optimization weight matrices described in Section 2.3 from the user interface. The graphical user interface should have the ability to edit and display maximum and minimum values used during the grade change optimization. The user interface should also be able to solve the grade change optimization problem with the defined weights and constraints.

Product prices and raw material cost should also be displayed to the user and the user should have the ability to change and edit the data. It should also be possible to save the product prices to a file and later import the data to the graphical user interface.

Importing Process Data

The graphical user interface should permit the user to import process data from the IP21 database containing measurement data. The timespan for the measurement values should be selectable by the user and a mean value displayed in the user interface. It should be possible to edit the displayed values and if necessary the user should be able to manually input every value needed to define a grade or to do a static calibration.

The measurement data is given to the graphical user interface as an ASCII text file. The user interface should have the ability to define the start time and duration for the data import before importing the data from the database. It should be possible to save the imported data, use it to specify a grade, calibrate the model with it or compare the data to optimization results.

Plotting and Comparing Results

A crucial part of the user interface is the ability to visualize data. Not surprisingly, a long list of specifications details the plotting capabilities of the user interface.

First of all, the graphical user interface should have the ability to plot the results from the grade change optimizations. The graphical user interface should also have the ability to plot old data from previous optimizations to enable the user to compare results from different optimizations. This can be achieved by letting the user save the results and later load it into the user interface. Additionally, the user interface needs functionality allowing the user to load process data to assist the user in comparing the optimized results with real grade transition data. The user interface should assist in comparing data from different sources by plotting similar data in the same plots. To improve the ability to compare real data with optimized results the graphical user interface should be able to shift the data along the time axis. Furthermore the graphical user interface needs to support having several variables plotted in the same plot.

Chapter 4

Graphical User Interface

In Chapter 3 the specifications for the user interface are described. The chapter contains an impressive list of features. Although the required functionality is thoroughly described nothing about the design is mentioned. The artistic work was deliberately unspecified. This chapter is dedicated to explaining the user interface. Both the design and functionality is described in detail. The chapter starts with a description of the main window and how the functionality is divided between different panels that are displayed to the user during different steps in the optimization process. The last section is dedicated to settings as well as external windows and dialogs.

4.1 Main Window

At startup the main window is opened. The main window contains core functionality and is used to open secondary windows. The main window consists of several panels containing sets of buttons and controls. The different panels are used to group similar functionality together and to select what is displayed to the user. To navigate between the different panels the buttons in the bottom corners are used.

The process is inspired by the standard installation procedure in Windows where the user clicks next a few times and then does something else while a progress bar slowly moves forward. The reason for using a familiar concept is to decrease the learning curve and make the interface more intuitive. The goal of the user interface is to convey information in an effective manner and not to test new innovative ways to display data. This is the reason for using standard ideas and why established concepts are preferred.

At startup the main window contains two panels as seen in Figure 4.1. On the left the grade selection panels is displayed. To the right the plot panel is shown. By using functionality from the different panels it is possible for the user to either do a new optimization or plot old or saved data. The *New* button in the lower right corner is used to open the next panel. The button on the

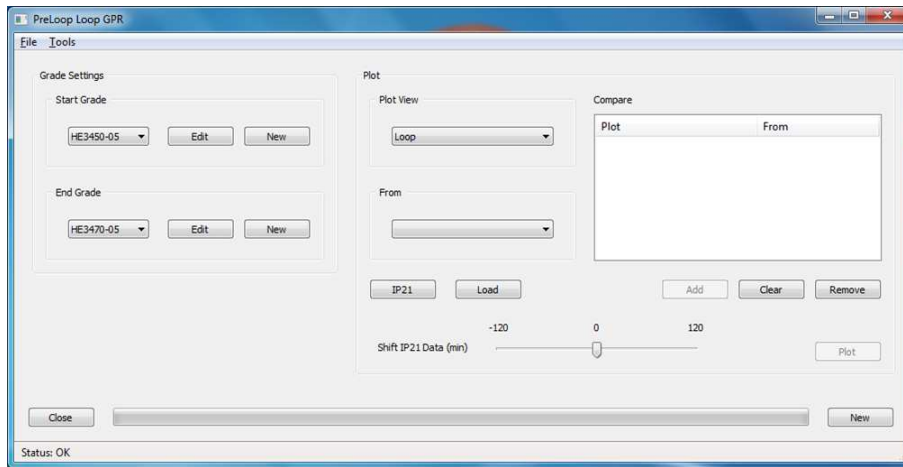


Figure 4.1: Main window at startup.

bottom left is usually used to go back to the previous panel but can also be used to close the application. The panels have the following order:

1. Grade selection panel/Plot panel
2. Calibration panel larger
3. Optimization settings panel
4. Optimization panel
5. Plot panel

4.1.1 Grade Selection

The grade selection panel is shown when the user interface is started. From the grade selection panel the transition is determined by selecting the appropriate start and end grade from the drop down lists. A new grade can be created by pressing the *New* button and old grades can be edited by pressing the *Edit* button. Before moving to the next panel both a start and end grade has to be selected. By pressing the *New* button in the bottom right corner the start and end grades are saved and the first step in the optimization procedure is completed. When the *New* button is pressed the calibration panel appears and both the plot panel and the grade selection panel vanish.

4.1.2 Calibration

The calibration panel is the first panel to appear after the initial grade selection panel. In the calibration panel the user can select to run a calibration by

pressing the *Calibrate* button. The calibration is optional but it might increase the accuracy of the results from the optimization.

The calibration uses grade data to calibrate the selected model variables. In the default configuration a steady state calibration is done with grade data from the start grade. The model variables used in the calibration can be selected from the settings window. A few predefined cases are available. The model variables that are selected are shown in the user interface with their current value as shown in Figure 4.2.

When the calibration is started the status bar at the bottom of the window will be used to display the current action. After the calibration is done the new values will be shown in the user interface. After an optional calibration the user can move to the next panel by pressing the *Optimization* button. This will open the settings panel where optimization settings are chosen before the optimization takes place.

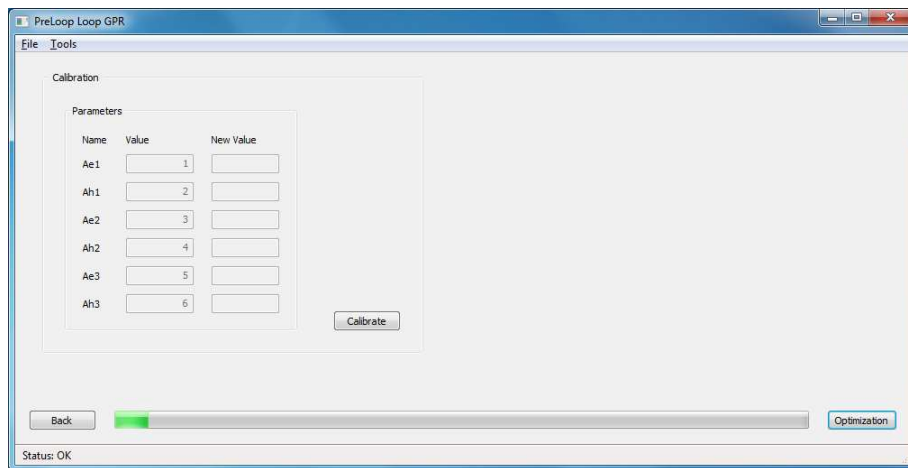


Figure 4.2: Calibration panel.

4.1.3 Optimization Settings

Before the optimization starts it is important to select appropriate optimization variables, constraints and weights. This can be done directly in the optimization settings panel shown in Figure 4.3. By clicking on the tabs at the top a different selection of settings are displayed. The box on the right indicates the selected start and end grade. At any time before the optimization has started the user can go back to the initial panel or calibration panel by pressing the *Back* button.

The optimization is started by pressing *Run*. The only way to abort an optimization is by closing the application. It is imperative that the settings are selected before the optimization is started. The *Back* button is not available during the optimization.

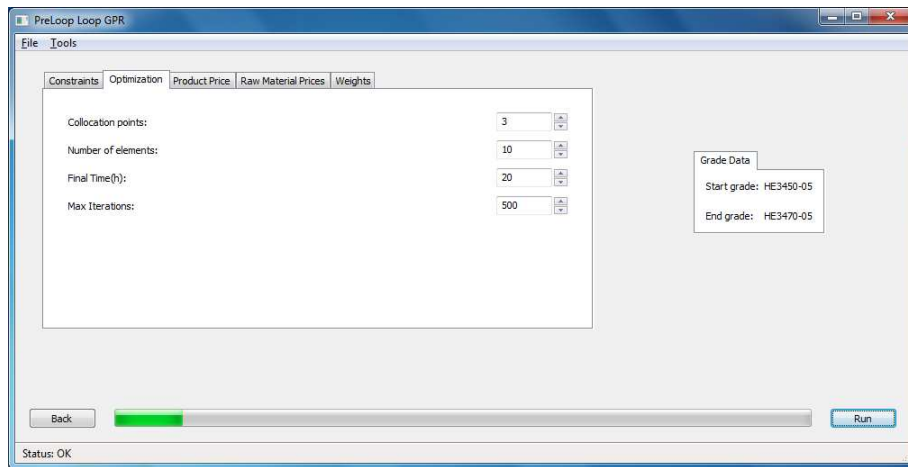


Figure 4.3: Optimization settings panel.

The time requirement for the optimization is dependent on the machine as well as the installed solver. The optimization can be made faster by selecting fewer collocation points as well as a smaller number of elements. Lowering the number of elements or changing collocation points will influence the results. The first time the optimization runs the model will be compiled which will also increase the optimization time.

4.1.4 Optimization

During the optimization the progress is indicated by the gauge at the bottom of the screen. The gauge values are updated when the user moves between panels to give a visual representation of the position of the current panel in the optimization progress. During the optimization the gauge values are updated when different steps are finished. The status bar at the bottom of the window describes the current action. Table 4.1 describes gauge values as well as the corresponding action. The final optimization is the most time demanding step in the optimization process. Because of the unpredictable nature of the time requirement of the optimization algorithm it is impossible to give accurate time estimations. The gauge values can therefore only be used to indicate the progress and does not reveal how long the optimization will take.

During the optimization the print stream from the optimization algorithm can be captured. The printed text can be shown in the optimization panel described in Figure 4.4. From the printed text it is possible to decipher information about the progress of the optimization.

Action	Gauge value (%)
Calibration panel	5
Calibrating	6
Calibration done	10
Optimization settings panel	10
Optimization panel	15
Compiling static start point	25
Calculating static start point	30
Compiling static end point	35
Calculating static end point	40
Simulating trajectory	45
Compiling optimization model	50
Calculating optimal grade change	60
Done	100

Table 4.1: Gauge values.

4.1.5 Plotting

When an optimization is finished the plot panel will automatically appear. The plot panel is identical to the plot panel shown when the main window is started. The only exception is that after the optimization the optimization results are already loaded into the *From* field. Figure 4.5 shows the plot panel as it is displayed after an optimization.

To create a plot the user has to define the plotted variables and select from which file the data should be collected. There are several predefined plots already in the *Plot View* list. User defined plots can also be added. The plots are divided into the following groups: *Loop*, *GPR* and *MyPlots*. If *Loop* is selected every plot in the *Loop* group is added.

Data can be plotted from several different sources. New data, saved data and measurement data can be compared and studied side by side. If the same data is plotted from different sources it will appear in the same plot.

Measurement data from the IP21 database can be loaded by pressing the *IP21* button. When the button is pressed a new window appears. The IP21 window is shown in Figure 4.6. In the IP21 import window the user has to specify the time interval for the data importation. By pressing *Import* the data is loaded and is added to the *From* list in the plot panel.

To increase the ability to compare transitions with measurement data the IP21 data can be shifted sideways. Instead of reloading IP21 data to get a perfect match the slider can be used to slightly move the IP21 data. The intension is to decrease the amount of times the user has to reload IP21 data by ensuring the initial selection does not have to be precise. Instead a rough time estimate

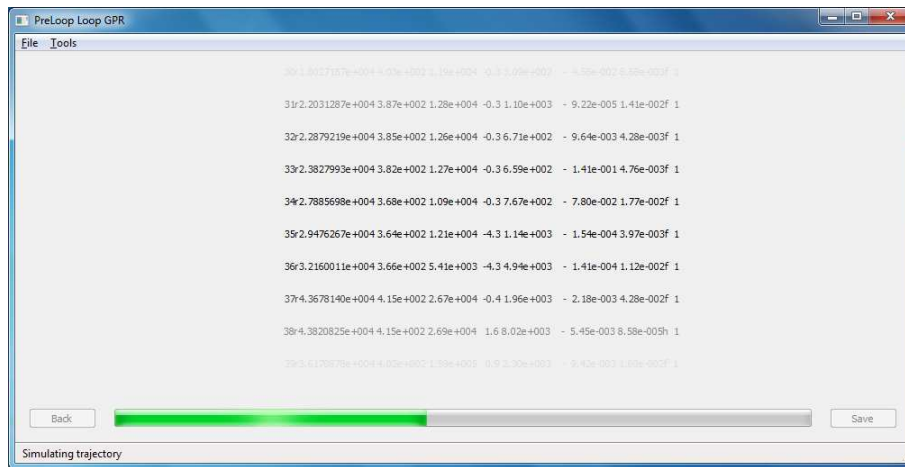


Figure 4.4: Optimization panel.

can be used and the fine tuning can be done later when the data is already imported.

Saved data can be loaded by pressing the *Load* button next to the *IP21* button. Data is loaded by choosing the data file and pressing *Open* in the file dialog window. It might take a few seconds to import the loaded data. The progress of the import is visualized by the gauge at the bottom of the window. When the gauge reaches 100% the data is fully imported.

When data is loaded or a new optimization has been completed the data can be plotted. A plot is created by selecting the plotted variables from the *View* list and also selecting the data source from the *From* drop down list. By pressing *Add* the plot is added to the compare list. The selected plot can be removed by pressing *Remove* or the entire list cleared by pressing *Clear*.

In the plot panel the buttons will become available when the corresponding action is a valid choice. The *Plot* button becomes accessible only when entries are present in the compare list. Similarly the *Add* button can only be selected when both a set of model variables and a source file is selected. The reason for restricting the user behavior is to limit errors caused by missing data or not fully imported files. By only allowing plotting when entries have been added it becomes impossible to plot data before the data has been imported.

When the *Plot* button is pressed one or several windows will appear depending on the number of entries in the compare list. If several plots are added they all emerge simultaneously. An example plot is shown in Figure 4.7. The plots are created using the matplotlib toolbox. In the plot window the plots can be saved or the user can zoom in to a selected section.

The entire plot procedure can be summarized in the following steps:

1. Import data or run an optimization

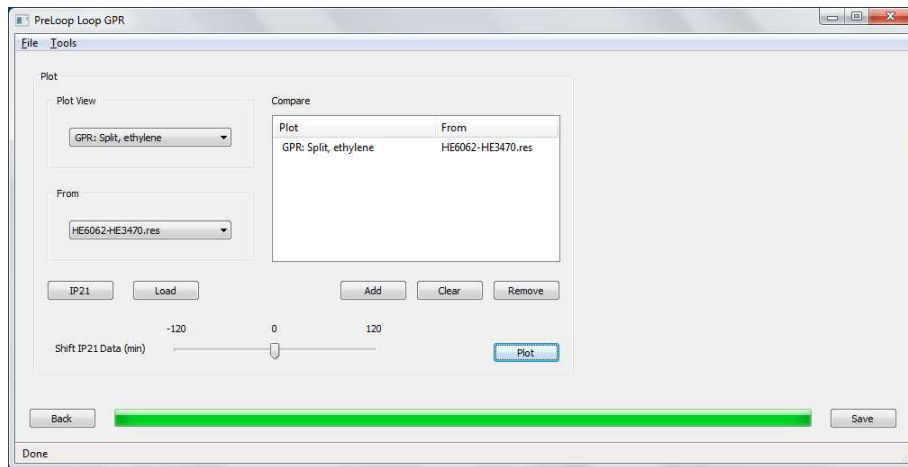


Figure 4.5: Plot panel after an optimization.

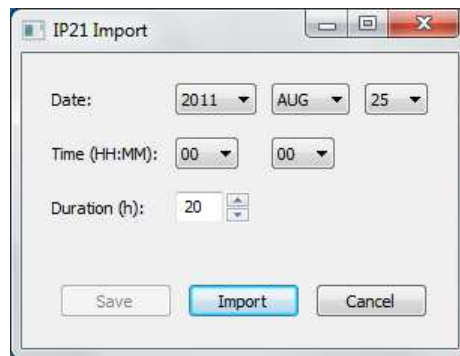


Figure 4.6: Window for importing IP21 data.

2. Select source file / Select model variables
3. Add plots
4. View Plots

When the optimization is finished the data can be saved by pressing the *Save* button in the plot panel. In the save dialog the default file ending is *.res*. The file type is arbitrary and is only used to aid the user in distinguish saved data from other files. In the plot panel the *Back* button is also available. The *Back* buttons opens the initial panel and can be used to do a new optimization.

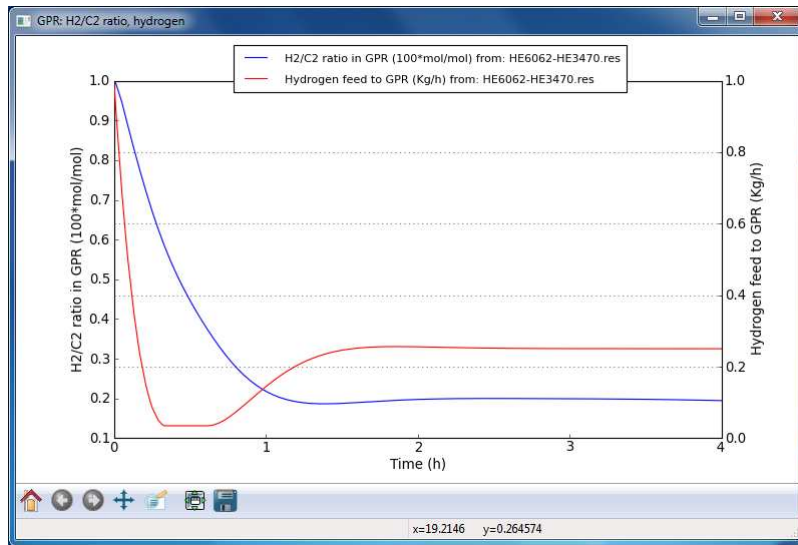


Figure 4.7: Example plot.

4.2 Features and Settings

4.2.1 Edit Grades

At startup all grades saved in the *Grades* folder are imported. A new grade can be created either by editing an old grade or by creating an entire new grade. Grades can be edited or created from the main window by pressing the *Edit* or *New* buttons on the select grade panel. To edit an already existing grade the grade has to be selected as either start or end grade from the drop down lists. The *Edit* button opens the grade window. Figure 4.8 shows an empty grade created by pressing the *New* button.

In a new grade all parameter values are zero. To facilitate the creation of a new grade, data can be imported from IP21. Instead of entering every value by hand a suitable parameter set can be imported by selecting a time interval where the grade was used and then importing measurement data. When data is imported the mean value is calculated over the time interval to get a more representative value and minimize the impact of measurement noise. The imported data is also used to calibrate the model. The data can be saved by pressing the *Save* button. To change the name of an existing grade or create a new grade with an old grade as base the *Save As* button can be used instead.

4.2.2 Menus

The menu in the main window has two sub menus. From the *File* menu it is possible to load an old optimization by pressing *Load*. The menu option is comparable to using the *Load* button in the plot panel. Figure 4.9 shows the

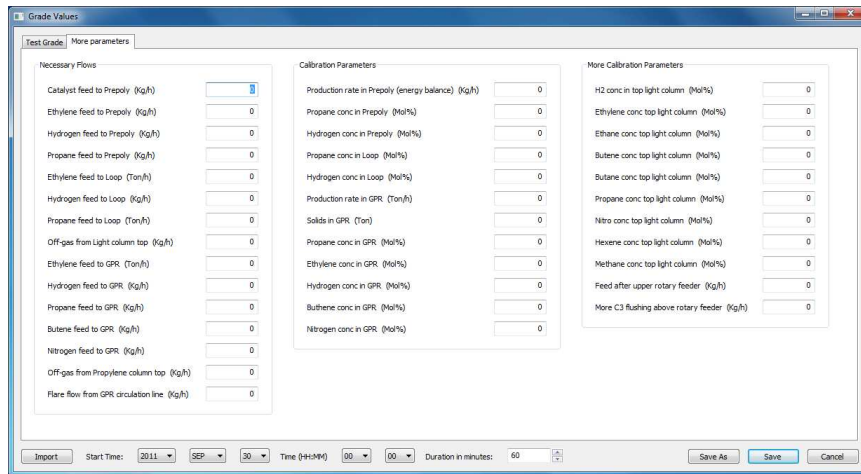


Figure 4.8: Grade window.

save option in the *File* menu. Although the *Save* option can be accessed at all times it is only possible to save the latest completed optimization and it is not possible to save an ongoing optimization to continue at a later time. It is also possible to close the application by pressing *Quit* in the *File* menu. The *Tools* menu can be used to access the settings window.



Figure 4.9: File menu in the main window.

4.2.3 Settings

The settings window is opened from *Settings* in the *Tools* menu. The Settings window is shown in Figure 4.10. The box on the left side of the settings window is a tree controller and by clicking a label the corresponding settings are shown to the right. If an item in the tree controller has child items it can be expanded by double clicking it.

In the settings window settings of similar type are grouped together in dif-

ferent panels. In panels with large amount of data, tabs are used to divide the information. Additional information is accessed by selecting the appropriate tab. In the tree controller a specific tab can be selected by clicking on a sub item.

Settings can be saved and shared. Settings can be saved by pressing the *Save* button and selecting a name in the save dialog. The default folder for saved settings files is the *Settings* folder. Old settings can be imported by pressing *Import* and selecting the saved settings file in the open file dialog. By pressing *OK* the settings are saved. *Cancel* will close the window.

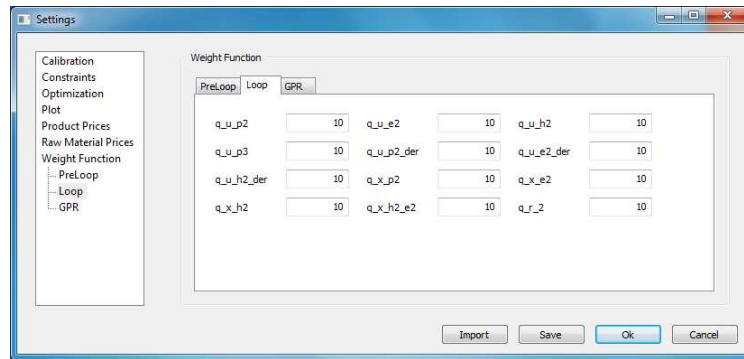


Figure 4.10: Settings window.

Calibration

In the calibration settings panel it is possible to select the parameter set used in the calibration. Only a few predefined cases are available. The predefined cases can only be changed by editing the default values in the *Data.py* file. When the selection is saved by pressing *OK* in the settings window the calibration panel in the main window is automatically updated.

Constraints

The constraint panel has five tabs with constraints. Both minimum and maximum values can be changed. For the feeds the maximum and minimum values for the derivative are also selectable. It is important to carefully select the constraints. By selecting values for the constraints that are located inside the range defined by the start and end values of the variable the optimization algorithm might fail to converge. If an end or start value is located outside of the constraints it can only be reached if the constraints are breached. If the optimization algorithm obeys the constraints the start or end values can therefore never be reached and a solution cannot be found.

Optimization

The optimization setting panel has four options. The number of collocation points and number of elements influences the accuracy of the optimization. More being better but also requiring a longer execution time. The final time option sets the end time for the optimization. The change of the final time requires a recompilation of the model. This will increase the execution time the first time the chosen value is used in an optimization. The maximum number of iterations makes it possible to limit the time it takes to run an optimization. The maximum number of iterations should only be reached if the optimization fails to converge.

Plotting

In the plot settings panel the user can edit existing plots or create new customized plots. At startup, predefined plots are loaded. Predefined plots can only be permanently saved by changing the Data.py file. New plots are added by typing a new in the name text box and selecting a primary and secondary plot variable. New plots will be added to the *MyPlots* group. The *Remove* button will remove the selected plot. Predefined plots cannot be removed permanently from the user interface.

The grouped plots checkbox decides how plots are positioned on the screen. If several plots are added, the default behavior is to divide the screen equally among the plots. If there are a square number of plots they will be placed in a square formation. Additional plots will be placed in an extra column to adapt to common screen resolutions. By unchecking the check box, the plots will instead be positioned on top of each other, as shown in the Figure 4.11.

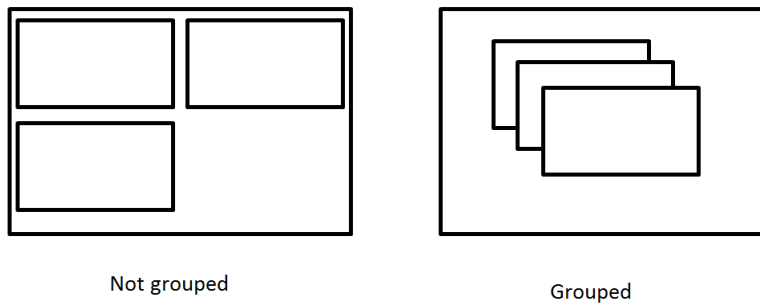


Figure 4.11: Plot options.

Product and Raw Material Prices

From the user interface product and raw material prices can be changed. These values are currently not used in the optimization algorithm. The prices will therefore not effect the results from the optimization. The product prices are

stored separately in a file in the *Settings* folder. They are loaded when the application is started. Saved files can be shared between users by copying the *Product prices* file in the *Settings* folder. By selecting a grade from the drop down menu the product prices can be edited. The *Save* button will save the values to the *Product prices* file.

Weight Functions

In the current optimization algorithm a quadratic cost function is used. Weights are used in the quadratic cost function to determine the influence of different variables. The default value for the weights is 10 but it can be changed from the user interface. The default value has no significance only the ratio between the weights is important. By increasing a weight the cost of the deviation from the end value will increase for the selected variable.

Chapter 5

Implementation

In this chapter the implementation of the graphical user interface is described. During the development, several complex problems were discovered. This chapter describes the solutions as well as the motivation for the resulting implementation. The chapter starts with a description of the model changes performed to ensure that the model can be used for arbitrary grade changes. The chapter also contains a description of how threads and processes are used in the graphical user interface. The chapter ends with a section detailing the underlying design principles and the interaction between classes.

5.1 Model Modifications

The model of the polythene production plant used in the thesis is only tested on a small number of hard coded grade transitions. To ensure that arbitrary grade transitions can be optimized, the Modelica model has been slightly rewritten. Additionally several modifications need to be made to the model to make sure that model parameters can be changed from the user interface.

5.1.1 Constraints

Every variable in the model has a maximum and minimum value. The default values are tailored after a few studied transitions. New transitions might have variable values outside of the default limits. This will cause the optimization algorithm to fail to converge. In bad cases the stationary solutions used to calculate the start and end points for the optimization algorithm will fail. Occasionally the optimization of the grade transition will fail as well. The reason for the convergence failure is that the solution is outside of the boundary limits and therefore it can never be reached. An example is shown in Figure 5.1.

The solution is to expand the default maximum and minimum values to encompass every typical value used by the different grades. Changing the default values will not effect the results from the optimization because an additional

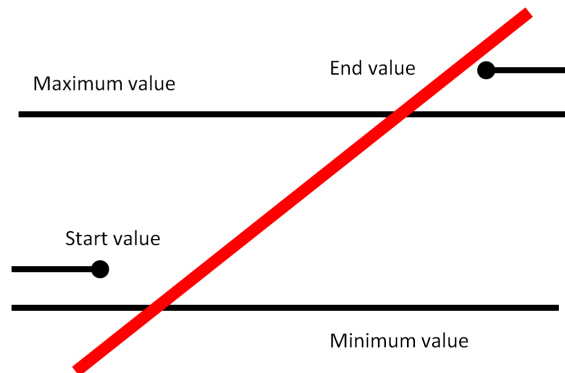


Figure 5.1: The end value cannot be reached because it violates the constraints.

set of constraints exist to ensuring that parameter values remain within safety margins.

Minimum and maximum values set in the Modelica model cannot be changed after the model has been compiled. While recompiling the model to change the boundary values is an option, the extra time required is unnecessary. Instead a different approach can be used to ensure that the user can set constraints from the user interface without the need to recompile the model. Constraints can be moved from the Modelica model to the Optimica extensions used to define the optimization problem. In the Optimica extension the constraints can be changed even after the model has been compiled. The drawback of having two sets of constraints is that they might interfere with each other and the Optimica constraints have to be within the limits determined by the values in the Modelica model.

5.1.2 Initial Trajectory

The initial trajectory used as a start guess for the optimization algorithm is created by using a simple ramp function between the start and end points as an input to a simulation. The trajectory generation is described in more detail in Section 2.3. In the original model the ramp function was created by using an if-statement. The ramp was constructed by defining the derivative. After the maximum value was reached the if-statement changed the derivative to zero.

The initial trajectory was conceived by simulating the model in Dymola with the ramp as an input to the simulation. While Dymola support discontinuous functions, JModelica.org does not. If-statements, sign functions or similar constructions can therefore not be used to create the ramp function. Instead a continuous approximation of the derivative was used to ensure that JModelica.org could be used to create the initial guess of the trajectory. Instead of an if-statement the following equation was used to create the ramp:

$$y = 1 - 1/(1 + e^{-k(t-t_{end})}) \quad (5.1)$$

At the time t_{end} the approximation will change value from one to zero. A larger k value results in a sharper curve as seen in Figure 5.2. The derivative of the ramp is calculated by multiplying (5.1) with the slope of the ramp.

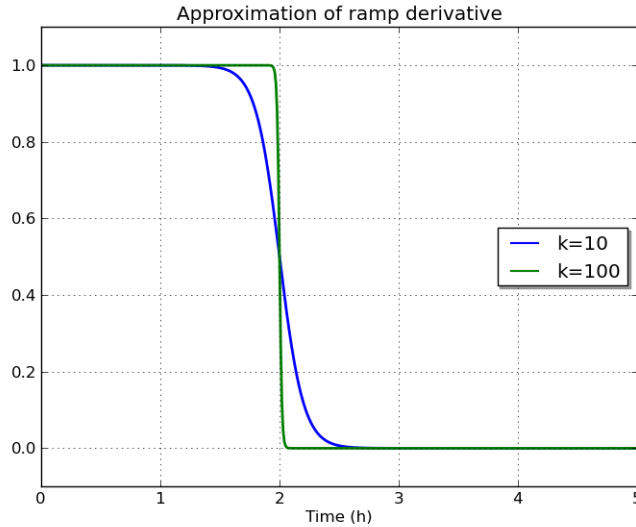


Figure 5.2: Continuous approximation used to calculate the derivative of a ramp.

5.1.3 Changing Model Values

The optimization process can be divided into three steps. In each step changes are made to the model. In the first step the model is initialized by calculating stationary solutions for both the start and end grade. Before the first step parameters defining the grades are saved to the model. Additionally calibrated model parameters are saved as well. The calibrated parameters have to be saved to the model before each of the three steps.

The start and end values calculated in the first step are used in both the initial trajectory estimate and in the final optimization. They are saved to the model before either the trajectory generation or optimization takes place. Before the optimization weights and constraints also need to be saved to the model. A schematic description of the changes to the model files is presented in Figure 5.3.

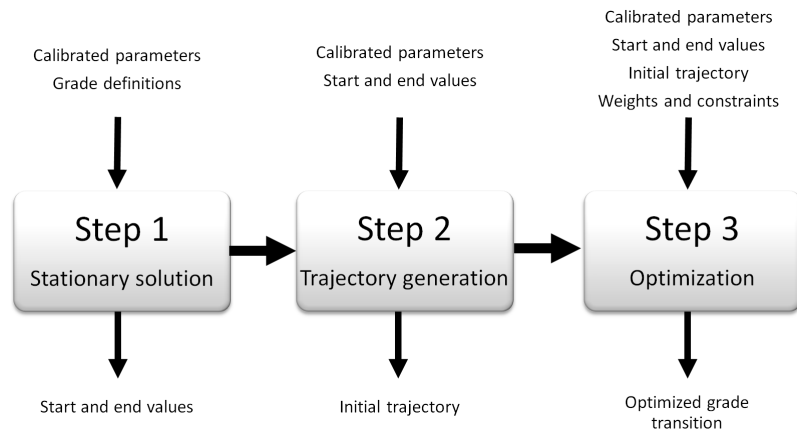


Figure 5.3: The model files are changed before each step in the optimization process.

5.2 Compiling

The compilation of model files takes a considerable amount of time. By limiting the need for compilation the total optimization time can be minimized. This is achieved by reusing compiled files as often as possible. Every optimization requires the presence of four compiled model files. The model files for the stationary solution for both the start and end grade needs to be compiled.

The model includes a few different grades with slightly different parameter values. When a new grade is created a preexisting grade is used as a mold for the new grade. A new grade will therefore not technically require a recompilation of the model files because it is based on an already existing grade. As a legacy feature every grade has a corresponding compiled model file despite the fact that need for one is limited. The grade specific files only need to be compiled once for every grade. After that they are reused.

Two other compiled files are necessary. The model for simulation of the initial trajectory generation and the optimization model need to be present. Both models only need to be compiled once with one exception. The change of the optimization interval requires a recompilation of the optimization files. This is caused by the fact that the optimization interval cannot be changed in the compiled model.

The first time an optimization or calibration is started all needed model files will be compiled. This will significantly increase the execution time. If a compiled version of a model file is present it will be detected by the application and reused. The reason for not providing precompiled model files is that they are operative system dependent. Precompiled model files will not work on different platforms.

5.3 Threads and Processes

To be able to ensure that the graphical user interface remains responsive while resource heavy calculations are running in the background, the execution of the user interface and the calculations need to be separated. In Python this can be done in several different ways. When external code is used the solution often involves either threads or processes. Both threads and processes are used in different parts of the implementation.

5.3.1 Threads

Threads within the same process share a common data space. The shared data space minimizes memory overhead and ensures that threads can communicate and share information. The simplified communication is the main advantage of using threads. Threads have a few disadvantages as well. The support for threads in Python is limited. It is not possible to easily prioritize, interrupt or terminate threads. Usually different creative implementations can be used to overcome most of the limitations.

The standard solution for combining a long running task with a user interface is to run the tasks in a separate thread see (Bolen, 2004). To communicate with the main thread events can be utilized. In graphical user interfaces events are used to react to user input. When a button is pressed an event is created and the *MainLoop()* will respond to the event by calling the predefined handler function. Similarly the sub-threads can generate a custom event that the main thread is listening to. This is used throughout the application to update the status of the calculations and to indicate when a task is finished. This is also described in Figure 5.4.

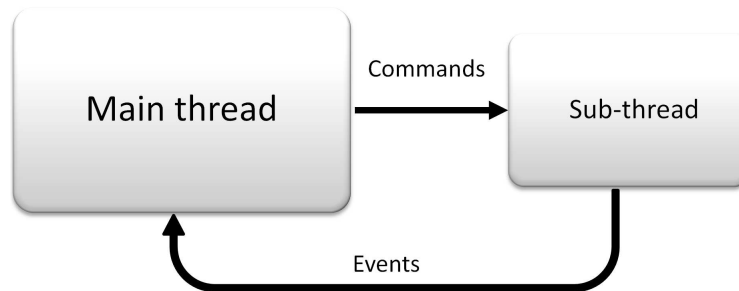


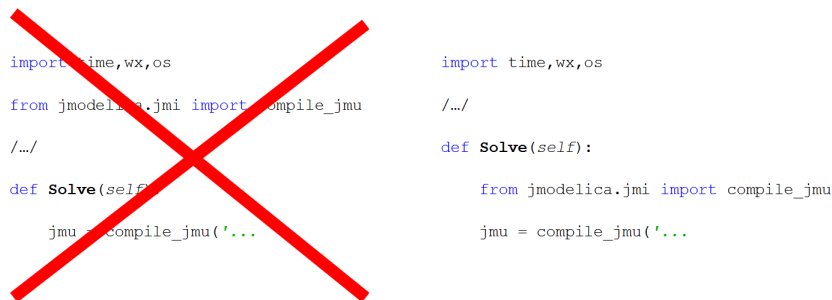
Figure 5.4: Schematic description of the communication between threads.

Using threads instead of processes makes the implementation easier but a few issues arise. One of them is that after the optimization has been started it is not possible to abort it. This has to do with the fact that killing threads is an inherently bad idea because the thread might be accessing a critical resource that needs to be closed properly. This is the reason why Python lacks an official

way to kill threads. Despite this there are several ways to terminate Python threads. Unfortunately none of the workarounds can be used to abort a Python thread while it is running external code. The only way to abort an optimization is therefore to kill the process running the application.

More problems arise from the fact that JModelica.org uses a Java Virtual Machine (JVM) to compile model files. Compiling has to be done in the thread containing the JVM. Trying to use JModelica.org to compile files outside of the thread containing the JVM will cause Python to crash. The solution is obvious but the implementation is not intuitive.

Usually a Python script starts with a list of modules to be imported. When JModelica.org components are imported the JVM is also started. The result is a JVM running in the main thread despite the fact that the JModelica.org functions are used in a different thread. To ensure that the JModelica.org functions always are used in the same thread as the JVM the JModelica.org components should only be imported when they are used. Importing the same module twice will not cause problems as Python remembers imported modules and only imports the module if it is missing. Figure 5.5 has a more in-depth description.



```
import time,wx,os
from jmodelica.jmi import compile_jmu
/.../
def Solve(self):
    jmu = compile_jmu('...')

import time,wx,os
/.../
def Solve(self):
    from jmodelica.jmi import compile_jmu
    jmu = compile_jmu('...')
```

Figure 5.5: The JVM is started when a JModelica.org module is imported. The import statement has to be in the same thread as the JModelica.org commands.

The difficulties in using the JVM is also a reason why it is not possible to run two different optimizations simultaneously as all the JModelica.org functions have to run in the same thread.

5.3.2 Processes

The mentioned difficulties with using threads can be solved by using processes instead. Processes can be terminated and because they are independent it would be possible to run JModelica.org functions in several different processes.

Processes are not superior in every way. They have a large disadvantage. A large amount of data has to be transferred between the user interface and the thread or process running JModelica.org code. With threads this is a trivial

task. Communication between different processes is nontrivial and as a result processes are only used in two special cases.

Processes are used to run external code and to capture printed text from IPOPT. In both cases text files are used to communicate between processes. In the latter case a process is only used as a last resort. The optimization algorithm uses external print commands to print text to a console. The only way to capture the printed text is to listen to the output from a process. By running the entire graphical user interface as a separate process it is possible to capture the printed text and save it to a log file or otherwise integrate it into the graphical user interface. A schematic description of how process output can be saved is shown in Figure 5.6.

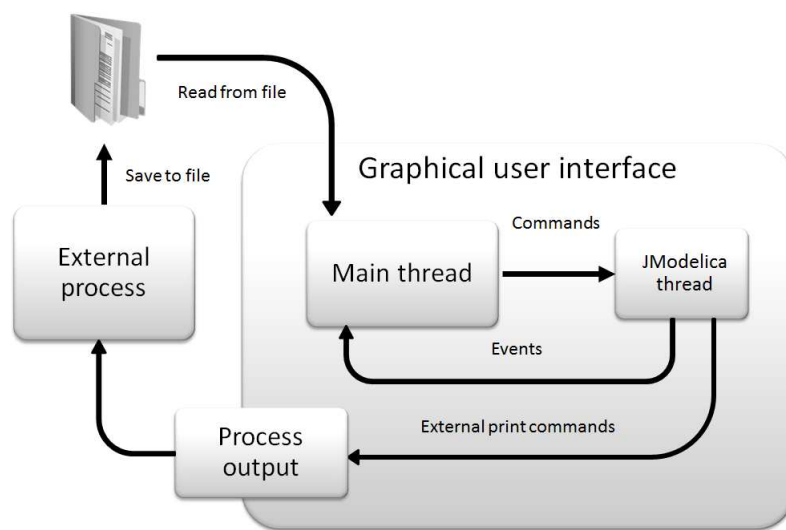


Figure 5.6: To save the printouts from the optimization algorithm an external process has to be used.

Another advantage of using processes is that after the optimization is finished the process can be terminated and the memory cleared. The memory footprint from using JModelica.org functions increase after several uses. This is caused by memory leaks. Running several large optimizations repeatedly might cause out of memory problems. By using processes the theoretical problem is eliminated.

Using processes instead of threads is often a superior solution. The reason for still using threads in certain situations is that the implementation becomes easier. The only current limitation to core functionality is the fact that an optimization cannot be aborted. Every other drawback is either limited to theoretical problems or solved by changing the structure of the code.

5.4 IP21 Communication

Borealis uses a database for measurement data called IP21. IP21 contains both historic and current measurement and target values. Measurement data from the IP21 database is used to define grades, calibrate the model and compare optimization results with real transitions. To communicate with the IP21 database, an external program written in Fortran is used. The reason for using an external program is that IP21 lacks a Python interface. The Fortran program developed by Borealis requires input in the form of a text file with a list of tag names. The text file has the following format:

```
01-MAR-11 00:00:00.0 01-MAR-11 20:00:00.0
FK442101-16_IP
AI470113-308E
FT443101-18
```

The first line contains the start and end time for the import. The time interval is followed by a list of tag names indicating variables to be imported. When the program is executed a result file is created containing values for every variable with a tag name. The following format is used to describe the returned file:

```
23-JAN-11 22:57:00.0 23-JAN-11 23:59:00.0
FK442101-16_IP AI470113-308E FT443101-18
51.9893875 9.6282234 87.8617325
51.9892812 9.6304234 87.7236432
51.9893765 9.6365324 87.6727221
```

The format is similar to the input file. The top row is a time interval followed by a list of tag names. For every minute one set of values are returned. The values are imported to the graphical user interface by reading the text file containing the data.

5.5 Saved Files

Both grade data and user settings are or can be saved as text files. There are a few requirements for the format used to save the data. The saved files need to be editable by hand and it should be easy to create and import the files.

Both grade data and settings are internally stored as large dictionaries filled with entries. A dictionary in Python is a data structure with pairs of keys and values. The following structure is used to create a dictionary in Python:

```
Dictionary = {'key': 1, 'key2': 2}
```

By using an identical format to save data the saved files can be created by simply converting the dictionary to a string and writing the string to a text file. Even importing is straightforward. The *eval()* function can be used to convert the string of text to a dictionary. Using the *eval()* function can be dangerous because it allows the user to run arbitrary code by changing the saved files.

This is not a problem in the application because if the user has the ability to edit files and run python scripts arbitrary code can be executed anyway.

By using a format identical to a Python data structure both saving and importing data becomes trivial. The drawback is that the format is not designed to be easy to edit when dealing with a large dictionary. The saved data is still saved in plain text so the option to edit the data still exists.

5.6 Design of the Graphical User Interface

The user interface is designed to be easy to change and upgrade. While the user interface is flexible it still has to be robust and the input from the user needs to be limited to only allow permitted behavior. A great challenge when creating a user interface is the ability to limit and screen user input without being intrusive. If it is possible for a user to input data in the wrong format it is only a matter of time before a mistake is made.

5.6.1 Restricting User Input

The user interface has to make sure that user mistakes will not propagate to the application and be saved. As a result default values are used every time the graphical user interface is started. While it is possible to change and save the values the changes will not be loaded automatically to minimize the risk that a mistaken value is used repeatedly without the knowledge of the user. The only exception is grade data that is entirely user defined and as such loaded on startup.

User input is also limited by using widgets that only allow input in certain formats. An example is the optimization settings. By using spin controls with maximum and minimum values instead of a normal text control the user can only input integers within the predefined limits. Another example is the date and time input used when importing data from IP21. Time input is problematic because different formats are used worldwide and a clear consensus has not been reached allowing dates to be defined in different formats depending on the context.

In the user interface the time format has to be both easy for the user to recognize and the input has to be checked for validity. This is achieved by allowing the user to only choose valid dates by selecting the year, month and day from drop down menus. The values in the drop down menus are updated after a selection is made. If the year 2012 and the month February are selected there will be 29 days in the drop down menu.

5.6.2 Increasing flexibility

To increase the flexibility of the user interface and allow changes to be easily made, the majority of the interface is not hard coded. Instead the interface is built from predefined lists of model variables when the application is started.

At startup, the user interface reads data from a file conveniently called `Data.py`. The data defined in the file is the cornerstone of the user interface. `Data.py` contains lists of variables used in different parts of the user interface.

A simple list of variable names is transformed into a grid filled with text controls. The variable values can be edited and when necessary imported from IP21 or saved to the model. By changing a name of a variable in the list the user interface will be updated, connections to IP21 database or model files will be automatically made and the new variable can be plotted. The reason for using this type of approach is to simplify routine maintenance and prepare the interface for future upgrades. The result is that the user interface only needs to be updated in one place to accommodate model changes. The `Data.py` file is the DNA of the graphical user interface and changes to the file will propagate throughout the application.

By having a centralized source of information regarding names, tags and corresponding model names the graphical user interface can be easily updated without the need to rewrite core functionality. In addition to default values, pre-defined plot options and unit conversion factors are also defined in the `Data.py` file. Default values can therefore be changed even by a novice user without any deeper knowledge about Python or the user interface.

5.7 Program Structure

The core of the application is the main window. From the main window all other windows are opened. The main window is the center of the application from both the perspective of the user and the code. A graphical description of all the classes can be seen in Figure 5.7.

The main window consists of five panels. Every panel is a separate class responsible for both displaying a part of the graphical interface as well as implementing the underlining functionality. The main window, as well as the entire user interface, is created in the main thread.

Several parts of the interface can be opened in separate windows. All secondary windows are separate classes. They are all independent of the main window. To transfer information between classes, a dictionary containing data is used. The dictionary is called settings. The settings dictionary is created from the content of `Data.py` file and updated when changes are made by the user. The settings dictionary is used to save data and user settings throughout the entire application and is a central source for storing or transferring information. The settings dictionary works as a database for measurement values, grade data and settings. Almost every module can access it.

The settings window consists of seven panels where each of the panels is a separate class. The reason for dividing the settings into different classes is that several of the setting classes are reused in the main window. Both the settings window class and the grade window class can save and load data from text files. The writing and reading of text files is separated into two classes, one class for reading and one class for writing.

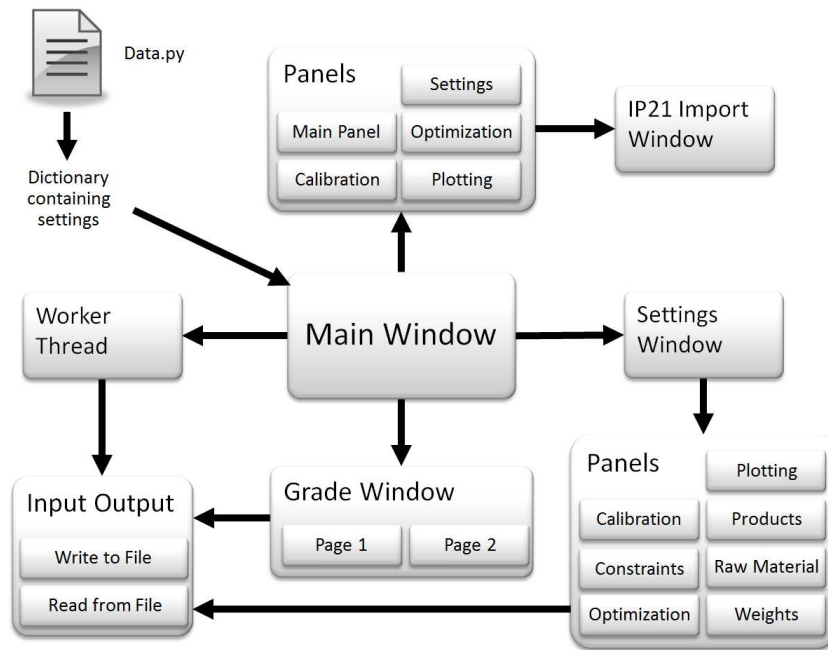


Figure 5.7: Graphical representation of the connection between classes.

From the main thread responsible for the user interface a worker thread is also started. The worker thread is made for running heavy calibration and optimization calculation without disturbing the user interface. The worker thread is a separate class. To save changes to model files before compiling them the worker thread also uses the write to file class. The connection between the classes is shown in Figure 5.7.

The `Data.py` file is loaded by the main window and passed down to the other windows and to the worker thread. This ensures that all major parts of the user interface share the data contained in the `Data.py` file. The main window is responsible for opening secondary windows and responsible to listen to events from the worker thread or user input in the main window. The secondary windows will respond to events created within the own window and occasionally call the main window to update the graphical user interface. The settings window will for example call the main window when a new set of settings are saved. The main window will then respond by updating its panels or redoing the layout to accommodate the settings changes.

Chapter 6

Epilogue

6.1 Discussion

In the end the usefulness of the application is determined by the results from the model calibrations and grade change optimizations. If the results are not comparable with real measurement data the graphical user interface loses its usefulness. Not only does the results need to be accurate they need to be reliable as well. A user of a graphical user interface take for granted that the underlying methods work and expects a result in every case regardless of input or user settings. This is a very challenging problem to solve as the optimization and calibration methods have been tested only on a few cases.

The optimization algorithms for the model without recycling are surprisingly robust. The optimization relies on good start and end values. If it is possible to calculate stationary solution for the start and end values the final optimization problem has a very high probability of succeeding even when weights are changed. In essence the problem is limited to calculating good stationary solutions. For the six grades that where studied as part of the thesis, stationary solutions where found for every grade. Every possible grade change between the existing grades where therefore theoretically possible to solve with the optimization algorithm.

The possibility to optimize a grade change does not guarantee that the solution is good. By changing the weights the performance can be increased by selecting the importance of variables but in the end the performance of the optimization is limited by the quadratic cost function. The quadratic cost function tries to minimize the cost by reaching the end point as fast as possible. No economic values are considered and the transition can therefore be wasteful.

In addition the model accuracy limits the performance. The accuracy of the model can of course be improved by performing a calibration before the optimization is started. The calibration can improve the model but not dramatically change it. In the end the results cannot be better than the underlying model.

The task of creating the user interface that combines the calibration and

optimization algorithms is trivial. The difficulties lie in the implementation of the optimization and calibration algorithms themselves. The step between solving a few selected grade transitions to solving all of them is huge. For the user interface to be usable the optimization and calibration have to be reliable. Achieving a high success rate on the optimizations is a demanding task, especially when the calibration can change the behavior of the model by changing model parameters. If only a few selected cases can be solved there is no point in having a user interface.

6.1.1 Code Structure

One of the goals when making the user interface was to make it flexible and easy to change. To achieve the high flexibility this user interface was built around a central source containing data and the changes to the central data would propagate throughout the program. This has the advantage that smaller changes can easily be made but there are several disadvantages as well. One of them is that the application becomes less modular as everything depends on the central source of information. Fundamental changes to the core data also become difficult as several different modules can depend on the same data. The central data source can therefore cause unpredictable problems when changed carelessly because the changes will effect the entire application.

The lesson to learn is that everything has a tradeoff. The sacrifice of making smaller changes easy is that the modules become dependent on each other. The general idea of having a central data source is on the other hand a very proven concept. Databases are present almost in every larger application and the prospect of hard coding everything is not very appealing. The result of having a data source similar to a database is that every module can easily access variable names or default values. It also becomes easy to translate between different forms of names such as model names and display names.

The system that is used to make the graphical user interface might not be ideal but it is very convenient. A new part of the graphical user interface can be created from a basic for loop and three rows of code that is reused from somewhere else.

6.2 Conclusions

The goal of the Master's thesis was to create a graphical user interface showcasing calibration and optimization methods developed at Lund University. This goal has been achieved. A graphical user interface has been created, capable of solving grade change optimization problems on a calibrated plant model. The graphical user interface can be used to change optimization constraints and optimization weights and the optimized trajectories can be visualized and compared to real measurement data.

6.3 Future Work

The scope of the work described in this thesis is limited and the user interface is by no means a final product. As such a lot of work remains. The model used in the optimization is not complete and to increase the accuracy of the optimization results the model needs to be expanded to include systems that were left out in the initial version of the user interface. In addition to changing the model additional features can be added to the graphical user interface. A few of those features are described below.

6.3.1 Dynamic Calibration

As part of the thesis only a static calibration method was included in the graphical user interface. A dynamic calibration method could be included as well. The dynamic calibration method minimizes the difference between measurement data and model simulations over a time period instead of only calibrating against measurement values from a steady state.

6.3.2 Expanding the Model

A crucial future step is to expand the model to also include the recycling system. To minimize waste and raw material consumption the plant in Stenungsund cleans and recycles materials. The recycling system is not used by the optimization in the work done as part of the thesis. By including the recycling in the model the results should be closer to measurement data and the accuracy of the optimization results should increase.

6.3.3 Output to Operators

When the model has been expanded to include the recycling new opportunities arise. The results from the optimization could be transformed to instructions that the plant operators can use. One idea is to parameterize the input feeds that the optimization algorithm manipulates to achieve the optimal trajectory. By making the inputs piecewise linear it would be easier for an operator to implement the optimal trajectory. By having the graphical user interface produce a list of values and times when inputs should be changed the application could be directly used to produce instruction for the operators. This would drastically increase the usability of the graphical user interface. The reason for not implementing the idea in the early phase of the project is that results from the grade change optimization methods will only be used on the real plant after extensive testing and after ensuring that the underlying model is good enough.

6.3.4 Economic Cost Function

The quadratic cost function has several disadvantages. One of them is that economic impact is not measured and the results from the quadratic cost func-

tion do not change depending on market prices. Instead the user has to change the weights manually to change the optimized trajectories. The quadratic cost function is not aware of the limits of the grade specifications and the drastic reduction in income when products outside of the specifications are produced is not included in the cost function. A remarkable improvement to the quality of the results could be achieved by using an economic cost function instead. An economic cost function that measures the profit as an expression of income and expenses could be used instead.

Bibliography

- J. Åkesson, T. Bergdahl, M. Gäfvert, and H. Tummescheit. Modeling and Optimization with Modelica and Optimica Using the JModelica.org Open Source Platform. In *Proceedings of the 7th International Modelica Conference 2009*. Modelica Association, Sept. 2009.
- N. Andersson, P.-O. Larsson, J. Åkesson, S. Haugwitz, and B. Nilsson. Calibration of a polyethylene plant for grade change optimisations. In *In 21st European Symposium on Computer-Aided Process Engineering.*, 2011.
- D. Bolen. Long running tasks, 2004. <http://wiki.wxpython.org/LongRunning/Tasks>.
- Borealis AG. Borealis Home Page, 2011. <http://www.borealisgroup.com>.
- P.-O. Larsson, J. Åkesson, S. Haugwitz, and N. Andersson. Modeling and optimization of a grade change for multistage polyethylene reactors. In *18th IFAC World Congress*, Leuven, Belgium, July 2011.
- G. Polo. Pygtk, pyqt, tkinter and wxpython comparison. *The Python Papers*, 2, 2008.
- Python Software Foundation. Python Programming Language - Official Website, 2011. <http://www.python.org>.
- The Modelica Association. The Modelica Association Home Page, 2011. <http://www.modelica.org>.
- A. Wächter and L. T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–58, 2006.
- wxPython. wxPython Home Page, 2011. <http://wxpython.org>.