TM Services: an architecture for monitoring and controlling the Square Kilometre Array (SKA) Telescope Manager (TM)

Di Carlo, M., Canzari, M., Dolci, M., Smareglia, R., Barbosa, D., et al.

M. Di Carlo, M. Canzari, M. Dolci, R. Smareglia, D. Barbosa, J. B. Morgado, J. P. Barraca, "TM Services: an architecture for monitoring and controlling the Square Kilometre Array (SKA) Telescope Manager (TM)," Proc. SPIE 10707, Software and Cyberinfrastructure for Astronomy V, 107071N (6 July 2018); doi: 10.1117/12.2309341

SPIE.

Event: SPIE Astronomical Telescopes + Instrumentation, 2018, Austin, Texas, United States

TM Services: an architecture for monitoring and controlling the Square Kilometre Array (SKA) Telescope Manager

M. Di Carlo*[a], M. Canzari[a], M. Dolci[a], R. Smareglia[b], D. Barbosa[c], J. B. Morgado[e], J.P. Barraca[d]
[a]INAF Osservatorio Astronomico d'Abruzzo, Teramo, Italy; [b]INAF Osservatorio Astronomico di Trieste, Trieste, Italy; [c]Instituto de Telecomunicações, [d]Universidade de Aveiro, Campus Universitario de Santiago, 3810-193 Aveiro; [e]CICGE, Faculdade de Ciências da Universidade do Porto, 4430-146 V. N. Gaia

ABSTRACT

The SKA project is an international effort (10 member and 10 associated countries with the involvement of 100 companies and research institutions) to build the world's largest radio telescope. The SKA Telescope Manager (TM) is the core package of the SKA Telescope aimed at scheduling observations, controlling their execution, monitoring the telescope and so on. To do that, TM directly interfaces with the Local Monitoring and Control systems (LMCs) of the other SKA Elements (for example, Dishes, Correlator and so on), exchanging commands and data with them by using the TANGO controls framework (see [1]). TM in turn needs to be monitored and controlled, in order its continuous and proper operation is ensured and this higher responsibility has been assigned to the TM SER package.

Keywords: SKA, TM, TANGO, SER, Services, LMC, Virtualization

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) [2]. 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. TM in turn needs to be monitored and controlled, in order its continuous and proper operation is ensured and this higher responsibility has been assigned to the TM Services (TM SER) package.

The problem of monitoring and controlling a software is an artificial intelligence problem consisting in the research in a state space characterised by:

1. an initial state;
2. a set of possible actions which transform a state to another one;
3. a path from a state to another state (list of actions).

This description of the problem helps in understanding what is needed to realize the architecture for the TM SER that is a set of actions and a set of monitoring data from which a state can be calculated.

* matteo.dicarlo@inaf.it; phone +39 0861 439711; fax +39 0861 439740; www.oa-abruzzo.inaf.it

Software and Cyberinfrastructure for Astronomy V, edited by Juan C. Guzman, Jorge Ibsen, Proc. of SPIE
Vol. 10707, 107071N · © 2018 SPIE · CCC code: 0277-786X/18/$18 · doi: 10.1117/12.2309341

The analysis carried out on TM requirements yielded the main system's functions shown in Figure 1 and Figure 2. They can be summarized in the following list:

- TM generic monitoring and fault management to detect internal failure and gather TM performance;
- TM lifecycle management to manage the versions of the TM and the TM applications which includes: configuration of TM software applications, starting, stopping and restarting of TM software applications, update and downgrade of TM software applications;
- TM Logging, which includes the control of the destination of log messages, the transformation of the message (if required) and the query GUI;
- Control of the virtualization system (see [3]).

Another important function of the system is the aggregation of the TM health status and TM State (of the various TM applications) and its reporting it to the Operator. This function can be considered an application of the present architecture (see [4]).

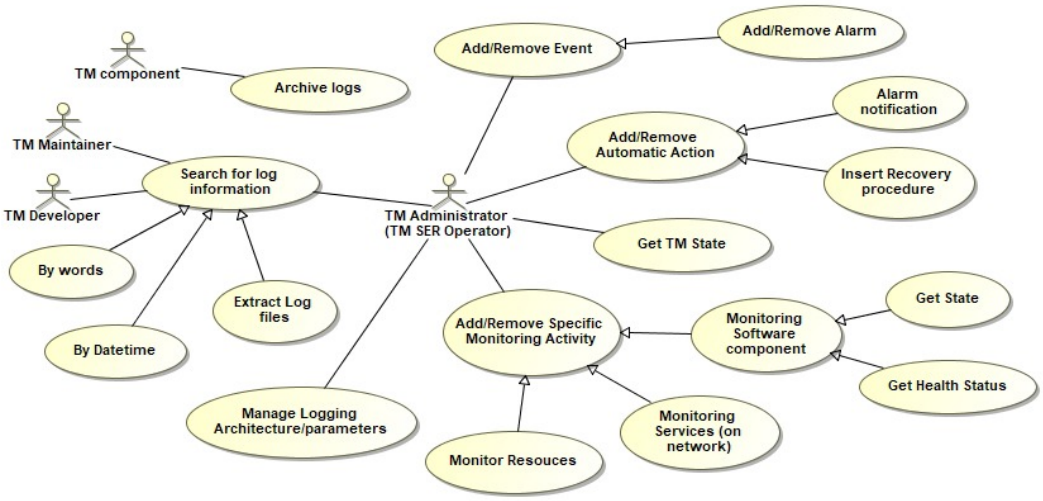

Figure 1. TM SER use cases, monitoring and logging.

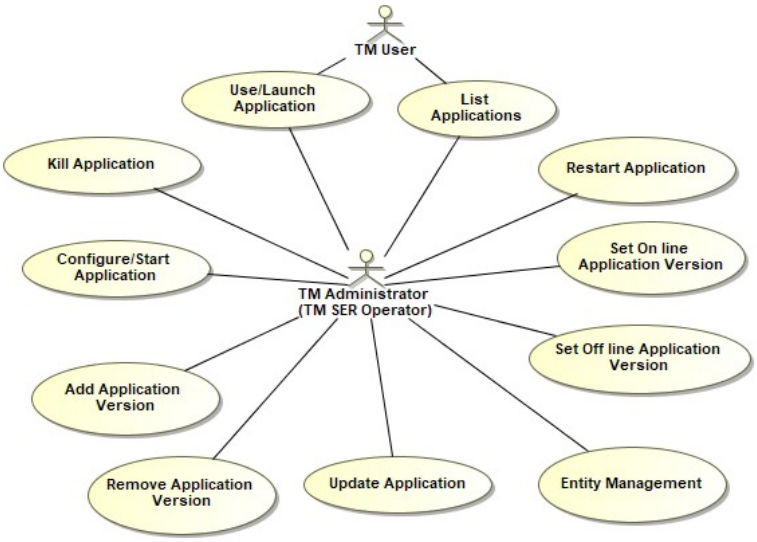

Figure 2. TM SER use cases, lifecycle.

2. CONTEXT

The TM Services take place in the middle between the domain logic and the infrastructure. In particular, Figure 3 explains the above concept with a layered structure:

- Domain/Business Layer: functional monitoring and control of business logic performed by each application;
- Services Layer: Monitors and controls processes on a generic level (non-functional) like web services, database servers, custom applications;
- Infrastructure Layer: Monitors and controls virtualization, servers, OS, network, storage.

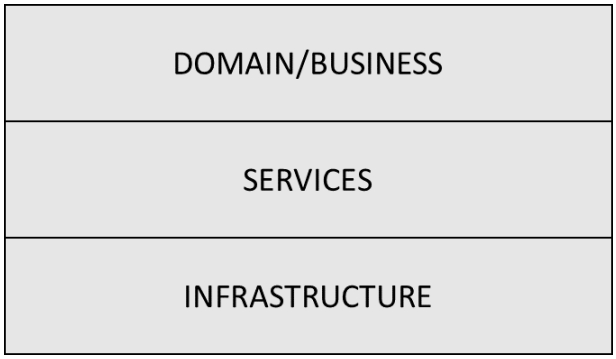

Figure 3. TM SER Context.

3. QUALITY ATTRIBUTES

The main quality that drove the development of the present architecture was the maintainability intended as availability (reliability and recovery), modifiability, testability and more in general the ability of a system to cope with changes.

Concerning availability, many tactics are implemented in the present architecture and in particular:

- Detecting fault: ping, monitoring activities, heartbeat and timestamp;
- Recovering from fault: active redundancy, software upgrade and reconfiguration;

- Preventing faults: predictive model and transaction (when accessing to repositories).

The modifiability is reached in the following areas of the system: monitoring activities, lifecycle scripts, logging rules and fault rules. In particular, cohesion has been increased and coupling reduced so that it is easy to add new version of an application and to add new monitoring activities.

The testability is reached by limiting the complexity of the system. In fact, it is easy to add a new monitoring activity to the system so that, if there is a new test to be performed, it will be possible to add a monitoring point for it that can represent a state, a measure or a simple message. Once the needed specific monitoring point is available, it is also easy to generate an event to intercept the problem raised with the test.

4. ENTITIES DECOMPOSITION

The entity managed by the TM SER package can be summarized in Figure 4. In particular, the central block of the diagram is the Entity, that is the main data wherewith every TM Services application refer to. It can be:

- a monitored process, that is an OS process that needs to be monitored and controlled or
- an application, that is an aggregation of MonitoredProcess selected according to a particular version with the block LogicalComposition or
- a monitoring activity, that is a process that monitors an entity and produces monitoring data or
- the virtualization, that is a generic term indicating a virtualization system (Openstack, Cloud system and so on) or
- a vResource, that is a resources managed by the virtualization including CPU, storage, and networking or
- a Template, that is a description of a set of instances (servers, VMs or containers) needed to run a particular configuration of an application.

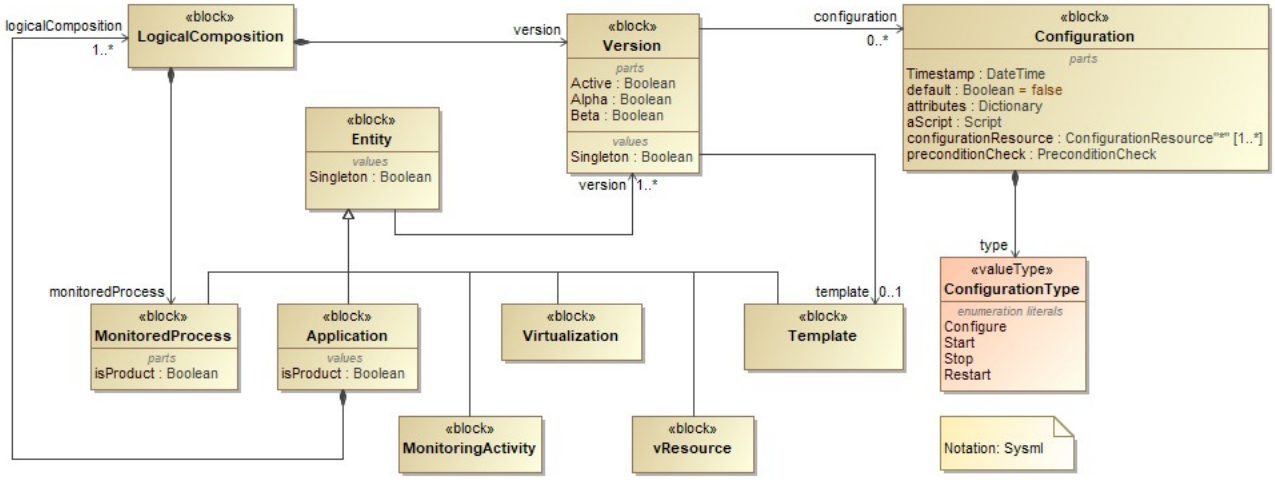

Figure 4: Entity decomposition.

5. MODULE DECOMPOSITION

Figure 5 shows the decomposition of the system into units of implementation and highlights the distinction between off-the-shelf software and built software.

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

engine (off-the-shelf) executes the lifecycle scripts directly into the client host (where the application run) and realizes a specific phase of the application lifetime.

In particular, 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, for instance, on web (web app) then the start phase corresponds 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 for doing that, for instance because the application goes offline or there is a new version of it, either resulting from the standard update cycle or from a redesigning stage triggered by new requirements released over the SKA life-time. All these activity can be done through an IT automation tool like puppet [5], chef [6], ansible [7] and so on.

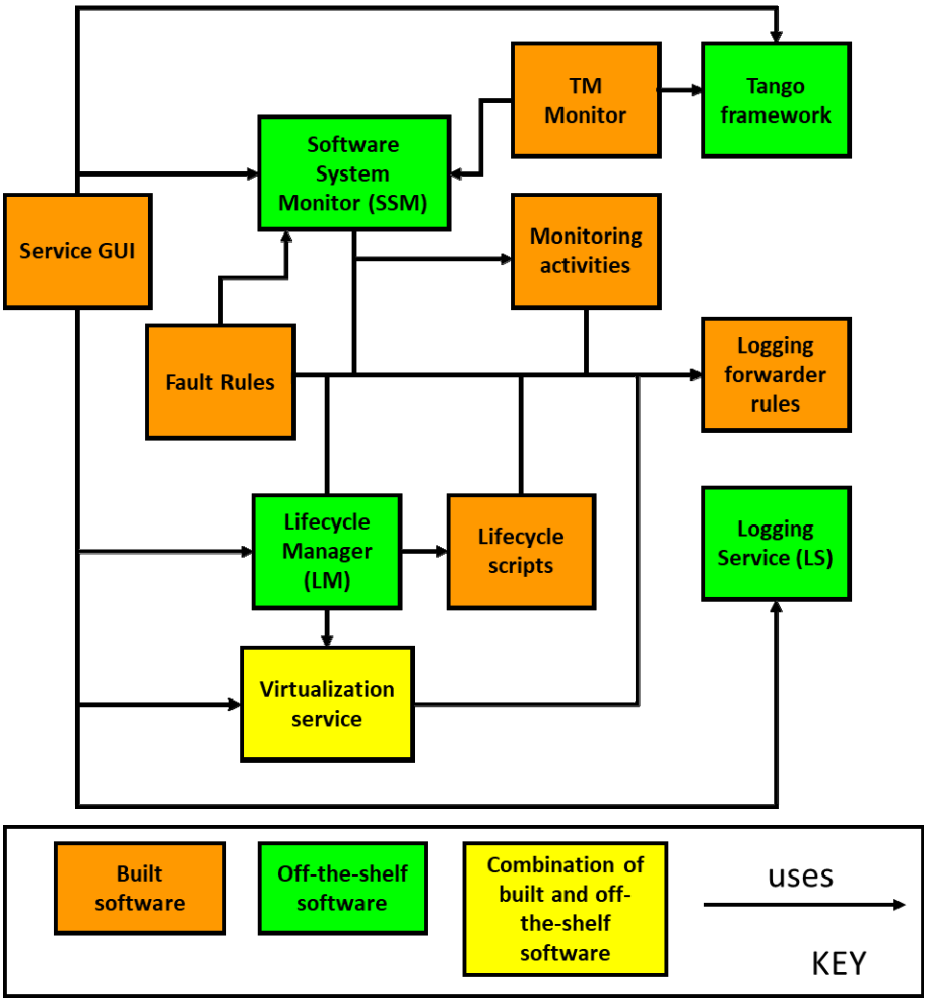

Figure 5. Module decomposition.

The Generic Monitoring is composed by a software system monitor (SSM) plus some specific monitoring activities in order to monitor:

- network services (SMTP, POP3, HTTP, NNTP, ICMP, SNMP, FTP, SSH);
- host resources (processor load, disk usage, memory, etc.);

- any hardware (like probes for temperature, alarms, etc.).

The SSM is a software component used to monitor resources and performance in a computer system: for every node of the TM network, there is a local agent that is able to collect information from the operating system and from the local processes of TM applications. The local agent executes one or more monitoring activities and reports the collected data to the SSM engine.

The SSM is also responsible of the aggregation of the TM health status and TM State (of the various TM applications). This function can be accomplished through the development of specific monitoring activities both for the local agent and for the server engine that collect state information (client) and aggregate them (server).

Every information collected by the monitoring system has to be reported to the Operator in order to give a clear picture of the functional and non-functional view of the system. For this reason, a TM Monitor has been developed as a Tango Device server (www.tango-controls.org) for reporting all the monitoring information into the control system (see Fig. 4).

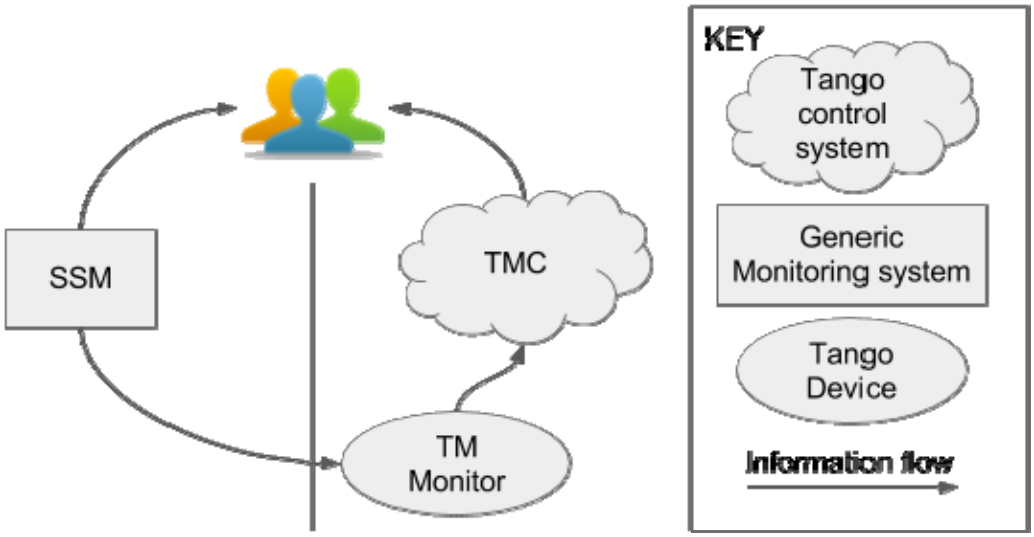

Figure 6. Reporting to Operator.

The fault rules are the basis of the fault management activity that uses the SSM and the LM in order to perform its duty that is:

- Detection that is the ability to understand if there is a fault in the system;
- Isolation, that is the ability to isolate a fault by understanding where it occurred;
- 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: 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.

A good logging service should focus on how many inserts can the architecture support (throughput) and how the system manages the growth of event data. In the module decomposition, two different modules are devoted to these two distinct aspects, one called Logging forwarding rule and one called Logging Service (LS).

The Service GUI is the entry point for the TM Services software package and will allow the Operator to access all the functionalities provided from one single UI (see [8]).

It is very important to notice that Monitoring activities, fault rules, Lifecycle scripts and Logging forwarding rules are separated from the execution engine (SSM vs Monitoring activities, SSM vs fault rules, Lifecycle Manager vs Lifecycle scripts, Logging service vs Logging forwarding rules) in order to increase the modifiability of the system. This also highlights the Client/Server architecture of the system.

6. RUNTIME VIEW

Figure 7 highlights the runtime components of the system and their relations. Highlighted in green are the runtime components of the logging service with a three entities architecture: the LS data repository, the LS engine and the LS forwarder. The LS forwarder is a service located near the TM applications that gives the possibility to forward the message to the central database cluster according to certain rules (logging forwarding rules).

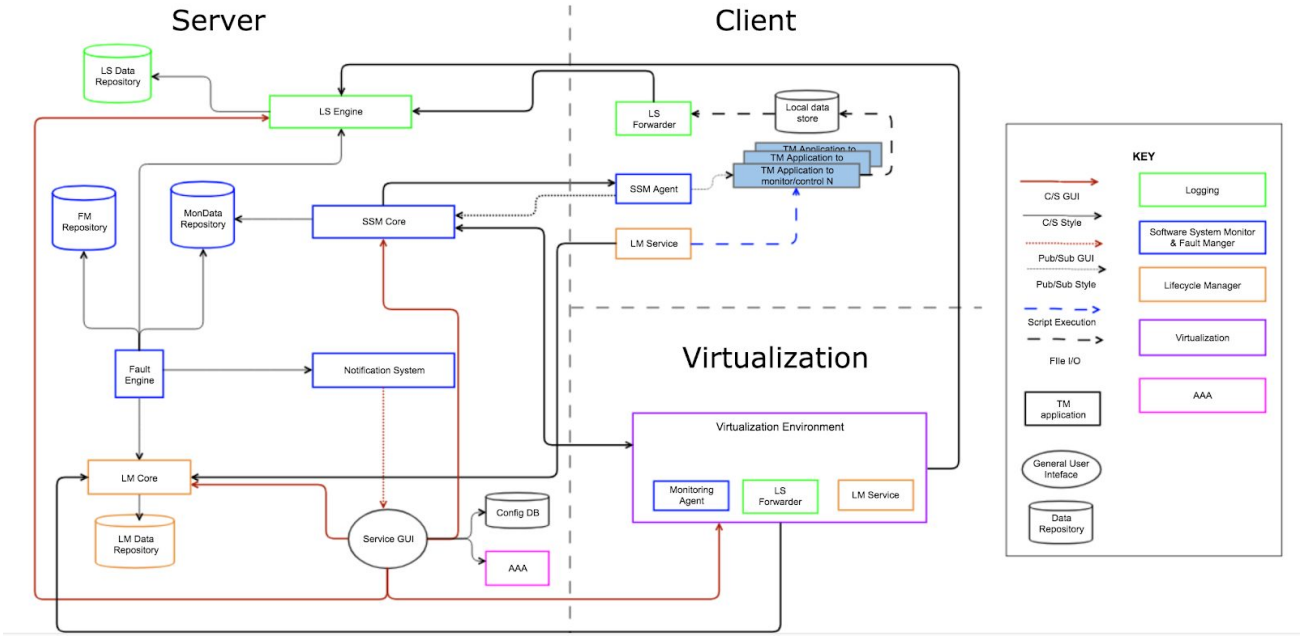

Figure 7. Runtime view.

The LS Engine is responsible for collecting and storing the log messages (transformed by the forwarder) into the data center together with the ability to receive queries from the external world and to answer in a timely manner. There are several best practises and possibilities for this service. It is possible to use simple files (as in the case of MeerKAT project, see [9]) or a relational DB (as for ASKAP, see for example [10]) or a NoSql DB (as for LHC, see for example [11] with ELK [12]). A performance evaluation of the three possibilities has been made and, focusing on fast write and centralized solution, the best solution resulted to be ELK. In fact, writing simple file is fast but is not a centralized solution, relational DB is a centralized solution but is not fast while the NoSql DB is both fast and centralized.

Highlighted in blue are the runtime components of the SSM that is composed by:

- the SSM Core, responsible for the interaction with the SSM Agents,
- the Fault Engine, responsible for the execution of the fault actions (if required),
- the Notification System to notify the SER Operator of any information and
- two repositories, the FM repository, responsible for the storage of the fault rules, and the MonData Repository, for the archiving of all the monitoring data.

Highlighted in orange are the LM runtime components: the LM Core, the LM repository and the LM Service. The LM Service retrieves from the LM Core the specific lifecycle script assigned to the particular TM application and applies it locally so that the requested action is performed. It is also possible to have an agentless lifecycle manager like for instance Ansible (see [7]). The LM repository contains all the versioned lifecycle scripts developed and it interacts with the LM Core only.

It is also important to notice that the Service GUI interacts with all the principal components of the SER modules together with a configuration DB (to maintain information like geographical information, entities information, version information, configuration information and so on) and the AAA (Authentication, Authorization and Auditing) package.

The Virtualization block manages resources (vResources) assigned to a specific configuration of an entity (see Figure 4 for the entity decomposition). Usually, every entity has associated a template that is a description of the set of instances (servers, VMs or containers) with an SLA (Service Level Agreement), user ACLs (Access Control Lists) and network ACLs needed by the entity.

A vResource can be:

- a vResourceCompute, that is a Virtualized computational resource (an Hardware, a vHardware, a Container or a Virtual Machine);
- a vResourceNetwork, that is a Virtual network for a cloud application;
- a vResourceStorage, that is a virtual storage for a cloud application.

The Template and the vResource blocks have a state associated that is collected and managed by the SSM. Figure 8 shows the data model for the use of the virtualization system made for TM.

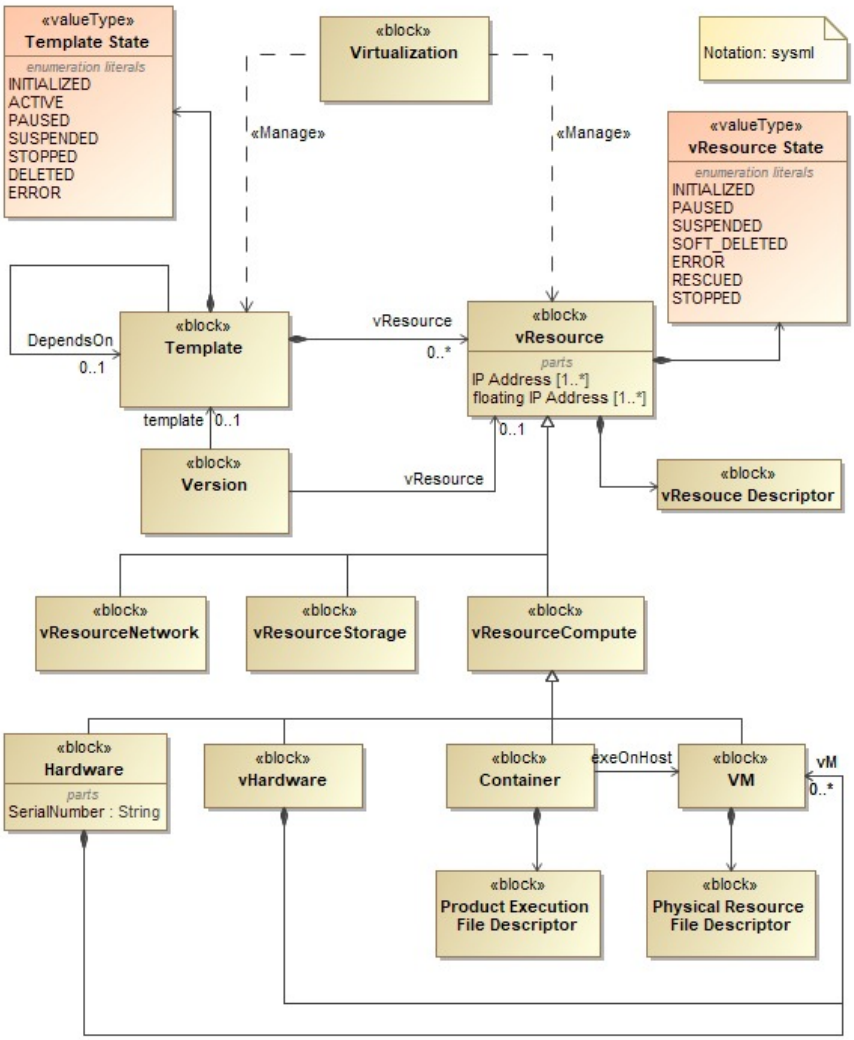

Figure 8. Virtualization data model.

7. CONCLUSION

The architecture developed for the TM Services package, devoted to monitoring and control the SKA1 Telescope Manager system, has been described in this paper. The work has successfully passed the SKA1 Pre-Construction Phase TM Critical Design Review and is ready for construction. The SKA1 Construction Phase is expected to start in 2019, while the overall SKA Telescope is expected to enter its fully operational phase in 2025.

8. ACKNOWLEDGEMENT

This work has been made possible thanks to the financial support by the Italian Government (MEF - Ministero dell'Economia e delle Finanze, MIUR - Ministero dell'Istruzione, dell'Università e della Ricerca). This research is also supported by the project Enabling Green E-science for the SKA Research Infrastructure (ENGAGE SKA), reference POCI-01-0145-FEDER-022217, funded by COMPETE 2020 and FCT, Portugal.

REFERENCES

[1] TANGO, www.tango-controls.org.
[2] Bridger, Alan et al., The SKA telescope manager software: a status update, Proc. of the SPIE Astronomical Telescopes and Instrumentation 2018, paper no. 10707-2 (this conference)
[3] Morgado, J. Bruno et al., Very large scale high performance computing and instrument management for high availability systems through the use of virtualization at the Square Kilometre Array (SKA) telescope, Proc. of the SPIE Astronomical Telescopes and Instrumentation 2018, paper no. 10707-20 (this conference)
[4] Dolci, Mauro et al., Achieving a rolled-up view of SKA TM health status and state: definition and analysis of aggregation methods, Proc. of the SPIE Astronomical Telescopes and Instrumentation 2018, paper no. 10707-19 (this conference)
[5] PUPPET, http://puppet.com
[6] CHEFF, http://www.chef.io
[7] ANSIBLE, http://www.ansible.com
[8] Canzari, Matteo et al., TM services GUI prototype: compliance with the user-centered design approach for the Square Kilometer array, Proc. of the SPIE Astronomical Telescopes and Instrumentation 2018, paper no. 10707-100 (this conference)
[9] Justin L. Jonas, MeerKAT-The South African Array with Composite Dishes and Wide-Band Single Pixel Feeds, Proceedings of the IEEE (Volume: 97, Issue: 8, Aug.2009), Page(s): 1522 - 1530, DOI: 10.1109/JPROC.2009.2020713.
[10] S. Johnston et al., Science with ASKAP The Australian square-kilometre-array pathfinder, Exp Astron (2008) 22:151–273, DOI 10.1007/s10686-008-9124-7
[11] S. Chatrchyan et al., (CMS Collaboration), Search for Signatures of Extra Dimensions in the Diphoton Mass Spectrum at the Large Hadron Collider, Phys. Rev. Lett. 108, 111801 - Published 12 March 2012, DOI:https://doi.org/10.1103/PhysRevLett.108.111801
[12] ELASTIC, https://www.elastic.co/