Monitoring and controlling the SKA telescope manager: a peculiar LMC system in the framework of the SKA LMCs

Di Carlo, Matteo, Dolci, Mauro, Smareglia, Riccardo, Canzari, Matteo, Riggi, Simone

Matteo Di Carlo, Mauro Dolci, Riccardo Smareglia, Matteo Canzari, Simone Riggi, "Monitoring and controlling the SKA telescope manager: a peculiar LMC system in the framework of the SKA LMCs," Proc. SPIE 9913, Software and Cyberinfrastructure for Astronomy IV, 99133S (9 August 2016); doi: 10.1117/12.2231614

SPIE.

Event: SPIE Astronomical Telescopes + Instrumentation, 2016, Edinburgh, United Kingdom

Monitoring and controlling the SKA telescope manager: a peculiar LMC system in the framework of the SKA LMCs

Matteo Di Carlo*[a], Mauro Dolci[a], Riccardo Smareglia[b], Matteo Canzari[a], Simone Riggi[c]
[a]INAF Osservatorio Astronomico di Teramo, Via M. Maggini snc, I-64100 Teramo, Italy; [b]INAF Osservatorio Astronomico di Trieste, Via G.B. Tiepolo, 11 I-34143 Trieste, Italy;
[c]INAF Osservatorio Astrofisico di Catania, Via S. Sofia 78, I-95123 Catania ITALY

ABSTRACT

The SKA Telescope Manager (TM) is the core package of the SKA Telescope: it is aimed at scheduling observations, controlling their execution, monitoring the telescope health status, diagnosing and fixing its faults and so on. To do that, TM directly interfaces with the Local Monitoring and Control systems (LMCs) of the various SKA Elements (e.g. Dishes, Low-Frequency Aperture Array, etc.), exchanging commands and data with each of them. TM in turn needs to be monitored and controlled, in order to ensure its continuous and proper operation – and therefore that of the whole SKA Telescope –. It appears indeed that, while the unavailability of one or more instances of any other SKA element should result only in a degraded operation for the whole telescope, a problem in TM could cause a complete stop of any operation. In addition to this higher responsibility, a local monitoring and control system for TM has to collect and display logging data directly to operators, perform lifecycle management of TM applications and directly deal - when possible - with management of TM faults (which also includes a direct handling of TM status and performance data). In this paper, the peculiarities presented by the TM monitoring and control and the consequences they have on the design of a related LMC system are addressed and discussed.

Keywords: SKA, TM, LMC, TM.LMC, Local Monitoring and Control, Telescope, Fault Management, Logging service, LifeCycle Management

1. INTRODUCTION

In the overall SKA architecture, each of the two telescopes (SKA MID and SKA LOW) is composed by several Elements covering all required functionalities: DISH and MFAA (Mid Frequency Aperture Array, for SKA MID) and LFAA (Low Frequency Aperture Array, for SKA LOW) are the front-end Elements for direct radiation detection, while elements such as CSP (Central Signal Processor), SDP (Science Data Processor), SAT (Synchronization And Timing), INFRA (Infrastructure) and SaDT (Signal and Data Transport) are devoted to all other operational and support functionalities. The global orchestration of this huge system is performed by a central element called Telescope Manager (TM).

SKA Elements (level 2) consist of multiple sub-elements (level 3), which in turn can be decomposed into applications (level 4), components (level 5) and so on, down to the line replaceable units (LRUs). Each SKA Element, in particular, is provided with a Local Monitoring and Control (LMC) system. TM interfaces with the each of the SKA Element LMCs to exchange commands and responses, gather monitoring data, events and alarms, and provide capabilities for diagnostics and upgrades. On the other hand, TM - like any other SKA Element - is provided with its own Local Monitoring and Control (TM.LMC), whose purpose is to support the operations of TM. This includes dedicated software that configures, monitors and controls the performance of the TM as well as ensures that diagnostics can be performed. LMC will also perform lifecycle control for the TM applications. In addition to these functions, LMC plays a special role in the overall "SKA LMC scenario". It must indeed essentially guarantee that, even in case of severe failures of the TM sub-elements (e.g. nodes failure), TM shall be accessible by external operators (engineers) for off-line monitoring data exploration, diagnostics and troubleshooting, in order to be restored as soon as possible. Therefore, TM.LMC must have the following responsibilities:

*matteo.dicarlo@inaf.it; phone +39 0861 439706;

Software and Cyberinfrastructure for Astronomy IV, edited by Gianluca Chiozzi, Juan C. Guzman, Proc. of SPIE Vol. 9913, 99133S · © 2016 SPIE
CCC code: 0277-786X/16/$18 · doi: 10.1117/12.2231614

1. Eliminate common failure modes with TM: if TM fails, the TM.LMC is still able to resolve and/or report the problem,
2. Monitoring and control of
    a) TM process applications and process initialization,
    b) Execution and stopping order,
3. Monitoring TM processes (process status) and hosts (Heartbeat),
4. Report TM element state of the overall TM element or components of it,
5. Configure TM element and its components and support re-configuring the runtime TM element.

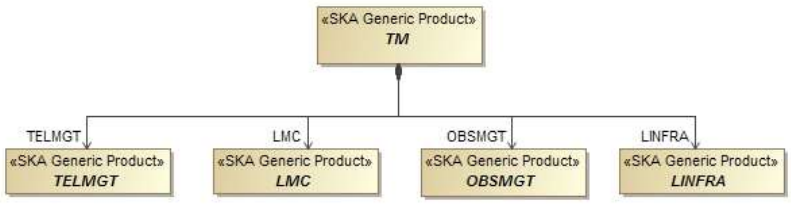

Figure 1 - TM PBS

Figure 1 shows the product breakdown structure of TM [10] that is composed by LMC and other three sub-elements:

- TELMGT, that is the responsible of the hardware telescope management;
- OBSMGT [12][13], that is the responsible of the observation management;
- LINFRA [14], that is responsible of the local infrastructure.

In general, a generic Element LMC is meant to provide a single point of control to TM because it enables TM to view the Element as a single entity and provide a standard interface to communicate with it. Applying the same view to TM.LMC, however, shows that TM needs a single point of control of itself that is very peculiar. In the following sections, the functional aspects and technical design of TM.LMC are analyzed and described.

2. FUNCTIONAL ANALISYS

2.1 Use cases

Figure 2 shows the use cases of the sub-element. They are mostly based on the following concepts:

- monitoring point, which is a specific kind of data that is representative of an aspect of the system and that can have an interest by an operator o by a component,
- event, that is a message indicating that something has happened (for instance an alarm is an event that require a user interaction),
- automatic action that is an LMC action (basically a lifecycle control command) that can be automated following some rules (defined by operators),
- monitoring activity that is a software library (for instance a script) that implements an activity that produce monitoring data.

Based on above concept, the following list explain the interaction between TM administrator, maintainer and developer (for logging there is also a TM component) with the TM.LMC application:

- Add/Remove specific monitoring activity: this case happens when an activity needs to be dynamically added (or removed) into (from) the system;
- Set Operating Mode: this case happens if sub-element implements the SKA Control model (see [10]) and the administrator wants to change it;

- Operating mode management: this case happens when the administrator wants to manage the relation between the valid profiles, valid commands and operating mode;
- Subscribe/Unsubscribe monitoring point: this case happens when a TM sub-element provide a monitoring point and the system needs to read it or not;
- Add/Remove Automatic Action: this case happens when the administrator wants to do a control operation automatically if, for instance, an event has occurred;
- Add/Remove Event: that case happens when the administrator wants to be updated with a message if something happened, e.g. when a monitoring point exceeded a specific threshold;
- Get Application status: this case happens when the administrator wants to retrieve the status of a running instance of an application;
- Manage Logging Architecture/Parameters: this case happens if the administrator wants to change the parameters of the logging service, like the priority of a log transfer;
- Archive logs: this case happens when a software component needs to archive a log message;
- Search for log information: this case happens when a maintainer needs to analyse the log database for an investigation and for this reason he can:
    a. extract log files;
    b. search by date time;
    c. search by words.

Section 3.4 explain the control that TM.LMC is able to do with the lifecycle manager application.

Figure 2 - LMC use cases

2.2 Abstract data model

The design and development of TM.LMC software products will be based on the Abstract Data Model shown in the Figure 3, which generally represents the SKA data model. In this diagram, the central block "Entity" can be an element, a sub-element, an application, a component or an LRU. An element is an aggregation of sub-elements. Sub-elements are, in turn,

aggregations of applications, each being made of components that finally are aggregations of LRUs. In software, an LRU must be replaceable based on version control, so it can be seen as a library or an assembly.

An entity is associated with a list of Configuration objects (at least one) and a list of Version objects. The first one can be composed by different things like a Dictionary (Hash table), a timestamp (useful to retrieve the last configuration of that entity) and a property indicating if the configuration object is default or not. Sometimes it is associated with a specific executable file (for instance in case the configuration object represents an item of an IT Automation Software like Chef [8] or Puppet [7]). The second one indicates if the Entity is active in a specific version and if it is alpha or beta (for testing reason).

The concept of Configuration in TM.LMC is different compared to those shown in other TM sub-element. In fact, TM.LMC will need (to access) the following information:

a) Information on the distributed nature of the application (for instance where the application runs);

b) Information for start-up configuration (local files, database or other objects needed);

c) Information on dependencies with other objects (for instance an application could depends of another application like the Tango-controls framework depends on ZeroMQ library).

The derived block Element is associated to a list of capabilities, which have a health status. The concept of capability and health status is the core of TELMGT sub-element and is out of the scope for the present paper.

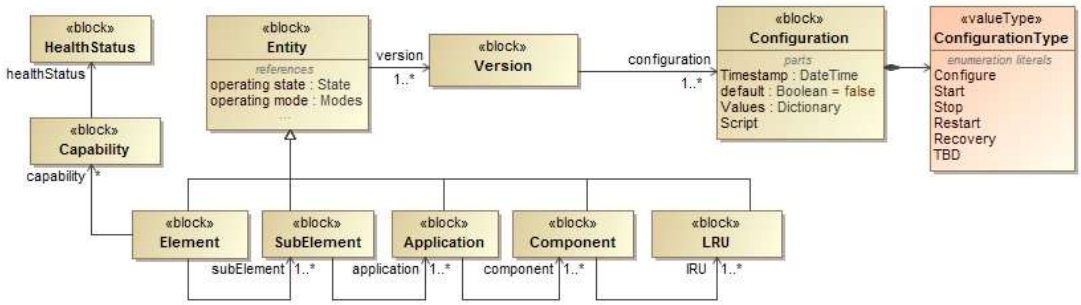

Figure 3 - Abstract Data Model

2.3 Product breakdown structure

The diagram in Figure 4 shows the product breakdown structure of the TM.LMC. The main products are software system monitor, TM monitor, lifecycle manager, launcher UI, help online, logging service, fault management, local configuration DB.

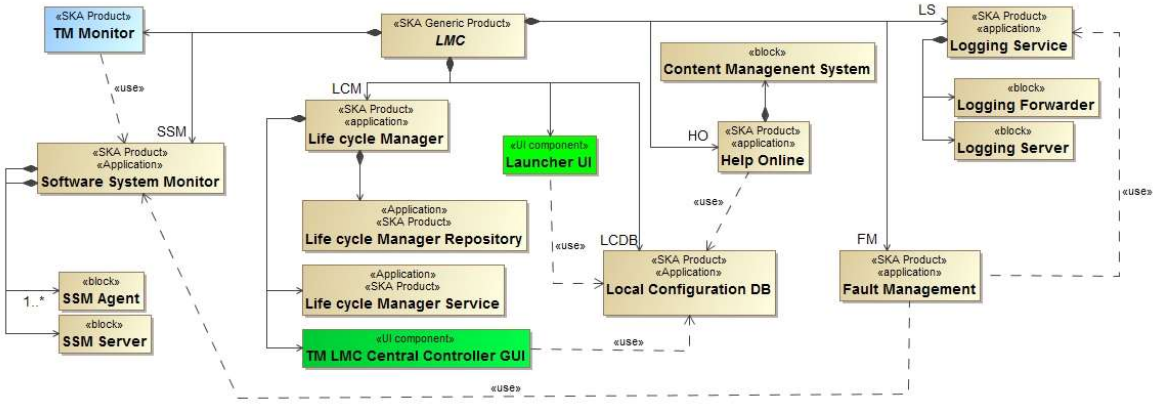

Figure 4 - LMC PBS for SKA Low and Mid

3. DESIGN ANALYSIS

3.1 Software system monitor

The TM.LMC monitoring system is composed by a Software System Monitor (SSM) and a list of specific monitor components provided by sub-elements. A software system monitor is a software component used to monitor resources and performance in a computer system: for every node of the TM network, there is going to be a local agent that is able to collect information from the operating system and from the local processes of TM Sub-elements. Having such a piece of software in every node of the network can easily provide the ability of monitoring other aspects of TM that are network services (for example ICMP, SNMP, FTP, SSH, TCP) or host resources (processor load, disk usage, system logs) and every monitoring points chosen and decided by TM teams. Next to this, in coordination with other TM sub-element teams that are in charge of the development of their piece of software, for every other sub-elements, there is the possibility to have a specific monitoring activity. The goal in this case is to find a list of monitoring points for every application so that it is possible to make a strategy (technique) for fault management (for instance black box/white box error handling). Figure 5 shows the abstract data model for the Monitoring System. The fundamental block is the MonitoringActivity that describes the process or computer to monitor; sub-element developer writes it together with an LMC administrator. It generates a set of MonitoringPoint that represents or a resource to measure or a service to test. The runtime result of the measure/test represents the Monitoring Data passed to the SSM Server.

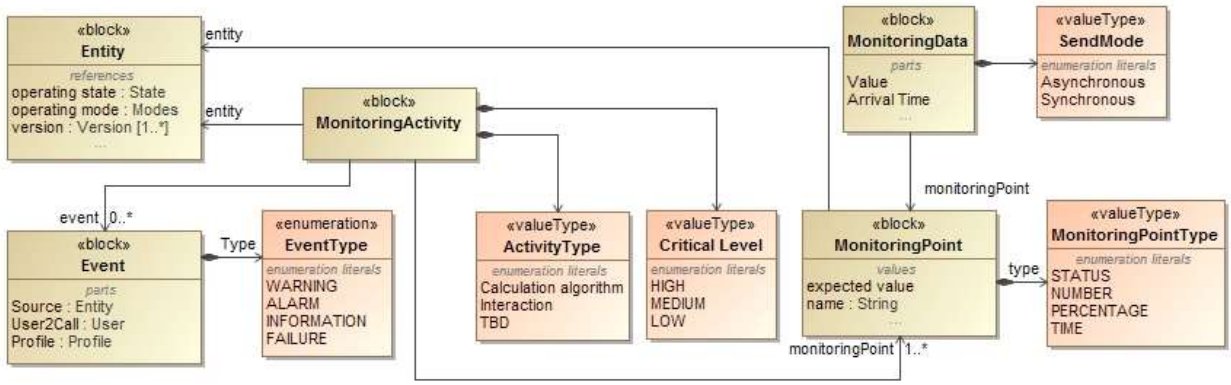

Figure 5 –Monitoring activity data model

Different implementation solutions have been considered by TM.LMC either based on open-source or commercial frameworks, among them Nagios [4], Zabbix [5], SolarWinds [6]. A prototype has been developed based on Nagios.

3.2 TM monitor

In the SKA project, each Element has a Local Monitoring and Control component (e.g. in the case of Dish Element, an LMC instance (named DSH.LMC) will be present for each dish antenna). LMCs report status, provide monitoring data handle events/alarms requiring prompt local response/reaction time and escalate alarm situations that need either non-local handling or operator visibility to TM, who performs system level monitoring and control at a higher hierarchical level. TM also performs orchestration of all the Elements in SKA to achieve science goals generating commands to Element LMCs, which implement the internal control logic needed to execute commands, capture responses, consume events from the Element LMCs, and be a broker for distributing events from some Element to other Element that may subscribe to them.

As for any other SKA element, TM needs its LMC even if it is very peculiar compared to others. In fact, while other element LMC accepts commands from TM, TM.LMC cannot do it because it could bring to a potential circular path problem (TM sends a command to itself). To avoid this, TM.LMC implements a component that expose every information relevant to TM: the TM monitor.

In order to ease the development of the LMCs, a common framework has been chosen by the SKA community: the tango control framework [1]. Therefore, the TM monitor application is a tango device server that exposes, dynamically, a list of attributes that report status, monitoring data and escalate alarm situations to operator (instead of TM).

3.3 Fault management

The monitoring of TM applications can be performed at three different levels of depth:

1. generic level (RAM allocation, CPU usage, …),
2. generic level + process status (start/end execution, stop execution, waiting for input, output sent, …),
3. correctness of operations (e.g. coordinate conversion by a devoted application).

The boundary of monitoring responsibility in TM was carefully analyzed. TM.LMC will be responsible for monitoring at level 1 and 2, while the third level is to be performed internally by TM sub-elements. The correctness of operations is a duty of the individual sub-elements. The Fault Management uses (as shown in the generic LMC PBS) the monitoring system (that is the Software system monitor and the specific activity that can be done thought it) in order to perform its duty that is:

1. Detection, that is the ability to understand if in the system there is a fault;
2. Isolation, that is the ability to isolate a fault understanding where it is;
3. Recovery that is the ability to recover the situation.

A monitoring activity together with alarm filtering (usually available in any software system monitor) realizes the detection activity. The same monitoring activity together with log information realizes the isolation while the recovery is essentially a control operation that TM.LMC can do: for instance an online action, which is a lifecycle command (reconfigure, restart, etc.) or an offline activity like raising a modification request for the software maintenance.

3.4 Lifecycle manager

The lifecycle management is the ability to control a software application in the following phases of its lifetime: configuration, start, stop, update, upgrade or downgrade (version control).

The first phase is the configuration, which is the ability to set all the parameter of a software application in order to start the product on the second phase (the configuration phase is the preparation of the start phase). Once the application has started, it is possible for a user to work with it. In the start phase, it is very important to consider the application typology (os service, web application, desktop application and so on). In fact, if the application is on web (web app), for instance, then the start phase correspond to the start of the web server and perhaps the database. Only after that (and after a test phase), a user can work with it. An application can also be stopped or killed if there is the need of it, for instance because an application goes offline or there is a new version of it, either resulting from the standard update cycle or from a re-designing stage triggered by new requirements released over the SKA life-time. Having a lifecycle manager software (LMS) brings to a similar way of controlling all TM sub-elements, so that TM.LMC can be the main entry point for TM and the users can see TM as one big application. All this activity can be done through an IT automation tool like puppet [7], chef [8], ansible [9] and so on in cooperation with other sub-elements. As shown in the LMC PBS (Figure 4), the lifecycle manager is composed by a repository of configuration items (for example in chef they are ruby scripts) and potentially an agent that execute the configuration needed. LMS has also the responsibility to ease the job of the administrators in all the phases of an application lifetime. This means that it should simplify the user job bringing him to finish its work correctly. For the above reason, the LMS includes testing mechanisms (for instance a checklist of things to test in order to configure and start a server application correctly).

Figure 6 shows the use cases for the lifecycle management.

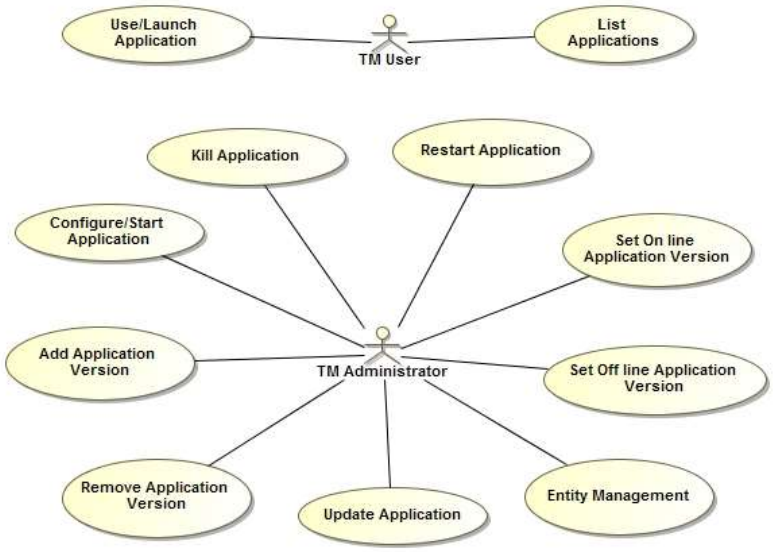

Figure 6 - Lifecycle management use cases

3.5 Help online

The TM maintenance plan [11], introduces the concept of modification request (MR) that is a generic term including the forms associated with the various trouble/problem-reporting documents (e.g. incident report, trouble report) and the configuration change control documents. A MR starts the maintenance process that is the responsible of the correct modification of the software and deliver the modified code and the related documentation, including the help. For this reason, there is a need for TM to have a help online that must include the correct (versioned) help information. This brings the needs, for the developers, to have a platform for the creation of help item (a set of information related to a specific application and coded into a file in a specific format, for instance xml or json). The benefit to have such a system is because using the same architecture will allow developers to concentrate their efforts on the content of their help.

A suitable solution for the help online is a content management system (CMS) that allow the following functionalities:

1. Generate a help item in the form of file (Figure 7);
2. Show the help item on request.

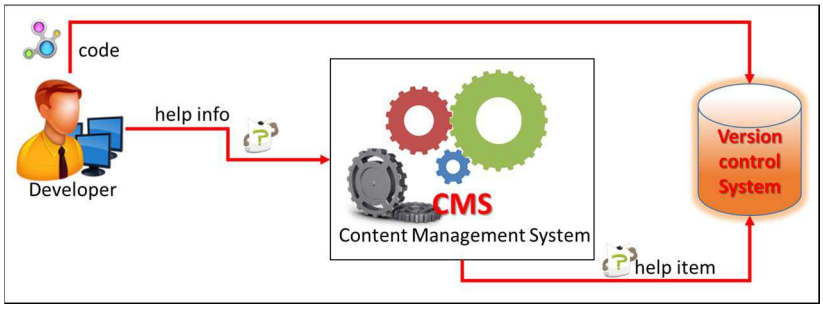

Figure 7 - A developer must write code and help information

3.6 Local configuration DB

The local configuration DB is an archive of data information supporting the LMC applications (lifecycle manager, monitoring system and so on). In specific, there are needs to store geographical information, entities information, version information, configuration information and so on.

3.7 Launcher UI

TM is composed by a set of applications which operate together to provide telescope functionalities to users. Since there are different users and different applications (potentially) written in different technology, the possibility to have a common access point for the system can be beneficial. In the overall UI concept, an important distinction is between:

- Online systems (the ones involved in the running of the telescope from within the control room environment),
- Offline systems (ones primarily used by office staff at HQ to prepare and schedule projects that do not directly affect the telescope)
- Proposal tools (generally available to interested members of the science community).

The Launcher UI is the online entry point for every applications (so online, offline and proposal tools) being aware of the location of the user in accessing the application list. In Figure 8, it is highlighted the LMC sub-elements as layer which connect the user to the applications. The LMC layer is colored in dark grey and highlight its components: the lifecycle manager, the monitoring system, the logging service, the padlock icon (which indicate the link to the A&A module that should be infrastructure for every tool), the help button (the online help) and the software configuration management (IT automation tool). The padlock icon has been inserted because not all the applications are available for all the users and in fact for TM.LMC it will be very important the knowledge of the authorizations of the user (to list applications the user can start). Another input for the application list is also the knowledge of the site location: if the user is near the telescope, it could access a specific application that potentially are not available at the head quarter. Another important aspect is the online help: every applications should have its own help that must be accessible together with the application the user want to work with.

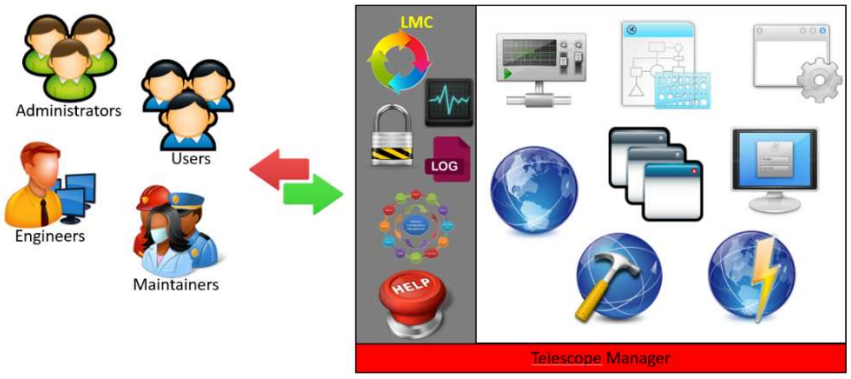

Figure 8 - One access point for many applications

3.8 Logging service

Figure 9 shows the data flow of log information from the client applications (all SKA Applications) to the server data center in the proposed architecture. The central component is a service (generally called Log Forwarder) located near the applications that give the possibility to forward the message to the central database cluster. The forward mechanism has a priority level for every message:

- Lower priority: the log messages (files) are copied from a local folder to the main data center without any decrease of network performance;
- Medium priority: the log messages (files) are directly written in a network path or in a cloud folder so that it can be synchronized with the main data center;
- High priority; the log messages (files) are directly sent through a TCP connection because there a need for the maintainer or the developer to look at them as soon as possible.

The proposed architecture is a best practice and composed by three main entities: the server (or data center), the client and the forwarder. The data center is the entity responsible for collecting and organizing the message from every applications; the client helps the applications to compose the log message (for instance with macro or template) and can be configured

remotely to use a forwarder which is the entity responsible for the transfer of the information to the server. Another important aspect is the possibility for the users of every TM applications to retrieve the log files that they are interested in. This possibility is realized by a web application usable by every user that can directly query the data center in order to retrieve entire log file or a collection of log files or a specific result of a query. The growth of event data should be controlled in order to avoid the storage of unused information and in order to maintain an enough amount of data persistent without the risk of data flooding. There are different possibilities and one of the most used one is to have a fixed size for data and drop all the messages that exceed that data. Other possibilities are with the use of some noSql technology like for instance MongoDB that allow the operator to have multiple DB instances and directly drop an instance when needed.

Since there are different valid possibilities on the market, there is no need for rebuild a system that is already available. Below, two different technologies are proposed for the logging architecture: syslog-ng and Elasticsearch.

Syslog-ng [2] is a scalable system logging for a centralized logging solution. It realizes the architecture described above with three types of entities: the server, the client and the relay. Within a client host there is an agent that collect every log messages from various applications and/or devices. In the client configuration there can be some global object like log paths, filters, network destinations and so on. The log path is used to connect source to destination and it can include one or more filter (usually regular expression used to select messages), a parser (for instance a json parser to store the log information in a mongo DB server) and rewriting rules (to completely change a message into another format). There are many possible destinations within this product like relational database, non-relational DB, Elasticsearch server, plain files and many others.

Elasticsearch [3] solves the problem with a full-text search and an analytics engine. Logstash is designed for collecting, aggregation, parsing data and putting them into Elasticsearch in order to run searches and aggregations to mine any information of interest. Kibana is designed for analytics/business-intelligence needs, to quickly investigate, analyse, visualize, and ask ad-hoc questions on many data (millions or billions of records). In fact, it is possible to build custom dashboards that can visualize aspects of the data that are important. According to the architecture described in Section 4, Elasticsearch is the data centre, Logstash is the forwarder and Kibana is a visualizer engine (the globe in Figure **9**). According to Elasticsearch documentation, there are different concepts that need to be understood to realize the product. The basic unit of information that can be indexed is a document that is expressed in JSON. An index is a collection of documents that have similar characteristics; within an index, it is possible to define one or more types which are a logical category (or partition) of your index (the semantic is up to the designer). Since log data can be considered as big data, an index can store a large amount of them and exceed the hardware limits of a single node. It is possible to subdivide the index into multiple pieces called shards. When defining an index it is important to define the number of shards as well for two main reasons: to scale the content volume horizontally and to distribute and parallelize operations across shards which will result in increasing performance/throughput. It is also possible to have replica shards (a copy of a shard) to provide high availability in case a node/shard fails and to scale out the search volume/throughput since they can be executed on all replicas in parallel. Elasticsearch provides a very comprehensive and powerful REST API to interact with the cluster for any type of operations.

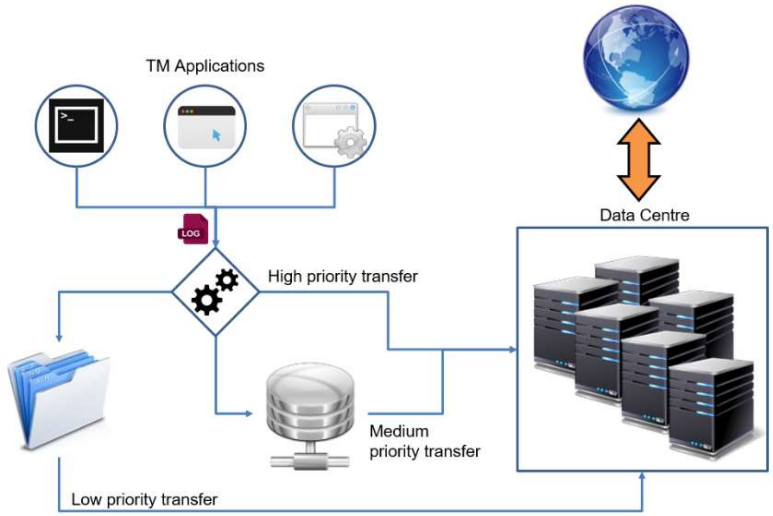

Figure 9 - Logging service strategy

4. CONCLUSION

This paper has discussed the peculiarities of the TM local monitoring and control and the design of a related LMC system. It is important to highlight that the detailed design is still ongoing and some evolution and refinement is expected. There are also system level changes and decisions expected that would trigger architecture changes and, for this reason, this paper should be seen as an overview of the current design more than the final design of the TM.LMC.

Acknowledgements: this work is supported by the Italian Ministero dell'Istruzione, dell'Università e della Ricerca.

REFERENCES

[1] www.tango-controls.org
[2] https://www.balabit.com/network-security/syslog-ng
[3] https://www.elastic.co/products/elasticsearch
[4] www.nagios.org
[5] www.zabbix.com
[6] www.solarwinds.com
[7] puppet.com
[8] www.chef.io
[9] www.ansible.com
[10] Swaminathan Natarajan et al. 2016: SKA Telescope Manager (TM): Status and Architecture Overview, Astronomical Telescopes and Instrumentation 2016, Proc. of the SPIE, paper 9913-1, 2016 (this conference)
[11] Mauro Dolci, Matteo Di Carlo, Riccardo Smareglia 2016: Challenges and strategies for the maintenance of the SKA telescope manager, Astronomical Telescopes and Instrumentation 2016, Proc. of the SPIE, paper 9913-90, 2016 (this conference)
[12] Alan Bridger, Stewart J. Williams, Mark Nicol, Pamela Klaassen, Roger S. Thompson, Cristina Knapic, Giovanna Jerse, Andrea Orlati, Marco Messina, Snehal Valame 2016: Observation management challenges of the Square Kilometre Array, Proc. of the SPIE Astronomical Telescopes and Instrumentation 2016, paper no. 9913-35 (this conference)
[13] Stewart J. Williams, Alan Bridger, Subhrojyoti R. Choudhury 2016: The SKA observation control system, Proc. of the SPIE Astronomical Telescopes and Instrumentation 2016, paper no. 9913-98 (this conference)
[14] Domingos Barbosa, João Paulo Barraca, Dalmiro Maia, Antonio Cruz, Gerhard Le Roux, Swaminathan Natarajan, Paul Swart, Yashwant Gupta 2016: A cyber infrastructure for the SKA Telescope Manager, Proc. of the SPIE Astronomical Telescopes and Instrumentation 2016, paper no. 9913-20 (this conference)