Summer Fellowship Report

On

Implementation of GUI for simulation

Submitted by

Anisha Patra

Under the guidance of

Prof.Kannan M. Moudgalya
Chemical Engineering Department
IIT Bombay

Acknowledgment

I, Anisha Patra, would like to express my sincere gratitude for being given the opportunity to intern at FOSSEE and work on the OPENMODELICA project. I would like to extend my heartfelt thanks to my mentors, Mr. Sumanto Kar and Mr. Nikhil Sharma. This wouldn't have been possible without their unwavering support and understanding despite the challenges faced. I am deeply grateful for their mentorship, which has imparted invaluable knowledge and provided me with a remarkable learning experience. I would also like to extend my thanks to Prof. Kannan M. Moudgalya for this opportunity.

Contents

Chapter 1

Introduction

1.1 OpenModelica

OpenModelica is a free/libre and open-source environment based on the Modelica modeling language for modeling, simulating, optimizing, and analyzing complex dynamic systems. Open Source Modelica Consortium supports its development. It runs on Windows, Linux and Mac OS.FOSSEE, IIT Bombay has taken up the initiative of promoting FLOSS (Free/Libre and Open Source Software), for education. We, the OpenModelica team at FOSSEE, IIT Bombay, promotes the use of OpenModelica as being accessible and readily available. The goal of this project is to enable the students and faculty of various colleges/institutes/universities across India to use Free/Libre and Open Source Software tools for all their modeling and simulation purposes, thereby improving the quality of instruction, learning and to avoid expensive licenses of commercial modeling and simulation packages for research and education.

1.2 Technologies Utilized

Language: Python
Libraries: PyQt6, Matplotlib, Pyzipper, os, shutil, sys, regex, xml.etree.ElementTree, subprocess, pathlib

1.3 Objective

Worked on building GUI which abstracts the modelica code.The models can be directly simulated through the GUI.

Chapter 2

Problem Statement

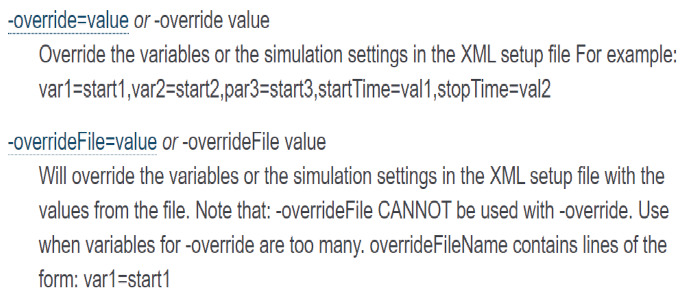
2.1 Simulating from the GUI

2.1.1 Problem Statement

Configuring the OpenModelica Compiler (OMC) environment was crucial for successful simulation. This required correctly setting up the OMC path and using specific simulation flags through the GUI application. Ensuring that the GUI could handle these flags to override simulation values was a significant challenge.

2.1.2 Outcome

The OMC was successfully configured.
The flags were correctly applied to override values through the GUI.It based on the below documentation



-override=value *or* -override value

Override the variables or the simulation settings in the XML setup file For example: var1=start1,var2=start2,par3=start3,startTime=val1,stopTime=val2

-overrideFile=value *or* -overrideFile value

Will override the variables or the simulation settings in the XML setup file with the values from the file. Note that: -overrideFile CANNOT be used with -override. Use when variables for -override are too many. overrideFileName contains lines of the form: var1=start1

It was tested to confirmed that the GUI could indeed handle the simulation process effectively, resulting in accurate and reliable simulation outputs.

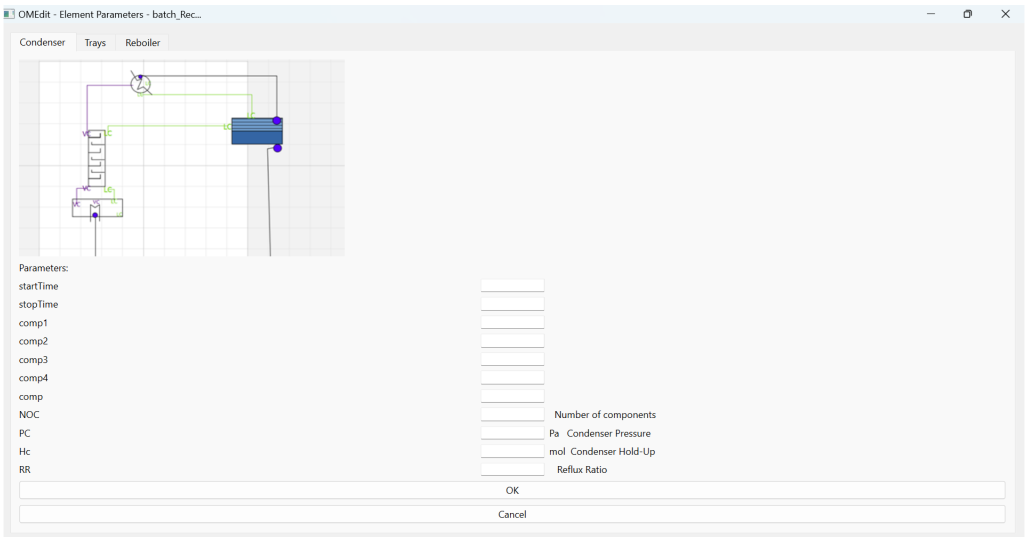
2.2 UI Design

2.2.1 Problem Statement

Designing a user interface (UI) that could take inputs efficiently and include a button to initiate simulations required thoughtful planning and execution. The UI needed to accomodate the image of the model and provide an seamless user experience.Also model 3 and model 4 had components so different tabs were used.

2.2.2 Outcome

A comprehensive UI design was developed and implemented, successfully addressing the requirements. The design incorporated user-friendly features and ensured that the interface was intuitive and responsive, enhancing overall usability.



Parameters:		
startTime		
stopTime		
comp1		
comp2		
comp3		
comp4		
comp		
NOC		Number of components
PC	Pa	Condenser Pressure
Hc	mol	Condenser Hold-Up
RR		Reflux Ratio

The above output shows the GUI of model3

2.3 Reflecting Values in OpenModelica

2.3.1 Problem Statement

Ensuring that values from the .mat file were accurately reflected in OpenModelica was essential for our simulation and analysis. We encountered a significant issue where, despite the correct values being plotted, these updated values were not being reflected in the variable browser of OpenModelica. This discrepancy was problematic because it meant that the values displayed in the variable browser did not match those in the plot, leading to confusion and potential inaccuracies in interpreting the simulation results.

The core of the issue seemed to be related to how OpenModelica was handling the data imported from the .mat file. The updated values were correctly processed and used in plotting, but the variable browser continued to display outdated or default values. This inconsistency suggested that there might have been a problem with how the values were being overridden or updated in the internal representations within OpenModelica.

2.3.2 Outcome

To address this issue, we undertook a comprehensive approach to ensure that the values from the .mat file were properly integrated and reflected in OpenModelica. We made modifications to the XML file using Python code to ensure that data was accurately displayed and utilized within OpenModelica. This involved ensuring that the XML configuration was correctly updated to reflect the changes from the .mat file.

By carefully adjusting the XML file, we were able to synchronize the data across both the plotting and variable browser views, ensuring consistency and accuracy. This also allowed us to set the default values as per our needs, resolving the issue of outdated values appearing in the variable browser. As a result, the values were correctly reflected in all relevant views within OpenModelica, ensuring that our simulation results were accurate and reliable.

2.4 Distributing the Executable

2.4.1 Problem Statement

We needed an efficient method to package and distribute the application as a standalone executable. The goal was to select a tool that would simplify distribution and installation for users.

2.4.2 Outcome

We used PyInstaller to package the application:

1. **Setup**: Created a virtual environment and installed the required libraries.

```
python -m venv OMGui
cd OMGui/Scripts
activate.bat
pip install pyinstaller PyQt6
```

2. **Packaging**: Used PyInstaller to build a standalone executable from `main.py`.

```
cd ../..
pyinstaller --onefile main.py
```

This process ensured that the executable was easy to distribute and install, as it was fully self-contained and did not require additional dependencies.

2.5 Finding Ways to Perform Simulation

2.5.1 Problem Statement

The presence of compounds could not be modified through the existing flags. Therefore, the existing approach had to be changed to perform the simulation.

2.5.2 Outcome

- Direct modification of the Modelica code was attempted but proved ineffective.
- The '.mos' approach enabled successful re-simulation and adjustments to compounds.

2.6 Using modelica script for simulation

2.6.1 Problem Statement

Re-simulation was necessary due to the presence of compounds that could not be modified through the existing flags.

2.6.2 Outcome

To address the issue and successfully re-simulate, the following steps were undertaken:

1. **Setup and Simulation**: The Modelica environment was configured and the necessary files were loaded. The simulation was executed with the updated parameters.

   ```
   loadModel(Modelica);

   cd("tmp/Modules");
   loadFile("package.mo");
   loadFile("Batch_Rectifier.mo");
   loadFile("ChemsepDatabase//GeneralProperties.mo");
   loadFile("Compounds.mo");
   simulate(Modules.Batch_Rectifier, startTime=0, stopTime=3);
   getErrorString();
   ```

2. **Results**: The '.mos' code was effectively utilized to address the limitations of flag-based modifications. This allowed for successful re-simulation and adjustments to compounds, resulting in accurate and comprehensive simulation outcomes.

2.7 Hiding of the Modelica Code

2.7.1 Problem Statement

To protect intellectual property and secure the Modelica code, various methods were explored to hide and encrypt the code. The goal was to ensure that the code remained inaccessible to unauthorized users while maintaining its usability during execution.

2.7.2 Outcome

- **Encrypted Folder Approach**: Initially, an approach was considered where the Modelica code would be stored in an encrypted folder. This method involved encrypting each file individually and required a specific password for decryption. Although this approach provided a high level of security, it proved to be slow due to the time required for the decryption process.
- **Encryption and Decryption Scripts**: Python scripts were developed to implement encryption and decryption:
  - `Encrypt.py`: This script encrypted the Modelica files using AES encryption. It created an encrypted version of each file and organized them in a new directory structure with modified names. The encryption process used a password-derived key and ensured the security of both file contents and filenames.
  - `Decrypt.py`: This script managed the decryption process, restoring the encrypted files to their original state in a temporary directory. It used the same password to derive the key and decrypt both file contents and filenames, ensuring that the files were accessible only during execution.
- **Password-Protected Zip File**: A more practical solution was to use a password-protected zip file. This approach involved compressing the Modelica code into a single archive and protecting it with a password. During execution, the zip file was temporarily extracted to a 'tmp' directory, where the files were processed. After processing, the temporary files were deleted. This method offered a balance of security and convenience, reducing overhead and simplifying file management.
- **Execution and Cleanup**: The password-protected zip file approach enabled secure temporary extraction of files during execution, while the encryption and decryption scripts ensured that the code remained protected when not in use. This method effectively hid the Modelica code and managed its accessibility in a secure and efficient manner.

2.8 Using Zip Files

2.8.1 Problem Statement

Managing and securing large directories containing Modelica files required an efficient method for both compression and password protection. The challenge was to create a mechanism that could bundle all necessary files into a compressed archive and protect them with a password, ensuring restricted access during simulations. Additionally, the process needed to support extracting the files for simulation, ensuring that the files remained secure and that the user could easily re-access them when necessary.

2.8.2 Outcome

The solution involved using a two-step process for creating and securing zip files, with the ability to extract them when required for simulation:

- **Compression and Password Protection**:
  - The directory containing Modelica files was first compressed using Pythons `zipfile` module, which created an intermediate zip file without password protection.
  - After compression, the `pyminizip` library was used to apply a password to the zip file, ensuring that the contents remained encrypted and secure. The original, unprotected intermediate zip file was removed after applying the password.
- **Extraction for Simulation**:
  - When required for simulation, the protected zip file was extracted using a password. The system first extracted an intermediate zip file, followed by extracting the actual contents.
  - The extracted files were temporarily placed in a designated folder for simulation purposes, and could be cleaned up after execution to maintain security.
  - This approach ensured that files were secure during storage and transport but easily accessible during execution.
- **Benefits of Using Zip Files**:
  - The zip file approach provided a straightforward and efficient method to compress and protect multiple files while ensuring security through password encryption.
  - The extraction process was designed to be secure, extracting files only temporarily for simulation and ensuring that sensitive data was not left accessible post-execution.

2.9 UI Design for GUI-2

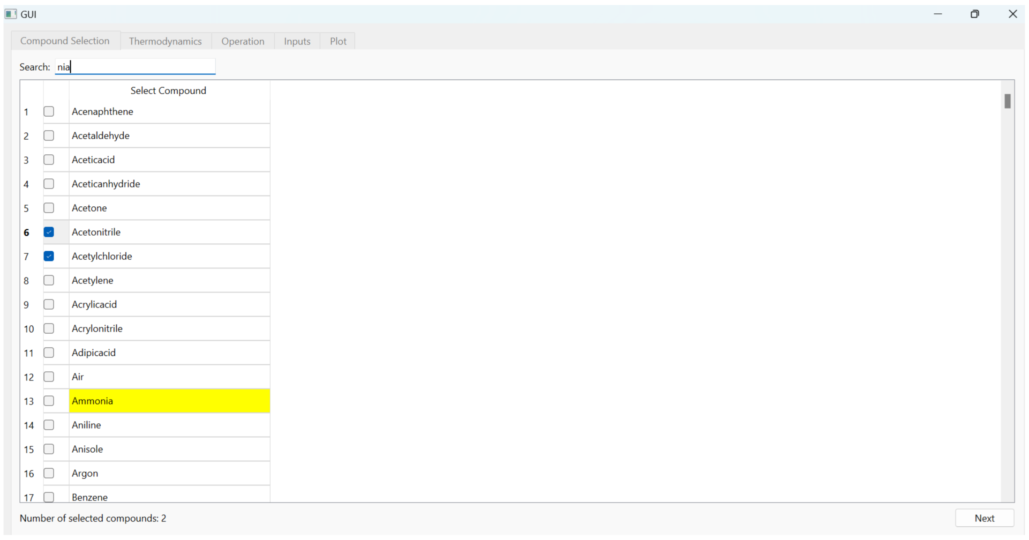
2.9.1 Problem Statement

The UI design for GUI-2 required a more detailed and organized layout due to the complex functionality needed. The design involved creating a multi-tabbed interface with the following components:

1. **Compounds**
2. **Thermodynamics**
3. **Operation**
4. **Inputs**
5. **Plot**

2.9.2 Outcome

- **Compounds Tab**: This tab allows users to select different compounds from a list. A counter at the bottom of the tab shows the number of selected compounds. The interface includes a search feature for easy location of specific compounds. To ensure data completeness, users are required to make a selection before proceeding to the next tab.

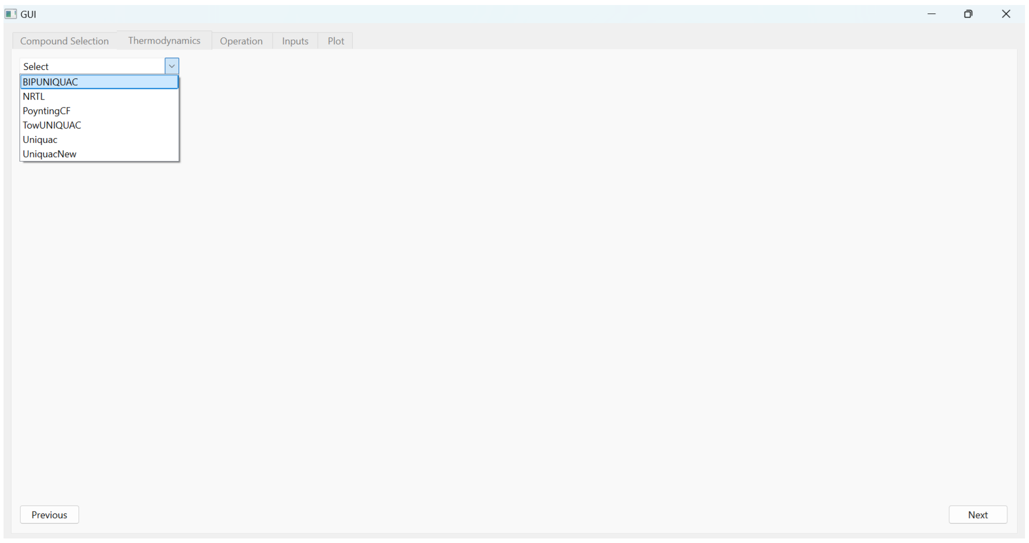


Search: nia

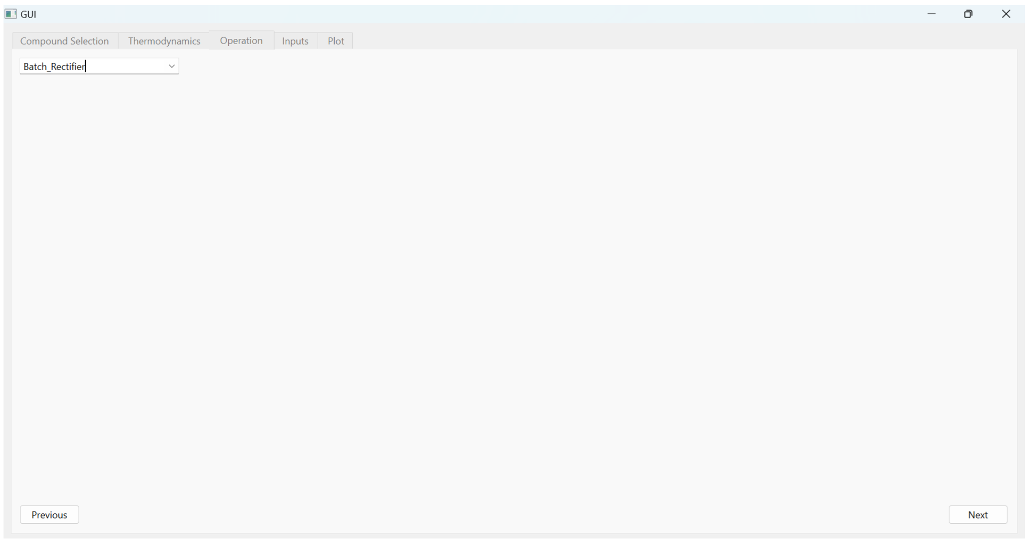
		Select Compound
1	☐	Acenaphthene
2	☐	Acetaldehyde
3	☐	Aceticacid
4	☐	Aceticanhydride
5	☐	Acetone
6	✓	Acetonitrile
7	✓	Acetylchloride
8	☐	Acetylene
9	☐	Acrylicacid
10	☐	Acrylonitrile
11	☐	Adipicacid
12	☐	Air
13	☐	Ammonia
14	☐	Aniline
15	☐	Anisole
16	☐	Argon
17	☐	Benzene

Number of selected compounds: 2

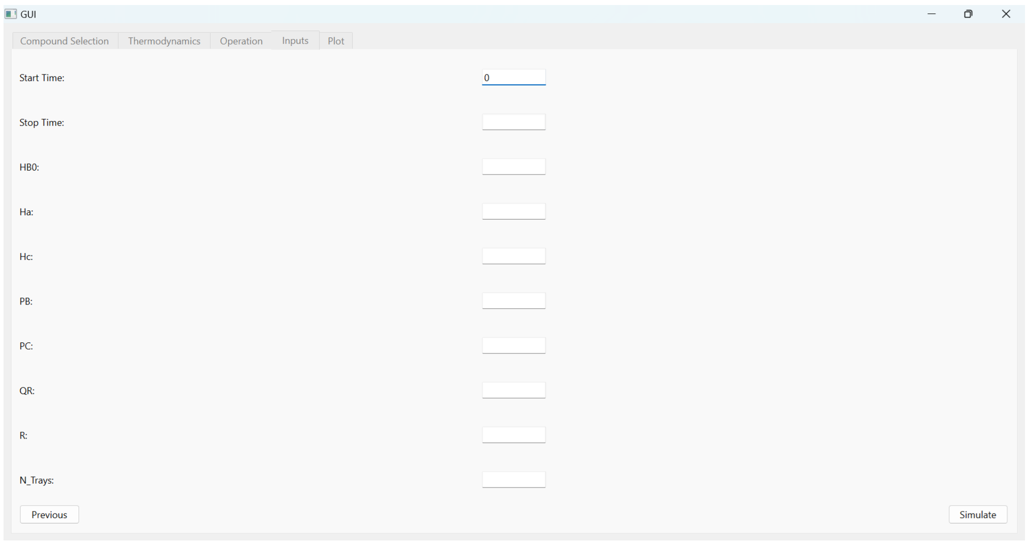
- **Thermodynamics Tab**: Users can choose thermodynamic properties from a dropdown menu. The tab also includes a search function to facilitate finding specific thermodynamic properties. This feature streamlines the process of selecting relevant properties for simulation.



- **Operation Tab**: Similar to the Thermodynamics tab, users select operations through a dropdown menu and can search for specific operations. This tab organizes available operations and simplifies the selection process for users.

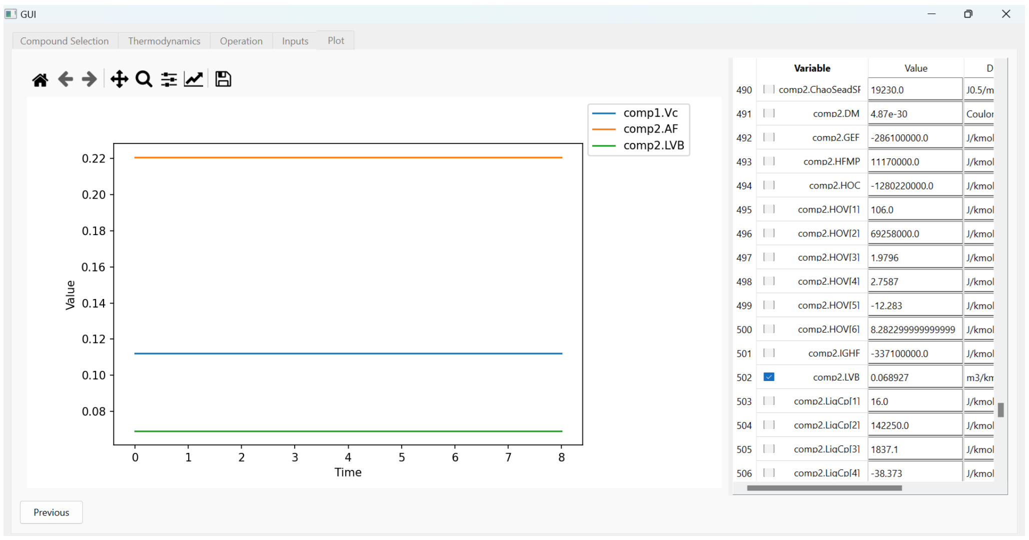


- **Inputs Tab**: This tab allows users to enter necessary input parameters. Start time and stop time are mandatory fields, with validation ensuring that the start time is earlier than the stop time. A dialog box provides feedback on the success or failure of the input submission, ensuring clear communication with the user.



Start Time:	0
Stop Time:	
HB0:	
Ha:	
Hc:	
PB:	
PC:	
QR:	
R:	
N_Trays:	

- **Plot Tab**: This tab features a plot display on the left side, with a menu on the right side for selecting different values and units to be plotted. The plotting functionality utilizes Matplotlib, allowing users to adjust the plot through a legend and various settings. This tab provides a visual representation of data, enhancing the analysis and presentation of simulation results.



		Variable	Value	D
490		comp2.ChaoSeadSF	19230.0	J0.5/m
491		comp2.DM	4.87e-30	Coulor
492		comp2.GEF	-286100000.0	J/kmol
493		comp2.HFMP	11170000.0	J/kmol
494		comp2.HOC	-1280220000.0	J/kmol
495		comp2.HOV[1]	106.0	J/kmol
496		comp2.HOV[2]	69258000.0	J/kmol
497		comp2.HOV[3]	1.9796	J/kmol
498		comp2.HOV[4]	2.7587	J/kmol
499		comp2.HOV[5]	-12.283	J/kmol
500		comp2.HOV[6]	8.282299999999999	J/kmol
501		comp2.IGHF	-337100000.0	J/kmol
502	✓	comp2.LVB	0.068927	m3/km
503		comp2.LiqCp[1]	16.0	J/kmol
504		comp2.LiqCp[2]	142250.0	J/kmol
505		comp2.LiqCp[3]	1837.1	J/kmol
506		comp2.LiqCp[4]	-38.373	J/kmol

2.10 Simulation Outcomes

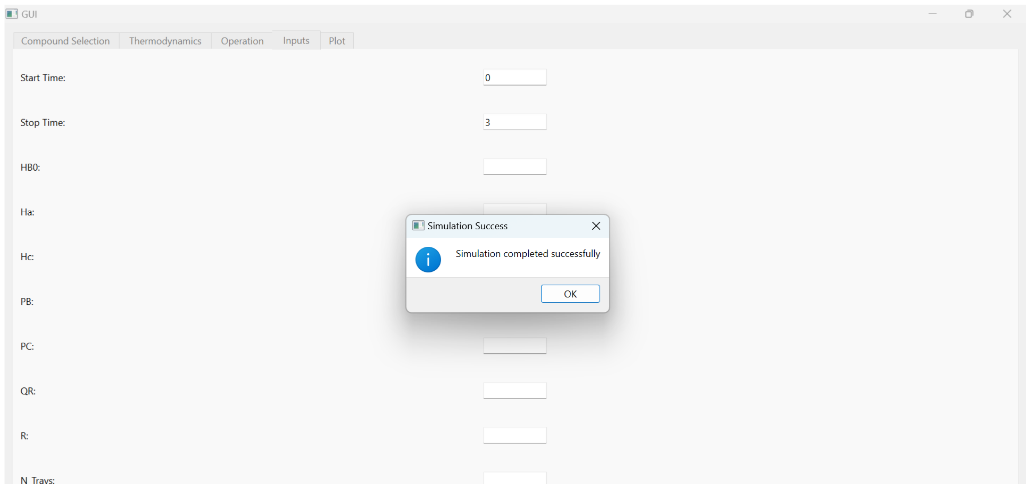
2.10.1 Problem Statement

The simulation process requires accurate inputs and a well-configured setup to ensure successful execution. The system needs to validate the inputs and provide clear feedback in the event of success or failure. This ensures users can troubleshoot issues and make adjustments as needed.

2.10.2 Outcome

The simulation outcomes are categorized into two primary results: success and failure. The system provides distinct feedback in each case:

- **Successful Simulation**:
  - If all the required inputs are correctly provided (e.g., start time, stop time, selected compounds, operations, and thermodynamic properties), the simulation proceeds without errors.
  - A success dialog box is displayed, confirming the simulation has started.
  - The plot updates based on the simulation results, displaying the selected variables with their corresponding units on the plot.
  - Users can interact with the plot, adjusting it using the legend and making further refinements if needed.


GUI

Compound Selection | Thermodynamics | Operation | Inputs | Plot

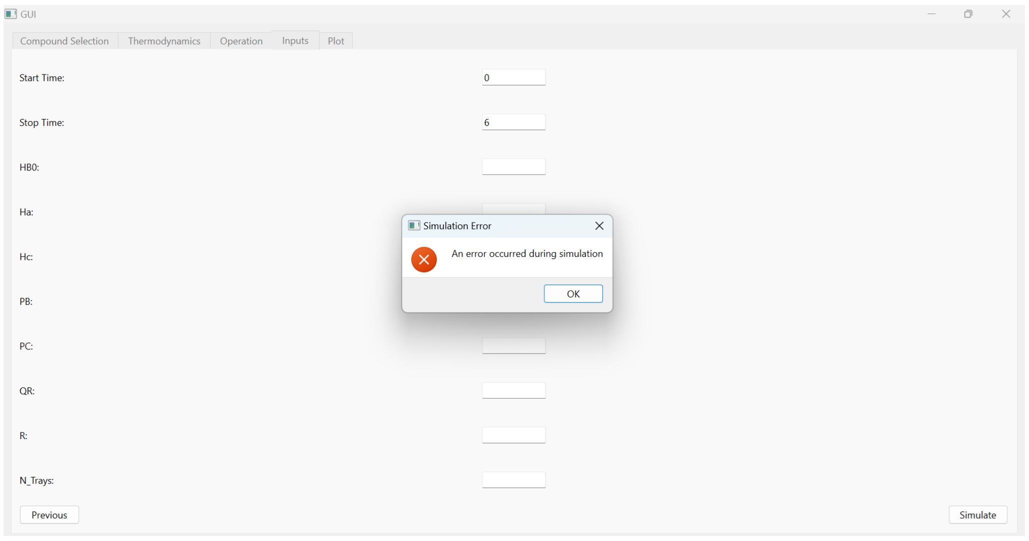
Start Time: 0
Stop Time: 3
HB0:
Ha:
Hc:
PB:
PC:
QR:
R:
N Trays:

Simulation Success

Simulation completed successfully

OK

- **Failed Simulation**:
  - If mandatory inputs are missing, such as start or stop times, or if the start time is not less than the stop time, an error message is displayed, prompting the user to correct the input.
  - In the case of missing selections (e.g., no compounds, operations, or thermodynamic properties selected), the system prevents moving to the next step and shows a warning dialog that informs the user to make the necessary selections.
  - If there are internal system errors during the simulation (e.g., conflicts in thermodynamic properties or unsupported operations), the system generates a failure report detailing the error and advising corrective measures.
  - Users are required to fix the errors and retry the simulation, with feedback provided at each step to guide them through the troubleshooting process.


GUI

Start Time: 0

Stop Time: 6

HB0:

Ha:

Hc:

PB:

PC:

QR:

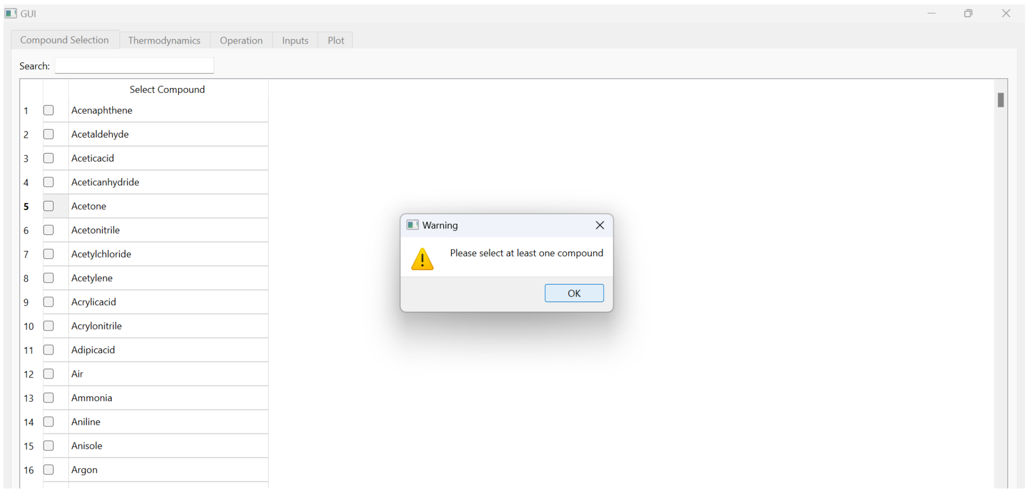
R:

N_Trays:

Simulation Error

An error occurred during simulation

- **Execution Feedback**: Both success and failure messages are designed to give clear feedback to the user. These messages appear in dialog boxes, ensuring that users understand the current state of the simulation and can take the necessary actions.

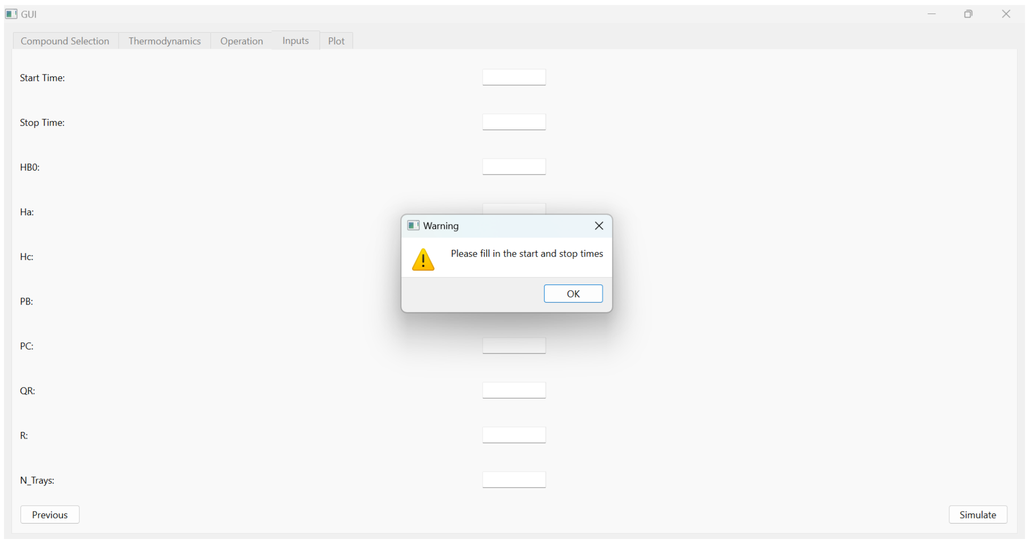

GUI

		Select Compound
1	☐	Acenaphthene
2	☐	Acetaldehyde
3	☐	Aceticacid
4	☐	Aceticanhydride
5	☐	Acetone
6	☐	Acetonitrile
7	☐	Acetylchloride
8	☐	Acetylene
9	☐	Acrylicacid
10	☐	Acrylonitrile
11	☐	Adipicacid
12	☐	Air
13	☐	Ammonia
14	☐	Aniline
15	☐	Anisole
16	☐	Argon

Warning

Please select at least one compound

Here no compound is selected so it displays a dialog box



Please fill in the start and stop times

Warning for no start time and stop time being entered

2.11 Modifying the .mo and .mos Files

2.11.1 Problem Statement

In the context of our project, it is essential to dynamically adjust certain parameters within the Modelica simulation files. This includes modifying the compounds used in the simulation, updating parameter values, and setting the appropriate start and stop times for the simulation. Direct modifications to these files are required to tailor the simulation to specific conditions or requirements. The process involves editing .mo and .mos filesModelica's model definition and simulation scripts, respectively. To achieve this, we utilize dedicated Python scripts that automate these modifications efficiently.

2.11.2 Outcome

The outcome of this process involves several key steps, each handled by a specific script designed to address different aspects of the modification task:

- **modifycompounds.py**: This script focuses on updating the compounds used in the simulation. By modifying the .mo files, this script ensures that the simulation uses the correct compounds based on the latest input . It involves parsing and editing the Modelica files to replace or adjust the compound definitions as needed.

- **modifybatchrectifier.py**: This script is responsible for changing parameter values within the '.mo' files. It updates parameters related to components such as batch rectifiers, ensuring that the simulation reflects the latest parameter configurations. This might involve setting new operational ranges or altering configuration settings specific to the rectifiers.

- **modifystartandstop.py**: This script handles the modification of simulation time settings in the '.mos' files. It adjusts the start and stop times for the simulation to ensure that the run time matches the desired duration.It identifies and replace time settings efficiently, ensuring that the simulation runs within the specified timeframe.

  All the scripts utilizes regular expressions (**regex**) library in order to make the changes.

Chapter 3

Implementation

3.1 First GUI

The implementation of the GUI involved several components, each playing a crucial role in the simulation process and interaction with Modelica. Below is a breakdown of the files and their respective purposes in the project:

3.1.1 Project Files

- **main.py**:
  - The central Python script for managing the GUI's functionality and interactions.
- **Test_init.xml**:
  - A configuration file used to initialize settings and parameters for the GUI.
- **Many bin files**:
  - Binary files required for the GUI's operation, including compiled resources or data files essential for the application.
- **main.exe**:
  - An executable version of the main Python script. It allows users to run the GUI without needing to install Python or other dependencies.

  However in order to run the exe there is need of the dll files.

3.2 Second GUI

The implementation of the GUI involved several components, each playing a crucial role in the simulation process and interaction with Modelica. Below is a breakdown of the files and their respective purposes in the project:

3.2.1 Project Files

- **main.py**:
  - The main Python script responsible for orchestrating the entire GUI application. It handles user input, interactions between different components, and triggers the simulation processes.
- **simulate.mos**:
  - The Modelica script that runs the actual simulation. It is triggered by the GUI once all necessary parameters are configured.
- **modules.zip**:
  - This zip file contains encrypted modules used during the simulation. These files are protected to ensure hiding the modelica code.
- **zip.py**:
  - A Python script that manages the compression and password protection of the Modelica module files. It ensures that the files are securely zipped before distribution and extracted securely when needed.
- **main.exe**:
  - An executable version of the main Python script. It allows users to run the GUI without needing to install Python or other dependencies.
- **Update Modelica Files (Folder)**:
  - This folder contains multiple Python scripts that modify different aspects of the Modelica files:
    - **modifybatchrectifier.py**: Modifies the parameters of the batch rectifier in the Modelica model.
    - **modifycompounds.py**: Handles updates to the compounds being used in the simulation.
    - **modifystartandstop.py**: Edits the start and stop time for the simulation, ensuring valid time ranges.
    - **modifytemp.py**: The ‘tempdir‘ parameter specifies the new directory path that replaces the existing path in the Modelica script. This ensures that the script uses the correct temporary directory during execution.

In order to run the main.exe we will require the modules.zip along with the simulate.mos file.

References

1 https://openmodelica.org/doc/OpenModelicaUsersGuide/latest/index.html

2 https://www.pythonguis.com/pyqt6-tutorial/

3 https://stackoverflow.com/

4 https://fossee.in/

5 https://om.fossee.in/

6 https://spoken-tutorial.org/tutorial-search/?search_foss=OpenModelica&search_language=En