Project Proposal

Constellation - Extending the user interface

Organisation: CERN HSF
Mentors: Stephan Lachnit, Simon Spannagel

Contact Information:
Full Name: Abhay Anil Varnekar
Degree: Bachelor of Technology in Computer Engineering (B. Tech)
University: Veermata Jijabai Technological Institute (VJTI) , Mumbai
Location: Mumbai, Maharashtra ,India
Time Zone: Indian Standard Time (UTC+5.30)
Email: abhayvarnekar05@gmail.com, aavarnekar_b23@ce.vjti.ac.in
Github Profile: @Abhay-Varnekar
Resume: Resume

1.1 About Me: Introduction & Background

Hello there! Abhay Varnekar here. I am an undergraduate student in my second year at Veermata Jijabai Technological Institute (VJTI), Mumbai pursuing Computer Engineering as my major.

1. As for relevant courses in the institute, I have taken courses such as Software Engineering, Data Structures and Algorithms and Object-Oriented Programming. These courses have provided me with a solid foundation in software development, problem-solving, and designing user interfaces
2. I have been developing projects and participating in various competitions since my first year.
    - ➔ 2D Car Simulation- This project simulates the evolution of cars using genetic algorithms over generations in a 2D terrain environment using JavaScript, HTML, and CSS.
    - ➔ Intrusion Detection System- A cybersecurity solution that integrates Suricata IDS with Wazuh for real-time monitoring and a chatbot for user assistance.
3. I have good experience with C/C++, Python, Git/Github . Also, Linux has been my daily driver for almost 2 years now and I am fairly comfortable with it.
4. I have been contributing to open source for a brief period of time. Here are my some of my open source contributions: MuseScore

1.2 Other Commitments

I can give 4 to 5 hours to the project everyday, compiling it to **30-35 hours per week** or even more than that if required. I do not have any strict commitments to fulfill during this summer and can **easily increase my duration** if required. I am flexible with any working hours and can adjust them as per the time zone of my mentors. I have my summer vacation from **20 May to 14 July** so a major part of the work will be easily completed in that duration. I will provide **regular updates** to my mentors regarding progress and will ask for their help whenever required. I'll continue to post updates on my blog every **two weeks** for reference.

2. Why me and my Motivation?

My passion for software development and solving real world problems has been the driving force behind my involvement in various projects. While solving different tasks and issues in development, my interest in contributing to large-scale, impactful projects has only grown stronger.

I've always aspired to work with organizations like CERN, where I can contribute to meaningful projects while continuing to learn and improve as a developer.

I'm excited by the idea of creating a Qt interface that simplifies tasks like chart generation. As this is crucial for users with limited programming experience who might struggle with complex setups such as Grafana. A more intuitive solution that would allow them to focus on their research and not troubleshooting.

I will be grateful to receive the opportunity to contribute as a Google Summer of Code student developer. This is a great opportunity and a head start for achieving success in my life.

3.1 Project Description

The main goal of this project is to develop a **dynamic and interactive user interface (UI)** that provides **real-time telemetry data visualization** for the Constellation system with just a few clicks which offers a much faster and more convenient solution. This interface will allow experiment operators and scientists to monitor critical system metrics from various instruments.

The UI will enable users to interact with the data, providing a dynamic and customizable dashboard that reflects their unique requirements for different experiments. The telemetry data will be continuously updated and displayed via interactive graphs, giving users the ability to monitor various system parameters and quickly react to any issues that might arise.

Key features of this project include:

- ➔ **Dynamic Telemetry Dashboard**: A customizable dashboard displaying real-time system metrics , with interactive graphs and charts to visualize telemetry data.
- ➔ **Service Discovery and Subscription UI**: A UI component for automating the discovery of services and subscribing to relevant topics, enabling users to monitor specific data streams.
- ➔ **User Preferences Persistence**: A feature to save and load user settings for a consistent user experience across sessions.

- ➔ **Seamless Integration with Constellation**: A Qt-based application that integrates smoothly with the Constellation framework, ensuring efficient real-time data updates and minimal performance impact even with multiple telemetry sources.

This project will focus on delivering a responsive and intuitive UI for experiment monitoring and control, tailored to the needs of users working with complex experimental setups.

3.2 Milestones

- ➔ Creating a new GUI to display monitoring data from devices using Qt Charts
- ➔ Modularization of UI elements into reusable Qt widgets
- ➔ Adding the monitoring widget to the existing GUI for device control

3.3 Implementation: Plan of Action

Basic Goal Of Project:

The goal of this project is to extend the **Qt-based interface** of the Constellation framework to enhance the user experience for monitoring and interacting with devices and services. The project aims to build a **user-friendly dashboard** that allows users to easily discover, subscribe to, and visualize real-time telemetry data and system events. The project will be implemented in **five phases**, ensuring seamless integration with Constellation while maintaining performance and usability.

- ➔ **Phase 1** - Implementing the StatListener class to subscribe to satellites and retrieve telemetry metrics using CMDPListener.
- ➔ **Phase 2** - Develop a Qt-based UI for selecting satellites and telemetry parameters with real-time subscription management.
- ➔ **Phase 3** - Integrate QChart to visualize real-time telemetry data with interactive graphs.
- ➔ **Phase 4** - Implement configuration persistence using QSettings and optimize UI performance.
- ➔ **Phase 5** - Final testing, bug fixing and creation of proper documentation.

Potential Solution:

To create a dynamic and interactive dashboard for the Constellation framework, the solution will integrate the newly implemented **StatListener** class with the **CMDPListener, CHIRP Manager** and **CMDP Protocol** codes with a **Qt-based UI**, ensuring real-time

telemetry data visualization and seamless system interaction. The following outlines the approach and technologies used to achieve this goal:

- ➔ **Service Registration and Discovery**: It will provide users with an easy way to discover and register relevant services for real-time telemetry data monitoring.
  - ◆ Using CHIRP, the system will be capable of auto-discovering available services within the Constellation network and subscribe to these satellites on the CMDP protocol.
  - ◆ A service registry will be displayed on the UI, where users can easily view and select services of interest. This will be built using Qt Widgets to list available services.
  - ◆ The UI will allow for dynamic updating in real time as services are added or removed, leveraging Qt's signals and slots to push updates automatically when the system changes.
- ➔ **Real-Time Feedback and Data Subscription Management**: This will enable users to manage their subscriptions to telemetry data and receive continuous updates.
  - ◆ StatListener Class will be developed in Phase 1 to allow the subscription to telemetry data streams using CMDPListener.
  - ◆ A real-time subscription interface will be built in Qt, allowing users to easily subscribe to and unsubscribe from specific telemetry topics or data streams.
  - ◆ To ensure real-time updates, the UI will leverage Qt's signals and slots to push updates as new telemetry data is received, allowing for immediate visualization of any changes or anomalies in the system.
- ➔ **Interactive Telemetry Data Visualization**: To provide real-time graphical representation of telemetry data to enable users to visually monitor the system's performance.
  - ◆ QChart will be integrated in Phase 3 to visualize telemetry data using interactive graphs and charts. These graphs will allow users to track system metrics such as satellite performance, telemetry stream data, or other relevant system parameters.
  - ◆ Graphs will update automatically as new data is received, with smooth transitions to ensure a seamless experience. Real-time data will be reflected on the graphs, ensuring users can easily monitor changes.
- ➔ **User Experience and Configuration Persistence:** Ensure a consistent, personalized experience by saving user preferences and optimizing performance.
  - ◆ Using QSettings, the UI will allow users to save and load their preferences across sessions. This includes custom dashboard layouts, selected data streams, graph configurations, and other UI settings.

- As the system grows and new telemetry sources are added, the UI will remain lightweight and responsive, with careful attention given to handling multiple data streams efficiently without compromising performance.

3.4 UI Design Overview

A Qt-based monitoring interface is ideal for users who need quick, accessible data visualization without the overhead of complex setup procedures. The simplicity, fast access to real-time data, and ease of use make it the perfect choice for operators working under time constraints. It allows users to start monitoring immediately, without wasting valuable time on configurations, making it an essential tool for quick and efficient experiment management.

Design Principles:

- **Clarity**: The interface should prioritize simplicity and ease of use.
- **Interactivity**: Real-time updates, intuitive controls, and customizable layouts to enhance user engagement.
- **Customization**: Users can personalize the dashboard by adding/removing graphs and selecting specific telemetry metrics.
- **Persistence**: Custom designs and preferences created by the user are saved in the config file, allowing users to retain their settings for future sessions.

UI Mockups:

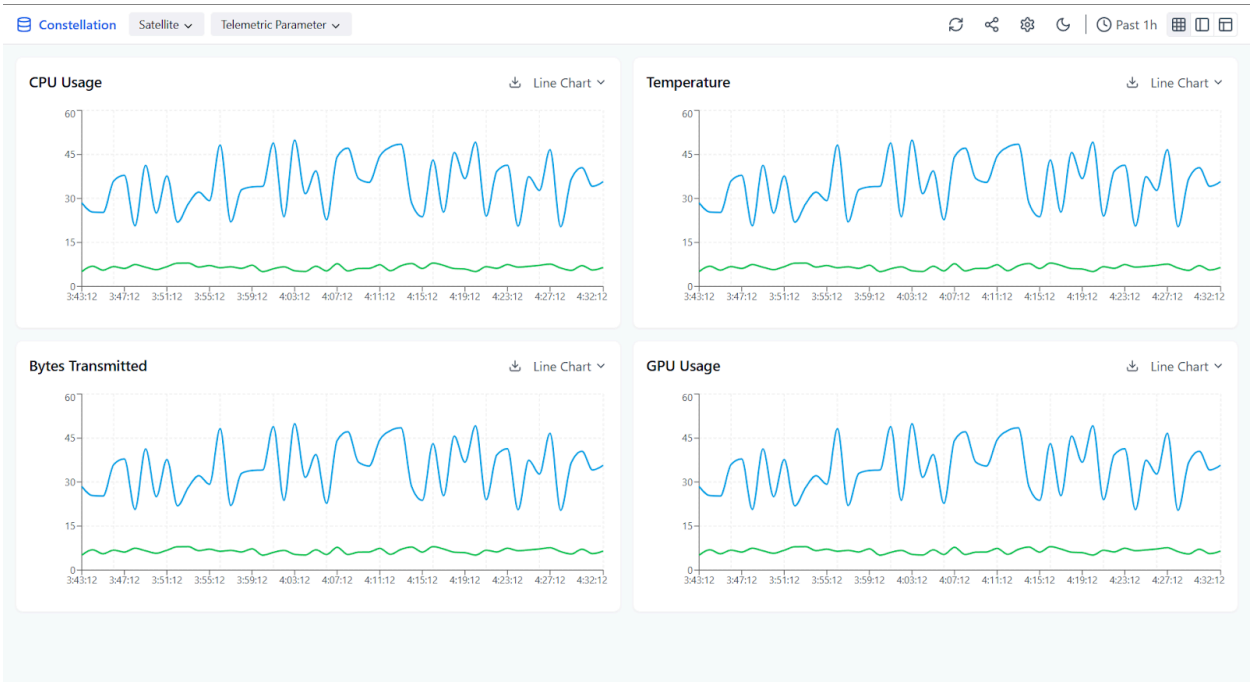

Figure 1: Grid layout(Light Mode)

Figure 2: Grid layout(Dark Mode)

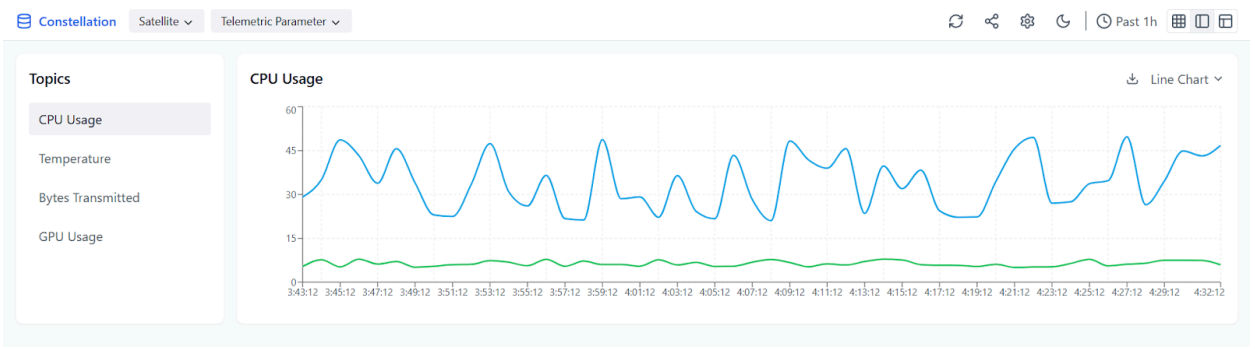

Figure 3: Sidebar layout

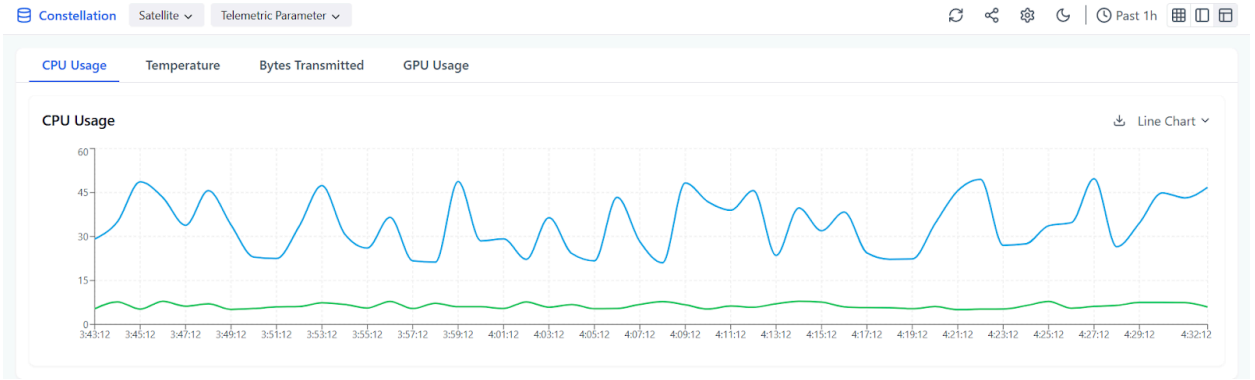

Figure 4: Tabbed layout

UI Components: The Constellation window can be divided into 2 main parts

- ➔ **Header Section**:
  - The top area will have a dropdown menu for selecting the satellite and telemetric parameters.
  - It will also include:
    - **Refresh Button**: To refresh and pull the latest Telemetry Data
    - **Share Button**: To provide an option to export the data or view for collaborative purposes.
    - **Settings Button**: To open a menu for users to configure various settings for the application.
    - **Dark mode Button**: To toggle dark mode (Fig 2).
    - **Time Frame button**: To adjust the time frame for the data being displayed.
    - **Layout Button**: To let users customize the layout of the window and choose between a Grid Layout(Fig 1,2), Sidebar Layout(Fig 3) or a Tabbed Layout(Fig 4).
- ➔ **Graph Area**:
  - A large area to display multiple graphs representing different telemetry metrics. Each graph will update in real-time as data is received.This will also contain:
    - **Download Button**: To download the charts.
    - **Dropdown Menu**: To switch between different data visualization types, such as Line Chart, Bar Chart, Pie Chart, and more.

3.5 Technical Approach

Qt Classes to be used:

- ➔ **QGraphicsView:** For handling dynamic graph updates and rendering the real-time telemetry data.
- ➔ **QComboBox:** For selecting satellites and telemetry metrics.
- ➔ **QWidget and QGridLayout:** For arranging graphs dynamically within the UI.
- ➔ **QTimer:** To update the UI periodically with new data from the telemetry sources.
- ➔ **QSettings:** To store and retrieve user preferences, such as graph layout and metric selection, in the Qt application config file.
- ➔ **QtCharts**: To display telemetry data in real-time using line graphs, bar charts, or pie charts.

Integration of CMDP:

The **CMDPListener** class subscribes to satellites using the **CMDP protocol** and listens for **STAT** topics, providing telemetry data for visualization in the UI. The **StatListener** class

(derived from **CMDPListener**) will handle these subscriptions and message reception, making the telemetry data available to the application.

- ➔ **Subscription to Topics:** The CMDPListener subscribes to various STAT topics provided by satellites, and this data is received asynchronously. The UI will utilize QComboBox for users to select the desired satellite and telemetry metric.
- ➔ **Real-Time Updates in the UI:** The data received from the CMDPListener will trigger updates in the UI. The telemetry values will be plotted on graphs using QtCharts or QCustomPlot, with the QTimer ensuring that the UI is updated regularly to reflect the latest received telemetry.
- ➔ **Dynamic Topic Management:** Users can interact with the list of available STAT topics through dropdown menus or checkboxes, allowing them to subscribe or unsubscribe to specific data streams. The CMDPListener will handle topic subscriptions, and any changes in the available topics will be reflected in real time.

Build System Integration:

Meson will be utilized as the build system to streamline the project's compilation and dependency management process. Meson, combined with Ninja as the default backend, offers high performance and parallel execution, ensuring quick builds. Meson's syntax is declarative, and it supports efficient configuration and build processes across multiple platforms, including Linux and Windows.

The build configuration will involve specifying **Qt 6** as the primary dependency using the Qt6 Meson module, which simplifies the integration of Qt's various libraries, such as **QtWidgets** for the user interface and **QtCharts** for dynamic graph rendering. Meson handles dependency resolution and ensures that the required versions of libraries like CHIRP and CMDP are correctly linked, providing a seamless integration process.

Reusable UI Widgets:

- ➔ **GraphWidget**: A reusable class for displaying metrics using QtCharts or QCustomPlot for line, bar, or pie charts. It will handle real-time updates via QTimer or QThread.
- ➔ **TelemetryManager**: Manages subscriptions to STAT topics from satellites using the CMDP protocol, and processes incoming data for display.
- ➔ **Layout Management**: A QGridLayout will arrange the graphs dynamically, allowing users to add, resize, and remove them. Graph placement and settings will be stored in QSettings.
- ➔ **MetricSelector**: Allows users to choose telemetry metrics and graph types, which are then linked to the corresponding GraphWidget.

Development Process:

The development process will be managed using **GitLab** for version control and **continuous integration (CI)**. The CI pipeline will automatically build the project, run tests, and perform static analysis with clang-tidy.
Additionally, GitLab CI will ensure that each commit meets the project's quality requirements, including proper code style and functionality checks.

3.6 Testing and Quality Assurance

Testing will be a crucial aspect of this project to ensure the system is reliable, performs as expected, and provides a seamless user experience. The testing process will be carried out throughout the development phases, focusing on functional correctness, performance, usability, and integration with the Constellation framework.
The testing process will be divided into several phases to ensure the system's integrity from initial development through final deployment.

- ➔ **Unit Testing:** Test individual components like the StatListener class, UI elements, and service registration in isolation using Qt's unit testing framework to ensure correct functionality.
- ➔ **Integration Testing:** Verify seamless integration between the StatListener class, data subscription system, and UI, ensuring real-time data flow and accurate telemetry stream handling in QCharts.
- ➔ **System Testing:** Test the system in a controlled environment with multiple telemetry streams, ensuring it handles simultaneous updates and maintains UI stability under heavy data load.
- ➔ **Performance Testing:** Test the system's performance under high-frequency data updates and multiple telemetry sources, measuring memory usage, CPU utilization, and response times to identify and address any performance bottlenecks.
- ➔ **Usability Testing:** Conduct usability testing with end-users if possible to gather feedback on the dashboard design, layout, and functionality, ensuring easy navigation, clear data visualization, and quick access to telemetry data.

3.7 Use Cases

- ➔ **Use Case 1: Monitoring System Health**

Scenario: An experiment operator wants to monitor the health of the experiment's computing resources. They open the telemetry dashboard to view metrics such as CPU usage, temperature, and memory consumption across multiple devices.
Expected Outcome: The operator can quickly glance at different graphs for each satellite (device and be alerted if any value goes out of expected bounds, aiding in troubleshooting before problems escalate.

➔ Use Case 2: Monitoring Data Transmission

Scenario: A scientist is conducting an experiment with multiple particle detectors and needs to ensure data transmission is consistent. They select the "bytes transmitted" metric and see live updates as each satellite's transmission performance is visualized.
Expected Outcome: Real-time updates provide immediate feedback on the health of the experiment's data flow, reducing the chances of data loss during critical moments.

➔ Use Case 3: Customizing the Dashboard

Scenario: A user wants to monitor specific telemetry metrics for their unique experiment setup, such as power consumption across different devices. They can drag and drop different graphs and select which metrics to visualize from the available satellites.
Expected Outcome: The dashboard is tailored to their needs, and the configuration is saved, allowing them to return to the same view later.

➔ Use Case 4 : Monitoring Data Quality in Real-Time

Scenario: An experiment operator wants to ensure the quality of the collected data during a time-sensitive experiment, such as a particle accelerator. They use the Qt interface to quickly access real-time telemetry metrics, such as instrument temperatures, to identify potential issues early.
Expected Outcome: The operator can immediately spot anomalies in data quality and take corrective action before the experiment suffers. The simplicity of the Qt interface allows for rapid monitoring without requiring complex setup like Grafana, providing real-time feedback with minimal effort.

➔ Use Case 5 : Multi-Experiment Monitoring

Scenario: A facility operates multiple experiments simultaneously, each requiring monitoring of different sets of telemetry data. The operator uses the interface to

monitor various experiments from a single dashboard(Fig. 1). They can switch between experiments and view the key metrics for each in real time.
Expected Outcome: The operator can efficiently track multiple experiments without needing to constantly switch between different systems or interfaces. This centralization makes it easier to manage and oversee concurrent operations.

3.8 Project Timeline

Pre Gsoc Work	➔ Completed the task given: Build a simple Qt application.
Up Till May 8, 2025	**Proposal accepted or rejected** ➔ Setup Constellation and familiarise myself with the codebase. ➔ Discuss with the mentors about the doubts related to implementation. ➔ A proper project structure will ensure smooth development.
May 8 to June 1	**Community Bonding Period** ➔ Ensure Constellation is set up properly for development on my local machine. ➔ Fix a few issues to familiarise myself with the Codebase. ➔ I will be having my End Semester Exams from April 30 to May 20, so although progress may be slow, the work will be completed in time.
Week 1 & 2 (Phase 1) June 2 to June 15	**Coding officially begins!** ➔ The project will start with understanding the existing infrastructure reviewing the CHIRP and CMDPListener codebase to understand the current implementation and then implementing the StatListener class using this existing infrastructure
Week 3 & 4 (Phase 2) June 16 to June 29	➔ The next part will be to develop a layout for the dashboard with Qt. ➔ Set up data flow from telemetry topics to the UI.
Week 5 June 30 to July 6	**Buffer Period** ➔ Try to complete any task mentioned before if not completed or if any unforeseen situation arises.

	➔ Coordinate with mentors, reinforcing the tasks to be done ahead.
Week 6 & 7 (Phase 3) July 7 to July 20	➔ Further I will proceed with implementing the ability to add/remove graphs dynamically. ➔ Also implementing user customization of the dashboard (selecting satellite, metrics, graph types).
Week 8 & 9 (Phase 4) July 21 to August 3	➔ Store user configurations in the Qt application config file. ➔ Integrate the new UI with the existing Constellation system.
Week 10 & 11 (Phase 5) August 4 to August 17	➔ Conduct extensive testing with real telemetry data from devices. ➔ Complete the final testing and bug fixing. ➔ Create the proper documentation of the development process and the user interface.
Week 12 August 18 to August 25	**Buffer Period** ➔ Complete any remaining tasks, fix any bugs that may arise previously. ➔ Try to fulfil all necessary requirements of mentors regarding the project.
Week 13 August 25 to August 31	**Final Submission!** ➔ Submit final project, mentor evaluation, and write a blog post summarizing implementation.
Until Pens Down	➔ Work on documentation and blog posts about my work.
Post Gsoc	➔ Continue to develop and improve the new UI and contribute to the project in general.

4. References

- Constellation source code
- Constellation documentation
- Qt Reference Documentation