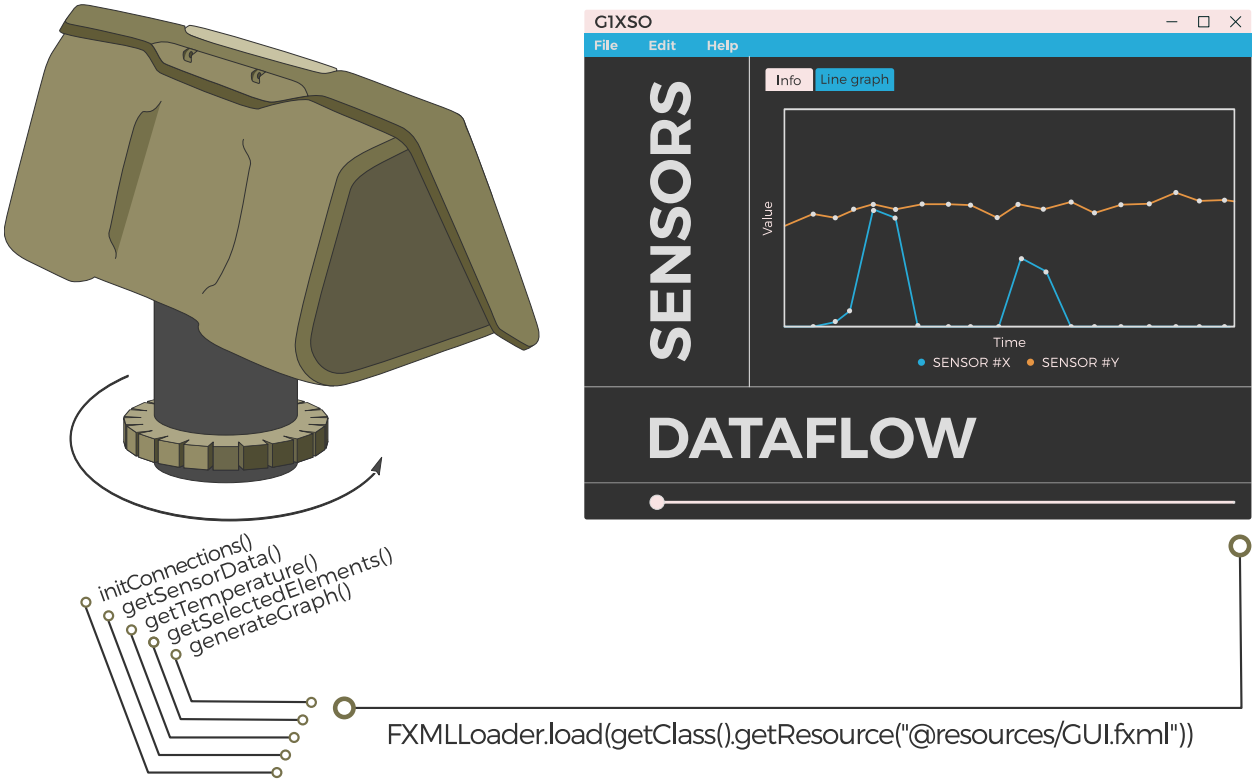
CHALMERS
UNIVERSITY OF TECHNOLOGY



G1XSO - a user interface for system overview

Visualizing data for the Giraffe 1X radar

Bachelor's Thesis in Computer Science and Engineering

FAWZI AIBOUD NYGREN
FILIP REUTERBERG

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
GOTHENBURG UNIVERSITY
Gothenburg, Sweden 2021
www.chalmers.se

Bachelor's Thesis 2021

G1XSO - a user interface for system overview

Visualizing data for the Giraffe 1X radar

FAWZI AIBOUD NYGREN
FILIP REUTERBERG

CHALMERS

Department of Computer Science and Engineering
Division of Computer Engineering
Chalmers University of Technology
Gothenburg, Sweden 2021

G1XSO - a user interface for system overview
Visualizing data for the Giraffe 1X radar
FAWZI AIBOUD NYGREN
FILIP REUTERBERG

Supervisor company: Lena Ernholm, SAAB
Supervisor: Sakib Sistek, Department of Computer Science and Engineering
Examiner: Peter Lundin, Department of Computer Science and Engineering

Bachelor's Thesis 2021
Department of Computer Science and Engineering
Division of Computer Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: The G1X and code snippets, transformed into a GUI

Gothenburg, Sweden 2021

Sammandrag

Ingenjörerna som arbetar med Giraffe 1X, en multifunktionell markradar byggd av Saab, använder idag ett flertal olika applikationer för att utföra verifiering och integrering. Processen är både komplex och tidskrävande. Projektets syfte har varit att skapa Giraffe 1X System Overview (G1XSO), en fristående applikation som tillhandahåller dessa applikationers mest kritiska funktionalitet. För att kunna distribuera mjukvaran i Saabs laboratorier, och återanvända befintlig kod, skrevs applikationen i Java 8. Den använder sig av det inbyggda biblioteket JavaFX och dess front end använder FXML. All kod skrevs i utvecklarmiljön Eclipse och FXML-filer genererades med mjukvaran Scene Builder. Resultatet är en applikation som kan läsa logg-filer från de tidigare nämnda applikationerna samt strömma realtidsdata från Giraffe 1X. I båda fallen samlas data in och sparas i flerdimensionella *Maps* och listor i applikationens databas. Användare presenteras med ett användargränssnitt som visar en översikt av alla sensorer som genererar data. Om applikationen strömmar data kan användaren välja vilka sensorers data som ska tas in. Om applikationen läser från fil öppnas istället operativsystemets utforskare. Användare kan välja vilka sensorer som ska visas och generera olika typer av grafer för dessa. Grafer kan lossas från huvudapplikationen för att visas sida vid sida, har stöd för zoom och uppdateras i realtid ifall applikationen strömmar data.

Keywords: grafiskt användargränssnitt, gui, radar, giraffe 1x, Saab, användarupplevelse, visualisering.

Abstract

The engineers working on Giraffe 1X, a multi-mission radar built by Saab, are today using multiple different software tools in order to perform verification and integration. This process is both complex and time-consuming. The purpose of this project has been to create Giraffe 1X System Overview (G1XSO) a stand-alone application that gathers the critical functionality of said software and merge it. In order to run on the hardware in the laboratories of Saab, and to re-utilize already existing code in existing projects, the application was written in Java 8. It makes use of the built-in library JavaFX and its front end uses FXML. All of the code was written with the integrated development environment Eclipse and FXML files were generated with the software Scene Builder. The result is an application that can read recorded log files from the previously mentioned applications, as well as real-time streamed data from the Giraffe 1X. In both cases, it collects the gathered data and saves it in multi-dimensional maps and lists within the application's database. The user is presented with a graphical user interface, displaying a system overview of all the sensors currently generating data. If streaming, users may select which sensors to collect data from. If reading from a log file, a file explorer is opened instead. The user may select sensors and generate different types of graphs. Graphs have zoom functionality and update in real time if streaming, and may be undocked from the main application to be viewed side by side.

Keywords: graphical user interface, gui, radar, giraffe 1x, Saab, user experience, visualization.

Acknowledgements

We would like to thank Lena Ernholm, Magnus Billström, Mikael Molin and Micael Andersson for taking time out of their busy days to aid and guide us in this process and showing us what Saab is about. We would also like to thank Sakib Sistek for pointing us in the right direction when writing our thesis. A special thanks goes out to Maria Köhler, for giving us the chance to write our Bachelor's Thesis for surface radar at Saab. Lastly, a big thank you to Victor Wallsten for making sure we could deliver a thesis written in English of the highest quality possible.

Fawzi Aiboud Nygren & Filip Reuterberg, Gothenburg, May 2021

Glossary

actionlistener in Java, a class that is responsible for handling all action events such as when the user clicks on a component. 3

back end belongs to the data access layer, usually a database/model and logic. 4, 9, 11, 24

cohesion a class with high cohesion is a class designed with a single, well-focused purpose. Refers back to the Single-responsibility Principle. 4

color wheel an abstract illustrative organization of color hues around a circle, which shows the relationships between primary colors, secondary colors and tertiary colors. 7, 23

concurrency the ability of different parts or units of a program, algorithm, or problem to be executed out-of-order or at the same time. 22

coupling refers to the degree of direct knowledge that one element has of another. Loose coupling means classes have little to no dependency on each other. 3, 10

dependency injection a technique in which an object receives other objects that it depends on. 27

front end belongs to the presentation layer, usually a user interface. v, vii, 4, 9, 10

observable an observable is an object which notifies observers about the changes in its state. 3

observer observer is a behavioral design pattern. It specifies communication between objects: observable and observers. 3, 4

proxy a server application or appliance that acts as an intermediary for requests from clients seeking resources from servers. 22

readme a text file that introduces and explains a project. 19

repositories tracks all changes made to files in your project, building a history over time. Often a cloud based repository such as github or gitbucket. 11, 22, 27

semaphore a synchronization tool used to control access to common resources by multiple processes. 5, 22

singleton a class that can have only one object (an instance of the class) at a time. 5, 10, 11, 27

user experience the process design teams use to create products that provide meaningful and relevant experiences to users. 7

version control also known as source control, is the practice of tracking and managing changes to software code. 10

Acronyms

CSS Cascading Style Sheets. 6, 9, 10, 14

FX Java FX. 5, 6, 9, 10, 22
FXML FX Markup Language. v, vii, 6, 11, 22

G1X Giraffe 1X. 1, 2, 11, 13–15, 27
GUI Graphical User Interface. 2, 3, 5–7, 9–13, 15, 22, 27

HTML HyperText Markup Language. 6

IDE Integrated Development Environment. 6, 10

MVC Model-View-Controller. 3, 4, 10, 11, 27

SB Scene Builder. 6, 9–11
SSH Secure Shell. 22

UX User Experience. 7

XML Extensible Markup Language. 5, 6

List of Figures

Contents

1
Introduction

This chapter will help the reader understand how this paper is structured and explain the underlying motivation for the thesis. Additionally, it presents the aim, purpose, question at issue and delimitations of the thesis.

1.1 Background and motivation

Saab's Giraffe 1X (G1X) is a multi-mission lightweight radar that has been developed for air defense and surveillance for both military and civil applications. The radar is equipped with several sensors within its hull that measures temperature, voltage, current and fan speed to name a few.

One of the difficulties with the systems used to retrieve data today is that the information is spread out across multiple applications. Radar data from the G1X is currently analyzed through a myriad of software applications and the information from sensors is often printed to a terminal window as plain text. This forces users to export data to other applications for interpretation and evaluation. Furthermore, it makes it difficult for developers to get an overview of the system status in one place. Needing to manually export data is time-consuming and not necessarily a straight-forward process.

There exists a need for an application that merge what the current systems show into a standalone application. It would eliminate many of the tedious steps taken in the current system and facilitate the workflow of the engineers that require this information in their day-to-day work.

1.2 Purpose

The aim is to create a user friendly application that displays an overview of the system's status and visualizes data retrieved from the G1X radar in a simple, yet informative way. Read data will be displayed as graphs, and the type of graph will vary depending on the type of data users want to visualize.

The purpose of the project is to develop an application that merges what the current separate systems show, in order to visualize retrieved data in a standalone application.

1.3 Scope

The end users will be developer- and test engineers at Saab's surface radar division. The application will aid the engineers and developers in identifying the origin of faults such as loss of connection and unexpected shutdowns. Focus lies on creating a well-structured solution that is modular and has low coupling for easy future implementation of new data types or changing the Graphical User Interface (GUI). Furthermore, the application should be user friendly, long-lived and easy to maintain. It should be possible to make minor customizations to the interface while the application is running in order for each end user to be able to view and prioritize what he/she seeks.

The main objectives of this project is to create an application that:

- Has a user-friendly and intuitive GUI
- Contains 10 data types as a foundation
- Shows a complete system overview of G1X's sensors
- Can stream live radar data and read from log files
- Has easily maintainable code by good usage of implementation standards

1.4 Delimitations

The number of supported data types will be limited to 10, as requested by the team of engineers at Saab. This will make the application modular and useful from the start. There will be no modifications to the hardware pertaining to the radar. Thus, the project will solely focus on software development.

The application will help engineers in visualizing the received data in the format requested by the engineer. However, it will not be used to identify where the source of faults are located, nor how to solve any issues. This is left to the engineers.

1.5 Report structure

The report is divided into five parts: Theory, Methods and Implementation, Results, Discussion and Conclusion.

The Theory chapter will introduce the reader to concepts such as software patterns, software and user experience to facilitate reading in the coming chapters. The Methods and Implementation chapter explains the thought and research process during the project as well as how the findings were implemented. The Results chapter presents the final results and tie back to the aims set at the start of the project. The Discussion chapter treats subjects such as difficulties encountered, design choices and ethics. It ties back to the purpose of the project. The last chapter, Conclusion, gives suggestions for further development of the application, looks into if the approach taken should have been a different one and summarizes what has been achieved.

2
Theory

This chapter will introduce the reader to concepts such as software patterns, programming languages and user experience to facilitate reading in the coming chapters.

2.1 Design patterns

This section explains two common software design patterns.

2.1.1 The Model-View-Controller

The Model-View-Controller (MVC) pattern is one of the oldest and most widely used architectural patterns in computer science. The underlying idea is to separate data (model) and presentation (view) in order to get loose coupling between the two [1]. Coupling refers to the degree of direct knowledge that one element has of another and according to good programming praxis, separate parts of an application should work independently of each other. The last part, the controller, acts as a bridge between the model with the view.

2.1.1.1 Model

The model contains the data used in the application. It knows nothing of the view or the controller. Being observable, the model notifies its observers when a change has occurred in the model [2]. A model may consist of single objects or structures of objects such as maps, sets or lists.

2.1.1.2 View

The view is a visualization of the MVC model and is often shown as an interactive GUI. The view does not initially show any data from the model. Once the controller has requested, data from the model and the model has performed the actions requested data is sent back to the view, via the controller, for presentation [1].

2.1.1.3 Controller

The controller takes input from the view through interactive elements such as a button or a menu item, which in turn use connected logic such as an actionlistener or a function to communicate with the model. Once the model is as completed the

request, it notifies its observers at which point the controller can pass data to the view [1]. An overview of the MVC model is shown in Fig. 2.1.

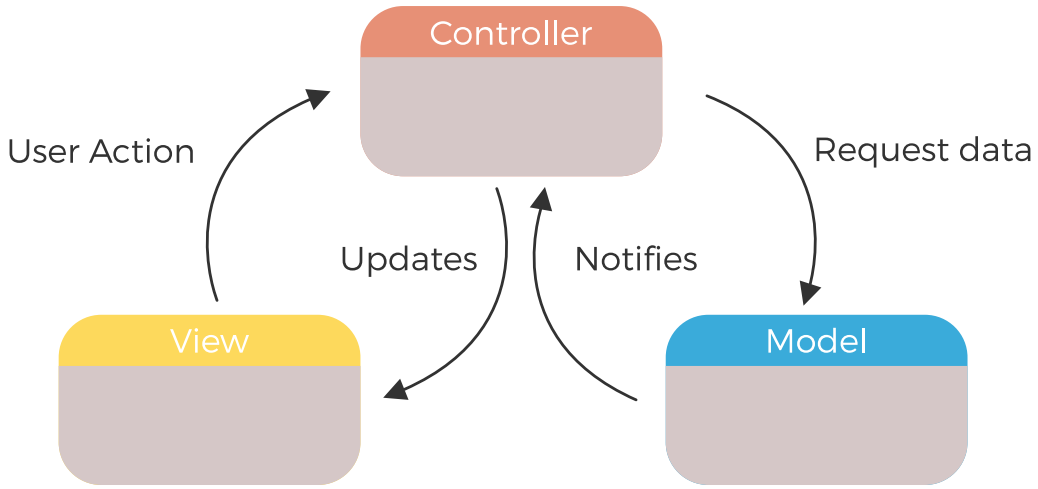


Figure 2.1: A graphic representation of how the different parts of the MVC pattern interact with one another. Note how the model and view has no direct contact with each other.

2.1.1.4 Benefits

The MVC pattern allows for parallel development due to separation and lack of dependency on its different parts. This makes it possible for part of a team of programmers to simultaneously work on the front end while another part works on the back end, without having to take into consideration each other's implementation [3].

Another benefit of the model is that multiple views can be created based on the same model, allowing for diversity in the way data can be presented [3]. It also decreases code duplication due to the separation of business logic and the presentation of data.

If properly implemented, the MVC pattern allows creating applications that are loosely coupled, have high cohesion and contains code that is easy to maintain and modify [1].

2.1.1.5 Drawbacks

One of the drawbacks of the MVC pattern is its complexity. Unless previously encountered or used, it can be quite challenging to implement the pattern [4].

The pattern is not suited for very small projects, as it will make things unnecessarily complex for the task at hand. In this case the benefits do not outweigh the drawbacks such as size and performance of the application [4].

The drawbacks are not nearly as many or big as what the benefits yield when using the MVC pattern, as long as it is implemented in an environment where it was intended to thrive.

2.1.2 Singleton

The singleton pattern is a software design pattern that limits instantiation of a class to a single instance. The singleton pattern is similar to a global variable in that it is globally accessible from all the other classes. It is useful when one and only one object is needed to coordinate actions across the application.

Singleton is also considered by many to be an anti-pattern. Since singleton classes are accessible from all other classes, it defeats the purpose of object-oriented programming where instances of objects are supposed to be passed around as parameters. It also goes against the design principles of object-oriented programming in that the class cannot be inherited, there is no direct control of creation and there is no way to inject dependencies [5] [6].

2.2 Binary semaphore

A binary semaphore is an integer variable shared between multiple processes, that is equal to either one or zero. The aim of using a semaphore is to control access to a common resource by only allowing a certain number of processes simultaneous access. A binary semaphore can be compared to a type of lock or mutex, and only allows a single process access to the common resource. Semaphores do not require busy waiting, and therefore do not waste any processing power [7].

2.3 JavaFX

Java FX (FX) is a Java library that facilitates building applications with GUIs [8]. Applications written with FX can run on multiple platforms such as desktop computers, embedded systems and tablets. It has a rich community that continuously updates and creates ready-to-use FX-based frameworks. Some examples of frameworks developed by the community are GANT chart rendering, geo-location map creation and 3D visualization [9].

2.3.1 XML

The Extensible Markup Language (XML) is, as the name indicates, a markup language. It defines a set of rules for encoding documents in a format that is both human- and machine-readable [10]. Although the design of XML mainly focuses on documents, the language is also widely used for representing arbitrary data structures commonly used in web services [11].

2.3.2 FXML

The implementation of XML in FX is called FX Markup Language (FXML). It is an XML-based user interface markup language for defining the user interface of FX applications. It is similar to how one would compose web GUIs in HyperText Markup Language (HTML), and allows for abstraction of program design from program logic in a simple yet usable way. FXML has support for Cascading Style Sheets (CSS), which is programming language used to define how elements are to be displayed on screen.

2.3.3 Scene Builder

Scene Builder (SB) is a free open-source visual layout tool that aids users in creating GUIs for FX applications [8]. SB consists of a drag and drop user interface and gives the programmer a graphical representation of the current state of the application. It has built-in tools for creating controls, charts and shapes available in its library. SB generates FXML files which can be integrated effortlessly with the FX application using any common IDE by binding the user interface to the application's logic. In addition SB provides a CSS analyzer which is used to inspect various parts of elements, such as the text inside of a button, or the button itself. Fig. 2.2 explains how a scene is created in SB and taken into Java to be displayed.

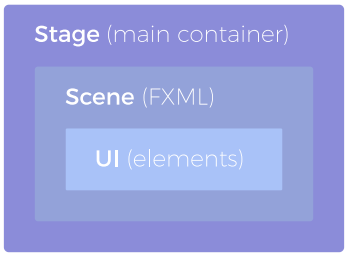


Figure 2.2: An example of how an FXML file is taken into a stage. UI elements are created inside the SB application and are grouped into containers. Examples of elements are buttons, text areas and sliders. An example of a container could be a grid pane or a vertical box. The scene is saved as an FXML file which is then loaded into a stage and displayed through Java code.

2.4 Version control

In software engineering, version control is a way to save different iterations of your work in different branches. A branch is merely a way to 'branch out' from the main project to work on a isolated copy of it [12].

2.4.1 Bitbucket

The hosting service Bitbucket is based on Git. It is a source code repository hosting service, designed to be used by teams. It is cloud-based and keeps a history of all the changes made to a project, allowing developers to revert to older versions if anything goes awry. It has support for multiple branches, which are copies of the main project. This is useful when many developers are working on the same project,

as changes to the branch does not affect the main project. Developers create a new branch for a specific feature they are working on and when the new feature is ready to be implemented, it may be merged into the main branch. This ensures that the main branch always contains an executable version of the application, available to the users of the application [13].

2.5 User Experience

When it comes to GUI's and user experience (UX), it can be designed differently depending on where in the world its target audience is located. In the West, things tend to be somewhat more minimalistic compared to the East where you have a lot more information in the same space [14]. While factors like these may not seem important they, together with colors, play a large role in how users perceive objects [15].

2.5.1 Color theory

Color theory is often used in UX design to instill a certain feeling or empower certain elements of a design [15]. Warm colors such as red, orange and yellow are generally associated with energy, action and danger. Cool colors such as blue, cyan and purple are associated with calm and peace. If used correctly, these colors may affect users, without the users ever being aware of it.

2.5.1.1 The color wheel

The color wheel is simply the color spectrum drawn in a circle. It is used to determine what colors look good together, so-called 'color harmony' [16]. There are many combinations of colors within the color wheel. Two possible color combinations are shown in Fig. 2.3.

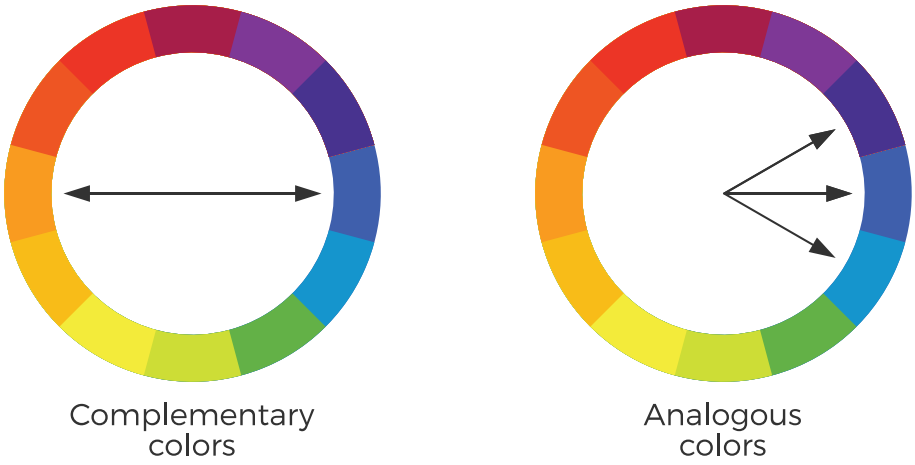


Figure 2.3: Complementary colors are located on opposite sides of the color wheel. Analogous colors are located on either side of the color chosen. These combinations are determined by the relative positions of different colors, and are two out of many combinations.

3
Methods and Implementation

This chapter includes the software, methods, patterns used to create the application and research conducted beforehand.

3.1 Research

Research began by looking into which programming languages would be suitable for our application. Deciding which frameworks and software patterns to use in the application was the second step in this process.

The primary idea was to code the back end in Java and front end in Python to make it compatible with already existing applications, while still being able to deliver a stylish and user-friendly design. The main reason Python was considered was its vast amount of libraries for plotting graphs [17].

It turned out to be harder than previously thought to combine the two languages this way. It was more common to do it the other way around - having the back end in Python and front end in Java, which was not an option due to the limitations of the laboratory equipment. It was decided that the front end would be written in Java as well by using the GUI library FX. By using Java as both the back end and front end the difficulties tied to how and where to separate the two languages were removed.

After deciding to develop the front end in FX, research shifted focus to reading up on what support FX offered in terms of visualization of data. It can be difficult to design a GUI with code alone, without a chance to see how the result looks before compiling. Research led to finding SB, which allowed for a way to create and alter a GUI through visual elements alone. It helped tremendously when designing the GUI and facilitated in producing a functional and user-friendly GUI. The built-in CSS analyzer allowed bypassing searching documentation manually and improved the workflow while designing the application.

3.2 Software development

Considering that the application would build on and use functionality from already existing software in use at Saab, the tools and software used was highly dependent on what the developers were already using. Because of this the development of the

application was done using a customized version of the Eclipse IDE, combined with Bitbucket for version control.

3.2.1 Programming language

The application was written in the programming language Java 8. This was the language other projects at Saab were written in and it facilitated the process since parts from the already existing applications could be re-used. It would also increase the longevity of our application due to the developers already being familiar with the language.

3.2.2 Design patterns

While developing the application two commonly used design patterns were used and referred to. These were the Model-View-Controller pattern and the Singleton pattern.

3.2.2.1 MVC

Early on during programming sessions, the MVC pattern was selected to structure the layout of the application and its classes. It was chosen because it is a software design pattern previously used in coursework at Chalmers. This approach seemed to be the most logical and it would aid in keeping coupling low between different parts of the application. By using the MVC pattern, clear guidelines existed that could be referred to when implementing new features, or modifying already existing ones.

3.2.2.2 Singleton

At a late stage of the project issues regarding how to pass information between controllers occurred. The project was far along when the problem was encountered and a decision was made to implement the most frictionless solution. This is how the singleton design pattern became a central part of the application. This resulted in transforming the MVC model into a singleton class, allowing controllers to access objects from any other class while making sure that only one model was able to exist during the execution of the program. The model implementing the pattern is provided by a separate, static, class which only purpose is to instantiate and provide access to the model from all the other classes.

3.2.3 Front end

For the front end the FX and controlsFX libraries were used, along with SB for designing the layout and CSS to skin elements. The first stage of developing the application was to create a simple GUI in SB . It was re-designed multiple times as the project went along, but having it helped in the continuous development of the application.

The GUI was created with FXML, where every view of the application corresponds to an FXML document. Each of the FXML documents are in turn linked to a controller. The FXML files were generated by SB while the controllers were created from scratch. Nearly all of the controllers communicate with the model, which acted as the backbone of the back end.

3.2.4 Back end

The back end was written solely in Java 8. The class serving as the model in the MVC implemented the singleton pattern. It set up the environment with the correct paths and connections, allowing the application to read from files as well as in real time directly from the G1X. The data that is read is passed on to a class serving as the database, storing the data in several multi-dimensional maps and lists used for different parts of the application. The stored data is available through the use of various methods to all of the other classes and controllers by utilizing the singleton model, which in turn as access to the database.

Graphs were generated in a certain class, which contains methods for generating all of the available graphs: line-, area- and scatter charts. Its methods were called by either one of the two controllers, each corresponding to one of the two main views in the application. One controller is used for streaming live data, and the other to read from log files. The generated graph is returned to the current controller and is in turn displayed through the view in the GUI. When creating a graph, the user is prompted with the view for the controller handling graph settings where he/she chooses which elements to display data from as well as what kind of graph to show. The data was fetched from the database through the model. If the application was connected to the G1X, it would continue to update the graphs with real-time data as soon as it arrived.

3.3 Implementation of modes

During the first half of the project, the focus was to create data structures that allowed the application to take in data in a way that would work for reading from a stream as well as when reading from a file. These modes are referred to as stream or file reading modes.

3.3.1 Reading from file

Despite receiving data it could not be read without a parser. The parser was located in a different repository which we did not have access to. After gaining access to more repositories, the original parser used to translate log files could be implemented with the application. Once it had been confirmed that the data taken in and shown in the original log file parser was the same as in the application, focus shifted towards implementing features such as graphs, an overview of the sensors and a settings menu.

Once the application could handle multiple types of data it was time to move on to reading from stream. This was not something that could be done remotely, but rather initially required a physical presence in the labs at Saab.

3.3.2 Streaming live data

The same approach as when reading from file was taken when moving over to reading from streamed live data. Code was added to the model and controllers to allow connections to the IP address of the computer connected to the radar. Parts of previously used software at Saab was also implemented to control which ports and sensors that should stream data. The connection ports and IP addresses were hard-coded as previous software was undergoing structural changes and could not be fully used.

3.4 Mode differences

The GUI is very similar in both modes, with a few exceptions. Aside from the color themes, the stream mode has a switch which allow users to select whether or not data should be printed in the list view and is always on when reading from file. Since this has no function when reading from a file, it is not present in file reading mode.

Another difference is updating of the generated graphs. While in stream mode, the graphs are updated with new data nodes as soon as the model has stored the read data in the database. This occurs in real time. When reading a file, all the data is read at once and the need for graphs updating in real-time is non-existent.

Lastly, the main menu bar only allow users to open files while in file reading mode and the option has been removed from stream mode.

3.5 Application environment

The application will be running on hardware inside of the radar laboratory at Saab, which are restricted as to which software and compilers can be used. Because of this, it was written solely in Java 8 using built-in libraries. Software was downloaded from Saab as the version of Java 8 available online was not compatible.

4
Results

The main objective with this project has been to create a GUI providing a system overview for the G1X. It was imperative to complete the application to such a degree that it could be implemented and used by the team of engineers at Saab within the given time frame of the thesis. It was also important to build the application in a way that made it possible for it to run as intended on the hardware available in the laboratory.

This chapter presents the final results and tie back to the aims set at the start of the project.

4.1 Software

The application is fully functional, and supports more than 10 types of data that can be displayed through three different type of graphs. The feedback from future users of the application has been positive. All of the objectives in the bullet point list available in section 1.3 were completed. Since the application was written in Java it is cross-platform and can run on all systems that support Java.

4.1.1 Additionally requested features

During the development of the application, additional features were requested. The features ranged from changing the size of the elements displayed in graphs, to more complicated ones such as being able to undock a graph from the main window, having a search bar for choosing elements to display and making it possible to zoom in and out on graphs. All of the additionally requested features were successfully implemented. The search bar can be seen in Fig. 4.4, and undocked graphs are shown in in Fig. 4.5.

4.2 The GUI

When launching the application users are prompted to select one of the two available modes, stream or file reading mode. The GUI consists of a main window, a graph settings window and the generated graphs.

4.2.1 The main window

The main window is shown in Fig. 4.1. The upper left area contains all unique sensors found in stream or file along with their values. This is the system overview. The terminal window at the bottom contain data that has been read, or is being streamed. This part of the application might seem unnecessarily wide, but some more advanced data types contain much more information, which is why the list view has support to scroll horizontally. This list view contains more information than the unique sensors, such as time stamps and origin of the data. The graphs generated are a visualization of the list view. Once a graph has been generated, a new tab is created and filled with said graph. Selected tabs are highlighted in the color theme of the mode users are in. By default, only the Info tab exists and it cannot be removed unlike generated graphs' tabs.

4.2.2 Stream mode

When opening up stream mode, users are presented with the view shown in Fig. 4.1. The mode automatically connects to the G1X radar and users may toggle whether or not they want to display read data in the list view. This option is present in the stream mode to reduce strain on the application while reading large sets of data. Furthermore, the stream mode utilizes an orange color theme, applied through CSS.

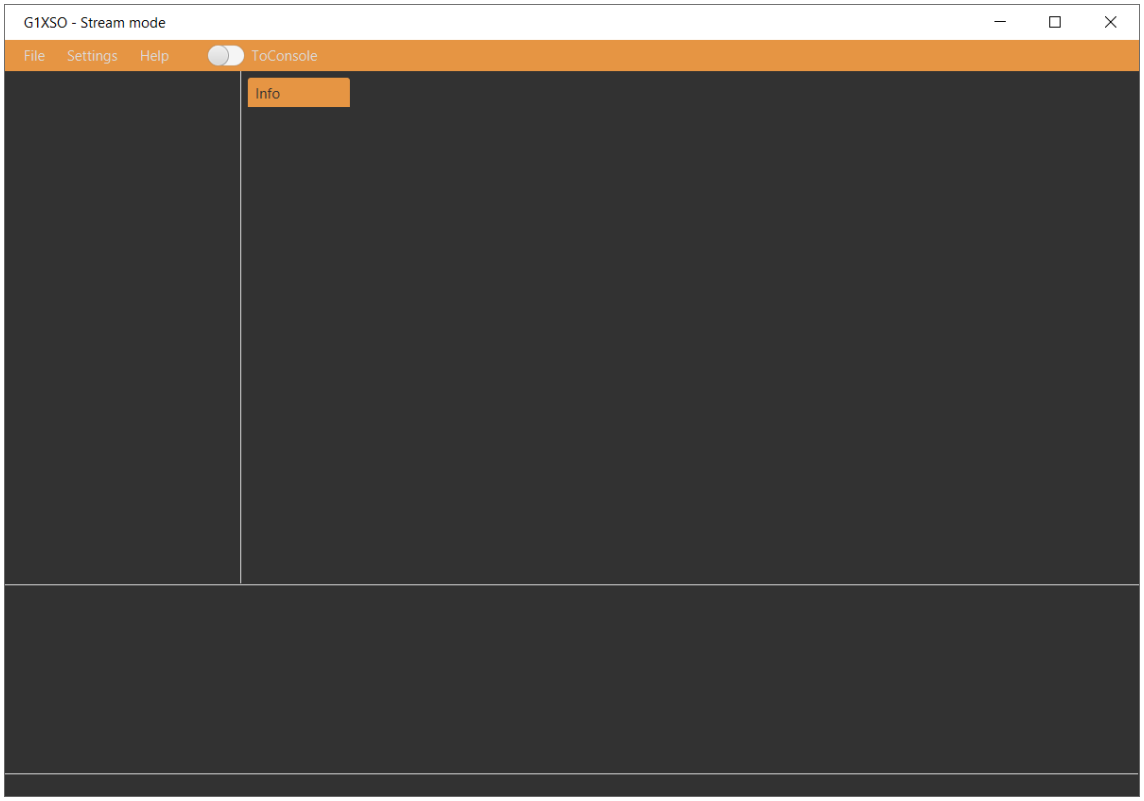


Figure 4.1: This is the stream mode which connects to the G1X radar to generate a live feed. The GUI will remain empty until the 'ToConsole' toggle button is pressed. Graphs may be generated without toggling the switch.

The theme is applied to important elements such as sensors and when elements are focused or toggled. Once the toggle switch in Fig. 4.1 has been toggled, the GUI starts updating the view with data from the model. The update is instantaneous for the sensors, the list view and the graphs, if any have been generated. This is shown in Fig. 4.2.

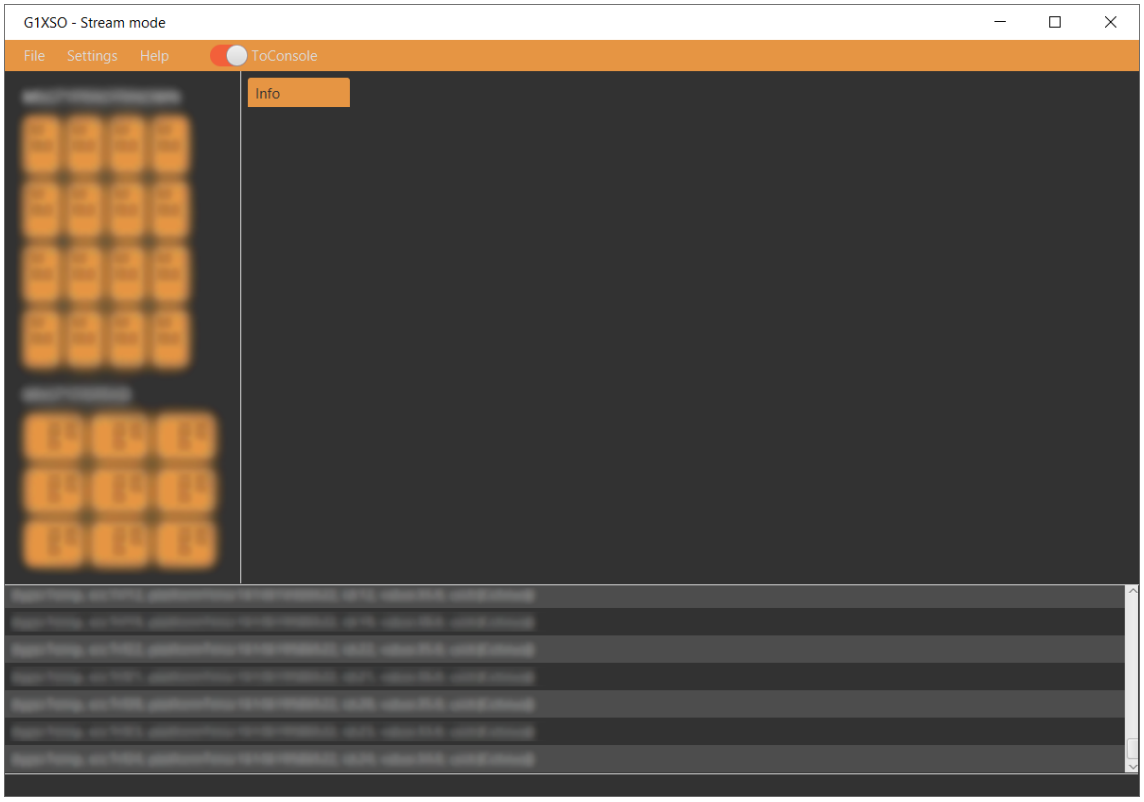


Figure 4.2: This is the stream view once the 'ToConsole' toggle switch has been toggled. Due to the sensitive nature of the data displayed it has been blurred out in order to be able to include the graphics in this thesis.

4.2.3 File reading mode

When selecting file reading mode, users are instead presented with Fig. 4.3. This mode is used to read saved log files containing data from a G1X radar session. Users may open a file through the main menu through the file explorer. Once the file has been read, the list view is filled and so are the unique sensors. Users have the option to generate graphs once the data is loaded into the model.

The view will not update until the whole file has been read, and this time varies depending on how large the recorded log file is and what data types it contains. Once loaded in, the view look similar to Fig. 4.2. The application is displayed in blue rather than orange, indicating that the application is in its file reading mode.

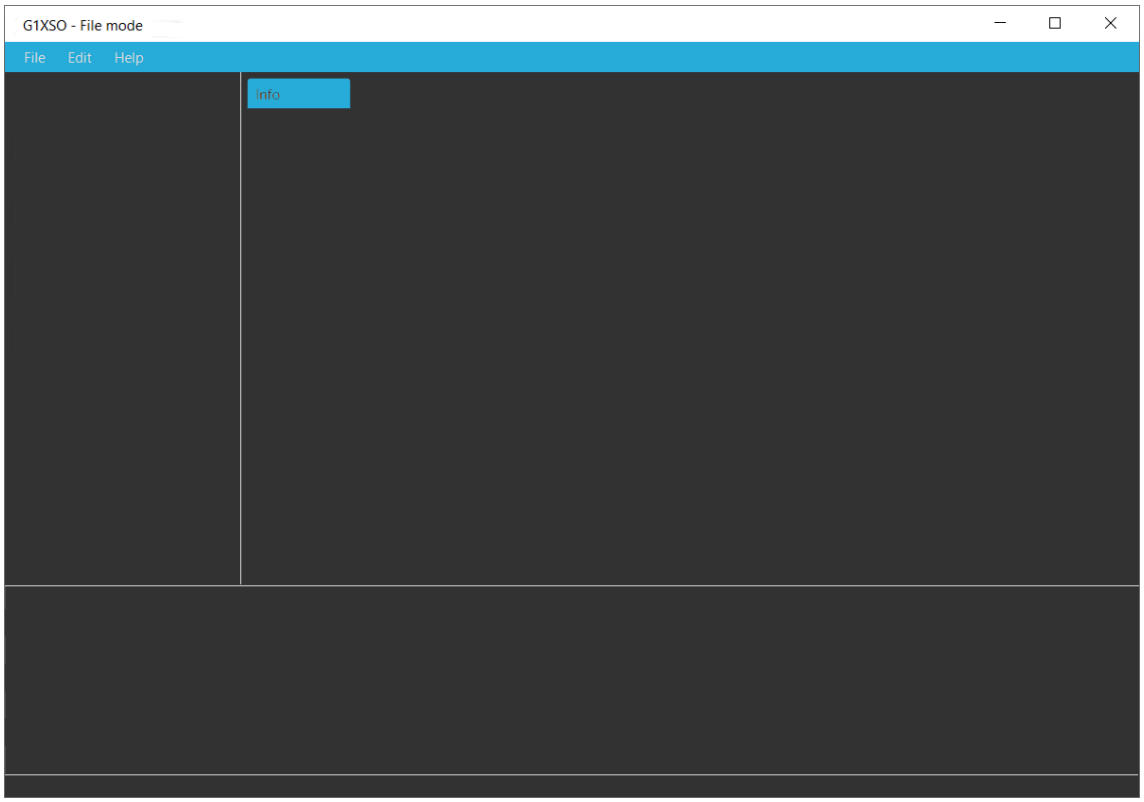


Figure 4.3: This is the file reading mode which read log files. The image depicts the application when a file has not yet been read and the GUI will remain empty until users have read a file.

4.2.4 Graph settings

To generate a graph, users must access the 'New graph' menu-item located in the main menu bar. The pop-up window produced by this action is shown in Fig. 4.4. Sensors found in the stream or from logged files are toggled and the type of graph rendered is also selected. Users may input a unique name for the new graph, which will become the name of the tab it is located on. If no name is selected, the application will default the name to the type of graph selected i.e 'Line Graph' for example.

There is also a search bar that will match and display sensors that partially or fully match its input, while hiding the remaining ones. The search bar accepts any part of the string, as long as it is present in the name of the sensor. This is useful when searching for sensors located in data types with longer names. If nothing has been entered in the search bar, the list shows all unique sensors found in the stream or read file are shown.

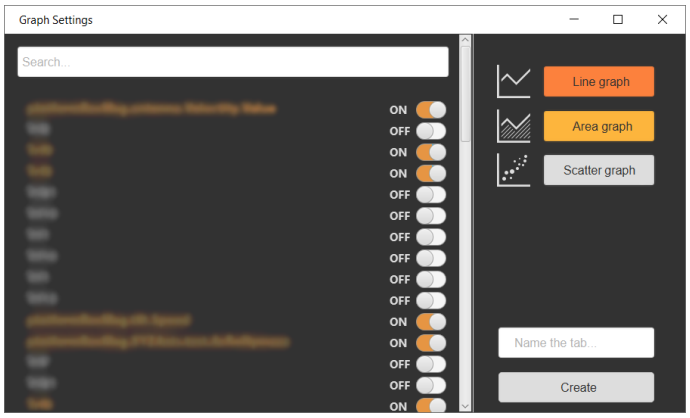


Figure 4.4: This is the graph settings window. Notice how the toggled element's names are orange, and the font is bold. The selected graph type is highlighted in a dark orange, while the hover effect is of a brighter orange. Although not visible in the image, the cursor is hovering over the 'Area graph' button. Due to the sensitive nature of the data displayed it has been blurred out in order to be able to include the graphics in this thesis.

4.2.5 Undocking graphs

As previously mentioned in section 4.1.1, a requested and implemented feature was to be able to undock a graph, which is shown in Fig. 4.5. The feature enables users to watch multiple streams of data, or sensors, simultaneously.

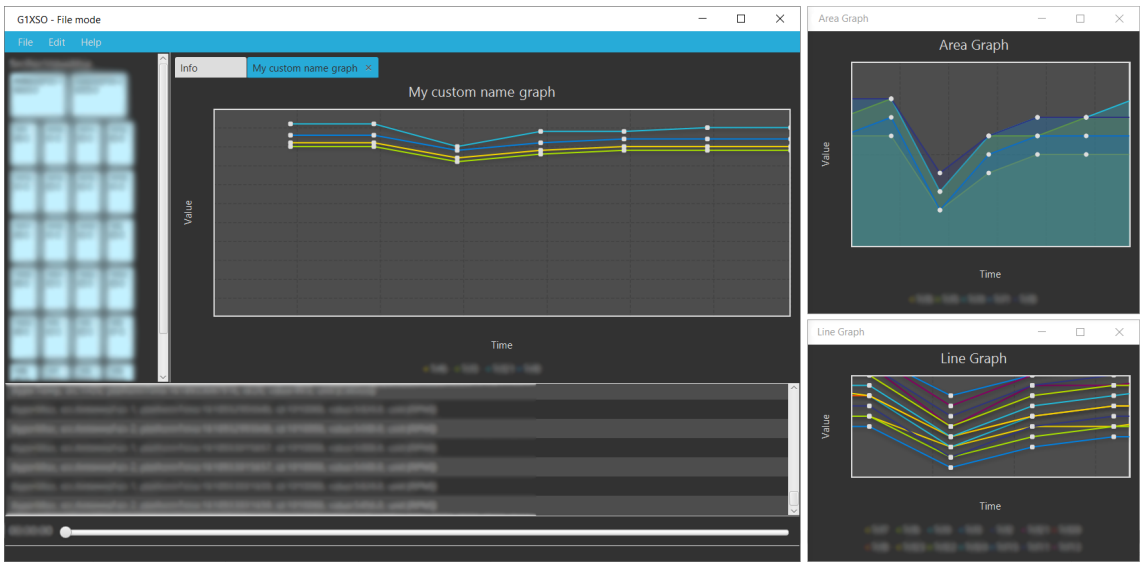


Figure 4.5: Example of the GUI. The GUI is currently in file reading mode, as indicated by its blue color. A file has been read and three graphs have been generated. Two of these have been undocked from the main application window and are located to the right. Users may right-click to undock a graph. Due to the sensitive nature of the data displayed it has been blurred out in order to be able to include the graphics in this thesis.

4.3 Types of graphs

It is possible to display the data in three kinds of graphs: line-, area- and scatter graphs, shown in 4.6 respectively. The graphs can all be maneuvered by zooming in and out on different parts of the graph by using the scroll wheel on the mouse combined with the shift-key. If a graph were to be undocked, it will keep the current view of the graph in the new window.

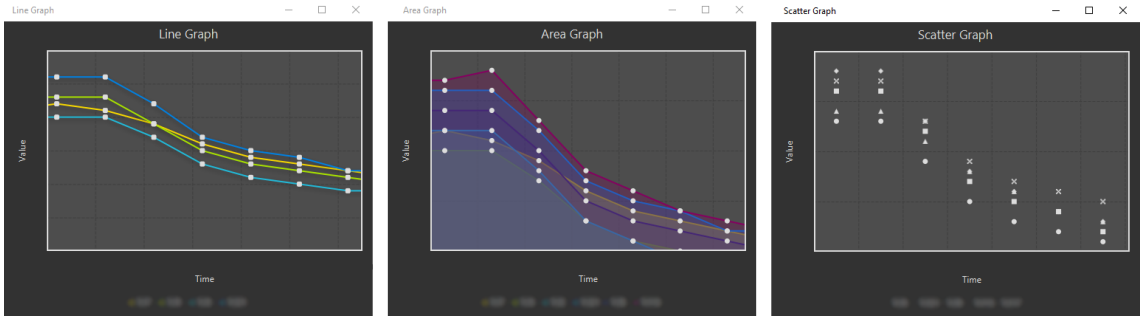


Figure 4.6: An example view of all the available types of graphs; line graph to the left, area graph in the middle and scatter graph to the right. Due to the sensitive nature of the data displayed it has been blurred out in order to be able to include the graphics in this thesis.

4.4 User friendliness

The application has been developed with user friendliness in mind. As a result of this, features have been implemented to be as versatile as possible. An example of this is the search bar. It accepts any numeric or literal input and matches any part of a string if the input is present in any of the names of the sensors.

Another example is that button effects, such as focus or hover, have been accentuated in order for users to easily recognize their functionality. This creates an effect to make it look like the button has been pressed and is now toggled. This effect can be seen in Fig. 4.4.

Furthermore, when selecting an element in the the sensor list inside of the graph settings, the name of the sensor changes color to match the current mode's theme color and the font becomes bold. This allows users to easily see which sensors have been selected. This gives instant feedback to users if they have chosen the right sensor or not.

4.5 Documentation

Documentation for the application exists in two forms. There is a 'Help' section in the 'About' menu item inside of the application. It contains a short tutorial on how to use the application, from reading from a file to displaying it as a graph.

There is also an external readme file which explains all parts of the application. Beyond this, the code for the application has thorough commenting throughout to allow for easier future implementation and understanding of classes or types of graphs.

5
Discussion

This chapter treats the difficulties that were encountered at different stages while developing the application, design choices made and what a data type was defined as. It also includes a brief section about ethics and the environmental impact of the application.

5.1 Difficulties

The difficulties encountered while programming were many, and not all were easily solved. Due to the sheer number of sensors and data types used, a lot of consideration went into how to design the database used by the data model. Assigning and accessing data to and from a node, in our case a point in time, required many trials and tribulations.

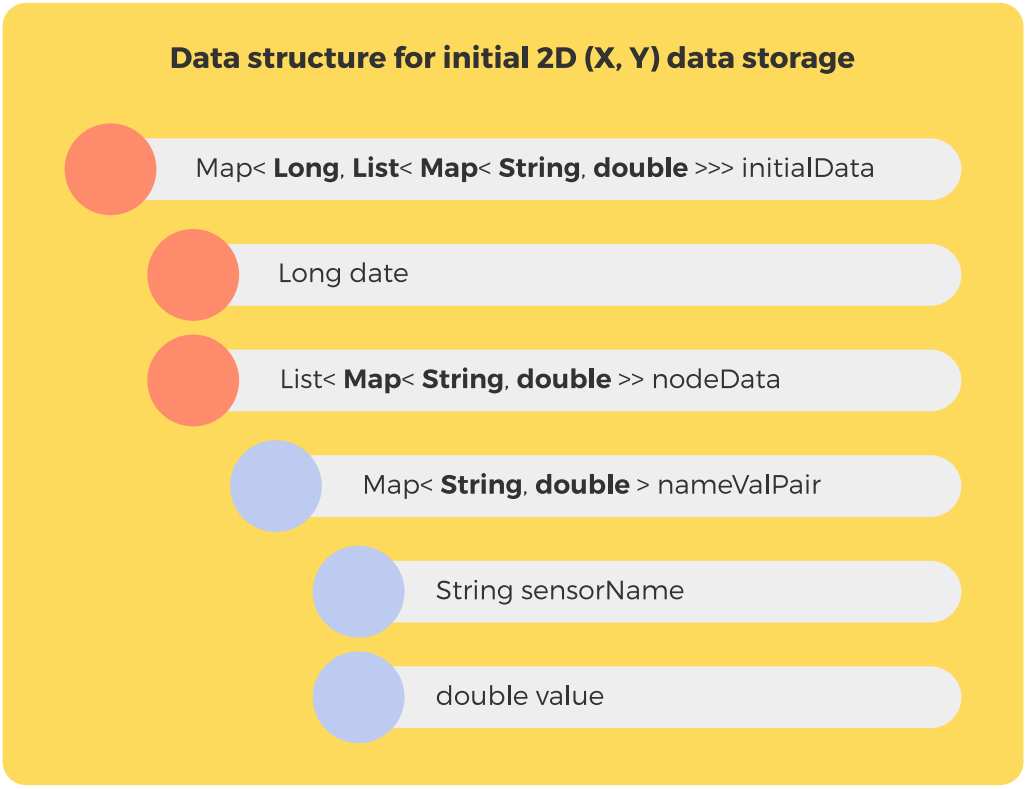


Figure 5.1: Example of data structure for reading data. Nodes are created for each time frame, and add all sensors that have data for the time frame. This data is fetched when generating graphs.

5.1.1 Threads

FX runs its own thread for related FXML elements unless specified to do otherwise. Placing tasks that run for a long time or require a lot of processing power on this thread inevitably makes the GUI unresponsive [18]. This problem first occurred when larger data sets were read into the application. We solved this by creating separate threads for handling the long-running tasks in order to reduce strain on the FX thread.

5.1.2 Java runtime and compilers

Due to miscommunication, the application was initially written in Java 11 which later caused problems when trying to deploy it onto the hardware in the laboratory. It was remedied by recompiling the whole project and swapping out certain libraries to ones already existing within packages of the repositories provided by Saab. This took up valuable time spent on site, and hindered us from spending more time on developing functionality for reading data in real time.

5.1.3 Reading data in real time

When implementing the second part of our application, streaming data from the radar, a myriad of issues were encountered. Initially we could not access the lab, despite having the proper Secure Shell (SSH) keys. We needed an additional name extension on the account we were trying to connect from when generating the SSH key. Once resolved, we needed to order additional software to connect to the lab hardware via a proxy server.

Once set up and connected, we ran into concurrency issues. The application crashed when creating a graph while reading real-time data. The issue occurred while trying to store and read data from the database, resulting in a concurrent modification exception. Multiple processes were trying to access a common resource at the same time. We managed to solve the problem by implementing a binary semaphore controlling access to the common resource - the database.

Every time a new graph was to be created, the FromStreamController acquired the semaphore and prevented other processes from accessing the database. For the duration the semaphore was held by the controller, new elements were placed in a queue. Once the graph was generated, the controller released the semaphore. All of the elements from the queue were then moved to the database and new elements were once again saved directly to the database.

One of the last problems solved was the right-click menu mouse event being consumed by the updating of a frame while streaming, which occurred too frequently for users to select available options. The event was changed to an onMouseClick built-in function. This also affected the ability to zoom in graphs, as a modifier key had to be pressed to zoom horizontally. A modified version of the built-in function onShiftDown was created in order to solve this.

5.2 Design choices

We wanted the application to be very minimalistic while still being able to contain large quantities of useful data. This forced us to make good use of all areas in the application by thoroughly thinking through its design in the early stages of development.

As for the design choices of the application a few ideas about button placement and color choices existed from the start. Utilizing the same application frame, with a few minor differences, for the stream and file reading we decided to use complementary color schemes for the different views. These are on opposite sides of the color wheel (as shown in Fig. 2.3) and represents the contrasts of streaming live, or reading logged data from a file. These colors are used for the menu bar, icons and selected texts associated with the respective application chosen as can be seen in Fig. 5.2. It was a very minor implementation, but was well received by the team of engineers. This facilitates identifying which mode the application is in quickly.

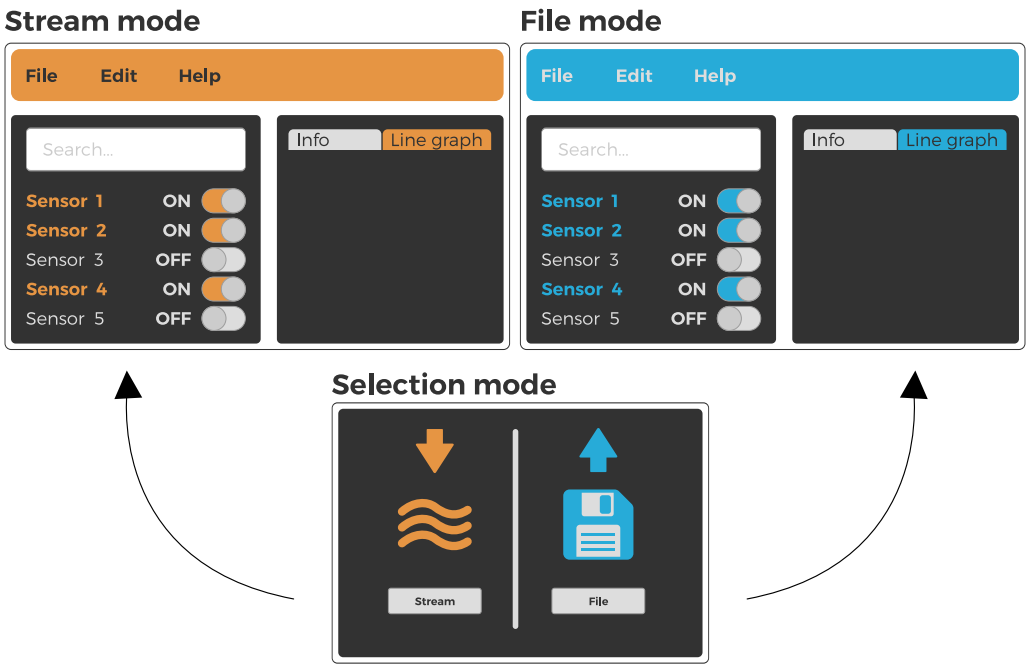


Figure 5.2: These are miniature replications of parts from the application and show all the elements affected by the two different color themes.

Throughout the application we have gone with a left-to-right, top-to-bottom design, as this is the way most Westerners read and take in information [14]. Elements such as getting a quick overview or selecting sensors, are located on the left-hand side, spanning from top to bottom. Buttons for creating graphs and accepting to open files are located in the bottom-right corner of its respective container. This can be seen in Fig. 4.5.

5.3 Data types

As mentioned in section 1.3, one of the ambitions was to read 10 data types as a foundation for the application. As the project went along we realized that data types were not well-defined, and Saab's definition of a data type differed from our own understanding of the term. The data types were in fact messages grouped by category.

The first message type, referred to as msgType 1, read from a logged file turned out to consist of more than one data type. It contained three pre-defined types, and a miscellaneous section that takes in undefined types of said class. Fig. 5.3 attempts to visualize the structure of this message.

The second message type, referred to as msgType 2, had a different structure compared to msgType 1. This made us realize that a generic way of reading data did not exist, as a majority of the msgTypes were unique in their structure. When adding a new data type to the application, a specialized way of parsing its msgType must also be implemented to be able to keep the back end unmodified.

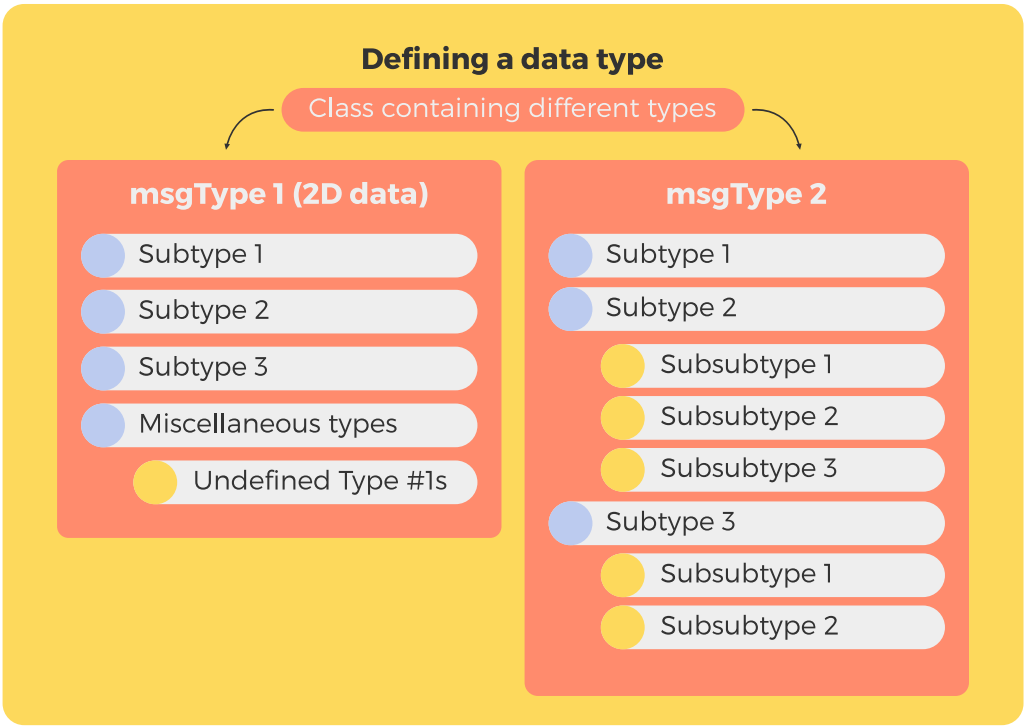


Figure 5.3: msgType 1 only contains 2D data. Examples of this is temperature and current, value over time types. An example of a miscellaneous type is fan speed. msgType 2 pertains to a certain part of the radar and its associated values. The subsubtypes are parts within the larger part with their own data structure and values.

5.3.1 Defining data types

All types are read properly by the application, but this raised the question pertaining to what data types should be defined as. We chose to define a data type as a sensor with a value attached to it, seeing as a message could contain a multitude of different types. These are the sub- and subsubtypes in Fig. 5.3. As an example, msgType 1 would count as three data types plus an undefined amount of miscellaneous types.

5.4 Environment and ethics

Regarding ethics the application developed will be used by integration and verification engineers at the Saab Radar department. The radar detection is used for military defense and is subject to the question if it is ethically correct to work on and support this kind of development. Considering its main functionality is used for defense, neither of the authors has an issue with it but rather support the ability to being able to defend yourself.

As for the environment, the impact of our application is not appreciable to a measurable degree of certainty. We have not conducted the proper measurements of performance or resource usage of previous applications used, or our own application. Our application may reduce resource usage at Saab by gathering data that would otherwise have required multiple applications for Saab's employees. This is merely an idea and implications remain difficult to estimate.

6
Conclusion

The purpose of this thesis was to create G1XSO, a standalone application that creates a system overview with options for how to visualize the data it receives. This is in accordance with our purpose, which was to develop a user-friendly application that merges functionality of current software used at Saab, with the added benefit of displaying data received from the G1X radar in various ways.

6.1 Further development

There are features that could be improved upon, such as the implementation of a dependency injection framework [19]. Graph navigation could use some more polish to ensure a smoother workflow for users of the application. The MVC is also something that could use some work to further improve in decreasing coupling by implementing thought-through interfaces to extend from.

6.2 Insights

About half of the allotted project time had passed before data needed could be accessed in a format that could be fed into the application. This was due to security scrutiny and gaining access to the various repositories. While this was something that could not be affected by us, knowing of it and preparing beforehand would have been helpful. This down time was used to work on the GUI and design the application structure.

As the development of the application went on, decisions taken in the beginning may not have been the best for the application. As mentioned in the section about further development, a dependency injection framework could have been used instead of the singleton pattern. This would have facilitated accessing data and improved overall structure, by letting the application better adhere to object oriented practises and for classes to be inherited, the negative aspects of the singleton pattern would be removed. In order to properly implement this, a more thorough briefing with Saab examining the data types and requirements at the start would have been required.

Despite this, the application is a solid base for what Saab sought out at the start of the project even though it is not completely finished.

References

[1] E. Doherty, “MVC Architecture in 5 minutes: a tutorial for beginners”, Educative, May 11, 2020. Available: https://www.educative.io/blog/mvc-tutorial [Online; accessed Mar. 28, 2021]

[2] Predrag, “The Observer Pattern in Java”, Baeldung, Oct. 1, 2020. Available: https://www.baeldung.com/java-observer-pattern [Online; accessed Apr. 21, 2021]

[3] “Six Benefits of Using MVC Model for Effective Web Application Development”, Brainvire, Unknown. Available: https://www.brainvire.com/six-benefits-of-using-mvc-model-for-effective-web-application-development/ [Online; accessed Mar. 25, 2021]

[4] Jithin, “What is MVC? Advantages and Disadvantages of MVC”, InterServer, Oct. 28, 2016. Available: https://www.interserver.net/tips/kb/mvcadvantages-disadvantagesmvc/ [Online; accessed Mar. 21, 2021]

[5] B. Button, “Why Singletons are Evil”, Microsoft, May. 25, 2004. Available: https://docs.microsoft.com/sv-se/archive/blogs/scottdensmore/why-singletons-are-evil [Online; accessed Apr. 29, 2021]

[6] https://testing.googleblog.com/2008/11/clean-code-talks-global-state-and.html [Online; accessed Apr. 29, 2021]

[7] Baeldung, “What is a Semaphore?”, Baeldung, May. 7, 2021 (Modified). Available: https://www.baeldung.com/cs/semaphore [Online; accessed May. 1, 2021]

[8] “JavaFX - Overview”, TutorialsPoint, Unknown. Available: https://www.tutorialspoint.com/javafx/javafx_overview.htm [Online; accessed Apr. 21, 2021]

[9] “JavaFX”, JavaFX, Unknown. Available: https://openjfx.io/ [Online; accessed Apr. 1, 2021]

[10] “Extensible Markup Language (XML) 1.0 (Fifth Edition)”, W3C, Nov. 26, 2008. Available: http://www.w3.org/TR/2008/REC-xml-20081126/ [Online; accessed Apr. 21, 2021]

[11] P. Fennell, “Extremes of XML”, XML London 2013, London, Great Britain, 2013, pp. 80-86. Available: https://xmllondon.com/2013/xmllondon2013-proceedings.pdf#page80 [Online; accessed Apr. 21, 2021]

[12] "Getting Started - About Version Control", Git, Unknown. https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control [Online; accessed Apr. 27, 2021]

[13] "A brief overview of Bitbucket", Bitbucket, Unknown. https://bitbucket.org/product/guides/getting-started/overview [Online; accessed Apr. 29, 2021]

[14] S. Malachi, "Cross Cultural Interface Design", Muzli, Jul. 1, 2014. Available: https://medium.muz.li/malachidigest-828e37f45117 [Online; accessed Apr. 29, 2021]

[15] K. Decker, "The fundamentals of understanding color theory", Muzli, Feb. 27, 2017. Available: https://99designs.com/blog/tips/the-7-step-guide-to-understanding-color-theory/ [Online; accessed Apr. 29, 2021]

[16] "Color wheel", Canva, Unknown. Available: https://www.canva.com/colors/color-wheel/ [Online; accessed Apr. 29, 2021]

[17] "Python Advantages and Disadvantages – Step in the right direction", TechVidan, Unknown. Available: https://techvidvan.com/tutorials/python-advantages-and-disadvantages/ [Online; accessed Apr. 30, 2021]

[18] "Concurrency in JavaFX", Oracle, Unknown. Available: https://docs.oracle.com/javafx/2/threads/jfxpub-threads.htm [Online; accessed Apr. 27, 2021]

[19] L. Vogel, "Using dependency injection in Java - Introduction - Tutorial", Vogela, Jan. 1, 2016.
Available: https://www.vogella.com/tutorials/DependencyInjection/article.html [Online; accessed Apr. 20, 2021]

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
GOTHENBURG UNIVERSITY
Gothenburg, Sweden
www.chalmers.se

CHALMERS