National Centre for Radio Astrophysics

NCRA • TIFR

Student Training Program 2024-25 Report

STP/2024/022

"Upgrading the Python-Based GUI Framework for the TGC (Tango based GMRT Control) System

(Duration: Dec 2024 -March 2025)

By

Mayur Jitendra Bhagade.

Under Supervision of:

Mr. Jitendra Kodilkar, Mr. Anil Raut

Objective of Project : The project "Upgrading the Python-Based GUI Framework for the TGC (Tango-based GMRT Control) System" focuses on modernizing the existing GUI by transitioning from Python 2 and PyQt4 to Python 3 and PyQt5. Since Python 2 reached its end-of-life in 2020 and PyQt4 is no longer supported, this upgrade ensures long-term stability, security, and compatibility with evolving technologies. The modernization will enhance performance, usability, and maintainability, making the system more robust for future developments. Key tasks include updating core libraries, refactoring the codebase, and implementing improved features to meet current and future operational needs

Revision	**Date**	**Modification/Change**
Ver. 0.9	March 25, 2025	Initial Draft version
Ver. 1.0	April 11, 2025	Revised version with Validation testing updates, and reviewed
Ver 1.1	April 16, 2025	Final Submission

Acknowledgement

I would like to express my profound gratitude to Mr. Jitendra Kodilar for giving me the opportunity to work on the "Upgrading the Python-Based GUI Framework for the TGC (Tango based GMRT Control) System" project. I also extend my heartfelt thanks to Mrs. Shubhangi Said, the head of the department, for allowing me to undertake this project.

I am deeply grateful to the NCRA Director, Prof. Y. Gupta, the GMRT Dean, Prof. Ishwar Chandra, and the Telemetry-Operation Group Coordinator, Mr. Anil Raut . I would also like to thank the NCRA and GMRT staff, including Mrs. Deshmukh, who assisted me throughout my project.

Working at GMRT was a wonderful experience. I had the opportunity to learn more about GMRT's work culture and to interact with the amazing people there. Through this project, I gained valuable insights into the Local Monitoring and Control System (LMC) for the GMRT Antenna. This experience has significantly enhanced my technical skills in python, UI and UX. And it's understanding of real-world applications in radio astronomy.

Executive Summary

- **Feasibility Study:** Upgrading the Python-Based GUI Framework for the Tango-based GMRT Control (TGC) System

This feasibility study explores the modernization of the **Python-Based GUI Framework** for the **Tango-based GMRT Control (TGC) System**. The current system uses outdated technologies, including **PyQt4** and **Python 2.7**, which have reached their end-of-life. Additionally, the GUI framework was initially implemented using repositories from **Ubuntu 16.04 LTS**, while the GMRT Local Monitoring Computers (LMCs) now operate on **Ubuntu 20.04 LTS**.

To ensure long-term support, improved security, and enhanced performance, this project proposes upgrading the framework to the latest versions of **PyQt5 or PyQt6** and **Taurus** using **Python 3**. The upgrade will also facilitate compatibility with the upcoming transition to **Ubuntu 22.04 LTS**.

The modernization will involve refactoring the existing codebase, integrating updated libraries, and ensuring seamless operation with the Tango-based control system. By achieving this, the project will enhance system stability, maintainability, and user experience while extending the system's operational lifespan.

Objective: The project "Upgrading the Python-Based GUI Framework for the Tango-based GMRT Control (TGC) System" focuses on modernizing the existing GUI by transitioning from Python 2 and PyQt4 to Python 3 and PyQt5. Since Python 2 reached its end-of-life in 2020, and PyQt4 is no longer supported, this upgrade ensures long-term stability, security, and compatibility with evolving technologies. The modernization will enhance performance, usability, and maintainability, making the system more robust for future developments while ensuring seamless integration with the Tango-based GMRT control system. Key tasks include updating core libraries, refactoring the codebase, and implementing improved features to meet current and future operational needs.

- **Methodology :**

On the latest Ubuntu 22.04 LTS, a feasibility study was done using the Tango Tool-kit based framework using the TANGO (cpp and PyTango) 9.4.2 and JTango (java) 9.7.2 libraries installed in the CONDA environment. Java Tango uses openjdk 21.1.0 for the Tango-framework and openjdk 1.8.0 for the LMC. For cpp-tango gnu compiler 13.3.0 is used and for python 3.9.18 is used. Using the PyQt 5.15.9 and the Taurus 5.2.0 framework based on PyQt5 is used for the GUI realization of the LMC.

The Scripting environment from the python-2 to python-3 shifted using the tool ***'2to3-3.9'*** python converter, and then manually editing the sixty python files for ***PyScriptManager*** Tango Device-server for the scripting, and ***UploadBox*** which consist of various script-files to be run for the LMC Tango-commands.

For the GUI environment, PyQt4 and Taurus5.2 python source-code base are converted to PyQt5 and Taurus-5 using the ***pyqt4topyqt5*** PyQt4 to PyQt5 valid-source code converter. This inevitably needed to change nearly 101 python-codes and verified 61 UI files in the PyQt5-designer.

The major changes in the python, PyQt5 and Taurus5.2 after conversion was associated with space/indentation, utf-8 and python-3 functionality changes, and the signal handling for PyQt4 to PyQt5 differences along with the Fully Qualified Domain Name related changes in the Taurus Model.

- **Findings:**

1. LMC validation testing : Variety of the Python3, PyQt5 and Taurus-5.2 code validation and functional testing performed which found to be working like the legacy LMC code which was based on Python2 and PyQt4, Taurus 3.7.

Thus, the upgraded framework is fully compatible with Ubuntu 22.04 LTS, ensuring seamless integration with the latest operating system updates.

2.Extended Support and Maintainability : Since **PyQt5** and **Python 3** are actively supported and maintained, long-term software support and maintenance is now ensured.

3. User Experience Enhancement : A refined and modernized GUI with improved graphical elements ensures that the LMC can be used in the new environment.

Acronyms

AN	Aggregation Node
API	Application Program Interface
CEB	Central Electronic Building
CG	Collaboration Group
CMC	Central Monitoring and Control
CORBA	Common Object Request Broker Architecture
CPP	C++ Programming Language
CPU	Command Processing Unit
DB	Database
DPU	Data Processing Unit
DS	Device Server
EMF	Eclipse Model Framework
EN	Element Node
EPICS	Experimental Physics and Industrial Control System
GAB	GMRT Analog Backend
GMRT	Giant Metrewave Radio Telescope
GSB	GMRT Software Backend
GUI	Graphical User interface
GWB	GMRT Wideband Correlator
HDB	Historic Database archiver
HTML	Hypertext Mark-Up Language
I/O	Input/Output
IDE	Integrated Development Environment
LDAP	Lightweight Directory Access Protocol

LMC	Local Monitoring And Control
M&C	Monitoring and Control
PANIC	Package for Alarms and Notification of Incidences from Controls
PET	Power Equalisation Tool
PoC	Proof of Concept
POGO	Programme Obviously used to Generate Objects
SCADA	Supervisory Control and Data Acquisition
SKA	Square Kilometre Array
SRS	Software Requirement Specifications
TACO	Telescope and Accelerator Controlled with Objects
TANGO	TACO Next Generation Objects
TCS	Tata Consultancy Services
TDB	Temporary database
TGC	Tango based GMRT Control System
TIFR	Tata Institute of Fundamental Research
TM	Telescope Manager
UI	User Interface
UoP	University of Pune
URS	User Requirement Specifications

INDEX

5. Summary :

5.1 Future scope of the work

References

1. Introduction

1.1 About the GMRT

The Giant Metrewave Radio Telescope (GMRT) is the largest radio telescope at metre-wavelength near Narayangaon, Pune in India. It's made up of thirty giant dishes, each of 45 meters in diameter, that can be moved in elevation and azimuth. The National Centre for Radio Astrophysics (NCRA) of the Tata Institute of Fundamental Research (TIFR),Mumbai founded and operates it. The telescope was designed and built under the leadership of Govind Swarup between 1984 and 1996. The GMRT telescope uses all 30 dishes separated by 25 km maximum distance for longer baselines, and less than 1 km for shorter baselines for earth rotation aperture synthesis (Interferometry technique) for making the radio images of celestial objects as well as study timing signals of celestial objects such as pulsars, quasars, high-gamma ray burst and neutron stars. The TIFR GMRT Sky Survey (TGSS) is a major project that used the Giant Metrewave Radio Telescope (GMRT) to map almost 90% of the sky at a frequency of 150 MHz (wavelength of 2 meters).

- **Antennas and subsystem :-**

The GMRT antennas receives Radio signal ranging from 150 to 1600 MHz signals in maximum bandwidth of 400 MHz using five sub-bands (Band-2 150 MHz, Band-3 300 to 550 MHz, Band-4 550 to 850 MHz , and Band-5 1000 to 1550 MHz) using variety of conical di-pole feeds. The receiving chain processes this signal using the GMRT Analog backend (GAB) system to make the signal at the operating region, select band-width filters from 100, 200 , and 400 MHz. The individual antenna spectrum signals are processed after digitization to make correlation (antenna signal products) , and beam-formers (addition of signal after putting delays) in the GMRT Wide-Backend (GWB) system.

The GMRT has a servo system with three layers of control to achieve very precise pointing, with an accuracy of about 1 or 2 arc minutes. The elevation axis is placed on top of the azimuth drive, which means the elevation drive can move the dish up and down, while the azimuth drive can turn it left and right. This setup allows the antennas to point at any location in the sky.

At the antenna base, individual sub-systems ranging from Front-end electronic system (at the turret associated with antenna-feeds), optical fiber, sentinel system and servo system are monitored and controlled using a modern computer which operates on the Linux operating system (Ubuntu). From the Central-control room, the Tango-based system control system coordinates all antennas operation to conduct the science observing and engineering experiments and testing sessions in an automatic way.

- **Purpose of GMRT :**

GMRT is a low-frequency radio telescope that helps investigate various radio astrophysical problems ranging from nearby solar systems to the edge of the observable universe. Radio telescopes detect and amplify radio waves from space, turning them into signals that astronomers use to enhance our understanding of the Universe. To detect the highly red shifted spectral line of neutral Hydrogen expected from protoclusters or proto-galaxies before they condensed to form galaxies in the early phase of the Universe is one of the objectives as well as study the variety of celestial objects such as pulsars , neutron stars, magnetic fields of galaxies and intergalactic medium .

1.2 Tango Based GMRT Control System

The **TANGO** control system is a free open source device-oriented controls toolkit for controlling any kind of hardware or software and building SCADA (Supervisory Control and Data Acquisition) systems. It is used for controlling synchrotron accelerators, lasers physics experiments. It is being actively developed by a consortium of more than 20 research institutes across the globe. TANGO tool-kit is also adopted as modern control-system for the Square Kilometer Array project (SKA) which is an international telescope build in Australia (Low frequency Array), and South Africa (Mid-frequency dishes) by the international consortia where the NCRA is play a key role in design and development of the Monitor and Control system for the SKA.

TANGO is a distributed control system. It runs on a single machine as well as hundreds of machines. TANGO uses two network protocols - the ***omniorb*** implementation of **CORBA (Common Object Request Broker Architecture)** and **Zeromq (** Message Passing system**)**. The basic communication model is using the client-server. Communication between clients and servers can be synchronous, asynchronous or event driven. CORBA is used for synchronous and asynchronous communication, and Zeromq is used for event-driven communication.

The concept of Tango Controls was developed in the European Synchrotron Radiation Facility (ESRF) in Grenoble, France about 20 years ago. Since then, the work on the core of the toolkit has begun. Though Tango Controls has proven itself as a mature toolkit with many users, as an open source Tango Controls toolkit is always being improved.
The software used in synchrotron facilities can be compared with the software for stock markets – huge amounts of data must be displayed on the monitor in real time and being processed and being saved in databases for post processing. As Tango Controls has existed for more than 20 years and becomes more and more popular among facilities, it has proved itself as a reliable toolkit. So, the toolkit was mainly developed for research facilities needs, but the idea and concept (philosophy) behind it was to create a framework. Tango Controls has been enriched with many applications (desktop and web based) that can satisfy almost all instrumental control and user needs. That means Tango Controls not only as a device-oriented controls toolkit and to write applications but also use it as a final product. Among Tango Control-kit applications, real-time alarm with alarm status, historical/real-time data monitoring , and Archiving is also available with the Web based UI. Tango Controls is a hardware independent toolkit. That means you can use your driver to connect hardware with Tango Controls. Taking that Tango Controls is

used in many facilities for many years, with high probability you can find the driver you need in a special device catalogue and use it for free.

1.2.1 What is Tango ?

The TANGO was developed in 1999 based on TACO (Telescope and Accelerator Controlled Objects), the next generation tool-kit based on object oriented is called Telescope and Accelerator Next generation Objects i.e. TANGO. Every control system, whether it is hardware or software system , is considered as an instance of a Tango Class (C++/Java or Python) where the instance/object is considered as a Tango device server (Tango DS) which is nothing but the Class instance or object.

The Tango Device-server can be modelled at high level as a command (Methods) and attributes (monitoring and control parameters) using a code-generation tool which generates a high level code automatically based on user choice in C++, Java or Python language. Every device across is having a unique name as Universal Resource locator across the network. The State of devices, monitoring attributes and the command of devices are exposed over the network at high level where devices can be fetched/accessed across the TANGO communication network based on CORBA. **Figure 1.1** shows a

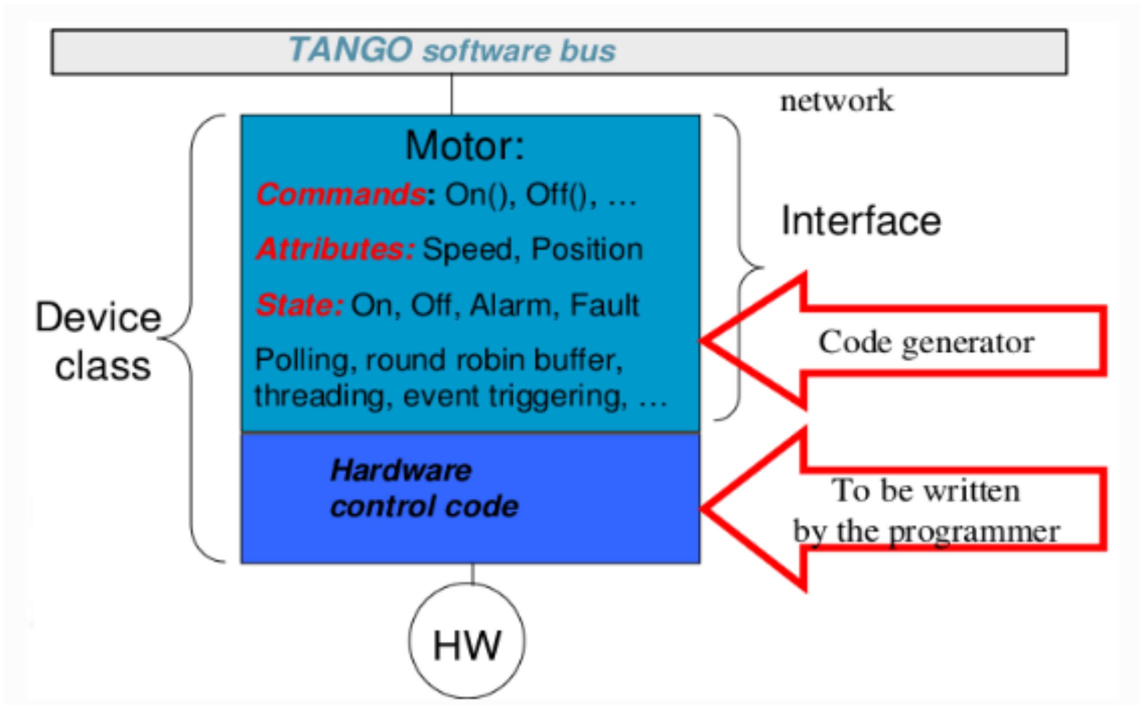

TANGO software bus
network
Motor:
Commands: On(), Off(), ...
Attributes: Speed, Position
State: On, Off, Alarm, Fault
Polling, round robin buffer, threading, event triggering, ...
Interface
Code generator
Device class
Hardware control code
To be written by the programmer
HW

Figure 1.1 Tango Device Server Model

Tango device-server model , The Tango Device Class or its instances can expose commands (for e.g. instrument ON or OFF) , attributes (Motor speed and encoder position as device’s attributes), and the instrument states over a Tango software bus which can be accessed for the instrument control.

A Tango-device server can be modeled for the commands, attributes and state etc using a automatic code generator tool (POGO), and lower level APIs to these device methods or attributes value set/read code shall be inserted by the s/w developer to communicate to the instrument under control.

Figure 1.2 shows a Tango Client-Server Model working. The Tango device at high level has the control-command along with the control-parameters (attributes), and the monitoring attributes have to be realized by writing a low level interface code which can communicate to the hardware instrument or software system.

The TANGO Client Server model work on following components :

(1) The Tango Database Device Server (DatabaseDS) :

The Tango Database server is nothing but the MySQL (Or Non MySQL File based also possible) where one can register the TANGO device server with the unique name so that the TANGO instance object reference is available over the TANGO communication bus.

(2) **TANGO Device Server :** Every Tango Device Server is nothing but the Object or instance of Tango Class exposed over network using the registered unique TCP/IP with fully qualified domain.

tango://<machine-name>:<Port>/Domain/Family/Device

For e.g. tango://e01:10000/MNC/LMC/E01

Where,

- *" e01" is host-name, network port is 1000,*
- *Domain is MNC - monitoring and Control Node*
- *Family is Local Monitoring Control (This can be regarded as a device type)*
- *E01 is device name, in this case testing antenna named 'e01'*
-

(3) **TANGO Client** : Tango Client can access the desired device using the Fully qualified domain name of the server by queuring the Tango-device DATABASE server for the resource locator ID of the desired device. Once it got the object reference of a Tango-device , then it can communicate to the Tango device (h/w instrument or software program/system) for monitoring parameters , knowing the operating state of the device, and controlling the device using the command.

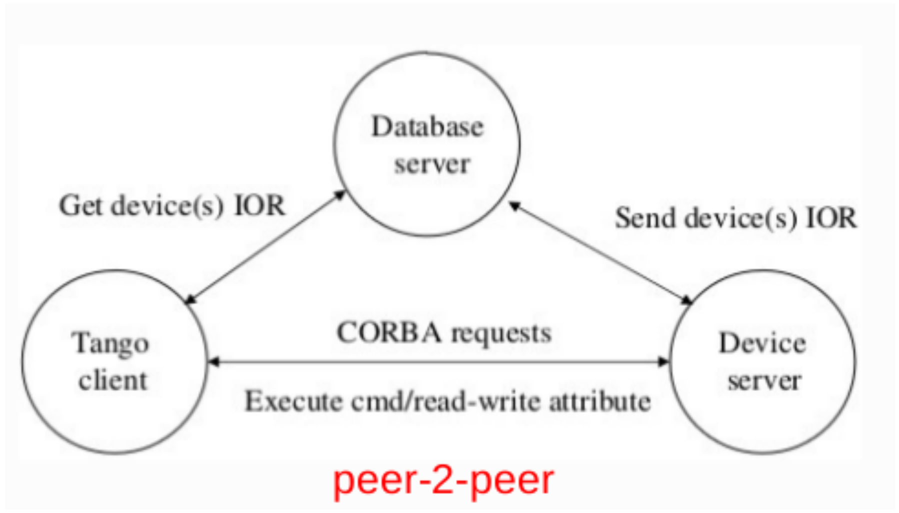


Figure 1.2 Tango Client-Server model.

Event-Based: Tango supports event-driven programming. Devices can send updates to clients asynchronously, allowing for real-time monitoring of device states and attributes.

Example Use Case:

Tango is often used in large scientific experiments where multiple devices need to be controlled and monitored from a central location, such as controlling and monitoring the operations of a telescope array, like the **GMRT (Giant Metrewave Radio Telescope)**.

1.2.2 TANGO Tool-Kit :

Tango is a comprehensive control system framework used to control and monitor hardware or software systems in real-time, typically in scientific and industrial applications. The TANGO system is a ready to use tool kit which comprises the generic tools required for the any SCADA kind of system such as a thick and thin client user-interfaces, Control and Monitor h/w or s/w system parameters it's operation state, archive the data of system under control, and Alarm-management system to notify the user about the warning/risk situations of the system.

Table-1 : Details Associated with the TANGO tool-kit .

SR NO.	Tango Tool-kit components	TANGO Tool	Remarks
1.	Alarms Management	PANIC(Package for Alarms and Notification of Incidents from Controls) and TANGO Alarms Suit	PANIC alarms suit is found more suitable.
2.	Data Archiving and Retrieval	Historical Data-base (HDB++)	It is used for data archiving in MySql as well as it supports non-mysql data as well.

3.	State Handling and Implementation	POGO (Program Obviously to Generate Objects)	For addressing M&C configurability needs, we need to adapt an alternate approach for specifying the state transition and so on.
4.	Command and Response	TANGO Commands and Attributes	Custom protocol is developed to address the GMRT M&C specific needs.
5.	Specification Driven Approach	TANGO DB, POGO, JIVE	Custom Database (DB) and configuration files are also used.
6.	Scripting	PyTango (Python API to TANGO)	Realised using Jython, Cython and PyTANGO
7	GUI	PyQT and Taurus frame-work for the GUI based on PyQT	The GUI thick client application consists of a variety of Widgets.

1.3 Local Monitoring Control (LMC) System - A Generic Control Monitoring system

The TANGO based Local Monitoring Control (LMC) System runs on a Linux based PC at the antenna base. The LMC system is designed as configuration or specification driven generic control-node architecture. The configuration or specifications defined in the Tango database are

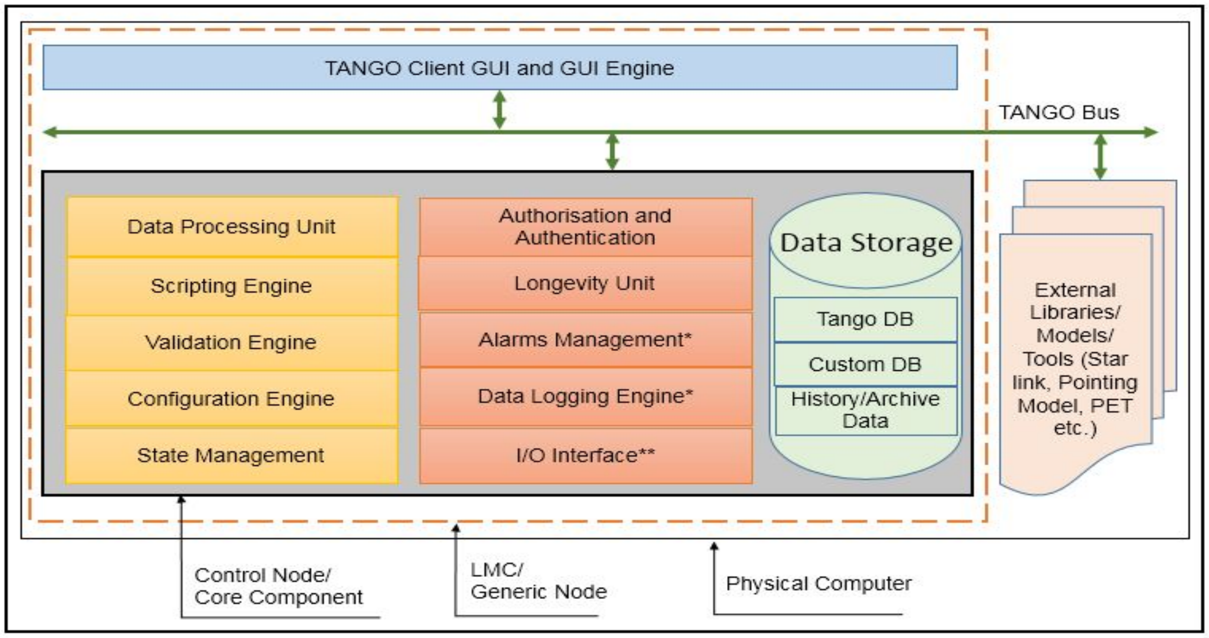


Figure 1.3 A Generic Control-Node Architecture.

used to realize the control-system for individual GMRT Antennas. The LMC s/w is same at every antenna along with the antenna-wise different configurations such as Antenna-base PC host-name and IP, Tango Fully qualified domain name and sub-systems configured under it.

The generic control-node software of the LMC read the TANGO database and identify the control role such as for which antenna (for e.g. C00, C01, E02, E03, or S02,S03.. etc) it is controlling. A Generic control node uses a various TANGO tool kit components to coordinate, monitor and control the various subsystems at the antenna base viz Servo, Frontend Commo-box (FECB), Optical-Fiber Sentinel System (FECB), Feed Positioning system (FPS), and signal conditioning system (IF-LO). The LMC Control node acts as a perent while controlling the child subsystems at antenna base, and communicates at the higher level Central Tango control-system running in the GMRT Control-room. Thus, the LMC is running at middle tier level in the GMRT architecture where it receives command from the Control-room and these commands are parsed and sent to the actual sub-systems of the antenna such as servo, FECB, OFCSNT etc. The LMC system also sends monitoring parameters , operational state, events and the Alarm-data at the control-room.

A few important modules or component from a TANGO-toolkit used in the LMC and antenna-base sub-system control & Monitoring are described here briefly as follows :

(a) **Data Storage :** Data storage uses MySQL database to store the three types of database. **(i) Tango Database** - It is used to register the Tango Device-server IDS running for the antenna, and its subsystems. Each Tango Device-server is registered and declared across the tango-bus for a unique-name for accessing.   **(ii) LMC Custom-database :** This database mentions specification for command, monitoring parameters, valid limits for control-parameters, alarming-rule the Antenna, servo, FPS, FECB and OFCSNT system. **(iii) Archiving database :** The monitoring data, command-response, and events are stored in the HDB++ archiving data for retrieval purposes.   For Archiving a separate HDB++ data-archive Tango device-server is running.

(b) **Alarm Management :**   The PANIC tool from the TANGO database is used to define the Alarm-rules based on which alarms can be generated and notified to the control-room for the antenna and individual subsystem for safety purposes. For this purpose PyAlarm Tango-device server is run by the PANIC tool.

(c) **Scripting Engine** : Scripting engine uses Python script to execute the algorithm required to realize the command. The MNCScriptManager Tango-device server developed by the GMRT is running for each subsystem of the antenna, and antenna LMC itself . The user level command in a batch kind of format using a python-file or script can be executed using the MNCSCript Manager Device-server.

(d) **Configuration Engine** : Configuration engine module is responsible for reading the configuration of TANGO, LMC_Custom_database to realize the LMC and

Sub-system I/O Tango device instances. Also, command & command arguments , monitoring limits are validated using the configuration engine.

(e) **Validation Engine** : Validation engine module uses command-syntax, command-arguments etc to validate the received command.

(f) State-Engine : State Engine mainly responsible to declare the sub-system devices or LMC device state whether it is initialized, running, maintenance, or shutdown. In particular STATE, only some commands are allowed. Also valid state-transitions like from init to run , run to maintenance or run to shutdown are allowed by the state engine.

(g) **Longevity Unit :** This module keeps the life-cycle management of Tango devices i.e. it starts or stops/shutdown the sub-system and LMC Tango-devices. Keep checking the running state of the devices.

(h) Authentication & Authorisation : Based on the Kerberos library , users are validated using the authentication, and for specific users (for e.g. servo engineer) only certain system commands (e.g. servo) are allowed.

(i) **External Tools or Libraries :** This supports integration with external resources and domain based s/w modules or libraries. Libraries such as **Starlink** ,and **PET** (Performance Evaluation Tools) provide additional functionality for the antenna operation. **Pointing Models** are used in systems like GMRT to ensure precise antenna positioning.

(j) **TANGO Client GUI and GUI Engine :** At the top of the architecture is the **TANGO Client GUI and GUI Engine**.It provides a graphical interface for users to monitor, control, and interact with various subsystems.It sends and receives commands using the **TANGO Bus**. The GUI engine ensures a seamless connection between the user interface and the backend components. The GUI is developed using the Python based PyQt and Taurus (PyQT based thick client GUI framework) .

Thus , based on generic configuration driven architecture (**Figure 1.3**), a Sub-system Tango Device servers (FPS, FECB, Servo, SIGCON, OFCSNT, and GAB systems) are developed in the C++ Tango framework, the LMC is developed in the JAVA Tango (JTango) Framework, and the scripting environment, and GUI uses a PyTango (Python interface to c++ tango).

1.4 GUI and Scripting Environment (PyScriptManager) :

A Graphical User Interface (GUI) in Python is a user-friendly interface that allows users to interact with applications using visual components like buttons, menus, windows, and text fields. Python offers a variety of libraries and frameworks to create robust and interactive GUI applications across different platforms.

- **Python's Role in LMC System :** Python plays a significant role in the development and operation of the GMRT LMC system. In the beginning the LMC system was developed on ***Ubuntu 16.04 LTS.*** The compatible python available in the Ubuntu repository was *Python 2.7.12 d*uring that time (2016-17), also for the GUI *PyQt4 4.8.7* and *Taurus 3.7.0* were used. Also, python is used in the Scripting environment to run the algorithms required for command via the python script or execute the batch of command using the python scripting. For this purpose it uses a PyTango Python-Library to access the Tango device. The *MNCScriptManager* scripting Tango-device itself uses PyTango 8.1.8 version. The PANIC alarming Device-server or tool is also based on python.

Therefore, as the LMC python based libraries are almost decade old, and either existing version is not available on the new Ubuntu OS , or no more supported since 2019-20.

A feasibility study is taken to upgrade the python 2 to 3 , along with this PyQt5 and Taurus 5.2 is also mandatory to change because PyQt4 GUI don't support python-3 (or up to 3.6 only which is deprecated version of Python-3 it self), also Taurus uses PyQt libraries for the API. Hence it became mandatory to change PyQt4 to PyQt5 and the Taurus 5.2 version while shifting from Python-2 to Python-3.

• Python based development for the GUI and Scripting -

Using Python is widely used in the many domain areas which includes data-analysis, data-science, modeling, even in the Web-framework, and of course user-interface and for the scripting. Thus main features of the Python is as follows :

- Simplicity and Readability: Python's clean and readable syntax makes it an excellent choice for GUI development.It reduces development time and makes debugging easier compared to other languages.

- Cross-Platform Support: Python is compatible with Windows, Linux, and macOS.Libraries like PyQt and Tkinter ensure platform-independent GUI applications.

- **Rich Set of Libraries :** Python offers various GUI frameworks such as PyQt / PySide: Based on Qt, ideal for building complex, professional-grade applications.

Tkinter: Lightweight and easy-to-learn, suitable for small applications.

Kivy: Supports multi-touch and cross-platform mobile development.

-Integration with Other Technologies: Python seamlessly integrates with databases (MySQL, SQLite), APIs, and data visualization tools (Matplotlib, Plotly).It's suitable for real-time monitoring systems like the GMRT LMC.

2. Objective of the LMC Upgradation

The objective of the GMRT Python upgrade project is to modernize the Tango-based GMRT Control (TGC) System by migrating it from Python 2 to Python 3, ensuring compatibility with modern libraries and tools. This upgrade aims to enhance system performance, maintainability, and security by addressing vulnerabilities associated with outdated software. It will ensure seamless integration with the latest versions of the TANGO Control System, while also improving the graphical user interface (GUI) for a better user experience. Additionally, the project focuses on providing a scalable and future-proof architecture to support further enhancements and minimize system downtime during astronomical observations.

2.1 Why Upgradation is needed specifically python and PyQT ?

The upgrade from Python 2 to Python 3 and PyQt5 (or later versions) is essential because **Python 2** reached its **end of life** in **January 2020**, meaning it no longer receives updates, security patches, or support. Continuing to use Python 2 exposes the system to security vulnerabilities and compatibility issues. Additionally, many modern libraries and tools are now exclusively compatible with **Python 3**, offering better performance, improved memory management, and new language features that enhance code efficiency and maintainability. **PyQt** also requires upgrading since older versions may not be compatible with Python 3. Upgrading to **PyQt5** or **PyQt6** ensures access to modern UI components, enhanced graphics rendering, better cross-platform support, and long-term maintainability of the graphical user interface (GUI). This upgrade also facilitates easier integration with other systems and future technological advancements at GMRT.

1. **End of Life Support**: Python 2 reached its end of life in January 2020, meaning it no longer receives security updates, bug fixes, or community support. Continuing to use it increases the risk of security vulnerabilities and system failures.

2. **Compatibility Issues**: Most modern libraries and frameworks are now designed for Python 3. Upgrading ensures compatibility with the latest tools, libraries, and dependencies required for efficient software development and system integration.

3. **Performance and Efficiency**: Python 3 offers significant improvements in performance and memory management. It includes optimized syntax, enhanced exception handling, and better concurrency support, leading to faster and more efficient code execution.

4. **Security Enhancements**: With Python 3, security features are regularly updated. Using outdated software like Python 2 exposes the system to potential cyber threats, while

Python 3 ensures improved protection through constant security patches.

5. **Improved Maintainability**: Modernizing the system with Python 3 ensures the codebase remains maintainable and easier to manage. Developers can leverage modern coding practices, write cleaner code, and reduce technical debt.

6. **PyQt Upgradation**: PyQt, a popular GUI library used for building graphical applications, also requires an upgrade. PyQt5 or PyQt6 provides enhanced user interface components, better rendering performance, cross-platform compatibility, and long-term support.

7. **Future-Proofing**: Upgrading both Python and PyQt provides a stable foundation for future enhancements and integrations. It ensures the system remains scalable, adaptable to emerging technologies, and aligned with GMRT's long-term operational goals.

8. **Reduced Downtime**: A well-planned migration to Python 3 and PyQt will minimize system downtime and ensure uninterrupted operations during astronomical observations. Thorough testing and validation during the upgrade process will ensure reliability and system stability.

2.2 A feasibility study - Software Upgradation :

The feasibility study ensures the project is technically and practically possible to build and deploy. While upgrading the Python 2 to Python 3, it is essential to study the change impact on the existing LMC software component in terms of compatibility, and functionality. The Python S/W Libraries and Tango web-sites are referred for the change impact of python-2 to Python-3, and which tango components/tools are needed to change are evaluated. The software compatibility study shows that for Python 2 to 3 changes, a complete Tango versions in C++, Python are needed to change as well.

The cascading effect of Python 2 to 3 is as follows :

- **Python 2.7.12 to Python 3.9 change shows that following s/w are need to upgrade :**

(i) Python 3

=> PyTango >= 9.4

=> cpp Tango version >= 9.4[1]

[1]https://tango-controls.readthedocs.io/projects/pytango/en/stable/versions/migration/to-9.4/deps-install.html

PyTango v9.4.0 is the first release which only supports Python 3.6 or higher. PyTango v9.4.0 moved from cppTango 9.3.x to at least cppTango 9.4.1. It will not run with cppTango 9.4.0 or earlier**.**

(ii) Python-3

=> PyQt5

PyQt4 binding end of Life in Aug 2018, PyQt4 was compatible up to Python 3.7 which is deprecated now, hence it is inevitable to change to PyQt5 for the Python version >= 3.7

(iii) Python-3

=> PyQt5

=> Taurus 5

Since PyQt5 is changed, it is mandatory to upgrade the Taurus, as the Taurus is a PyQt5 API based GUI Framework. Taurus 4 is no longer supported, Taurus 5 also requires python-3 and is generally integrated with PyQt5.

Thus, while shifting from the Python 2 to 3, it became mandatory to upgrade the libraries and s/w repositories available on the Ubuntu 22.04 LTS. To study feasibility a Tango framework is installed using the **CONDA** package environment[2]. ***Table 2.1*** shows the LMC Tango framework installed on machine 22.204 LTS.

Feasibility testing -
After installing the cpp Tango framework, Java framework, Python, Qt framework a basic testing of the Tango Tool-kit was performed.

- Mysql with Tango database tested with the Jive interface to register and connect test devices.
- Using a python shell, PyTango version, and a tango device such as DatabaseDS ping and send a state/status command.
- PyQT , Taurus's examples given in the tutorials on home-page of respective site ran and code tried to understand.

[2] Conda package environment installation work on the LMC machine is done by J. Kodilkar

Table 2.1 : The upgraded LMC Tango framework for Ubuntu 22.04 LTS

Package/OS/Tools	New	Mid	old
OS	Ubuntu 22.04.5	Ubuntu 18-20.04 LTS	Ubuntu 16.04 LTS
MySQL	8.3.0	8.0.33	5.7.33 - 5.8
Tango-Controls (cpp)	9.4.2	9.2.5a , 9.3.3	9.2.5a
Omniorb (cpp)	4.2.5	4.3.0	4.2.1
zeromq	4.3.5	4.3.0	4.0.0
Python	3.9.18	2.7.18	2.7.15
PyTango	9.4.2	9.3.3	8.1.1
Boost, boost-cpp	1.78.0	1.76.0	1.58 - 1.65
Java open-jdk	21.0.2/ 1.8.0 _432(LMC)	1.8.0_422	1.8.0_292
JTango, JTangoServer	9.7.3	9.5.0	9.5.0
Qt	5.12.8	4.8.7	4.8.7
PyQt	5.12.9	PyQT-4.12.1	PyQt-4.11.4
Taurus-core Taurus-Qt	5.2.0	3.7.3	3.7.0

Technical Feasibility :

- Python 3 offers enhanced performance, security, and modern libraries, making it a robust choice for future system sustainability. Migrating to Python 3 will involve refactoring legacy code to ensure compatibility with updated libraries.
- PyQt 5.4 ensures better GUI support with improved performance, cross-platform compatibility, and support for advanced UI components. Additionally, it supports the latest C++ features, essential for integrating with Tango 9.4.
- Taurus 5.4 provides an up-to-date framework for building control and monitoring applications, ensuring smooth interactions with the upgraded Tango control system.
- Tango 9.4 (C++) enhances system communication, device management, and real-time monitoring capabilities, offering better scalability and stability.
- Java-Tango 5.7 will ensure compatibility with Java-based applications, providing a more robust middleware for managing control commands and data flow.

2.3 Difference between python-2 & Python-3 :

The difference between Python 2 and Python 3 can have significant implications for the **GMRT TGC (Tango-based GMRT Control)** system, especially when it comes to performance, upgradation, and maintenance compatibility.

Upgrading the GMRT TGC system from Python 2 to Python 3 would offer several advantages, including improved security, better performance, access to modern libraries, enhanced string handling, and better compatibility with the Tango control system. The migration to Python 3 is essential to ensure the long-term sustainability, reliability, and security of the system, as Python 2 is no longer supported and its continued use could expose the system to risks and limitations.

Here are the key differences and how they would impact the GMRT TGC system:

1. End of Life

Python 2: Python 2 reached its official end of life on January 1, 2020. No more updates, bug fixes, or security patches are provided for Python 2.

Python 3: Python 3 is actively maintained and receives updates, security patches, and new features. It is the recommended version for all new projects.

Impact on GMRT TGC: Since Python 2 is no longer supported, continuing to use it in the GMRT TGC system poses security risks and could lead to difficulty in maintaining the system over time.

2. Syntax Differences

(a) **Print Statement vs Print Function :**
Python 2: `print "Hello"` *(Without parentheses)*
Python 3: `print("Hello")` *(With parentheses)*

(b) **Integer Division:**
Python 2: `5 / 2 = 2` *(Returns Integer if both operands are integers)*
Python 3: `5 / 2 = 2.5` *(Always returns a float)*
Python 3 supports `/` for integer division: `5 / 2 = 2`

(c) **Unicode Handling:**
Python 2: Strings are **ASCII** by default: `str`
Python 3: Strings are **Unicode** by default: `str`

- **UTF-8 vs ASCII vs Unicode and Python 3**

(i) ASCII (American Standard Code for Information Interchange)
Definition: ASCII is a character encoding standard that represents English characters using 7 bits (0 to 127). Hence, only supports 128 characters (including letters, numbers, punctuation, and control characters). It is not suitable for representing characters from other languages. Example: `'A' = 65` in ASCII.

(ii) Unicode Definition: Unicode is a universal character encoding standard designed to represent characters from all languages using a unique code point for each character.
Character Support: Supports 1.1 million characters across different languages, symbols, and scripts.
Encoding Methods: Unicode can be represented using different encoding formats like UTF-8, UTF-16, or UTF-32.
Example: `'अ' = U+0905` in Unicode.

(iii) UTF-8 (Unicode Transformation Format - 8 bit)
Definition: UTF-8 is a variable-length character encoding that uses 1 to 4 bytes to encode Unicode characters. Efficient: It is backward compatible with ASCII since ASCII characters are represented in 1 byte.
Character Support: Can encode all Unicode characters.
Advantages: Space-efficient for text with mainly ASCII characters and is widely used in web pages, APIs, and file storage. Example: `'A' = 1 byte`, `'अ' = 3 bytes` in UTF-8.

(d) Python 3 and UTF-8

Python 2 used ASCII encoding by default, which often led to encoding errors when dealing with non-English characters.
Python 3 uses UTF-8 as the default encoding for both source code and string handling. This means Python 3 supports all Unicode characters seamlessly. It provides robust handling of international text, supports multi-language applications, and is widely compatible with modern systems and APIs.

(i) String Handling

Python 2: In Python 2, strings are ASCII by default. Unicode support is available but needs explicit declaration (u"string").

Python 3: In Python 3, all strings are Unicode by default, which is important for handling multilingual or special characters.

Impact on GMRT TGC: With Python 3, the GMRT TGC system would have better support for handling non-ASCII characters, which can be crucial for processing diverse data sources.

(e) Performance Improvements:

Python 2: Python 2's performance is generally slower compared to Python 3 in many use cases, due to older internal optimizations.

Python 3: Python 3 includes many performance improvements, such as better memory management and more efficient handling of certain operations.

Impact on GMRT TGC: Python 3 would likely provide better performance for the GMRT TGC system, particularly in areas that involve intensive data processing or real-time control tasks.

(f) Tango Controls Compatibility

Python 2: The Tango control system historically supported Python 2, with many older implementations and libraries built on this version.

Python 3: Tango has since evolved to support Python 3, and newer versions of Tango libraries and tools are designed for Python 3.

Impact on GMRT TGC: Since the future of Tango development is focused on Python 3, upgrading the GMRT TGC system to Python 3 ensures continued compatibility with new Tango features, bug fixes, and improvements.

Table 2.2 : Python-2 and Python-3 Difference

Aspect	Python2	Python3	Difference
Support & Updates	End of life, no updates	Actively maintained and updated	Python 3 ensures long-term support
Performance	Slower with limited memory management	Improved memory management and speed	Faster and more efficient code in Python 3
Library Compatibility	Many libraries no longer supported	Most modern libraries support Python 3	Easier integration with modern tools
Unicode Support	Limited Unicode support	Native Unicode support	Better multilingual data handling
Syntax Improvements	Legacy syntax, e.g., print as statement	Improved, e.g., print as function	Cleaner and more readable code
Security	Vulnerable with no security patches	Regular security updates	Improved security and data protection

Future-Proofing	Deprecated	Long-term stability	Ensures system longevity

Ref : https://riverbankcomputing.com/software/pyqt/intro

2.4 PyQT4 to PyQT5 upgradation :

PyQt is a set of Python bindings for The Qt Company's Qt application framework. The bindings are implemented as a set of Python modules and contain over 1,000 classes. PyQt5 supports Qt5 and runs on Windows (Intel), macOS (Intel and Apple Silicon), Android, iOS and Linux (Intel).
Existing Scenario (PyQt4) : PyQt4 was a set of Python bindings for the Qt4 application framework. It was widely used for cross-platform GUI applications. Reached **End-of-Life (EOL)** on **Aug, 2018**. Compatible only with **Python 2** (limited support for Python 3).

PyQt4 :
PyQt4 is a set of Python bindings for the Qt cross-platform GUI toolkit, developed by Riverbank Computing Limited, that allows developers to create graphical user interfaces (GUIs) in Python using the Qt framework.
PyQt4 is a set of Python bindings for Qt 4, a popular cross-platform application development framework that is primarily used for creating graphical user interfaces (GUIs). Qt 4 was a major version of the Qt framework and was widely used for desktop application development.

PyQt 5 :
There are so many options provided by Python to develop GUI applications and PyQt5 is one of them. PyQt5 is a cross-platform GUI toolkit, a set of python bindings for Qt v5. One can develop an interactive desktop application with so much ease because of the tools and simplicity provided by this library. A GUI application consists of Front-end and Back-end. PyQt5 has provided a tool called 'QtDesigner' to design the front-end by drag and drop method so that development can become faster and one can give more time on back-end stuff.

Upgrading from **PyQt4 to PyQt5** is a significant step for modernizing Python-based GUI applications. Since **PyQt4** is deprecated and no longer supported, moving to **PyQt5** ensures better performance, security, and compatibility with current systems

Table 2.3 : PyQt4 and PyQt5 differences :

Aspect	**PyQt4**	**PyQt5**	**Difference**
Support	No longer supported	Actively maintained	Access to latest

			features and fixes
Widgets and Components	Limited GUI components	Enhanced UI components	Improved UI experience
Compatibility	**Not compatible with Python 3**	**Fully compatible with Python 3**	Seamless upgrade with modern Python versions
Graphics and Multimedia	Basic support	Advanced support for OpenGL and multimedia	Improved rendering and multimedia applications
Cross-Platform Support	Limited on some platforms	Improved cross-platform compatibility	Easier deployment across Linux, Windows, macOS
Licensing	GPL and commercial licenses	Similar licensing	No major licensing concerns

2.5 Taurus 5.2 :

Taurus is a software framework designed for building graphical user interfaces (GUIs) for controlling scientific instruments and systems, often used in complex research environments such as telescopes, particle accelerators, and other large-scale experimental setups. Taurus is built on top of **PyQt,** and is specifically designed for creating control and monitoring interfaces in scientific systems. While **PyQt** is a general-purpose library for building GUIs in Python, **Taurus** is specialized for control systems and integrates well with scientific hardware and frameworks like **Tango** and **EPICS**.

- **Key Features of Taurus:**

Python 3 Support: Fully compatible with Python 3, unlike older versions.
PyQt5 Support: Integrates seamlessly with PyQt5, supporting modern UI development.
Enhanced Widgets: Provides improved Taurus widgets for visualization and control.
Improved Performance: Faster data handling and optimized UI components.
REST API Support: Can communicate with TANGO devices using REST APIs.
Backward Compatibility: Maintains support for older Taurus 4.x applications with minor adjustments.

- **Use Cases:**

(a) **Scientific Instrument Control**: Taurus is commonly used to create interfaces for controlling and monitoring scientific instruments like telescopes, accelerators, detectors, etc.
(b) **Data Acquisition**: It is also used in data acquisition systems where the real-time collection and visualization of sensor data is crucial.
(c) **Laboratory Automation**: Researchers use Taurus to automate experiments, monitor equipment, and collect data from various devices in a lab.

Example of Using Taurus: A simple example of creating a graphical interface using Taurus to control a device might involve setting up a control panel to adjust the settings of a motor or a sensor in an experimental setup.

Taurus 5.2 Upgradation

Existing System: Taurus (Older version, Taurus 3.7)
Proposed System: Taurus 5.2 (Framework for creating control and monitoring GUIs using TANGO)

Table 2.4 : Difference between Taurus 3 and Taurus 5

Aspect	Older Taurus 3.x	Taurus 5	Difference
Support	Limited support	Actively maintained	Better community support and updates
TANGO Compatibility	**Compatible with older Tango versions**	**Fully compatible with Tango 9.4**	Seamless integration with updated control system
GUI Features	Basic UI components	Enhanced UI for complex systems	Improved visualization of data
Performance	Limited optimization	Optimized for large-scale systems	Faster and smoother operation
Python Compatibility	**Python 2 support**	**Python 3 compatible**	**Modern software development**
Extensibility	Difficult to extend	Modular and extensible	Easier to customize and expand

3. Methodology for the upgradation

The methodology for upgrading the Python-based GUI framework for the LMC of Tango-based GMRT Control (TGC) system involves a structured and systematic approach to ensure a successful transition from Python 2 and PyQt4 to Python 3 and PyQt5. Steps followed for the upgradation of python and PyQt4 for the LMC GUI are as follows :

1. Requirement Analysis : Analysed the functional and non-functional requirements of the current GUI system. Identified **limitations** in the existing PyQt4 and Python 2 framework, including deprecated functions and outdated libraries. Determined the key goals for the upgrade, focusing on **performance improvement**, **long-term support**, and **maintainability**.

2. Environment Setup using the CONDA[3] : Created a **dedicated development environment** using the CONDA package on the Ubuntu 22.04 LTS. The MySql Database, Tango CPP , PyTango, and JTango along with the respective OMNIORB (used in the tango-bus communication) and ZMQ Libraries of message parsing. Python 3.9 , PyQt5.15 , and Taurus 5.2 installed and tested using the examples.

Under the Conda environment for the TANGO installation, the LMC software for LMC, and GUI, Scripting, and astronomical libraries were compiled.

3. Code Migration : Performed a **code assessment** to identify modules and dependencies requiring modification.Upgraded code syntax from Python 2 to Python 3 using tools like ***2to3-3.9***.

Pyqt4to5 program used to transfer the GUI python programs, and later converted into the Python3. Later many manual corrections were done in the python files.

4. Integration and Testing : Integrated the upgraded GUI with the **Tango Control System** for real-time monitoring and control. Conducted **unit testing** using frameworks. Performed **system testing** to ensure compatibility with the GMRT infrastructure.Simulated operational scenarios to detect and resolve bugs.

5. Continuous Monitoring and Feedback : Monitored system performance during real-world operations. Applied incremental updates and patches as needed.

3.1 Python-2 to Python-3 Code Conversion : Py2to3 _3.9 :

[3] TANGO and LMC software installation was done by J Kodilkar under the CONDA package environment on Ubuntu 22.04 LTS.

The python-2 to python-3 conversion program 2to3-3.9 is used to convert the MNCScriptManager Tango Device-server code in "***/opt/tangoworkspace/ControlNode/Scripting"*** area using the command.

2to3-3.9 -W -o /opt/angoworkspace/ControlNode/Scripting

2to3-3.9 -W -o /opt/angoworkspace/ControlNode/GUI/src

The 2to3-3.9 program converts python-2 code to python-3 for the mentioned directory with option "-o" whereas -W writes the backup files of the modified code.

After python3 code conversion there were many errors which were manually corrected and code made python-3 compatible. Mainly, in many files indentation was not correctly put, hence each python file reviewed, also some missing functions, method types , return data type examined carefully which is debugging point of view hard task as to trace the errors. Major modifications done in python-3 codes after conversion are as follows :

(i) indentation spaces and import libraries :

(a) Indentation and spaces were not correct in many created either logical execution errors or misleading method errors. Hence, each python file is reviewed and compared with the old to correct the spaces and indentation errors systematically.
(b) ***"from .import"*** for locally import file was not working, for local files **"import <modulename>"** added correctly
(c) Comments with -- **'''** -- replaced with -- **"""** -- characters.

(ii) Data type errors, method APIs replacements :

(a) Default ***division*()** in python-3 return type is ***float***, hence manually casted to ***int*** data-type wherever it necessitates.
(b) **filter**() and **map**() not in python-3 list hence methods are replaced to work with the filter() and map() functions.
(c) **sort**() method is not available for the dictionary data types, hence even in some python-core libraries **sort**() is replaced by **sorted**() method.
(d) **Log4py** Class functionally are missing in python-3, hence log4py modules compatible with python-3 re-written. I.e ***whole APIS for log4py class changed.***
(e) ***iteritems***() method for python-2 used in the dictionary was memory efficient, but not available in the python-3 hence it is replaced by ***dict.items()*** methods.
(f) ***sys.stdout.encoding = sys.getdefaultencoding()*** This method was not working in the python-3 as the sys.stdout.encoding is readonly. Hence, in the **logger-class** of MNCScriptManager declared with ***sys.stdout.isatty = Lambda: False***

(iii) Python-3 unicode - 'utf-8' data type related changes :

(a) MySQLDB for the POOLED connections (mysqldb threads for accessing the MySQLDB in the python Interface) : used option use_unicode='True' made 'utf-8'. At some-places also the byte-conversion to utf-8 default for python-3 was removed manually.

(b) For ***split(), find(), match()*** , bytecode comparison is required, hence argument of ***str("xxxx"***) data type changed to ***b"xxx"***

(c) ***str(obj).rstrip('\n')*** function was returned in byte-data type (i.e. utf-8) , hence data type "***str***" given manually.

(d) ***/opt/conda/anaconda3/envs/tangoexp/lib/python3.9/site-packages/flake8/formatting/base.py*** was not having "buffer" attributes in python-3 , hence it is removed in the library.

- **Thus, while shifting from the python-2 to python-3, there were 20 python programs changed in the PyScriptManager, and 46 scripting files changed in the UploadBox.**

3.2 PyQt4 to PyQT5 code conversion using the code-converter tool "pyqt4topyqt5" :

The ***pyqt4topyqt5***[4] tool installed locally for the PyQt4 to PyQt5 code conversion of the GUI which contains the Taurus framework call as well. First python-2 to python-3 code conversion done for the GUI python-code, and then the **'pyqt4topyqt5'** converter tool used.

> pyqt4topyqt5 --nolog ./<pyqt4_code> -o ./<pyqt5_code>

The pyqt4topyqt5 converter tool writes the PyQt5 code into a separate directory.

In addition to Python-2 to Python-3 code related changes described in the ***section 3.1***. After PyQt5 code was created, there were many errors related to functions, data-types, signal handling , and FQDN (Fully Qualified Domain Name) related changes for the Taurus Model. A brief summary for major changes in the PyQt5 code are as follows .

(i) PyQt4 to PyQt5 Signal Handling :

In PyQt4, signals were handled using the SIGNAL() and SLOT() macros. The method associated with the GUI object (such as button) can be called using the connect call in the class.

```
from PyQt4.QtCore import SIGNAL

# for the 'button' widget , upon clicked() event, on_button_click() method is called.
self.connect(self.pushButton, SIGNAL("clicked()"), self.on_button_click)

# In PyQt5 new style of signal-slot connection is used.
self.pushButton.clicked.connect(self.on_button_click)
```

Major difference for the PyQt4 and PyQt5 signal handling are as follows :

[4] https://github.com/rferrazz/pyqt4topyqt5

Table 3.1 Difference between PyQt4 and PyQt5 for the signal handling.

Feature	PyQt4	PyQt5
Signal connect	self.connect(obj, SIGNAL(...))	obj.signal.connect(...)
Signal emit	self.emit(SIGNAL("sig"), args)	self.signal.emit(args)
Signal declaration	pyqtSignal(type)	pyqtSignal(type) (unchanged)
Syntax style	C-style macros, string-based	Pythonic, object-oriented
Type safety	❌ Error-prone (string-based)	✅ Stronger type safety
Debugging & IDE help	❌ No autocompletion, hard to debug	✅ Autocompletion, easier to debug

(ii) Phonon Library Removal in Qt5 :

Phonon was a multimedia framework originally used in the KDE and supported in **PyQt4** for audio and video playback. The **Phonon module is completely removed** in PyQt5. PyQt5 includes `QtMultimedia` and `QtMultimediaWidgets` for audio/video. Hence, in the GUI B2/B2.py code , phonon related code is removed, and QtMultimedia code changes are done.

(iii) Widget SuperClass changed in the PyQt5 :

In converted PyQt5 many QtWidget objects were missing because their super/parent class from which the QtWidget objects are derived were changed. Manually , the GUI widget objects parent class found using the google-help or from the PyQt5 documents' web-sites.

Table 3.2 PyQt4 to PyQt5 Widget super-class related changes :

PyQt4	PyQt5
QtGui.QDialog	from PyQt5.QtWidgets import QDialog
QTableWidgetItem	QtGui.QTableWidgetItem
QStringListModel()	QtCore.QStringListModel()
self.lineEditSource.text().***isEmpty()***	if not self.lineEditSource.text()
# For message parsing on the GUI at many places QString = unicode	

self.plainTextEdit.appendHtml(“<FONT color=xxx>%1</FONT>”).**arg(Message)**
 	
self.labelValidationStatusMessage.setText(“<FONT color=xxx>%1</FONT>”)**.arg(Message)**
	self.plainTextEdit.appendHtml(**QString**(“<FONT color=xxx>**{}**</FONT>”).**format(Message))**
	
	self.labelValidationStatusMessage.setText(“<FONT color=xxx>**{}<**/FONT>”.**format**(str(Message))

attributeModel = <hostname>:1000/domain/family/device/attribute_name	*# Attribute Model required FQDN name in PyQt5* *# Hence appended tango:// in many modules* *# wherever Taurus attribute model variables are.* **attributeModel = tango://**<hostname>:1000/domain/family/device/attribute_name

- **There were 37 PyQt5 , Taurus code files that were manually changed.**

Details of the code modification in the Python-3 and PyQt5 after conversion is tabulated in the **Appendix-I.**

4. Validation And Functional testing

The validation testing in the UI for the LMC of Tango based GMRT Control system was performed by dry-running the GUI, and correcting the code wherever it failed to run. The validation testing although the actual GMRT sub-system not connected was performed by login to the GUI, checking catalog widget, message-console, and alarms widget working or not.

The Functional testing includes the testing of the UI features **behave as expected** based on the software's functional requirement. In this case , the Optical-fiber and Sentinel sub-system (OFCSNT) was connected and actual monitoring attributes and commands were tried to send.

As the LMC of the TGC sub-system is a generic configuration software, only one system testing in the telemetry Lab is considered to be sufficient for the testing.

4.1 Number of UIs validation testing

In validation testing total 12 Main-windows, and 45 UI were validated to test it run properly. Following table shows the result of validation testing.

TABLE 4.1 : Validation Testing of the LMC GUI in PyQT5 and Python-3 Modified Code.

UI Category	UI-ID	UI Filename	Description	Test
1 .Login-console	**001**	**1.login.ui**	User login page enters username and password.	**PASS**
	002	**2.signUp.ui**	Fill all info and get to login	**NA**
	003	**3. contactus.ui**	Contact information shows	**PASS**
	004	**4.about.ui**	Write to the destination and small info of the centre	**PASS**
	005	**5.lock.ui**	Getting a username and password for the user to login.	**PASS**
	006	**6.R1.ui**	LST,IST time zone and date showing top layer.	**PASS**
	007	**7.gmrtVersion.ui**	To show the GMRT version	**PASS**
2. view->dashboard ->LMCStatus	**008**	**8.pageSubsystemStatus.ui**	Show drill-down monitoring of the GMRT Sub-systems, at first-page a high level	**PASS**

			sub-system status, and alarm for the systems (SERVO, FPS, SIGNCON, FECB, OFCSNT etc), and upon click, detailed parameters for each sub-system	
	009	**9.pageSubsystemIntrospect.ui**	Show detailed configured parameters for each sub-system like hardware interface(Rabbit, PC104 card), hostname, IP, Sub-system name, PORT connected etc.	**PASS**
	010	**10.help.ui**	A Help dialog to give information	**PASS**
	011	**11.titlebar.ui**	Title bar shows LMC or Sub-system name and updated Time for the monitoring, and button for 'introspect' and navigation for drill-down or going to the parent window	**PASS**
3 . Control -> TuneReceiver	**011**	**11.tuneReceiver.ui .**	Get current input and execute command-selection. The Input can arguments (setup) for the sub-system can be configured either - (i) setup file (ii) Band-centre (iii) From given Task	**PASS**
	012	**12.selectTask.ui**	Not Applicable for the LMC	**NA**
	013	**13.bandCenter.ui**	Show and Load the band-centre settings. (*The Tune-receiver code contains a double-event and immit call signal which is giving an error.)*	**FAILED**
	014	**14.tuneReceiverTabContainer.ui**	Shows the loaded parameters for the sub-system	**PASS**
4. Control -> Sub-system	**015**	**15.subsystemWindow.ui 9.**	LMC name selection and frequently used commands	**PASS**
	016	**16.subsytemWindowTabContainer.ui**	Shows the commands button for the selected LMCsys or Sub-system	**PASS**

	017	**17.subsytemPoupWindowDailog.ui**	Shows the command panel for the command-click to get the arguments	**PASS**
5. Control-> Common-Command-Environment	**018**	**18.commonCommandEnvironment.ui**	Check antenna env. To select them using a dropdown list and type command in the console.	**PASS**
6. Control -> Expert Console	**019**	**19.expertConsoleBackend.ui**	Expert console is like a command line interface where upon selecting a sub-system a list of associated commands appear. Upon selecting and executing, it shows the detailed message of command and response.	**PASS**
7. Control -> Observation Program	**020**	**20.observationProgram.ui**	Upload the scripting file, validate it for the execution. And start execution upon 'execute' button clicked.	**PASS**
	021	**21.scriptStatus.ui**	check script status for name , id,status,user name,time .	**PASS**
8. Control -> System Variables	**022**	**22.systemVariables.ui**	Upload LMC or Subsystem configuration files (Like FPS.csv, AntPara.csv etc) and allow modifying values and save them on the disk.	**PASS**
9. Control -> LMC Master Control	**023**	**23.lmcMasterControl.ui**	Various fields are there GMRT status,IST,Park,LMC Mode,Change mode,LMC services,LMC Subsystems	**PASS**
10. Control -> LMC Operation Control	**024**	**24.lmcOperationControl.ui**	Main console of lmc to check source catalog and track in / out of source and sets azimuth and elevation	**PASS**
	025	**25.Catalogs.ui**	Read Item 5. Utilities->Catalog. Upon selecting the object in the catalog it load in the 'LMC Operational Control'	**PASS**
10. Monitor->Message Console	**026**	**26.MessageConsole.ui**	Message console is a page of some info. And also export history option to save in device.	**PASS**

11. Utilities-> Catalogs	**027**	**27.Catalogs.ui**	Manage , view and edit catalog , also upload new in browse	**PASS**
	028	**28.RiseSet.ui**	Show the rise-set and transit time for the given date and Time	**PASS**
	029	**29.TypeofCatalog.ui**	Declare System or User Catalog and Type of the Catalog (Type-1 and Type-2)	**PASS**
	030	**30.catalogsEditSource.ui**	Edit the Fields of selected Source Object in the Catalog	**PASS**
	031	**31.plotTrajectory.ui**	Show the selected source object trajectory for a day for AZ and EL axis	**PASS**
	032	**32.DopSet.ui**	Calculate Doppler Settings	**NA**
	033	**33.precessTo.ui**	Precess the RA-DEC for a given date and time Epoch	**PASS**
	034	**34.riseAndSetTime.ui**	Select the Date-Time	**PASS**
	035	**35.subsystemFileDialog.ui**	Choose the catalog from Disk	**PASS**
	036	**36.TimeConverter.ui**	Convert IST to LST	**PASS**
	037	**37.catalogsNewSource.ui**	Add the new Source in the selected CATALOG	**PASS**
12. Other windows in the MainWindow application	**038**	**39.b2 History.ui**	Shows the past alarm messages for a configured sub-systems	**PASS**
	040	**40. B2.ui**	Shows the current Alarm list	**PASS**
	041	**41.b1History.ui**	Show the history of the LMC and sub-system detailed messages of command-response and events	**PASS**
	042	**42.B1.ui**	Show the LMC and sub-system detailed messages of command-response and events	**PASS**

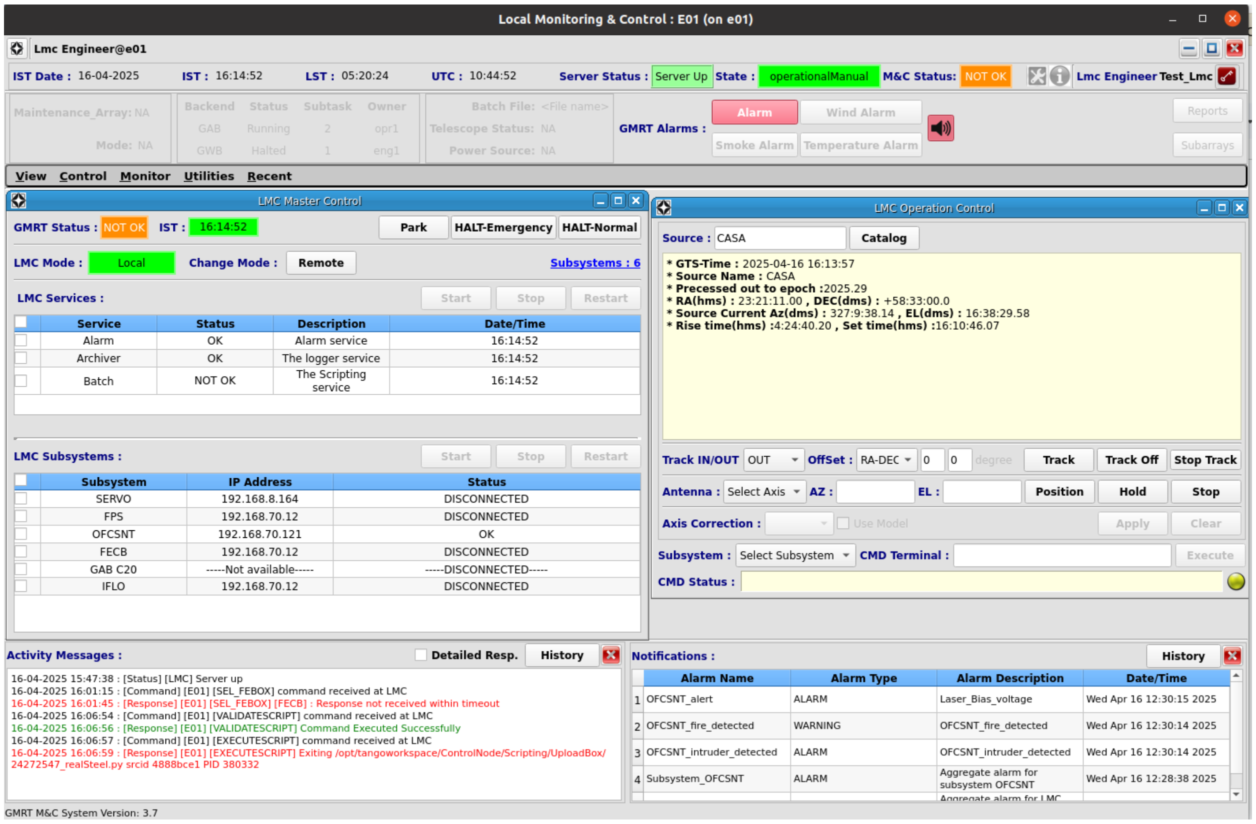
	043	**43.R1.ui**	Show the IST, LST, UTC, Server status, LMC , M&C Status, and logged user along with switch or re-login user	**PASS**
	044	**44. R2. UI**	Show panel of ALARM , sound mute option	**PASS**
	045	**PANIC Alarm panel**	Invoke Panic Alarm window	**PASS**

4.2 Functional testing at the GMRT antenna.

In a functional testing, one sub-system, optical-fiber and sentinel sub-system connected in the LAB to E01 antenna , and monitoring, and command parameters were tested. Since the LMC Sub-systems are configured in a generic specification driven, it is considered to test a single sub-system. Along with the major functionality of catalog, operational control window and LMC Master control window for the LMC system testing itself.

Following screen-shots of functional testing of the LMC UI configured for E01 antenna is shown for accomplishing the result of functional testing.

(a) Figure -1 : LMC Master-Control , and LMC Operational Control :



IST Date : 16-04-2025 IST : 16:14:52 LST : 05:20:24 UTC : 10:44:52 Server Status : Server Up State : operationalManual M&C Status: NOT OK Lmc Engineer Test_Lmc

Maintenance_Array: NA
Mode: NA

Backend	Status	Subtask	Owner
GAB	Running	2	opr1
GWB	Halted	1	eng1

Batch File: <File name>
Telescope Status: NA
Power Source: NA

GMRT Status : NOT OK IST : 16:14:52
LMC Mode : Local Change Mode : Remote

LMC Services :

	Service	Status	Description	Date/Time
☐	Alarm	OK	Alarm service	16:14:52
☐	Archiver	OK	The logger service	16:14:52
☐	Batch	NOT OK	The Scripting service	16:14:52

LMC Subsystems :

	Subsystem	IP Address	Status
☐	SERVO	192.168.8.164	DISCONNECTED
☐	FPS	192.168.70.12	DISCONNECTED
☐	OFCSNT	192.168.70.121	OK
☐	FECB	192.168.70.12	DISCONNECTED
☐	GAB C20	-----Not available-----	-----DISCONNECTED-----
☐	IFLO	192.168.70.12	DISCONNECTED

Source : CASA

```
* GTS-Time : 2025-04-16 16:13:57
* Source Name : CASA
* Precessed out to epoch :2025.29
* RA(hms) : 23:21:11.00 , DEC(dms) : +58:33:00.0
* Source Current Az(dms) : 327:9:38.14 , EL(dms) : 16:38:29.58
* Rise time(hms) :4:24:40.20 , Set time(hms) :16:10:46.07
```

Track IN/OUT OUT OffSet : RA-DEC 0 0 degree
Antenna : Select Axis AZ : EL :
Axis Correction : ☐ Use Model
Subsystem : Select Subsystem CMD Terminal :
CMD Status :

Activity Messages :

```
16-04-2025 15:47:38 : [Status] [LMC] Server up
16-04-2025 16:01:15 : [Command] [E01] [SEL_FEBOX] command received at LMC
16-04-2025 16:01:45 : [Response] [E01] [SEL_FEBOX] [FECB] : Response not received within timeout
16-04-2025 16:06:54 : [Command] [E01] [VALIDATESCRIPT] command received at LMC
16-04-2025 16:06:56 : [Response] [E01] [VALIDATESCRIPT] Command Executed Successfully
16-04-2025 16:06:57 : [Command] [E01] [EXECUTESCRIPT] command received at LMC
16-04-2025 16:06:59 : [Response] [E01] [EXECUTESCRIPT] Exiting /opt/tangoworkspace/ControlNode/Scripting/UploadBox/
24272547_realSteel.py srcid 4888bce1 PID 380332
```

Notifications :

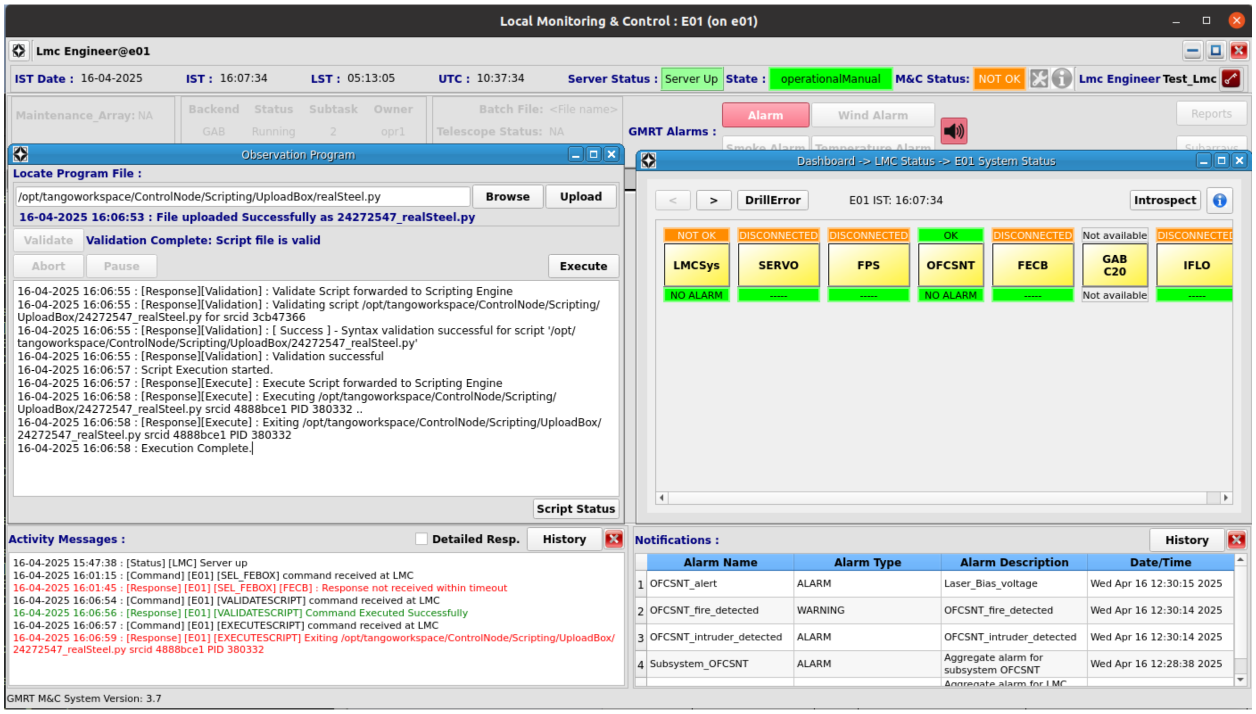
	Alarm Name	Alarm Type	Alarm Description	Date/Time
1	OFCSNT_alert	ALARM	Laser_Bias_voltage	Wed Apr 16 12:30:15 2025
2	OFCSNT_fire_detected	WARNING	OFCSNT_fire_detected	Wed Apr 16 12:30:14 2025
3	OFCSNT_intruder_detected	ALARM	OFCSNT_intruder_detected	Wed Apr 16 12:30:14 2025
4	Subsystem_OFCSNT	ALARM	Aggregate alarm for subsystem OFCSNT	Wed Apr 16 12:28:38 2025
			Aggregate alarm for LMC	

GMRT M&C System Version: 3.7

Figure -1 shows LMC Master control mode in Local mode along with the status of Alarm, Archiver, and BATCH (Batch not OK because of GAB MNCScriptManager run on the gab1 machine and not configured for the ‘E01’ antenna).

LMC-Master control shows the IP addresses and Status of each sub-system, the OFCSNT shows ‘OK’ status as it is connected.

(b) Figure-2 : LMC Script Execution in Observing Program and Dashboard view showing a high level status



```
/opt/tangoworkspace/ControlNode/Scripting/UploadBox/realSteel.py
16-04-2025 16:06:53 : File uploaded Successfully as 24272547_realSteel.py
Validation Complete: Script file is valid
```

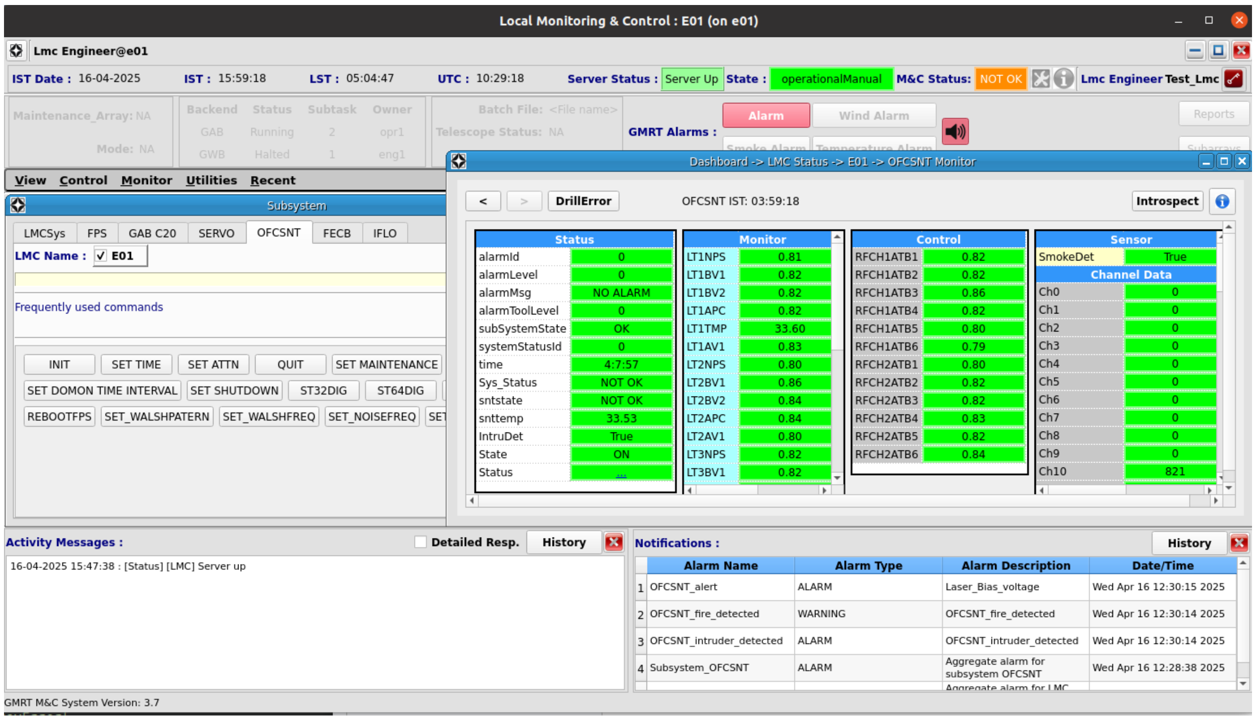
```
16-04-2025 16:06:55 : [Response][Validation] : Validate Script forwarded to Scripting Engine
16-04-2025 16:06:55 : [Response][Validation] : Validating script /opt/tangoworkspace/ControlNode/Scripting/
UploadBox/24272547_realSteel.py for srcid 3cb47366
16-04-2025 16:06:55 : [Response][Validation] : [ Success ] - Syntax validation successful for script '/opt/
tangoworkspace/ControlNode/Scripting/UploadBox/24272547_realSteel.py'
16-04-2025 16:06:55 : [Response][Validation] : Validation successful
16-04-2025 16:06:57 : Script Execution started.
16-04-2025 16:06:57 : [Response][Execute] : Execute Script forwarded to Scripting Engine
16-04-2025 16:06:58 : [Response][Execute] : Executing /opt/tangoworkspace/ControlNode/Scripting/
UploadBox/24272547_realSteel.py srcid 4888bce1 PID 380332 ..
16-04-2025 16:06:58 : [Response][Execute] : Exiting /opt/tangoworkspace/ControlNode/Scripting/UploadBox/
24272547_realSteel.py srcid 4888bce1 PID 380332
16-04-2025 16:06:58 : Execution Complete.
```

```
16-04-2025 15:47:38 : [Status] [LMC] Server up
16-04-2025 16:01:15 : [Command] [E01] [SEL_FEBOX] command received at LMC
16-04-2025 16:01:45 : [Response] [E01] [SEL_FEBOX] [FECB] : Response not received within timeout
16-04-2025 16:06:54 : [Command] [E01] [VALIDATESCRIPT] command received at LMC
16-04-2025 16:06:56 : [Response] [E01] [VALIDATESCRIPT] Command Executed Successfully
16-04-2025 16:06:57 : [Command] [E01] [EXECUTESCRIPT] command received at LMC
16-04-2025 16:06:59 : [Response] [E01] [EXECUTESCRIPT] Exiting /opt/tangoworkspace/ControlNode/Scripting/UploadBox/
24272547_realSteel.py srcid 4888bce1 PID 380332
```

	Alarm Name	Alarm Type	Alarm Description	Date/Time
1	OFCSNT_alert	ALARM	Laser_Bias_voltage	Wed Apr 16 12:30:15 2025
2	OFCSNT_fire_detected	WARNING	OFCSNT_fire_detected	Wed Apr 16 12:30:14 2025
3	OFCSNT_intruder_detected	ALARM	OFCSNT_intruder_detected	Wed Apr 16 12:30:14 2025
4	Subsystem_OFCSNT	ALARM	Aggregate alarm for subsystem OFCSNT	Wed Apr 16 12:28:38 2025
			Aggregate alarm for LMC	

The Observing window shows the scripting status of execution, and dash-board shows a high level view of the sub-system. ‘GAB_C20’ don’t exist as it is not configured for the E01 antenna.

(c) Figure-3 : Sub-system window for the command, and Detailed Monitoring for the “OFCSNT” Subsystem -



IST Date : 16-04-2025 IST : 15:59:18 LST : 05:04:47 UTC : 10:29:18 Server Status : Server Up State : operationalManual M&C Status: NOT OK Lmc Engineer Test_Lmc

Maintenance_Array: NA

Mode: NA

Backend	Status	Subtask	Owner
GAB	Running	2	opr1
GWB	Halted	1	eng1

Batch File: <File name>

Telescope Status: NA

GMRT Alarms :

LMC Name : ✓ E01

OFCSNT IST: 03:59:18

Status	
alarmId	0
alarmLevel	0
alarmMsg	NO ALARM
alarmToolLevel	0
subSystemState	OK
systemStatusId	0
time	4:7:57
Sys_Status	NOT OK
sntstate	NOT OK
snttemp	33.53
IntruDet	True
State	ON
Status	...

Monitor	
LT1NPS	0.81
LT1BV1	0.82
LT1BV2	0.82
LT1APC	0.82
LT1TMP	33.60
LT1AV1	0.83
LT2NPS	0.80
LT2BV1	0.86
LT2BV2	0.84
LT2APC	0.84
LT2AV1	0.80
LT3NPS	0.82
LT3BV1	0.82

Control	
RFCH1ATB1	0.82
RFCH1ATB2	0.82
RFCH1ATB3	0.86
RFCH1ATB4	0.82
RFCH1ATB5	0.80
RFCH1ATB6	0.79
RFCH2ATB1	0.80
RFCH2ATB2	0.82
RFCH2ATB3	0.82
RFCH2ATB4	0.83
RFCH2ATB5	0.82
RFCH2ATB6	0.84

Sensor	
SmokeDet	True
Channel Data	
Ch0	0
Ch1	0
Ch2	0
Ch3	0
Ch4	0
Ch5	0
Ch6	0
Ch7	0
Ch8	0
Ch9	0
Ch10	821

Activity Messages :

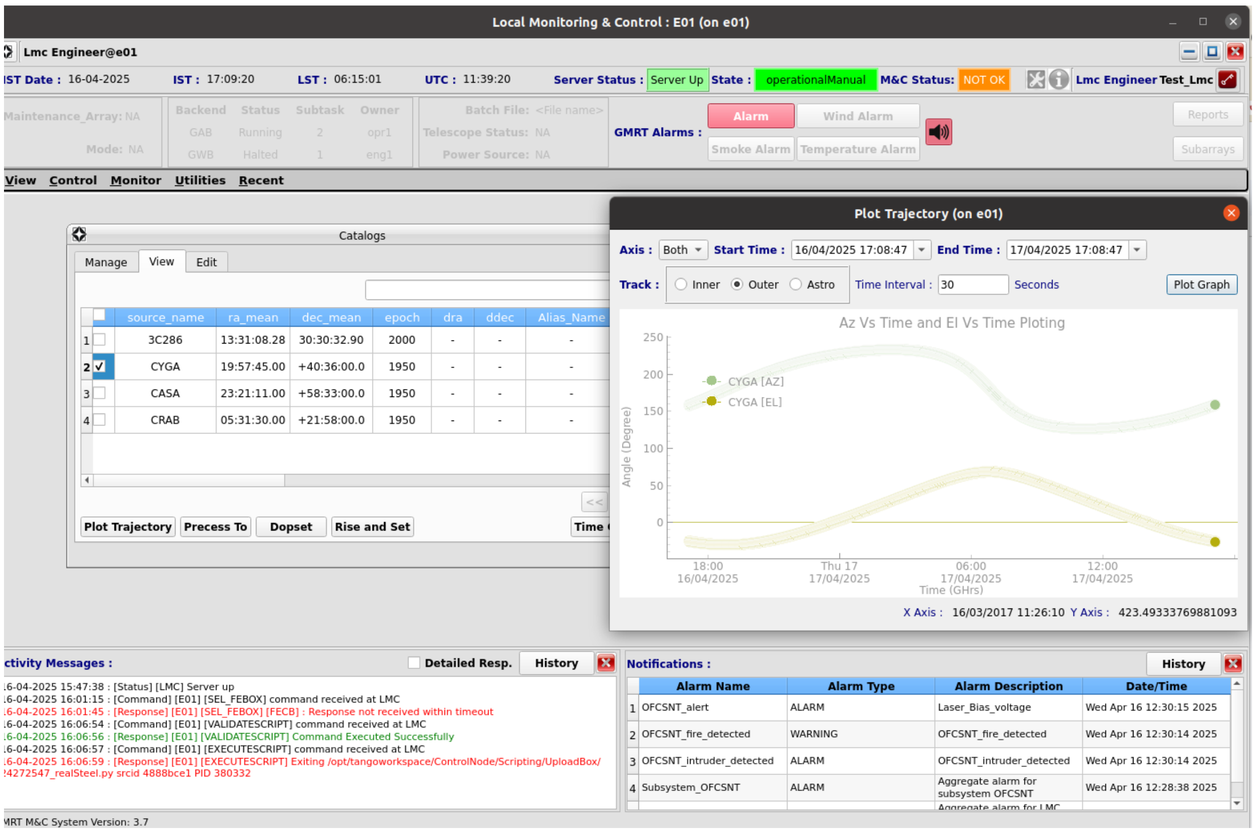
```
16-04-2025 15:47:38 : [Status] [LMC] Server up
```

Notifications :

	Alarm Name	Alarm Type	Alarm Description	Date/Time
1	OFCSNT_alert	ALARM	Laser_Bias_voltage	Wed Apr 16 12:30:15 2025
2	OFCSNT_fire_detected	WARNING	OFCSNT_fire_detected	Wed Apr 16 12:30:14 2025
3	OFCSNT_intruder_detected	ALARM	OFCSNT_intruder_detected	Wed Apr 16 12:30:14 2025
4	Subsystem_OFCSNT	ALARM	Aggregate alarm for subsystem OFCSNT	Wed Apr 16 12:28:38 2025
			Aggregate alarm for LMC	

GMRT M&C System Version: 3.7

(d) Figure-4 : Catalog functionality



IST Date : 16-04-2025 IST : 17:09:20 LST : 06:15:01 UTC : 11:39:20 Server Status : Server Up State : operationalManual M&C Status: NOT OK Lmc Engineer Test_Lmc

Maintenance_Array: NA

Mode: NA

Backend	Status	Subtask	Owner
GAB	Running	2	opr1
GWB	Halted	1	eng1

Batch File: <File name>

Telescope Status: NA

Power Source: NA

		source_name	ra_mean	dec_mean	epoch	dra	ddec	Alias_Name
1	☐	3C286	13:31:08.28	30:30:32.90	2000	-	-	-
2	✓	CYGA	19:57:45.00	+40:36:00.0	1950	-	-	-
3	☐	CASA	23:21:11.00	+58:33:00.0	1950	-	-	-
4	☐	CRAB	05:31:30.00	+21:58:00.0	1950	-	-	-

Axis : Both Start Time : 16/04/2025 17:08:47 End Time : 17/04/2025 17:08:47

Track : ( ) Inner (●) Outer ( ) Astro Time Interval : 30 Seconds

X Axis : 16/03/2017 11:26:10 Y Axis : 423.49333769881093

ctivity Messages : ☐ Detailed Resp. History

```
.6-04-2025 15:47:38 : [Status] [LMC] Server up
.6-04-2025 16:01:15 : [Command] [E01] [SEL_FEBOX] command received at LMC
.6-04-2025 16:01:45 : [Response] [E01] [SEL_FEBOX] [FECB] : Response not received within timeout
.6-04-2025 16:06:54 : [Command] [E01] [VALIDATESCRIPT] command received at LMC
.6-04-2025 16:06:56 : [Response] [E01] [VALIDATESCRIPT] Command Executed Successfully
.6-04-2025 16:06:57 : [Command] [E01] [EXECUTESCRIPT] command received at LMC
.6-04-2025 16:06:59 : [Response] [E01] [EXECUTESCRIPT] Exiting /opt/tangoworkspace/ControlNode/Scripting/UploadBox/
24272547_realSteel.py srcid 4888bce1 PID 380332
```

Notifications : History

	Alarm Name	Alarm Type	Alarm Description	Date/Time
1	OFCSNT_alert	ALARM	Laser_Bias_voltage	Wed Apr 16 12:30:15 2025
2	OFCSNT_fire_detected	WARNING	OFCSNT_fire_detected	Wed Apr 16 12:30:14 2025
3	OFCSNT_intruder_detected	ALARM	OFCSNT_intruder_detected	Wed Apr 16 12:30:14 2025
4	Subsystem_OFCSNT	ALARM	Aggregate alarm for subsystem OFCSNT	Wed Apr 16 12:28:38 2025
			Aggregate alarm for LMC	

MRT M&C System Version: 3.7

5. Summary

The objective of this project is to upgrade the graphical user interface (GUI) framework of the Tango-based GMRT Control (TGC) System by transitioning from Python 2 and PyQt4 to Python 3 and PyQt5. The current system, built using outdated technologies, poses security risks, lacks modern support, and restricts the implementation of new features. This upgrade will ensure long-term support, improved performance, and enhanced user experience.

In the CONDA package environment TANGO Libraries (cpp tango 9.4.2 , PyTango 9.4.2 and Java 9.7 with java openjdk 21.0.2, and for LMC programs compiled under openjdk 1.8.0_432) on **Ubuntu 22.04 LTS** found to be working. Since it was a feasibility study, the Java program needs to be updated further. The Python 3.9 and PyQt 5.12 and Taurus 5.2 Libraries and many dependencies were installed.

Total 45 UIs were tested for the validation and functional testing, and Nearly 57 python files need to be modified even after python-2 to python-3 and PyQt4 to PyQt5 converter tools were used.

To Make the LMC fully functional along with the stable Ubuntu 22.04 LTS OS version, the LMC software has a further scope of the work which is briefly listed as follows :

5.1 Future Scope of the Work :

- **(a) LMC Tango DS compilation under Openjdk 21.0.2 :** At present , the LMC Tango Device server is compiled under openjdk 8.0.1 which is unable to read response events from the Sub-systems CPP IO-Tango device server. Hence, the Java Tango Device-server gives command timeout response, however command and response is received and command gets successfully executed at the CPP TANGO IO Device Servers.
- **(b) LMC Software containerisation :** To run the LMC under a stable and safe environment, a LMC software needs to run under the docker kind of software containerisation.
- (c) **Web-based UI :** A web-based simple UI can be considered as a thin-client for the LMC user-friendly UI experience.

Reference :

[1]https://www.tango-controls.org/

[2]https://wiki.python.org/moin/PyQt4

[3]https://conf1.ncra.tifr.res.in/event/5/attachments/87/186/ngmnc_demo_nov15_2019-Jitendra.pdf

[4] https://www.geeksforgeeks.org/python-introduction-to-pyqt5/

[5] https://github.com/rferrazz/pyqt4topyqt5

Appendix-I

(I) Python-3 code modification done after 2to3-3.9 converter used.

#	Code Change description	Python files for code changes
1.	.import syntax corrected	
2.	Indentation errors and space errors are resolved	
3.	Handled float division using int() casting	
4.	Wrapped filter() and map() with list()	
5.	Replace iteritems() with items()	
6.	Updated MySQLDB connection encoding	Remove bytecode
7.	Updated log4py class API usage	
8.	Change comment format (’ ’ ’ to ” ” ”)	MNCScriptManager.py , MNCApiBridge.py , helper.py
9.	Upload Script spacing and exception fixes	
10.	Handled missing issaty function for stdout	MNCScriptManager.py
11.	Avoided writing to read-only sys.stdout.encoding	
12.	Fixed byte/string mismatched error	validation.py
13.	Fixed string rstrip() on byte objects	base.py , MNCScriptManager
14.	Use sorted() instead of dict.sort()	

(I) PyQT5 code modification done after the ‘pyqt4toqt5’ converter was used.

#	Code Change Description	Python Files for code changes
1.	Resolved Syntax error (global devlog before variable usage)	offload.py , mainWindow.py , R1.py

2.	Fixed deprecated get_instance() method , initialize direct instance creation Logger()	mainWindow.py , R1.py , DeveloperLogger.py
3.	Replaced the Phonon library with QMediaPlayer	mainWIndow.py , offload.py , B2.py
4.	Fixed pextension compilation	offload.py , mainWIndow.py , menuGenerator.py , lmcMasterControl.py , validator.py , scaled_format.py
5.	Linked required external libraries (Cdist,GuiLibs), Enabled compiled gnovas extension	Dir- src
6.	String replace functions , login.ui file errors , Signal connet errors	tuneReceiver.py ,bandCenter.py , Trajectory.py , plotTrajectory.py , correlator.py , newProject.py
7.	Expert console	Catalog.py
8.	Subsystem Command Execution	Subsystem.py
9.	Observation Program	base.py
10.	Signal Changing function	Catalogs.py , cmdRespChangeEvent.py
11.	QLineEdit.text().isEmpty Issue	lmcOperationControl.py
12.	CommonCommandEnvironment – QTableWidgetItem	CommonCommandEnvironment .py
13.	Update def getAttributeModel function to FQDN (Fully Qualified Domain Name)	lmcMasterControl.py , dashboardControl.py