ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOBPP04

THE ELT M1 LOCAL CONTROL SOFTWARE: FROM REQUIREMENTS TO IMPLEMENTATION

L. Andolfato†, J. Argomedo, C. Diaz Cano, R. Frahm, T. R. Grudzien, N. Kornweibel, D. Ribeiro Gomes dos Santos, J. Sagatowski, European Organisation for Astronomical Research in the Southern Hemisphere (ESO), Garching bei Muenchen, Germany
C. M. Silva, Critical Software, Coimbra, Portugal

Abstract

This paper presents the ELT M1 Local Control Software. M1 is the 39m primary mirror of the Extremely Large Telescope composed of 798 hexagonal segments. Each segment can be controlled in piston, tip, and tilt, and provides several types of sensor data, totalling 24000 I/O points. The control algorithm, used to dynamically maintain the alignment and the shape of the mirror, is based on three pipelined stages dedicated to collect the sensors' measurements, compute new references, and apply them to the actuators. Each stage runs at 500Hz and the network traffic produced by devices and servers is close to 1.2 million UDP packets/s. The reliability of this large number of devices is improved by the introduction of a failure detection isolation and recovery SW component. The paper summarizes the main SW requirements, presents the architecture based on a variation of the estimator/controller/adapter design pattern, and provides details on the implementation technologies, including the SW platform and the application framework. The lessons learned from deploying the SW on CPUs with different NUMA architectures and from the adoption of different testing strategies are also described.

INTRODUCTION

The European Southern Observatory is building the Extremely Large Telescope (ELT): one of the largest optical/near-infrared telescope in the world that will gather 13 times more light than the largest optical telescopes existing today. The telescope is located on top of Cerro Armazones in the Atacama Desert of northern Chile.

One key component of the ELT is the concave 39m primary mirror (M1) made of 798 quasi-hexagonal mirror segments of approximately 1.45m in size. M1 segments are controlled by the M1 Local Control System (M1LCS).

M1LCS prototyping activities started in 2011 with the goal to validate and consolidate the system design [1]. Final design review was passed in October 2017 and one month later started the development of the control SW. After less than 2 years of development, the first version of M1LCS control SW is being released.

SYSTEM DESCRIPTION

A detailed description of the M1 local control system is given in [2]. The segmented primary mirror of the ELT (Fig. 1), is composed of six sectors with 133 segments each. Segments within a sector are organized in flowers. One flower groups up to seven segments which are connected to a segment concentrator cabinet (SegC). There are 132 SegC cabinets and each cabinet hosts:

- the controllers for the field electronic devices (FE): edge sensor (ES), position actuator (PACT), and warping harness (WH);
- a Programmable Logic Controller (PLC) and a power supply unit (PSU) for power distribution control and temperature monitoring;
- a network switch connecting the PLC and FE controllers to the sector distribution (SecD) network switch that is connected to the computer room (CR).

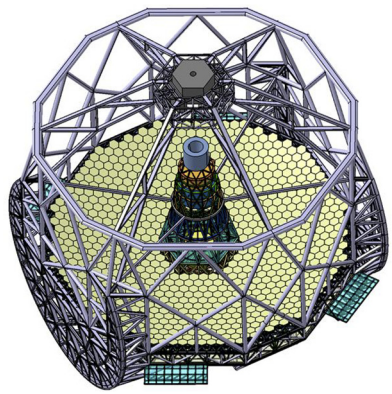

Figure 1: M1 primary mirror made of 798 segments within the ELT main structure.

Field Electronic Devices

The ES measure the relative out-of-plane (piston), and in-plane translation displacements (gap, shear) of a segment with respect to its neighbours. Each segment is equipped with six edge sensors with nm resolution.

The PACT are driving dynamically the segment in piston, tip and tilt in order to keep them aligned within the required accuracy under variable load conditions and disturbances. They are high accuracy linear actuators attaining nm resolution along a stroke of 10mm with internal feedback control.

The WHs are used to change the shape of the segment by applying different axial support forces to the mirror segment. This is achieved by a set of nine motors integrated in each segment support (Fig. 2).

† landolfa@eso.org

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOBPP04

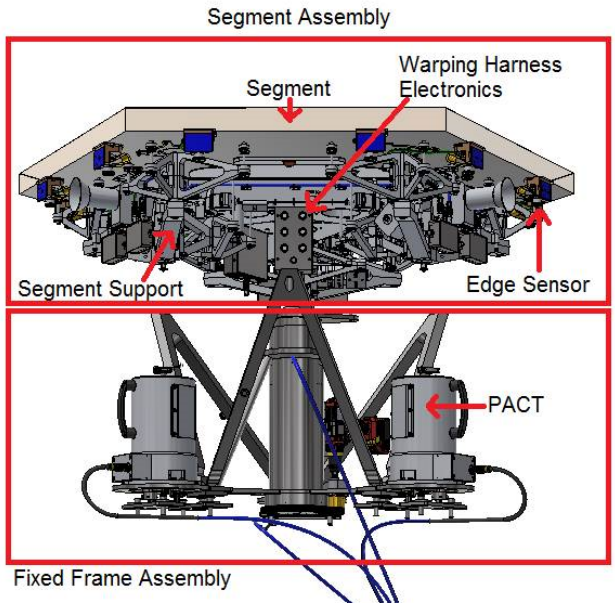

Figure 2: M1 Segment Assembly with ES, PACT, WH.

Power Distribution and Control

Within one flower, the power to the FE devices and the SegC network switch is provided by a PLC connected to the power supply unit. The PLC also monitors the temperatures, the door, and the maintenance switches of the SegC cabinet (Fig. 3).

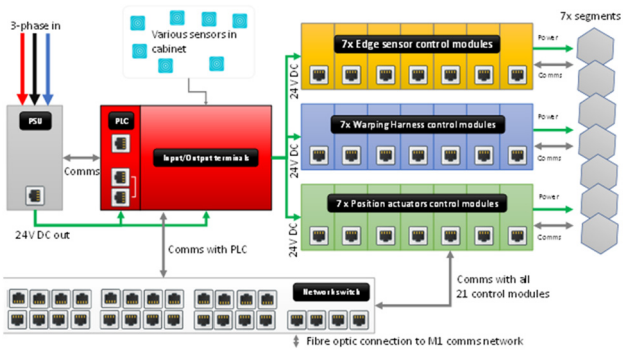

Figure 3: Electronic components within a SegC cabinet: PSU (in grey), PLC (in red), switch (bottom), and ES/PACT/WH control modules (in yellow/blue/green).

Network Infrastructure

The M1LCS communication infrastructure consists of a control network and a deterministic network. The control network is used by the operators and the servers for the non-real-time communication. The deterministic network is used for the real-time traffic and it is based on a star topology connecting:

- the FE devices and PLCs to the 132 SegC 1G switches;
- the SegC switch to a SecD 10G switch;
- the SecD switches (one every two sectors) to the 10G computer room (CR) switch;
- the servers to the computer room switch.

Deterministic network fault tolerance is improved by adding redundant CR and SecD switches as illustrated in Fig. 4.

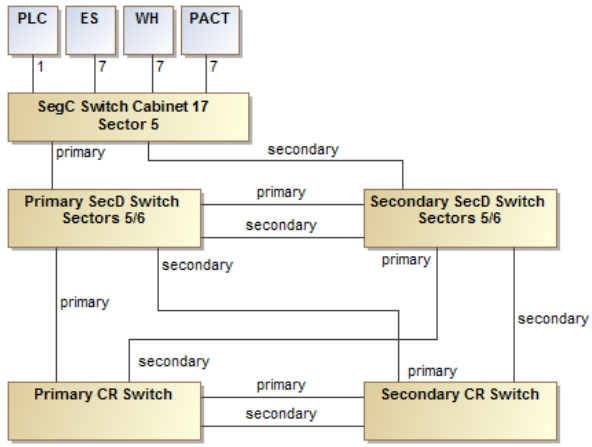

Figure 4: Redundant deterministic network topology connecting PLC/ES/WH/PACT to SegC, SecD, and CR.

Time Reference System

The M1LCS servers are synchronized to the observatory time reference signal using the Precision Time Protocol (PTP) [3]. The field electronic devices (ES, PACT, WH) are synchronized with a UDP packet (SYNC), generated by the M1LCS control SW, containing the observatory time. The PLCs are synchronized using the Network Time Protocol (NTP) [4]. The PTP grandmaster and the NTP use the same time reference generator.

REQUIREMENTS

The main responsibility of the M1LCS is to control and monitor the 798 segments. M1LCS does not control the shape of the whole M1. This task, called Figure Loop, is assigned to the M1 Local Supervisor SW (M1LSV) that uses the services provided by M1LCS to maintain a given optical quality for the duration of the observation. The most challenging SW requirements include:

- the generation and the distribution of the SYNC UDP packet to the FE devices and to M1LCS servers every 2ms, with 10µs accuracy;
- the ability to receive, combine, and deliver the ES and PACT measurements (1596 UDP packets) to the M1LSV in less than 2ms;
- the distribution of the PACT reference positions (798 UDP packets), received from M1LSV, within 2ms;
- the periodic collection of the FE performance and telemetry messages (7182 UDP packages every 1 to 10s)
- the detection, isolation, and recovery of faulty ES measurements. New synthetic ES measurements have to be provided to M1LSV within 1s from the failure;
- the configuration management of 2394 FE devices, 132 PLCs, and 140 network switches.

As additional goal, the solution should minimize HW and SW obsolescence costs by adopting widely used COTS components and avoiding vendor specific solutions.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOBPP04

ARCHITECTURE

The ELT Control Software Overview

The ELT Control SW is based on an hierarchical layered architecture having at the top level the instrument driving the observation and using the services provided by the telescope control SW (TCS). Within the TCS, a global supervisor (GSV) coordinates the work of each subsystems: dome, telescope main axes, M1, M2, etc. Each subsystem is made of two parts: the Local Supervisor (LSV) and the Local Control Software (LCS). The LSV operates in the telescope domain while LCS operates in the device domain. For example, the M1LSV is responsible for controlling and maintaining a certain optical quality of the whole telescope primary mirror surface. The M1LCS, instead, is in charge of controlling the M1 actuators and sensors making sure that the failure of a single component or segment does not affect the operation of the complete system. Figure 5 shows a simplified version of the ELT SW layers.

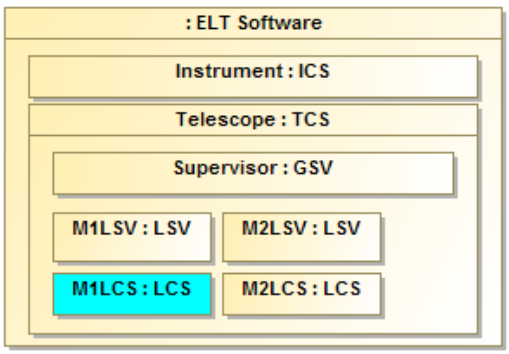

Figure 5: A simplified view of M1LCS within the ELT control SW (only M1 and M2 subsystems are shown).

Communication Patterns

One key aspect of the M1LCS system is the adoption of UDP multicast for the distribution of measurements. This allows to add any number of measurements recipients in parallel simply by configuring a network switch. This approach introduces the possibility of scaling performance by adding more servers (or more NICs within a server), improve robustness by adding redundant applications, measure performance and debug communication by adding sniffers. Commands and references are, instead, sent via UDP unicast. The M1LCS SW uses the request/reply communication pattern for the control flow and pub/sub for the data flow.

Monitor, Controller, and Adapter Pattern

The M1LCS SW adopts some of the architectural patterns defined in State Analysis [5]. In particular, the estimator-controller-adapter pattern is used to distribute the responsibilities between device monitoring, device management, and device access as shown in Fig. 6. A manager (mgr) is an application that can send commands to one or more device via the device adapter (devAdapter). A monitor (mon) is an application that can monitor the measurements, status/health, and performance of one or more devices via the devAdapter. The devAdapter is a library translating the manager's commands into the device protocol and the information produced by the device into a protocol understandable by the monitors. Monitors can store the received data in a "common database", the Runtime DB, using the dbAdapter library. Managers can access information stored in the common Runtime DB via the dbAdapter API.

Managers and monitors applications can be coordinated (initialized, started, stopped) by a supervisor (e.g. M1LSV) as illustrated in Fig. 7.

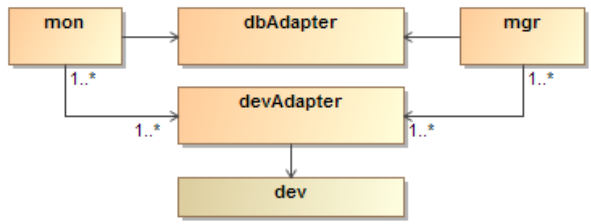

Figure 6: Adaptation of the State Analysis estimator-controller-adapter pattern.

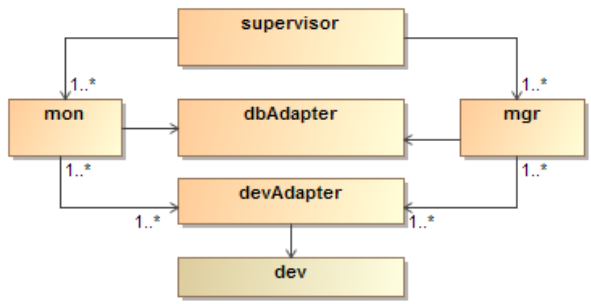

Figure 7: Estimator-controller-adapter pattern with Supervisor.

Fault Detection, Isolation, and Recovery

In order to provide fault detection, isolation and recovery functionalities, a dedicated manager (FDIRmgr), similar to the goal monitor of State Analysis, has been introduced as shown in Fig. 8.

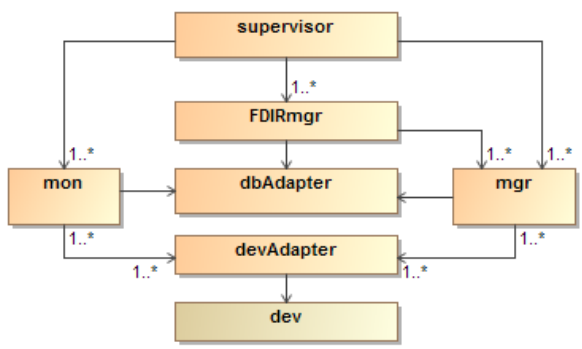

Figure 8: Estimator-controller-adapter pattern with Supervisor and FDIR manager.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOBPP04

The (LSV) supervisor distributes the commands to the managers and the goals to achieve to the FDIR manager(s) via a dedicated SetGoal command. The FDIR manager is in charge of monitoring periodically the information written by the monitors in the Runtime DB and verify whether the goals have been achieved or not. In case of missed goals, the FDIR manager can start recovery actions by sending commands to the managers. The (LSV) supervisor is responsible for avoiding conflicts with the FDIR manager when sending commands to the other managers.

Generic Application State Machine

Monitor and manager applications share the same basic behaviour defined in Fig. 9 which consists of three states: IDLE, RUNNING, and ERROR. IDLE means that the application has been launched and initialized. With the Start command, the application enters the RUNNING state and starts the working thread(s). For monitoring applications, the working thread(s) are used to receive the measurements via the device adapter library while for manager applications the working thread(s) send the commands to the devices.

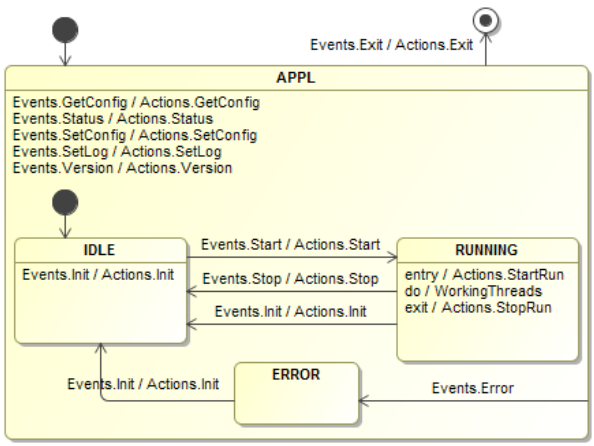

Figure 9: Monitors and Managers application behaviour.

System Partitioning

The M1LCS SW has been partitioned into seven subsystems:

- Field Electronics (FE) to manage ES/PACT/WH devices. It includes: FeAdapter library to talk to devices, FeMeasMon to collect the measurements, FeRefMgr to send the references, FeInfoMon to collect the performance and telemetry messages, FeCmdMgr to send commands, FeConfig to deal with devices configuration.
- Sync (SYNC) to synchronize M1LCS devices and applications. It includes: PtpAdapter to retrieve PTP time, SyncAdapter to receive the SYNC message, SyncMgr to generate the SYNC UDP message, SyncMon to collect period and jitter statistics on the SYNC message.
- Power Distribution and Control (PDC) to manage PLCs. It includes: PdcAdapter library to talk to the PLCs, PdcMgr to send commands to the PLCs, PdcMon to collect information produced by the PLCs.
- Network switches (NET) to manage the network switches. It includes: NetAdapter library to talk to the switches, NetMon to collect information on the switches.
- Fault Detection Isolation and Recovery (FDIR) to trigger alarms/corrective actions in case of faults. It includes: the FdirNpm library implementing a computationally expensive algorithm to detect invalid ES measurements and the FdirMgr to process the information collected by all monitoring applications, detect faults, and trigger alarms/corrective actions.
- Common libraries to access the runtime DB, serialize/deserialize messages, to deal with system configuration, and other common services.
- Tools containing engineering GUIs, scripts, and simulators used for testing.

IMPLEMENTATION

Application Stack

The M1LCS SW has been developed using the Rapid Application Development framework (RAD) on top of a Software Platform and the ELT Development Environment as illustrated in Fig. 10.

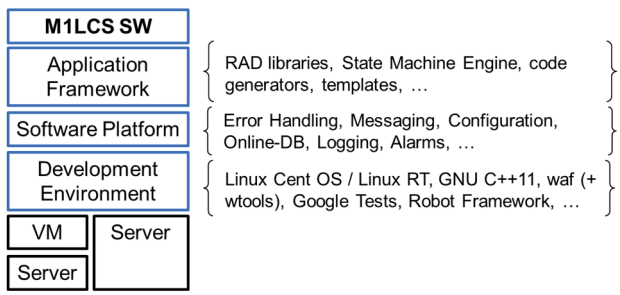

Figure 10: M1LCS SW application stack.

The ELT Development Environment [6] includes:

- OS: Linux CentOS with RT patch (CERN distribution)
- Languages: GCC C/C++, Python
- Building system: waf + ESO wtools
- Documentation: doxygen
- Unit tests: Google Tests, Unittest
- Integration tests: Robot Framework
- Continuous Integration: Jenkins
- Configuration Management: SVN
- Deployment: Nomad/Consul
- Other Tools: cpplint, cppcheck, valgrind, pylint

The Software Platform provides services common to all M1LCS applications. Since the official ELT Software Platform, the Core Integration Infrastructure, has not been released yet, a temporary platform, which includes the components described in Table 1, has been selected.

All M1LCS applications shares the same design defined by RAD, the application framework developed by ESO. RAD is a toolkit that allows to quickly create event driven applications based on the Software Platform described

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOBPP04

above and with the support of a state machine engine. RAD provides an event loop based on Boost.Asio [7] integrated, via AZMQ [8], with ZeroMQ [9] to be able to react to incoming requests/replies and pub/sub topics. RAD encapsulates ZMQ messages into events which are injected into a state machine engine based on an SCXML interpreter [10]. Events are defined using a Domain Specific Language (DSL). The SCXML state machine definition can be generated from UML/SysML models using the COMODO tool [11]. For example, the application behavior shown in Fig. 9 is converted into an SCXML model which is loaded and interpreted at run-time. The initial version of a RAD based application is generated from pre-defined templates using the Cookiecutter tool [12].

Table 1: M1LCS Temporary Software Platform

Service	Product	Description
Messaging	ZeroMQ OPC-UA	Request/reply, pub/sub. PLC communication.
	Goolge ProtoBuf	Message serialization/deserialization.
Configuration	YAML	Based on local files + Redis.
Runtime DB	Redis	In memory key-value store (Redis).
Logging	EasyLogging	Local logging.
Erorr Handling	N/A	Based on exceptions.
Alarms	N/A	Based on Runtime DB
GUI	PySide2	QT for Python

Runtime DB

M1LCS applications exchange data such as measurements, status, statistics, etc., using pub/sub communication pattern. The Runtime DB adapter libraries provide two ways of publishing information:

- over shared-memory for the low-latency transfer of ES/PACT measurements from M1LCS to M1LSV.
- over ZMQ to transfer non real-time information between M1LCS applications and GUIs and M1LSV.

Moreover, M1LCS applications publish information to Redis [13] to make the system observable. Information stored in Redis can be used to: debug the system, provide data to the GUIs, and get a complete overview of the system status. Redis is also used to store some global system configuration parameters such as the communication parameters to talk to the devices.

Enabling the Figure Loop

The core of the M1LCS SW is represented by the FeAdapter library. This library, using a standard Linux network stack, is able to receive 798000 UDP pkt/s from the ES and PACT devices and, at the same time, send 399000 UDP pkt/s to the PACT devices. This is achieved by using:

- Two NICs on the same server: one dedicated to receive ES measurements and one to receive PACT measurements and send PACT references.
- Intel Ethernet Flow Director to steer the incoming packets to different network queues [14].
- NIC interrupts affinity to pin network interrupts to specific CPU cores [15].
- RAW sockets configured with the PACKET_FANOUT option [16] to distribute the packets arrived on a given CPU core to a well-defined FeAdapter thread.
- Isolated CPU cores by applying the real-time profile to specify which core should not be used by the OS. Cores to be isolated have to be selected considering the NUMA architecture of the servers with the goal of minimizing the memory access latency/distance between the memory used by NIC/PCI, the cores executing the interrupt handlers, and the cores running the FeAdapter threads.
- Setting some BIOS options and OS services: configure memory channel interleaving (all memory channels should be occupied), hyper-threading disabled, irqbalance service disabled.

Figure 11 shows the flow of the incoming ES/PACT measurement UDP packets to specific FeAdapter threads.

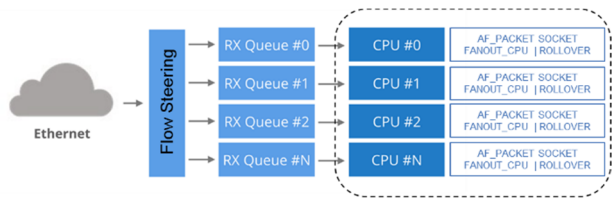

Figure 11: Ethernet packets flow.

The FeAdapter library is the result of previous prototyping activities [17] and it is used by the FeMeasMon and the FeRefMgr applications. The FeMeasMon combines the ES/PACT measurements received by the FeAdapter threads into a vector which is written to the shared-memory. The LSV Figure Loop controller reads from the shared-memory the measurements and computes the new PACT references. The resulting vector is written into shared-memory and used by the FeRefMgr to send, via the FeAdapter, the new positions to the PACT devices. This three-stage activity (measurements acquisition, references computation, references distribution) is pipe-lined as illustrated in Fig. 12.

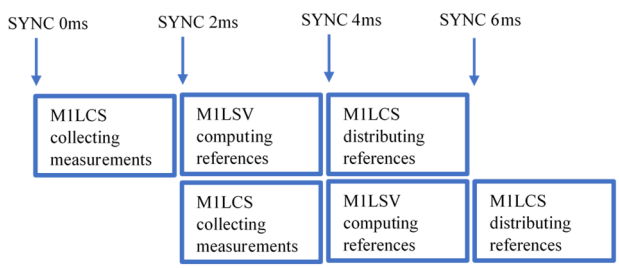

Figure 12: Figure Loop stages.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOBPP04

The FeAdapter library is also used by FeInfoMon application to receive telemetry and performance messages produced by the ES, PACT, and WH devices at a lower rate (4788 pkts/s).

Fault Detection, Isolation, and Recovery

FDIR is implemented as a manager application with a RUNNING state specialized with orthogonal regions [18]. Each region defines the state of an evaluator using the EVAL_ENABLED and EVAL_DISABLED sub-states. Each evaluator is implemented with a dedicated thread that, when enabled (i.e. in the EVAL_ENABLED state), is responsible for evaluating one or more goals.

A goal represents a condition to monitor. It is described by an identifier, a time interval indicating when to evaluate the goal, a flag to indicate whether the goal has to be evaluated or not, and a status flag indicating whether the goal is satisfied, i.e. the condition to monitor is satisfied, or not. Goals are enabled, disabled and configured via the SetGoal command.

When a goal is enabled, the associated evaluator thread will subscribe to the topic(s) published by M1LCS monitoring applications and required to evaluate the goal; then it will periodically compute the goal status. If the goal is not satisfied, an alarm containing the reason is published. In the case of faulty ES, the evaluator provides also an estimation of the expected ES measurement to avoid breaking the Figure Loop.

Configuration Management

M1LCS SW provides dedicated applications to retrieve, compare, and apply the configuration of the HW under control. For the FE devices, FeConfig application uses the FeCmdMgr to get/set, asynchronously, the ~50 parameters of the 2394 devices. For the TwinCAT PLCs [19], a dedicated application, TcPutty, has been developed to configure and download the PLC code to all 132 PLCs. Similarly, the NetConfig application has been implemented to get/set the 140 network switches configuration files.

Graphical User Interfaces

Engineering panels have been developed in Python using PySide2 [20]. They retrieve information to be displayed by polling Redis or by subscribing to ZMQ topics. Commands are sent to the applications via the ZMQ request interface.

The following GUIs have been developed:

- dbBrowser to read, write, monitor, and record Redis keys/values.
- m1StatusGui to monitor M1LCS SW applications and send commands. It can be configured with the applications to monitor, the Redis attributes to display and plot, and the commands/parameters to send (Fig. 13).
- m1ExecGui to execute scripts based on sequences of commands.
- m1MonitorGui to monitor M1LCS devices.
- m1SegmaintenaceGui to follow up segments maintenance operations (Fig. 14).

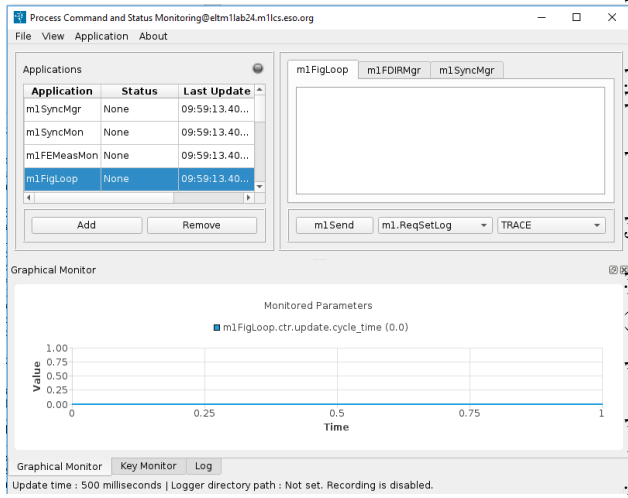

Figure 13: m1StatusGui

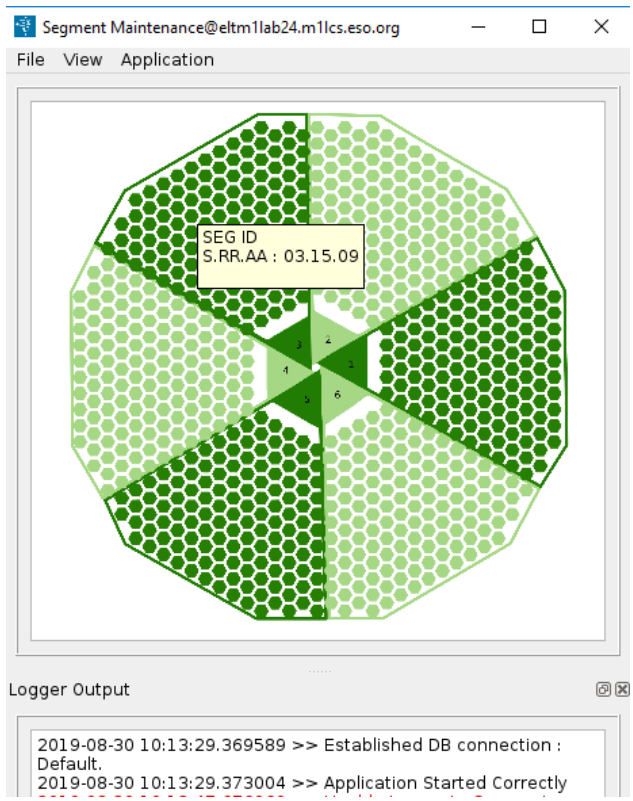

Figure 14: m1SegmaintenanceGui.

Simulators

Since the final number of devices, network switches, and PLCs will be available only during integration at the observatory, quite some effort was invested in the development of simulators and testing tools to be able to properly verify the M1LCS SW. To validate the performance requirements, the following tools have been developed:

- Traffic Generator: this application is able to generate the full (or a subset of) M1 ES/PACT/WH UDP traffic including measurements, announce, telemetry, and performance packets within the time constraint of the

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOBPP04

real devices. It uses 4 ports of a 10G NICs to send the packets. The content of the packets is configurable.
- Figure Loop: this application is a simple implementation of LSV Figure Loop control algorithm.

To verify interfaces and for scalability tests, the following simulators have been developed:
- ES, PACT, WH device simulators used to test the interface between M1LCS and the FE devices.
- PLC simulator to simulate the 132 PLCs.

For the simulation of the network switches the SNMP Simulator tool [21] has been adopted.

DEPLOYMENT

M1LCS SW can be deployed on one or more servers depending on the performance to be achieved. Since the low latency interface between M1LCS and M1LSV is implemented via shared memory, enough server resources have to be reserved to M1LSV to execute the control algorithm.

We tested different deployment scenarios: some based on servers equipped with dual CPU Intel Xeon E5-2699 2.3GHz, for a total of 2 NUMA nodes and 36 cores, and others based on servers with dual CPU AMD EPYC 7551 2.0GHz, for a total of 64 cores and 8 NUMA nodes [22]. All servers have dedicated 10G NIC(s) to connect to the deterministic network. The best results were obtained using the AMD EPYC running on CPU-0 the M1LCS FeMeasMon and FeRefMgr applications and on CPU-1 the M1LSV Figure Loop control algorithm simulator. The eight cores associated to NUMA node 0 were left to the OS and to low priority threads while all the other cores were isolated. FeMeasMon was running the receiving threads on cores associated to NUMA nodes 1 and 2. FeRefMgr was running the sender threads on NUMA node 3. We used two 10G NICs installed on the PCIe slot connected to NUMA node 1 and 2 respectively: one to receive the ES measurements, the other to receive the PACT measurements and send PACT references. The Figure Loop simulator was configured to use eight worker threads to run in parallel matrix-vector multiplications plus one thread to combine the partial results. Each worker thread was running alone on one AMD EPYC die (also called zeppelin) to increase memory bandwidth (i.e. less conflicts when accessing the memory channels). Using only one core every four may seems a waste of resources but it increased dramatically the performance.

Currently, the M1LCS SW is deployed on VMs for local development and in M1 laboratory to verify performance, scalability, and interfaces. The lab deployment consists of a SegC cabinet with 7 WH and 7 ES controllers, PLC/PSU, SegC network switch, a SecD switch, a CR switch, and five servers running M1LCS applications. In addition, one server is dedicated to the Traffic Generator simulator and one PC with a 10G Myricom is used as a sniffer. Servers in the lab are provided with PTP time reference signal.

It is foreseen to add an additional deployment setup to control a small scale M1 made of seven segments fully equipped with ES/PACT/WH sensors and actuators connected to a dedicated SegC cabinet.

TESTING AND RESULTS

Testing Strategies

M1LCS SW is integrated with the ESO ELT continuous integration (CI) infrastructure based on Jenkins. CI is triggered every time a modification is archived and it executes 691 unit tests and 127 integration tests. For the PLC code a special unit test infrastructure has been developed in-house.

In addition to the CI tests, system validation tests, to be executed in M1 laboratory, have been developed to verify performance, interfaces, and scalability. They use the traffic generator and Figure Loop simulator to measure control loop latencies, and the ES/PACT/WH simulators to test scalability and interfaces.

In parallel to the testing activities carried out at ESO, the M1LCS SW is verified and validated also by an independent contractor.

Performance

Both Intel and AMD servers comply with the 2ms spec. Using the AMD Epyc processor, sensor's measurements can be acquired in ~1.4ms, new references can be computed in <0.8ms and sent to the actuators in <0.8ms. With respect to the Intel, the AMD gains 1.2ms on the computation stage while it loses 0.4ms in acquisition (Fig. 15). CPU cores load is always below 50% for AMD, and below 60% for the Intel

.

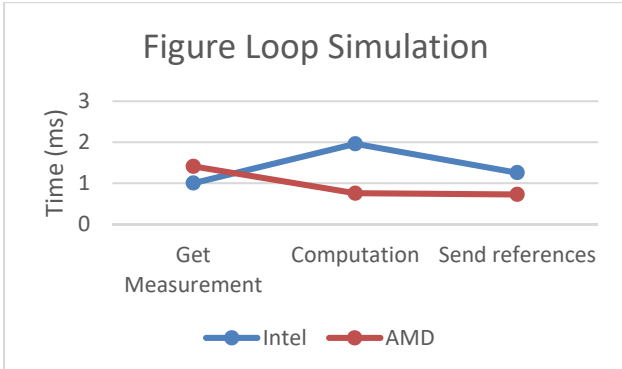

Figure 15: Intel/AMD comparison of latency for each stage of the Figure Loop simulation.

The jitter on SYNC UDP packet measured by the sniffer is ±12µs, close to the ±10µs spec.

FDIR most demanding algorithm to detect wrong ES measurements can run on one isolated core/single thread in 11ms which is well below the 1s requirement and can be improved by scaling on more cores/threads.

CONCLUSIONS

In this paper we have summarized the main characteristics of the M1LCS SW and the results obtained so far on two type of servers. The achieved performance confirms the theoretical analysis on the Figure Loop control algorithm complexity: the problem is limited by the memory bandwidth and not by the CPU. The AMD solution has

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOBPP04

been selected since it provides a flexible NUMA architecture. This flexibility can be used to further improve the system performance.

The SW architecture based on the estimator-controller-adapter pattern and the goal monitoring approach for FDIR showed to be robust and fulfilling the requirements. It was flexible enough to accommodate unforeseen implementation problems and requirement modifications.

REFERENCES

[1] M. Dimmler, J. Marrero, S. Leveque, P. Barriga, B. Sedghi, and M. Mueller,"E-ELT M1 Test Facility", in *Proc. SPIE Ground-based and Airborne Telescopes IV*, Amsterdam, Netherlands, Sep. 2012, vol. 8444, pp. 692-706. doi:10.1117/12.926146

[2] M. Dimmler *et al.*, "Getting ready for serial production of the segmented 39-meter ELT primary: status, challenges and strategies", in *Proc. SPIE Ground-based and Airborne Telescopes VII*, Austin, Texas, USA, Jul. 2018, vol. 10700, pp. 1325-1344. doi:/10.1117/12.2312073

[3] *Precision clock synchronization protocol for networked measurement and control systems*, IEEE 1588-2008.

[4] *Network time protocol*, RFC 1305, version 3.

[5] M. D. Ingham *et al.*, "Engineering complex embedded systems with state analysis and the mission data system", in *Journal of Aerospace Computing, Information, and Communication,* vol. 2, Dec. 2005, pp.507-536. doi:10.2514/1.15265

[6] F. Pellegrin and C. Rosenquist, "The ELT linux development environment", in *Proc. 16th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'17*), Barcelona, Spain, Oct .2017, pp. 1125-1130 doi:10.18429/JACoW-ICALEPCS2017-THBPL05

[7] Boost.Asio, https://www.boost.org/doc/libs

[8] AZMQ, https://github.com/zeromq/azmq

[9] ZeroMQ, https://zeromq.org

[10] SCXML, http://www.w3c.org/TR/scxml

[11] L. Andolfato, G. Chiozzi, N. Migliorini, and C. Morales, "A platform independent framework for statecharts Code Generation", in *Proc. 13th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'11)*, Grenoble, France, Oct. 2011, paper WEAAULT03, pp. 614-617.

[12] Cookiecutter, http://cookiecutter.readthedocs.io

[13] Redis, https://redis.io

[14] Introduction to Intel Ethernet Flow Director and Memcached Performance, White Paper, https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/intel-ethernet-flow-director.pdf

[15] IRQ affinity, https://www.kernel.org/doc/Documentation/IRQ-affinity.txt

[16] PACKET_FANOUT, http://man7.org/linux/man-pages/man7/packet.7.html

[17] J. Argomedo, N. Kornweibel, T. Grudzien, M. Dimmler, L. Andolfato, and P. BarrigJ. Argomedo, "Prototyping the E-ELT M1 local control system communication infrastructure" in *Proc. SPIE, Software and Cyberinfrastructure for Astronomy IV*, Edinburgh, UK, Aug. 2016. Vol. 9913, pp. 635—644. doi:10.1117/12.2232817

[18] D. Harel, "Statecharts: A visual formalism for complex systems", Journal Science of Computer Programming, vol. 8, no. 3, pp. 231-274, 1987.

[19] TwinCAT, https://www.beckhoff.com

[20] PySide2, https://pypi.org/project/PySide2

[21] SNMPSimulator, http://github.com/etingof/snmpsim

[22] Advance Micro Devices, "NUMA Topology for AMD EPYC Naples Family Processors", May 2018.