Implementation Flowchart-Module for OCTANE

Tobias Bleymehl Tobias Bodmer Diana Burkart
Matthias Lüthy Sebastian Weber
Jonathan Schenkenberger

October 21, 2020

Contents

1 Introduction

This document shows the Implementation-Phase of our Flowchart-Module for the Open-Source simulation environment OCTANE. The document focuses on the overall structure and execution of the implementation of our product and the changes to the design. For the changes themselves we examine why they were made and what impact they have to the structure and functionality of the Flowchart-Module. Overall the document only gives a brief overview since doxygen provides a detailed documentation of our code.

2 Planning of the Implementation

Before starting implementing the Flowchart-Module we first planned it, to be able to execute it as efficient as possible. The result was an implementation plan showing the deadlines of each part of the project.

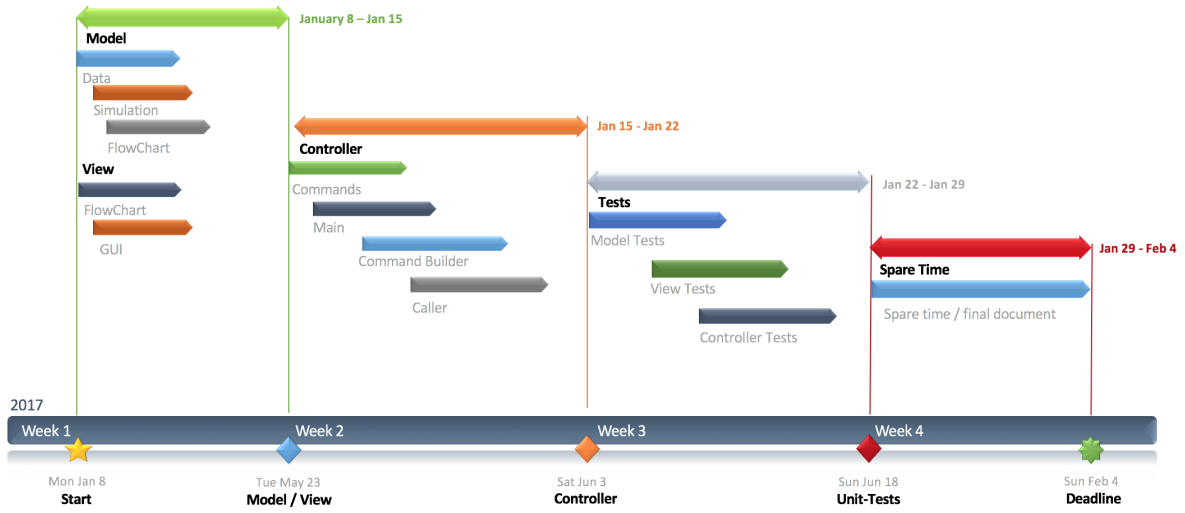

Figure 1: Implementation Plan - The figure shows our plan for the Implementation-Phase

Planning and actually doing is quite different. We realized that quite quickly facing many problems in the first stages of the implementation. Especially the first deadlines were missed because of occurring problems with building libraries and IDEs. The problems themselves and how we solved them are further explained in the corresponding sections.

3 Progression of the Implementation

This section describes the progression of the Implementation-Phase. Each part is split into two showing our goal and perspective of the problem and the challenges and difficulties we faced and how we solved them.

3.1 Libraries

Since the program is supposed to run on both Linux and Windows we used some cross-platform libraries. To be able to parse XML documents we used the parser library Xerces, which in contrast to other parsers also provides validation. For building the graphical user interface the library wxWidgets was quite suitable since it is open-source and provides vast functionality. The last library we used is Poco. It gives us possibility to dynamically load DLLs and also made us change our Warning System since it provides many ways for generating logs and notifying the user.

Building the libraries was much harder than we expected since the used compilers and compiler options have to match during the build itself and the project build. On Linux the builds were quite easy without any major problems. On Windows and Mac on the other hand, the libraries were not building. After a couple of tries the libraries first ran on Windows and eventually, after rewriting part of the libraries, on Mac. Includes and Links were in general a harder problem than we expected since it differs quite a bit from Java.

3.2 Model

The Model was our first target during the implementation. Everything builds on top of the Model and so we implemented this first. The Model itself is split into the DataLib and the Flowchart, providing the data, modules and other elements in the flowchart, are passing between themselves, as well as the abstraction of the flowchart and all its elements. The flowchart also contains the Simulation package which is an effective implementation of simulation modules in various ways. Additionally the Model includes the GraphicalRepresentation which describes how each element of the flowchart is viewed and presented in the flowchart. One can clearly see how our model is designed to be expandable. It is structured clearly and easy to understand and work with.

During the implementation of the Model we faced many problems. First of all we had to change our own design idea since there were better alternatives arising. These changes are explained later in the document. The simulation did not have to many problems since it does not depend on many other classes therefore being easy to implement and test. The flowchart abstraction was also quite forgiving since its main purpose and therefore its implementation are easy and straight forward. DataLib faced more problems being quite complex in functionality. Overall

there was nothing to challenging but rather time taking since the classes have quite some lines of code. One of the greater challenges we faced were in the GraphicalRepresentation. One of the main problems was that we started to late recognizing that many things depend on the GraphicalRepresentation and therefore other parts of the project had to wait for it to get done. All in all we implemented a functional and easy to use Model which fully supports the functionality we wished for. Unfortunately many features in the Model cannot yet be used since the Graphical User Interface does not support all operations yet and therefore we would recommend you warmly to expand our Graphical User Interface to use that functionality.

3.3 View

While the Model was being programmed on the one side of the project we concurrently started implementing the Graphical User Interface on the other side of the project. Our goal here was to have a fully functional user interface which would only have to be connected to the Controller and therefore to the Model. The View provides a rather simple user interface to be able to manipulate the flowchart and create own flowcharts. The user is able to start simulations and connect modules from the interface to simulate his own modules and flowcharts.

During the implementation of the View the main problems were that for creating functionality the necessary Controller and Model part of the project were not implemented yet. This slowed down the creation of the Graphical User Interface as well as the fact that we did not know anything about wxWidgets yet. We had to learn how to create such an interface and more importantly how to interact with it the way we want to. Overall we created an graphical user interface which supported the basic creation of flowcharts and their simulation yet is still to be expanded since for many things that we would like to have implemented we just did not have enough time.

3.4 Controller

The Controller part of the program was planned as one of the last parts during the implementation since is highly depends on functionality of the View and the Model and forges them together to build one whole program. The Controller is supposed to support all actions where the GUI interacts with the model. It encapsulates the functionality of passing parameters and calling the correct method of the Model in one interface quite well.

The biggest problems during the Controller's implementation were missing methods in the Model and that we started implementing the Controller way to late. The View and GUI were drastically slowed down by not being able to communicate their actions to the non-existing Controller. In the later stages of the progress of implementing we then finished the Controller which now works quite well. It now nicely fits inside our program doing the communication work between View and Model which was further strengthened by the fact that many command we thought necessary in the design were reduced to be handled by the GUI itself.

4 Design Changes

This section is the key section of this document showing the changes we made to the design of the Flowchart-Module and explaining why we made them. We do not guarantee this section to be complete since many changes where minor changes and are therefore not mentioned here.

4.1 Controller

The Controller has been changed to the favour of comprehensibility and usability.

class	attribute/method	change	reason
CommandBuilder		removed	1
Caller		runs on own thread, owned by the tabs	2

(1) The CommandBuilder class is not needed anymore since commands are built in the GUI

(2) The Caller runs on an own thread now. It has the functionality of executing the commands committed by the GUI. This is implemented with a producer-consumer architectural pattern which nicely separates the GUI thread from the Controller thread which changes the model.

The names of the commands have been changed for reasons of comprehensibility. Many commands are not implemented since they were only combinations of the other commands.

4.2 View

The changes in the View were mainly caused by problems we had with wxWidgets and our observer pattern connecting View and Model.

4.2.1 FlowChart

class	attribute/method	change	reason
GraphicalRepresentationView and all subclasses	method delete	removed	1
GraphicalRepresentationView and all subclasses	method draw	wxDC parameter added	2
ConnectionLine		removed	3
CornerPointView		inherits from GraphicalRepresentationView	4
LabelView		removed	5
StdInputPinView		removed	6
StdOutputPinView		removed	6
RigidPinView		removed	6

(1) The method is not needed anymore.

(2) A parameter of the type wxDC is added to the method, because the drawing in wxWidget is done on a device context. Drawing directly to a window is not possible.

(3) The class is not needed anymore.

(4) The class inherits from the GraphicalRepresentationView class, because it is a view of a graphical representation element.

(5) The functional requirement (F170) is not implemented, so this class is not needed anymore.

(6) The classes StdInputPinView, StdOutputPinView and RigidPinView are not needed anymore. There is no reason to divide the various ModulePinView, because there are no differences between the subclasses. So they are combined in the class ModulePinView.

4.2.2 Graphical User Interface

The Graphical User Interface was changed quite a bit because much functionality was not implemented and with wxWidgets it was easier to implement some visual parts than others. The changes are quite visible by comparing the GUI with the GUI from the planning phase. Since they are not functional we will not further examine these changes here.

4.3 Model

This subsection shows the changes we made to the Model.

4.3.1 Model Facade

The Model Facade is the interface to access all functionality of the Model which explains why it changed quite a bit comparing to our design.

class	attribute/method	change	reason
ModelFacade	attribute m_lastFilePath with getter and setter	added	1
ModelFacade	methods stopSimulation, pauseSimulation, stepForward	added	2
ModelFacade	method stepForward	added	3
ModelFacade	method connectWithStdArrow	renamed to addStdArrow	4
ModelFacade	method connectWithRigidConnection	renamed to addRigidConnection	4
ModelFacade	method validateFlowChart	not implemented	5
ModelFacade	methods initializeStoreLoad, initializeFlowChart	removed	6

(1) The attribute is added with an getter and a setter method to save the last file path.

(2) The methods are used to stop or pause the simulation.

(3) The method is used to step forward in the simulation.

(4) The methods are renamed for reasons of comprehensibility.

(5) The method is not yet implemented. The overall structure of the validation is implemented but we did not have the time to implement this detail.

(6) The methods are removed, because the functionality was taken over by the constructor of the ModelFacade class.

4.3.2 FlowChart

The FlowChart subsection shows the changes to the FlowChart part of our project which represents the abstraction of each flowchart.

class	attribute/method	change	reason
FlowChartElement	attribute activeWarnings methods reportWarning, warningResolved	removed	1
FlowChartElement	methods serializeInternalConfig, restoreSerializedInternalConfig, serializeSimulationState, restoreSerializedSimulationState	removed	2
FlowChartElement	method setGraphicalRepresentation	added	3

(1) Warnings are logged through the Poco library.

(2) The functional requirement F110 is not implemented and F120 is changed to not module internally undo and redo functionality.

(3) A setter for the GraphicalRepresentation of the FlowChartElements is needed since flowchart elements can be assigned a graphical representation which this setter makes possible.

4.3.3 Graphical Representation

class	attribute/method	change	reason
FlowChartGR		removed	1
GraphicalRepresentation	attribute flowChartGR	removed	1
StdInputPinGR		removed	2
StdOutputPinGR		removed	2
RigidPinGR		removed	2
PinOrientation enum		renamed OrientationType	3
ModuleGR	attributes ENUM_LENGTH, ROTATE_LEFT, ROTATE_RIGHT	added	4
ModuleGR	getter and setter methods for attributes m_inputPinOrientation, m_outputPinOrientation	added, instead of one setter method for both attributes	5
ModuleGR	attribute m_orientation with getter	added	6
ModuleGR	attribute hasCustomIcon with getter and setter	added	7
ModuleGR	methods getInfoText, getInternalConfig	moved to Module class	8
ModuleGR, ModulePinGR	attributes defaultIcon, selectedIcon	type changed	9
CornerPointGR		inherits from GraphicalRepresentation	10
ConnectionLineGR		inherits from GraphicalRepresentation	10
ConnectionLineGR	getter methods for positionA and positionB	added	11
ConnectionGR	attribute m_lines	added here instead of in the subclasses	12
ConnectionGR	attribute vector of bool m_isToPinLines with method getToPinConnectionLines	added	13
ConnectionGR	attribute diversionPoints	added	14
ConnectionGR	method getInfo	removed	15
ConnectionGR	method moveLine	added here instead of in the subclasses (not abstract)	16
ConnectionGR	methods addLine, removeLine	added here instead of in the subclasses	16
ConnectionGR	methods addLines and removeAllLines	added	17
ConnectionGR	method calculateAllRoutes	added here instead of in the subclasses, renamed the former calculateRoute (not abstract)	18
ConnectionGR	attribute m_color with getter and setter	added	19
StdArrowGR	attributes defaultArrow, selectedArrow	removed	19
LabelGR		removed	20

(1) The class FlowChartGR is not needed anymore, because the FlowChart class holds the relevant information. Therefore the attribute flowChartGR of the GraphicalRepresentation class is also removed.

(2) The classes StdInputPinGR, StdOutputPinGR and RigidPinGR are not needed anymore. There is no reason to divide the various ModulePinGR, because there are no differences between the subclasses. So they are combined in the class ModulePinGR.

(3) The Enum PinOrientation is renamed to OrientationType, because it is now also used as the orientation of the Module.

(4) Constant integers are added as attributes of ModuleGR to make it easier to make changes to the orientation of the Module. To set more possibilities of orientations only the ENUM and the ENUM_LENGTH need to be adjusted. If someone wants the Module to turn halfway only the attribute ROTATE_LEFT and ROTATE_RIGTH need to be set to (-2) and (+2) for example.

(5) The methods are added to so that the input and output pin orientation can be set individually.

(6) The attribute holds the information about the current rotation of the Module.

(7) The attribute holds the information whether an icon has already been set by the user or not.

(8) The methods getInfoText and getInternalConfig are moved to the Module class, because this does not belong to the graphical representation of the module.

(9) The type of the attributes defaultIcon and selectedIcon changed from wxImage to wxBitmap.

(10) The classes are graphical representations of specific parts of the View.

(11) The ConnectionGR class needs to access the attributes positionA and positionB.

(12) Both subclasses need to hold a vector of ConnectionLineGRs.

(13) Saves whether the ConnectionLineGR has a connection to a toPin of a module or not. To allow, for example, to put an arrow or other symbol at the end of a connection.

(14) To save where the user wants the Connection to branch off.

(15) This does not belong to the graphical representation of the module.

(16) Both subclasses need this methods.

(17) The addLines method adds several given ConnectionLineGRs at once to the ConnectionGR. The removeAllLines method removes all lines from the ConnectionGR.

(18) The method calculateAllRoutes isnot abstract anymore in ConnectionGR and is renamed from the proper calculateRoutes method, because both subclasses need the same functionality of this method.

(19) The attributes defaultArrow and selectedArrow are removed and the functionality is realized through a new attribute m_color.

(20) The functional requirement (F170) is not implemented, so this class isn't needed anymore.

4.3.4 Simulation

class	attribute/method	change	reason
Simulation	method finish	added	1
Simulation	attribute stepperActivated	moved to SchedulingStepper	2
Simulation	methods activateStepper, deactivateStepper	removed	3
Simulation	attributes m_isPaused, m_isStopped	added	4
Simulation	attribute m_simulationThread	added	5
Simulation	attribute m_flowChart	added	6
Stepper	abstract method stepForward with no parameters	added	7
Stepper	abstract method pause	added	8
Stepper	abstract method reset	added	9
SchedulingStepper	attribute m_modules	added	10
SchedulingStepper	attribute m_frequencyInverted with getter	added	11
SchedulingStepper	attribute m_flowChart with setter	added	12
SchedulingStepper	protected method step	added	13
SchedulingStepper	attribute m_timing	added	14
SchedulingStepper	attribute m_clock	added	15
SchedulingStepper	method pause	added	8
SchedulingStepper	method reset	added	9
SchedulingStepper	method stepFoward	added	7

(1) The method lets the simulation calculate until it is done.

(2) Only in the SchedulerStepper class needed.

(3) The method is not needed anymore.

(4) The attributes are added to save if the simulation is not running.

(5) The attribute holds the thread on which the simulation is running.

(6) The attribute holds the FlowChart on which the simulation is running. The attribute is set by the constructor.

(7) The method stepForward with no parameters steps until the user stops the simulation and therefore also the stepper.

(8) The method pauses the stepper, but do not reset the clock.

(9) The method resets the clock and stops the execution of the stepper.

(10) The attribute holds the modules that the SchedulingStepper should step through.

(11) The attribute holds the frequency of the SchedulingStepper.

(12) The attribute holds the FlowChart on which the stepper is stepping. The setter method sets the FlowChart of the SchedulingStepper.

(13) The method performs exactly one step. It is running the modules that have to run in this step in the right order.

(14) The attribute counts the internal virtual time of the simulation. On every step the time one step is running virtually is added to the attribute. So it contains the complete virtual time the simulation is running.

(15) The attribute keeps the clock in which the simulation is. Every step of the simulation counts the clock up by one.

4.3.5 Store and Load

The DOM class is realized through a parent class with different child classes. The DOM and its derived classes save persistent data of an object of the flowchart. Since every object needs different data to be restored, the DOM is implemented with inheritance instead of templates.
Restoring a saved flowchart is not implemented.The dynamic module loader returns a module instance and a module parser dom, which can be used to get language specific data. New module instances were created by a clone method that has to be implemented in every derived module class.
The dynamic loading of modules works on windows but not on linux/mac. A module loaded dynamically cannot be connected yet, because loading dynamically requires specific data handling due to the different storage behaviour of a executable and a dll.

4.3.6 Data types

class	attribute/method	change	reason
BasicCheckable		inherits of Data	1
RecursiveCheckable		inherits of Data and abstract class instead of an interface	1
Data	method getUcid	method outsourced	2
Data and subclasses	method toString	added	3
OneSizeDataSignature		added	4
Data and subclasses	method isCheckingRecursive	added	5
UCID		renamed DataIDsEnum, set as enum	6

(1) The classes inherits of the Data class to avoid code duplication. It also allows and simplifies necessary casts.

(2) The method is outsourced in the DataDetail and the DataSignatureDetail classes in a sub-namespace detail. The C++ protected keyword does not mean at the same time package-private, so the detail namespace is created to realize to make the method package-private.

(3) The method creates and returns a string representation of the data.

(4) The class is added as a private class in SizesDataSignature. It makes it possible to deactivate one separate dimension.

(5) The method determines whether this data signature is checking recursively or not.

(6) The DataIDsEnum simplifies the implementation and avoids code duplication.

4.3.7 Units

The Unit classes were provided by OCTANE and included in this project. No changes were made to this part of the project.

4.3.8 Function module

class	attribute/method	change	reason
Primary		added	1
Number		inherits of Primary	2
Variable		inherits of Primary	2
Negate		added	3
FunctParser		written by ourselves	4
Scanner		added	5
FunctInternalConfig	attribute m_configUsingInt	added	6

(1) The class inherits of the FormulaElement class.

(2) The Number class and the Variable class now inherits of the added Primary class.

(3) The class is added and inherits of the UnaryOperation class. It makes it easier for the user to negate components of the function.

(4) The class is written by ourselves, because the parser has to provide an additional functionality in order to be able to include units.

(5) The class is added to divide a string into atomic parts for the FunctParser class.

(6) The attribute says whether to calculate with int or double. If the boolean is true it calculates with integer, otherwise it calculates with double.

FormulaParser

Before the implementation phase, this parser was thought to be implemented by a third party library. Unfortunately there were no libraries available where you can scan basic expressions and treat the expressions as a composite of formulas, thus as objects. This lead to a problem, because we needed to assign not only numbers and variables but also units to each of them. Fortunately the unit system was provided by Octane. So it was an obvious choice to implement a new parser and scanner, using the already existing system to fit better into octane and the requirements specified.

Overall structure
The parser consists of a Scanner, Parser and a FormulaElement structure. The FormulaElement composite is representing the structure of a formula, by splitting each element into a object which can hold FormulaElement structures for itself.

Scanner
The scanners job is to take a input string and split it into the most atomic part that the scanner can make sense out of it. This atomic element of a formula is also called a Token. It holds basic informations like what it represents and a value that can be interpreted differently, depending on the token type. For example the scanner receives a '+' and interprets a PLUS-token out of it. On the other hand the scanner also handles recognizing different number types and stores the recognized number in the correct Token type. Currently the scanner supports interpreting int and double values.
A nice fact about the scanner is also, that if the user inputs false characters or messes with not yet implemented mathematical strings (e.g. $sin(..)$) then the scanner can step in and refuse the input. This is a big relief for the parser, which contains the most logic and would be otherwise polluted with error handling.

FormulaElement
The FormulaElement class hierarchy is a composite structure which enables to store any mathematical formula as a composite of calculation operators. Each calculating operator has its own

class which can then hold the rest of the formula beneath itself. This also means that the order of calculation is specified by the composite structure itself. - No brackets needed. The Structure brings also some (partially not yet existent) functionality with it. There is a method to check the mathematical validity of the formula structure. It checks whether there are units in potencies or the user tries to add seconds with meters and so on. Unfortunately there was not enough time left to sufficiently develop and test this feature, so that is now left half-implemented. To be able to continue our work, we will briefly describe the idea behind this methods implementation:
When check() is called, it should propagate down to each child element to be checked as well. For each check() call you can specify what the specific child elements must or must not do. So you can forbid every type of unit (e.g. in potencies), or specify the unit signature the child must calculate. (compatibility with '+' and '-' operations)
Of course it is also possible to give the formula structure some input and it calculates the result. This of course works on every composite element, if you wish to do so.

Parser
The parsers job is to take a list of tokens and create a equivalent composite structure of FormulaElemtents from them. This is achieved by defining a chompsky-type-2 grammar (aka. context free grammar) which holds every legal structure of formula configurations. Generally we decided to use a recursive parser structure, even though it limits in terms of grammar acceptability. So is important that the grammar has the right form, to be able to traverse it recursively. You can find the current grammar in a separate text file with the parser or you can look it up in the doxygen documentation. A big advantage of this parsing structure is that it is very easy to add in new behaviors or even new formula elements. You would not have to change much code to insert a $sin()$ operator into the parser and have it recognize properly and even to execute in the right way. Each of these context free grammar rules has their own method which is called whenever this specific rule can be applied.

A word of warning
The whole implementation of these described structures have originally not been our task. But they have proved necessary to achieve the asked specifications. We have to demonstrate the vast functionality somehow so we implemented this module to do so. The parser and its components are still in a work-in-progress state, and will remain this way. The structure is rarely tested and also highly unstable when called. Each wrong syntax input throws invalid_argument exceptions. Be also sure to never calculate a formula, that has not been validated properly. Because otherwise the UnitD class of oct::unit will handle illegal math operations, which will result in a assertion failure.

4.3.9 Warning System

The planned Warning System as a whole was removed due to the fact that the POCO library provides vast functionality to log warnings. The whole system was unfortunately not implemented since we did not have enough time to do that but would appreciate it if an ambitious reader would do so. (Visualising would be great.)

5 Fulfilment of the requirement specifications

The following pages contain all functional requirements from the requirement specification presenting which features were implemented and which we did not implement. We would be glad if somebody would find our project worth to further implement all our features we wished to implement. The section shows all parts of the project were parts were not implemented. Not implemented parts are coloured red.

5.1 Module instances and modules

F10 Inserting module instances in the flow graph
F11 Selecting module instances in the flow graph
F12 Deselecting module instances in the flow graph
F13 Removing module instances in the flow graph
F14 Moving module instances in the flow graph
F15 Turning module instances in the flow graph
F16 Scaling module instances in the flow graph
F17a Adjust inputs and outputs
F18a Providing a function module
(W)F60 Providing a derivation module
(W)F90 Encapsulate a flow chart in a module
(W)F65 Providing a plotter module

5.2 Connections

F20 Connecting module instances
F21 Automatical routing of connections
F22 Adjusting the connection when rotating or moving a module instance
F23 Displaying data types on the connections
F24 Checking the data types on creation of connections
F25 Providing rigid connections
F26 Providing arrows as a connection type
F27 Splitting arrows between two module instances (not accessible from the GUI)
(W)F70 Connecting module instances by dragging
(W)F71 Routing-algorithm for better routes
(W)F72 Filtering connection types

5.3 Simulation

F30 Perform simulation cycle
F31 Start simulation
F32 Pause the simulation
F33 Reset simulation
(W)F110 Jumping back to previous states of the simulation

5.4 Store and Load

F40 “Save” a flow chart
F41 “Save as” for a flow chart
F42 Loading a flow graph
F43 Adding a module to the module list
F44 Updating the module list
(W)F80 Importing a flow graph
(W)F81 Exporting a flow graph
(W)F100 Displaying a preview picture of the flow graph

5.5 Others

F50 Info-tab of the module sidebar
F51 Representation-tab of the module sidebar
F52 Module list sidebar
(W)F120 Undo and redo (not module internally)
(W)F130 Search function in the module list sidebar
(W)F131 Search function for the flow graph
(W)F140 Move the view
(W)F141 Switch grid on and off
(W)F150 Warning button/ warning window
(W)F170 Label elements

6 Conclusion

Overall the implementation of our project did not go quite as planned and as we hoped it would. We had countless problems and every time we solved one of them two new ones occurred. In the end we had to postpone the deadline of the phase by two weeks. Looking back we thought whether we had planned the implementation correctly and executed it well enough despite all the missing features of the project. Today we know that we could not have done more that we have. We surely planned far too many features, but we worked every day of the last six weeks hours after hours to get the project to where it is now. We have created a strong project which is highly expandable. We look forward to the testing phase to get some of our yet implemented but still not working features done to provide a project which surely is not done by any means but can be easily brought to a state which makes it possible to use it in various ways.