Acta Scientific COMPUTER SCIENCES

Volume 7 Issue 1 April 2025

Case Study

Software Redesign for Better Adaptation, Robustness, Performance and Modularity: Case Study

A Jain[1]*, Padmini S[2] and S Srivastava[1]

[1]*Electronics Division, Bhabha Atomic Research Centre, Mumbai, India*

[2]*Electronics Division, Bhabha Atomic Research Centre (Ret.), Mumbai, India*

***Corresponding Author:** A Jain, Electronics Division, Bhabha Atomic Research Centre, Mumbai, India.

Received: February 14, 2025

Published: March 28, 2025

Software redesign [1] involves the thorough evaluation, analysis, and modification of an existing software system to transform it into a new form, followed by the implementation of the new form. The goal of software redesign is to provide more robust, scalable, fault tolerant, efficient software while preserving its basic functionalities. While the re-development process is tedious and expensive, it is compensated by improvement in software performance, reduction in maintenance efforts and ease of troubleshooting problems.

In this paper, we describe design analysis of existing software of centralized console server of MACE Telescope, which we used as test case. Initially, the software's design appeared reasonable, but upon investigating the feasibility of implementing new features, several significant flexibility-related challenges were uncovered, necessitating adjustments. The issues were analyzed to observe their impact on the flexibility of the software. In the process of solving these issues, we discovered usage of various design principles and design patterns [2] not only resolves these issues, but also makes design scalable and robust, reducing future maintenance effort.

Software responsibilities

MACE is a mega science experiment to study the Cherenkov light associated with an extensive air shower, developed as a result of a primary γ-ray emission from various cosmic, galactic and extra galactic astrophysical sources [3]. MACE telescope with 21m diameter and nearly 180-ton weight is a distributed system consisting of several subsystems with their designated responsibilities. The major subsystems include - high resolution imaging camera for recognition and acquisition of Cherenkov events, control system for moving the telescope towards specific sources at proper orientation, sky monitoring system for quantifying the sky transparency level and checking the tracking accuracy, mirror alignment system for focusing the Cherenkov light onto the camera, weather system for monitoring of wind speed and weather parameters, calibration system for relative gain calibration of imaging camera and data archiver for saving observation data. These subsystems are controlled by Central Camera Controller (CCC) [5], Telescope Control Unit (TCU) [6], Sky Monitoring System (SkMS), Automatic Mirror Control System (AMCS) [7], Weather Monitoring System (WMS), LED Calibration System (LCS), Data Archiving System (DArS) [8] respectively.

MACE Console server acts as centralized server for controlling each of these subsystems by issuing commands to all of them. Each of the subsystem controller accepts commands from centralized MACE Console Server, interprets it and perform desired actions corresponding to the received command. MACE operator console (OC), which acts as a client to MACE console server, is responsible for overall control, monitoring and error handling of the telescope. It provides user the facility to schedule, conduct and monitor observations.

To ensure safety of overall system, all functionally diverse subsystems within the MACE telescope exclusively function within well-defined and recognized states. The MACE Console server manages a state machine for each subsystem based on their respective functionalities. When a command is received from the MACE OC, it undergoes validation within the appropriate state machine to ensure its safe execution within the current state of the subsystem before it is transmitted to the intended subsystem controller as shown in Figure 1. MACE Console server also provides connection status of all subsystems to MACE OC periodically. MACE OC maintains the overall state of the MACE telescope by coordinating among the subsystems during the observation runs.

Original design

Original MACE console server software was designed using object-oriented methodologies. Each subsystem was defined as an

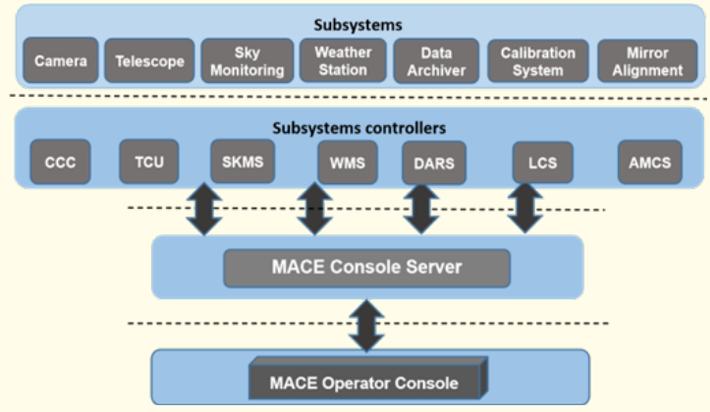

Figure 1: Block Diagram [9].

object with attributes and properties. The Subsystem object interacted with the actual subsystem controller for executing different commands and receiving the data. Each subsystem object had an associated Finite State Machine (FSM) for validation of commands. An elaborate FSM [12], consisting of all possible process states with all possible transitions as per subsystem functionality, was implemented using a third-party library. These FSMs [10] addressed both normal and abnormal conditions of the subsystems. Each subsystem had its well-defined parameters. The FSM also maintained information of connection state of all subsystems which is periodically communicated to MACE OC.

Planned extensions

MACE telescope is commissioned at Hanle (32.80 N, 78.90 E, 4270m asl) in the Ladakh region of North India. Elevated and less habituated remote site improves sensitivity and offers low background urban light with provision of approximately 260 spectroscopic nights annually, ensuring a consistent and evenly distributed sky coverage for observing celestial sources. The remote location and extreme weather conditions of the site impose constraints of minimum number of operational staff and lesser working period in a year, creating a lot of challenges for engineering during design, commissioning, testing and operation. The requirement of observations to be performed during dark and clear nights constrains configurations, troubleshooting and maintenance related activities, creating a strong need of remote operation capability.

The remote operation occurs over a shared satellite network with a restricted bandwidth allocation of 128 kbps.

Therefore, it becomes crucial to optimize communication mechanisms to prevent any disruptions or delays in the transmission of data to and from the remote OC. Security concerns forbid multicast mechanism in satellite communications, which enforces all the communications to be unicast. It was also necessary to address the requirement of multiple OCs, co-existing at the same time, and coordinating control and monitoring tasks, enabling multiple researchers from diverse geographic locations to conduct or observe the experiment simultaneously.

Need of redesign

- **Delays in communication mechanism:** Optimized communication mechanism requires fast response time, robust connection management and earliest detection of connection loss followed by quick re-connection. The existing software didn't provide the early and reliable detection of connection losses, causing delay in communication chain.
- **Elaborate third party state machine:** Each subsystem object had an elaborate state machine, evaluating every possible condition in all the states, introducing delay in command processing. This reduced the overall responsiveness of the system. State machines were created using a third-party tool, which provided FSM of each subsystem in the form of a library. This made the future development and troubleshooting difficult.
- **Serialized processing:** The original software serialized receiving commands from OC client, exchanging command and their respective responses from various subsystems, sending back the response to OCs and providing connection status to all OC clients. Functionality of connection management and status update gets blocked while a command is getting executed. To provide the facility of multiple co-existing OC's, server should be capable of receiving commands from one of the clients and sending status to all of them in unicast manner maintaining synchronization. Also, Connection management with multiple OC clients as a server and a number of subsystems as

clients should be accomplished concurrently. In case of detection of connection loss at one of the interfaces, it should be possible to reestablish connection while the other interface is doing data exchange. For instance, connection loss with camera should not interrupt communication with OC or TCU. While communication with OC or TCU is taking place, detection and recovery of connection with camera should happen concurrently without any interruption to the communication in progress.

- **Prohibited Multi-cast Mechanism:** In the original software, connection status of all the subsystems is periodically multicasted to OC, which is forbidden in satellite communication used for remote operation. Also, the mechanism of connection status update is tightly coupled with command response mechanism. It was required to change the existing scheme as well as provide the periodic status update to all the clients in unicast manner in multiple OC environment.
- **Data Coupling:** In the original software, parameter server used to maintain a global copy of shared parameters resulting in data coupling. The synchronization mechanism was required for any change in these parameters inducing unnecessary software bottlenecks. This needless blocking time reduced the overall responsiveness of the software. The use of global shared data impacted any future developments and reduced its maintainability.
- **Poor Design:** In the original software it was not possible to test each of the software functionality independently by keeping a stub for each of the interface. The design was not modular which increased efforts involved in software testing and maintenance.

New design

Design concepts

Well established software design principles have been adopted while designing the new software for MACE console server. Following the single responsibility principle, each module is designed by assigning single well-defined responsibility. Inter component communication is conducted through events and notifications to reduce coupling. Each of the subsystem State Machine is replaceable by the STM base class following Liskov substitution principles. The modules use interfaces to achieve Open-closed principle. These Interfaces are segregated so that none of the class is forced to depend upon interfaces that it doesn't use. Usage of SOLID principles [2] has helped in reducing side effects and made the design more stable and scalable. Multi-threaded programming architecture has been adopted to achieve maximum efficiency. Processing of commands at various stages, preparing subsystem connection status and communication with various subsystems and multiple OC clients are done concurrently.

Software architecture

MACE Console Server has been designed in a layered architecture comprising of dedicated packages [11] with reusable modules and classes. The important components of the software are described below:

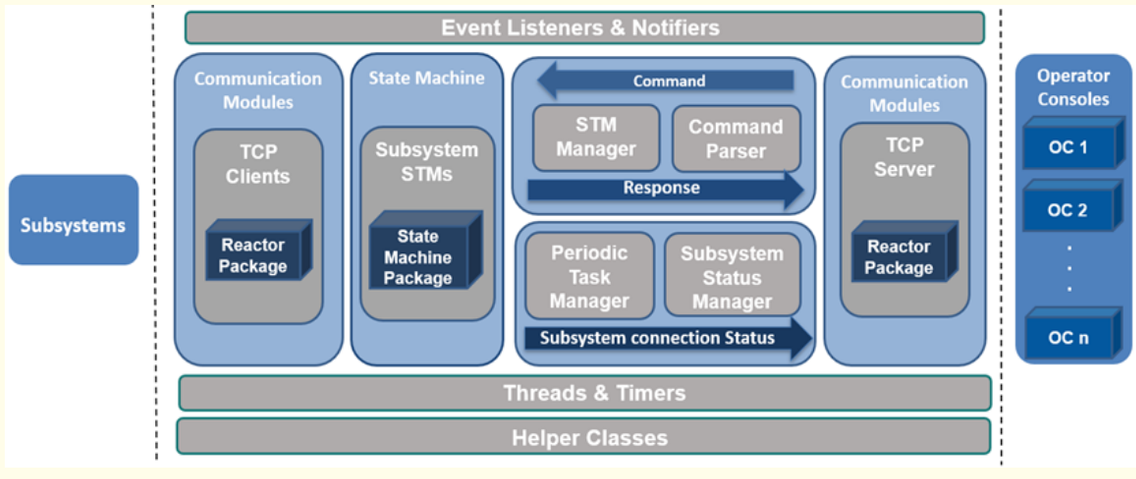

Figure 2: Architecture Diagram.

Communication package

This package contains TCP server, TCP client, Service IO Handler modules and classes for IP Address, IP End Point, IP END Point Array and various data structures like pipe and collection of connection handles.

TCP server

TCP Server module manages the communication with multiple MACE OC clients by maintaining a collection of connection handles and their connection status. The command received from one of the clients is dispatched to the command parser module. After receiving notification of response event, the response to the command is sent to all the OC clients. It also sends connection status of all the subsystems to all OC clients, on receiving notification from periodic task manager. TCP server runs on a separate thread, which is not blocked on IO. The communication mechanism with multiple OC clients is made non-blocking by using SELECT function to multiplex over the TCP sockets for getting the ready descriptor [12].

TCP client

MACE console server maintains a TCP client for communication with each of the subsystem controller. Each TCP Client has a TCP Connector which manages the connection with the subsystem controller. As a command is validated by state machine it is dispatched to corresponding TCP Client for sending it to designated subsystem controller. The response received from the subsystem controller is directed back to the corresponding state machine. Each TCP client also provides connection status with its subsystem controller to TCP sever on being notified by periodic task manager. TCP Clients run on a separate thread and uses non-blocking communication mechanism for performing IO.

Communication modules are based on Reactor Design Pattern [12] for accomplishing A robust and optimized communication mechanism, achieved through a single-threaded event loop that awaits resource-generating events and subsequently dispatching them to their respective handlers and callbacks. These handlers/

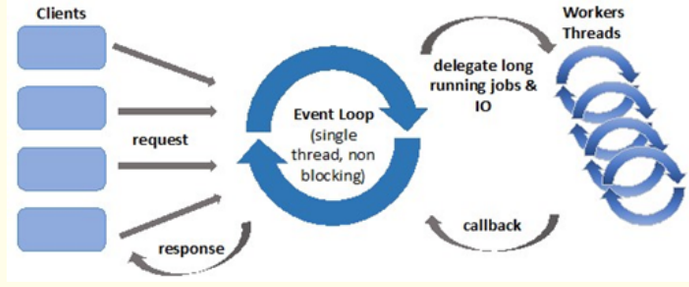

Figure 3: Reactor Pattern [4].

callbacks utilize a worker thread pool for actual execution of work instead of using a thread per client, which remains blocked most of the time waiting for network IO.

The mechanism instantly detects loss of connection eliminating the wait for a read or write. Every disconnection is quickly followed by a reconnection; making communication mechanism robust and efficient. Each unused connection descriptor is immediately released eliminating the possibility of any connection descriptor leaks.

Packet based communication mechanism is used, where each packet being transmitted from source to destination is assigned a header with packet id, packet length, checksum and payload. Unique packet id helps in discarding redundant packets received over dual redundant communication links provided for some of the subsystems.

State Machine (STM) package

This package contains an abstract STM factory, an STM factory and state machine manager. It contains classes for states, events, handlers, transitions and necessary data structures for maps and tables for creation of state machines.

For each of the subsystem an STM [14] is created consisting of three distinct states, connected via six state transitions as shown in table 1. These abstract level state machines maintain the connection and current state of subsystems for regulating issue of command to the subsystem. Camera state machine is described in Figure 4, which replaces the complex and detailed state machine in the original software as described in [10].

The State machine Manager maintains all state machines and their maps. Once a command is received from TCP server, it parses the command id through the maps to identify the destined subsystem and directs it to the corresponding state machine.

STM States		Connected		Disconnected	Response Awaited	
STM Events	Single Events	Connect		Disconnect	Timeout	
	Range Events	Issue Command		Receive Response		
STM Handler		Connect	Disconnect	Request	Response	Timeout

Table 1: State Machine Elements.

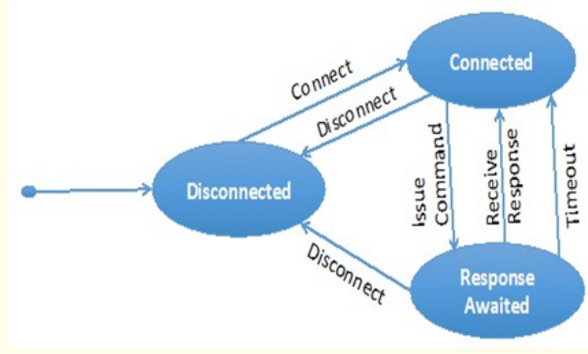

Figure 4: CAMERA Subsystem State Machine [4].

Since, behavioral functionality of subsystems changes along with their states, the State pattern is employed to establish a structured and loosely coupled approach to accomplish this. It is realized through the implementation of the Context and State, which permits the behavior to be dynamically adjusted depending on the current state at runtime. Hence, each of these state machines is created using state machine design pattern [14]. The implementation of state machines removes dependency on any other third-party tool making it easier to debug and integrate.

Since, we need separate state machines for each of the subsystems based on a similar template, factory design pattern is used for creation of various state machine instances.

Periodic task manager

It is a timer-based module, which manages the periodic activities in the software. This module runs on a separate thread and facilitates execution of periodic task of all the modules by providing them timer notification periodically.

Subsystem status manager

This module collates connection status of all subsystems and provides the information to all MACE OC clients on receiving notification from periodic task manager.

Command parser module

This module is responsible for parsing the command received from TCP server and passing it on to corresponding subsystem state machine.

Multi thread package

This Package comprises of classes and interfaces, which encapsulate OS thread implementation. This module is used for efficient management of threads by reducing overhead of thread creation. It creates and keeps some predefined number of worker thread in a thread pool and provides an interface to retrieve them whenever required. This reduces the need for dynamic thread creation during run time thereby optimizing unnecessary delays.

Event dispatcher

The Event Dispatcher facilitates application components to communicate with each other by dispatching events and listening to them. When an event is dispatched via dispatcher, it notifies all listeners registered with that event. It allows encapsulation of data as event arguments with notifier to be passed on to listener. It runs on a separate thread.

Figure 5 demonstrate full sequence of command processing from MACE OC to subsystem.

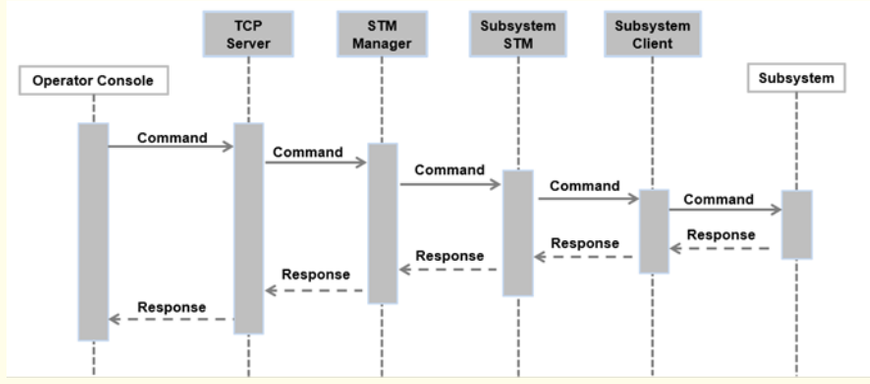

Figure 5: Sequence Diagram of Command execution.

Enhancements and benefit by new design

The enhancements in new software design has played a pivotal role in attaining peak performance. The benefits gained by incorporating various design changes are described below:

Improves scalability and adaptability

New MACE Console server design is capable of accepting multiple OC clients from different geographical locations. Decoupling of command interface and subsystem status update module provides flexibility of assigning different roles to geographically separated OCs, while continuing to send subsystem connection status periodically.

Based on open close principle the design is adaptive to addition of any new subsystem by simply adding an instance of TCP client for communication and creating another state machine by state machine factory.

Improved performance

Multithreaded architecture followed in the new design provides concurrent execution of tasks enhancing run time performance. The separation of the command and status interfaces from the OC creates space for simultaneous processing of commands and status updates. Non-blocking communication mechanism with multiple OCs and various subsystems provides faster network IO reducing overall response time, which is a critical requirement for remote operation. Abstraction of state machines reduces command-processing delays incorporated at console sever. Facility of shared pool of pre-allocated worker threads removes thread creation latency and readily provides worker thread for delegation of jobs.

Increased robustness

In the new design usage of Select pattern ensures quick detection of connection loss, followed by quick reconnection without incorporating significant delay in communication channel.

Increased modularity

Modular design of MACE Console server makes each of its independently developed general purpose component reusable in a variety of applications. Separation of communication modules for server and clients makes design modular and decouples communication with subsystems and OCs.

Improved testability and maintainability

Each of the module of newly designed console server is tested independently. All the state machines and subsystem clients are tested with individual subsystems keeping stub for other subsystems. Modular design has improved the testability of the system, as individual module undergone any kind of update can be independently tested before integrating with the system. Modular design of MACE Console server has also increased the maintainability as each of package can be independently modified, redesigned and extended without affecting other packages.

Decreased personal dependency

Usage of well-established design patterns [16] has helped in making design generalized. Modular design, usage of intuitive naming convention, dividing responsibility among classes and packages-based development has increased the readability of the code and is helpful in reducing dependency at developer level.

Conclusion

Redesign proves to be a valuable instrument for transforming outdated, obsolete systems into more efficient and streamlined ones. Its applications extend to enhancing flexibility, maintainability, robustness, modularity, performance, and testability while also reducing reliance on specific individuals. The comprehensive elucidation and rationale behind the changes, based on design principles as outlined in this paper, can prove beneficial when undertaking a redesign of other software systems in need of enhanced flexibility and optimization. These modifications can additionally function as examples for preparing software systems to accommodate future requirements. The discussed issues are common; appear in various scenarios, therefore described solutions can be employed in similar situations.

Software developers frequently find themselves inheriting existing software systems, making the issues outlined in this paper quite common. Consequently, this analysis can be valuable to programmers facing similar scenarios. Furthermore, by following recommended design patterns [15] for commonly occurring software scenarios during the initial development phase, it's possible to proactively prevent such issues. Recognizing and avoiding typical design challenges mentioned earlier, whether in the design or maintenance phase, has the potential not only to lower maintenance expenses but also to enhance system efficiency.

Bibliography

1. Fowler M., *et al*. "Refactoring–improving the design of existing code". (2002).

2. Robert C Martin. "Design Principles and Design Patterns". [www.objectmentor.com], (2000).

3. M Sharma., *et al*. "Sensitivity estimate of the MACE gamma ray telescope". Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 851 (2017): 125-131.

4. A Jain., *et al*. "MACE Telescope: Observation, Data Acquisition and Monitoring". BARC Newsletter: Universe of MACE Telescope at Hanle 383 (2022).

5. S Srivastava., *et al*. "MACE camera controller embedded software: Redesign for robustness and maintainability". ELSEVIER Publications, Astronomy and Computing 30 (2020).

6. P Kurup., *et al*. "MACE Telescope Servo Controller Design". *National Symposium on Nuclear Instrumentation* (2010).

7. P Kurup., *et al*. "Active Mirror Alignment Control System for the MACE Telescope". *National Symposium on Nuclear Instrumentation* (2010).

8. D Sarkar., *et al*. "A Generic High Data Rate Archiving Software Solution: In Context of an Astronomy Experiment". *Acta Scientific Computer Sciences* 3.7 (2021): 72-82.

9. A Jain., *et al*. "Autonomous Observation, Control, Data Acquisition and Monitoring of MACE Telescope". *Astroparticle Physics* 157 (2023): 102922.

10. S Bharade., *et al*. "State Based Control Design of MACE Console System". National Symposium on Nuclear Instrumentation (2010).

11. Package-Based Software Development, Proceedings of the 29th EUROMICRO Conference "New Waves in System Architecture" (EUROMICRO'03), IEEE, Merijn de Jonge, 1089-6503/03, (2003).

12. "The Reactor: An Object-Oriented Interface for Event-Driven UNIX I/O Multiplexing (Part 1 of 2)". D. C. Schmidt, C++ Report 5 (1993).

13. M Ackroyd. "Object-oriented design of a finite State machine". *Journal of Object Oriented Programming* (1995).

14. Shalyto N., *et al*. "State machine design pattern". Proc. of the 4th International Conference on. NET Technologies (2006): 51-57.

15. E Gamma., *et al*. "Design Patterns: Elements of Object-Oriented Software". Addison Wesley, (1995).

16. Armeet Singh and Syed Imtiyaz Hassan. "Effect of SOLID Design Principles on Quality of Software: An Empirical Assessment". *International Journal of Scientific and Engineering Research* 6.4 (2015): 1321.