ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA207

Tm SERVICES: AN ARCHITECTURE FOR MONITORING AND CONTROLLING THE SQUARE KILOMETRE ARRAY (SKA) TELESCOPE MANAGER (Tm)*

M. Di Carlo †, M. Canzari, M. Dolci, INAF Osservatorio Astronomico d'Abruzzo, Teramo, Italy;
R. Smareglia, INAF Osservatorio Astronomico di Trieste, Trieste, Italy;
D. Barbosa, B. Morgado, J.P. Barraca, Instituto de Telecomunicações, Campus Universitario de Santiago, 3810-193 Aveiro

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 (e.g. Dishes), exchanging commands and data with them by using the TANGO controls framework.

TM in turn needs to be monitored and controlled, in order its continuous and proper operation is ensured. This higher responsibility together with others like collecting and displaying logging data to operators, performing lifecycle management of TM applications, directly deal - when possible - with management of TM faults (which also includes a direct handling of TM status and performance data) and interfacing with the virtualization platform compose the TM Services (SER) package that is discussed and presented in the present paper.

PRINCIPLE OF WORK

The principle that drove the development of the present architecture is to study the best practises and known architectures that could solve the problems highlighted by the TM and SER requirements and reuse those concepts whenever possible. This means that only if there are no proven solutions then a new concept or pattern would be developed.

RESPONSIBILITIES

From the requirements analysis (done with the help of the entire TM in the numerous discussion held) it has been extracted the main system's functions that are described in the use cases document.

They can be summarized in the following list:

- TM generic monitoring and fault management in order to detect internal failure and gather TM performance;
- TM lifecycle management in order 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;
- Controlling of the virtualization system, according to the interface provided by the LINFRA (local infrastructure) team ("Instituto de Telecomunicações", Portugal, Lisbon).

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

CONTEXT

The TM Services take place in the middle between the domain logic and the infrastructure. In particular, the Figure 1 explains the above concept with a layered structure:

- Domain/Business Layer: functional monitoring and controlling of business logic performed by each application [1, 2];
- Services Layer: Monitors and controls processes on a generic level (not functionality) like web services, database servers, custom applications [3];
- Infrastructure Layer: Monitors and controls virtualisation, servers, OS, network, storage.

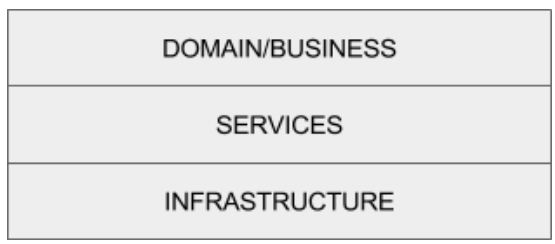

Figure 1: TM SER context.

ENTITY DECOMPOSITION

The entity managed by the TM SER package can be summarized in Figure 2. 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

* Work supported by Italian Ministry of University and Research (MIUR)
† matteo.dicarlo@inaf.it

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2017). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA207

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2017). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

- 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.

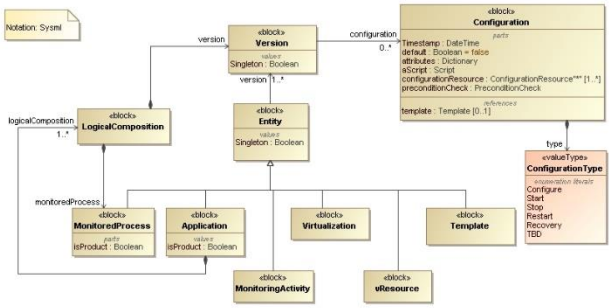

Figure 2: Entity decomposition.

QUALITIES FOR TM

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 it has been increased cohesion and reduced coupling so that it is easy to add new version of an application and 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 perform, it is possible to add a monitoring point for it that can represent a state, a measure or a simple message. Once it is available the specific monitoring point needed, it is also easy to generate an event to intercept the particular problem raised with the test.

MODULES DECOMPOSITION

Figure 3 shows the decomposition of the system into units of implementation and highlights the distinction between off the shelf software and built software. In yellow, it is shown the collaboration with the TM LINFRA (local infrastructure) team ("Instituto de Telecomunicações", Portugal, Lisbon) for the virtualization service.

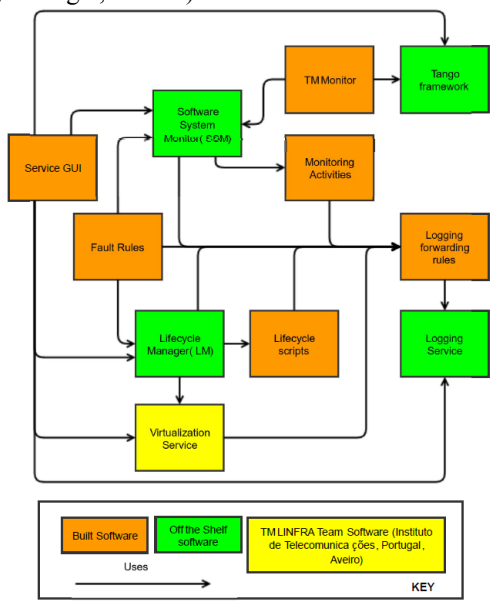

Figure 3: Module decomposition.

The lifecycle Manager (LM) realizes the *lifecycle management* that 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 LM engine (Off the shelf) execute 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 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 redesigning stage triggered by new requirements released over the SKA life-time.

All this activity can be done through an IT automation tool like puppet [4], chef [5], ansible [6] and so on in cooperation with other sub-elements.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA207

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 going to be 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 report the collected data to the SSM engine.

The SSM is also responsible of the aggregation of the TM health status and the TM State (of the various TM applications). This responsibility can be solved with 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, it has been developed the TM monitor that is a Tango Device server (www.tango-controls.org) for reporting all the monitoring information into the control system (see Figure 4).

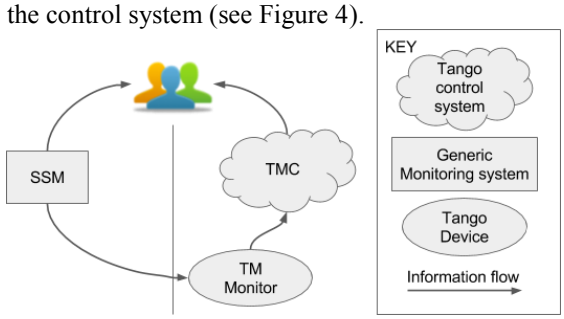

Figure 4: Reporting to Operator.

The fault rules represent 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 in the system there is a fault;
- Isolation, that is the ability to isolate a fault understanding where it is;
- 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.

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, the distinction made highlights two different modules, one called Logging forwarding rule and one called Logging Service (LS). The detail of the logging architecture are explained in the next section of the present paper.

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. In order to develop it the focus would be to make the user work with a limited number of steps to reach his various functionalities and on a high integration that is the user may need or want to compose his specific GUI.

Because of the high integration guideline, it is crucial to take part of the construction of the tango web application (Feature Request # 6 – TANGO web application) that is an effort in refactoring of the generic TANGO web app to be an open platform (third parties will be able to implement plugins for the platform to resolve their needs) [7]. Therefore, the development of the Service GUI will correspond to the construction of some plugins into the TANGO web platform.

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.

RUNTIME VIEW

Figure 5 highlights the runtime components of the system and their relations.

In green, it is highlighted 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 give the possibility to forward the message to the central database cluster according to certain rules (logging forwarding rules).

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2017). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA207

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2017). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

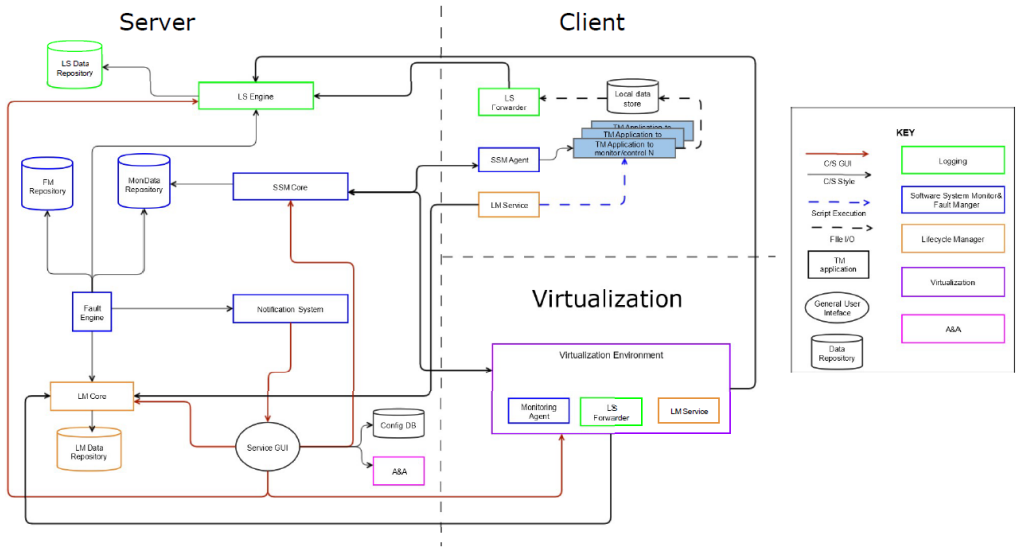

Figure 5: Runtime view.

The LS Engine is responsible for collecting and storing the log message (transformed by the forwarder) into the data center together with the ability to receive query from the external world and answer in a timely manner.

According to the principle of work exposed above, there are many best practises and possibilities for this service. In particular, it is possible to use simple files (MeerKAT, see for example [8]) or a relational DB (ASKAP, see for example [9]) or a NoSql DB (for instance LHC, see for example [10] with ELK [11]). A performance evaluation of the three possibilities has been made, and focusing on fast write and centralized solution, the best solution was the use of ELK.

In blue, it is highlighted 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 of 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 of the storage of the fault rules, and the MonData Repository, for the archiving all the monitoring data.

In orange, there is the LM runtime components that are 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 action requested is performed. It is also possible to have an agentless lifecycle manager like for instance Ansible (see [3]). The LM repository contains all the versioned lifecycle scripts developed and it only interact with the LM Core.

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 maintains information like geographical information, entities information, version information, configuration information and so on) and the AAA (Authentication And Authorization) package.

VIRTUALIZATION SERVICE

The Virtualization block manages resources (vResources) assigned to a specific configuration of an entity (see Figure 2 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 resources (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 6 shows the data model for the use of the virtualization system made for TM.

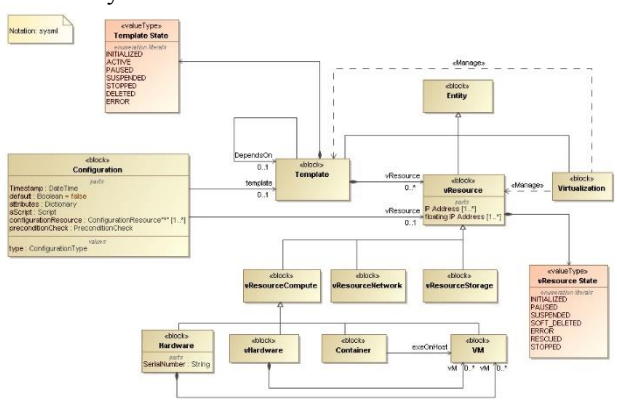

Figure 6: Virtualization data model.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA207

CONCLUSION

This paper has discussed the architecture of TM SER system. It is important to highlight that the review phase is ongoing and some evolution and refinement is expected.

ACKNOWLEDGEMENT

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

REFERENCES

[1] Swaminathan Natarajan, Domingos Barbosa, Joao Paulo Barraca, Alan Bridger, Subhrojyoti Roy Choudhury, Matteo Di Carlo, Mauro Dolci, Yashwant Gupta, Juan Guzman, Lize Van den Heever, Gerhard LeRoux, Mark Nicol, Mangesh Patil, Riccardo Smareglia, Paul Swart, Roger Thompson, Sonja Vrcic, Stewart Williams, SKA Telescope Manager (TM): Status and Architecture Overview, Proc. of SPIE Vol. 9913, 991302, doi: 10.1117/12.2232492

[2] M. Dolci, M. Di Carlo, R. Smareglia 2016: Challenge and strategies for the maintenance of the SKA telescope manager, Proc. of SPIE Vol. 9913, 99132J, doi: 10.1117/12.2231642

[3] M. Di Carlo, M. Dolci, R. Smareglia, M. Canzari, S. Riggi 2016: Monitoring and controlling the SKA telescope manager: A peculiar LMC system in the framework of the SKA LMCs, Proc. of SPIE Vol. 9913, 99133S, doi: 10.1117/12.2231614

[4] PUPPET, http://puppet.com

[5] CHEFF, http://www.chef.io

[6] ANSIBLE, http://www.ansible.com

[7] 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.

[8] 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

[9] 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

[10] ELASTIC, https://www.elastic.co/

[11] TANGO, http://www.tango-controls.org/community/roadmap/

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2017). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.