







LMC HARMONISATION THROUGH TELESCOPES

Document number 000-000000-003
 Document type PLN
 Revision 01
 Author A.Cremonini, C.Knapic
 Date 2016-02-26
 Document Classification UNRESTRICTED
 Status Released

Name	Designation	Affiliation	Signature
Authored by:			
A. Cremonini Et al.	System Engineer	SKAO	
			Date: Mar 1, 2016
Owned by:			
A. Cremonini	System Engineer	SKAO	
			Date: Mar 1, 2016
Approved by:			
T.J Stevenson	Chief Systems Engineer	SKAO	
			Date: Mar 1, 2016
Released by:			
A. McPherson	Head of Project	SKAO	
			Date: Mar 1, 2016

DOCUMENT HISTORY

Revision	Date Of Issue	Engineering Change Number	Comments
A	2015-11-18	-	Approved and Released Template
01	2016-02-26	-	First release

DOCUMENT SOFTWARE

	Package	Version	Filename
Word processor	MS Word	Word 2007	000-000000-003_01_LMCharmonisationThruTel
Block diagrams			
Other			

ORGANISATION DETAILS

Name	SKA Organisation
Registered Address	Jodrell Bank Observatory Lower Withington Macclesfield Cheshire SK11 9DL United Kingdom Registered in England & Wales Company Number: 07881918
Fax.	+44 (0)161 306 9600
Website	www.skatelescope.org

TABLE OF CONTENTS

1	INTRODUCTION.....	6
1.1	Purpose of the document	6
1.2	Scope of the document.....	6
1.3	Credits	6
2	REFERENCES	7
2.1	Applicable documents.....	7
2.2	Reference documents	7
3	EXECUTIVE SUMMARY	8
4	TANGO TRAINING AND CONTROL SYSTEM BEST PRACTICES.....	9
5	USE CASES DISCUSSION OUTCOME.....	9
5.1	TANGO Paradigm	9
5.2	TANGO States.....	10
5.3	Single Point of access between TM and Elements.....	10
5.4	TANGO Scope	10
5.5	Naming Convention	10
5.6	Architecture Leadership.....	10
5.7	Workflows	10
5.8	Logging and historical information:	11
5.9	Security	11
6	LMC HARMONISATION PROCESS DESCRIPTION.....	12
7	AGREEMENTS, ISSUES AND ACTIONS.....	15
8	APPENDIX A – USE CASES AND MODERATOR COMMENTS	16
9	APPENDIX B - AGREED IMPLEMENTATION SOLUTIONS.....	42

LIST OF FIGURES

Figure 1: Peer review process description (H. Schnetler) 14

LIST OF TABLES

Table 1: Proposed Peer review Meeting dates and Venues 13
Table 2: Workshop Template for List of Agreement, Issues and Actions 15

LIST OF ABBREVIATIONS

LIG.....	LMC Interface Guidelines
LMC.....	Local Monitoring and Control
SKA	Square Kilometre Array
SKAO	SKA Project Office Organisation
TANGO.....	TAcO Next Generation Object, a control system based on TACO
TM	Telescope Manager

1 Introduction

The responsibility to design the SKA Telescope has been shared between Consortia. Each Consortium has in charge to deliver the design of a specific element. Each element is responsible to provide monitoring and control capabilities from low to high-level functionalities, except for Telescope Manager which implements the higher level telescope functionalities and coordinate the activities of the telescopes. The lack of coordination in the process of developing a consistent architecture has produced a non-homogeneous implementation. In order to recover this situation and to push implementation of monitoring and control system, a harmonisation process has been setup. During a meeting held in Trieste in March 2015, it was decided that TANGO was the best-suited control system framework for SKA purposes. About one year later, again in Trieste, a three day workshop aimed to address three main areas: Provide Advance TANGO Best Practices Training Session, discuss SKA Use Cases proposed by SKA LMC teams with recognised TANGO experts and draft a Strategy for proficient use of TANGO for SKA. The outcome of this workshop and the future LMC peer-review sessions, which are part of the harmonisation process described in the document with more details, will be collected in this document, with the aim to bring out possible inconsistencies in the TANGO implementation for SKA, by exploiting previous experiences in different application fields. Hence, the outcome of the workshops would provide guidance for the community and collect implementation solutions. Moreover, it shall provide hints of eventual aspects not covered by the current SKA LMC vision. Under the perspective to participate in TANGO community, SKA could contribute to expand the native TANGO features.

1.1 Purpose of the document

This document describes the effort taken by SKAO and Consortia address the activities around the high-level architectural design and fill the gaps of information and coordination between elements to agree on a common Control and Monitoring Framework. It describes also the process used to start-up a collaborative community with the aim to exploit communalities in implementation. It highlights the present identified gaps in the SKA monitor and control. This document shall be regularly updated after each peer review meeting or when relevant number of issues will be addressed.

1.2 Scope of the document

The document contents apply to Both MID and LOW Monitoring and Control system at any level of the hierarchy.

1.3 Credits

Authors want to recognise the extremely and fundamental contribution of the SKA Local Monitoring and Control System community as well as the SKAO Engineering Team, which provides input for this document as well as the contribution of Andy Gotz (ESRF) and Lorenzo Pivetta (ELETTRA). They provided, with their expertise, an invaluable insight and guidance in TANGO based control systems. Authors also want to recognise the support of Riccardo Smareglia and his group in Trieste, which kicked-off this activity and continuously support it.

2 References

2.1 Applicable documents

The following documents are applicable to the extent stated herein. In the event of conflict between the contents of the applicable documents and this document, **the applicable documents** shall take precedence.

[AD1] SKA-TEL-SKO-0000002_02_SKA_Baseline_Design

2.2 Reference documents

The following documents are referenced in this document. In the event of conflict between the contents of the referenced documents and this document, **this document** shall take precedence.

- [RD1] LMC Tiger team, "LMC Middleware Evaluations SKA_CSP_MEMO_0010_middleware_contest
- [RD2] LFAA LMC Middleware Infrastructure (DeMarco)
- [RD3] LMC Standardisation Workshop Report, Trieste 2015
- [RD4] LIG LMC Interface Guidelines Document
- [RD5] SKA1 LMC Scope and responsibilities, RevA, SKA-TEL.TM-TMC-MEMO-001-A

3 Executive Summary

After a brief *ex-cursus* describing the facts and considerations that have driven the decision to use TANGO as the common Control System Framework for SKA, this document collects the outcome of harmonisation process. The First step of this process assembled LMC Developers and LMC Team Leader in Trieste (16-18 Feb 2016). The aim of this meeting can be so summarised:

- Provide Advanced TANGO Best Practices Training Session
- Discuss SKA Use Cases with experts in Distributed Control systems, TANGO in particular
- Draft a Strategy for proficient and consistent use of TANGO across the SKA
- Give a general overview of the benefit about using a common infrastructure for SKA Local Monitoring and Control system

The main outcomes of the workshop can be so described:

- The Advanced TANGO Training Session has been widely appreciated, and the need for further training activities recognized. The TANGO ecosystem provides many tools that in several cases already cover SKA needs, such as for example the SEQUENCER component. Keeping in touch with TANGO experts is foreseen as essential.
- 20 SKA related Local Monitoring and Control use cases have been proposed by workshop attendees which represent the SKA LMC architects and developers community. This means that we had a good coverage of the control needs for the system. Only the INFRA consortia were not represented.
- Use cases have been reviewed by widely recognized experts in Distributed Control systems, in particular TANGO, which provided comments, solutions and hints regarding how correctly implement them following the TANGO paradigm, and TANGO best-practices.
- The final discussion, very well attended, and with lively participation, provided a list of proposals in order to progress the SKA Local Monitor and Control system.

In more detail, the participants agreed to the following points, related to the architecture and other aspects of the control system:

- Given TANGO as the chosen Control System Framework, the TANGO paradigms **MUST NOT** be violated; that is, control has to be implemented following TANGO patterns, and avoiding TANGO anti-patterns.
- Best Practices of Control System design suggest new states **MUST NOT** be introduced, but if truly necessary, they should be implemented in TANGO Kernel. However, states can have additional metadata that can qualify the state.
- It has been discussed if it is reasonable having a single point of access of all nodes exposed to TM from each element. That is, given the TANGO architecture, the high-level supervisor would access top-level TANGO device servers, which will provide rolled-up attributes, and commands for high-level coordination, but access to lower level devices can still be had for debugging and drill-down purposes, but not for coordination.
- Naming convention and definitions —including the high-level control hierarchy— are necessary and they have to be under configuration control.
- TANGO has been designed to be a control system, not to perform other types of software operations like for example HPC.
- While TANGO is identified as a dominant control system, it does not exclude the possibility to manage non-TANGO subsystems through interfaces to TANGO
- TANGO does not natively provide security features, as for example encryption. The best practices on distributed control system suggest network has to be designed to take into

account this characteristic¹. Moreover, usually in demanding processes, encryption reduces performances and has to be taken into account. A comprehensive security policy would be discussed.

- It has been suggested the creation of a common repository for the code developed for the SKA. GIT or GITHUB were suggested. The repository must not be public, as even when it will be hosting prototype information, the control hierarchy can be deduced from it, and could help potential attacks. SKA configuration information or hostnames rather than IP Addresses, have not to be hardcoded or available as properties.
- The need for a Tiger Team and a Control System Architect to implement the high-level architecture has been recognised. Roles and responsibilities of those figures have been discussed and proposed
- The suggested peer review process has been generally accepted, and recommendations for improvement and making it more formal where given. Dates, Venue, and element under review have been coarsely agreed.
- Some actions have been agreed in order to clarify the Control Architect and architecture situation, establishing the Tiger Team and related high priority topics that this team has to tackle, and to setup the logistics for the two following venues (Madrid and Edinburgh)

4 Tango Training and control system best practices

Advanced TANGO training has been provided in a day-long session that covered many different aspects, from how to use kernel functionalities natively embedded in TANGO to the use of tools designed for non TANGO-native, but connect, specific functionalities.

5 Use Cases discussion outcome

The workshop attendees submitted about 20 SKA LMC use cases, which are collected in Section 8. These cases have been reviewed by widely recognized TANGO experts², which provided direct feedback during the meeting, and also provided the written feedback which can also be found Section 8. Solutions have been provided regarding how to correctly implement those cases adhering to the TANGO paradigm. Further considerations are elaborated in the following sections.

5.1 TANGO Paradigm

From the review of the Use Cases, a new interpretation of the LMC Interface Guidelines (LIG) started to emerge, that did not force the TANGO paradigm. The benefits of a common bus and the limits imposed by the unique interface between elements and TM have misunderstood the right approach to the tango paradigm of using the native types and do not encapsulate structured information in a one single self-describing object. This introduces a risk for maintenance and scalability. By following closely the TANGO-way, even if at the beginning it can be more difficult, but would lead to more uniform code, easier to be maintained, updated, and improved, and with more compatibility with the developments made by the TANGO community.

¹ This point was discussed late in the meeting, and was not part of the original set of discussed elements. Examples of design can include enabling tunnelling using SSH-keys, or allowing connections only between particular device servers, instead of having all devices with access to the network.

² Lorenzo Pivetta (ELETTRA) and Andy Gotz (ESRF).

5.2 TANGO States

The need to clarify the correct approach to provide new states in the TANGO State Machine emerged. Best Practices for control systems suggest to REFRAIN from introducing new states, unless strictly necessary. The current state machine states are a distillation of a large number of complex states. The states indicated in the LIG must be analysed and aligned to TANGO states, and use the additional state metadata to implement the LIG semantics.

5.3 Single Point of access between TM and Elements

It has been discussed if it is reasonable having a single point of access of all nodes exposed to TM from each element, as stated in the LIG. The result of the discussion was that it was more appropriate to have every element provide TM with the rolled-up status and high-level control of the element (as appropriate), and shall allow TM to access status of individual devices (equipment, software) at all levels of hierarchy (drill-down) when required (for diagnostics, troubleshooting, etc.). In that way, the drill-down capabilities do not need to be coded, but would make use of native TANGO tools and clients.

5.4 TANGO Scope

TANGO has been designed to be a control system, not to perform, for example, computations. For those cases, a dedicated system should be designed to expose to TANGO what is necessary for control purposes. While TANGO is identified as the dominant control system, it does not exclude the possibility to manage subsystems just through interfaces to TANGO. However, those instances need to be properly reviewed and authorised by the SKA Architect, as they need to be minimised.

5.5 Naming Convention

In order to harmonise the control practices, it is sensible to become a community of practice. As a community, it is necessary to have a common vocabulary, agreed definitions, and naming convention that go across telescopes, and that can be put under configuration control. In particular, the naming and control hierarchy needs to be established. Such common agreements improve dramatically the sustainability of the system in the long term, and make debugging easier.

5.6 Architecture Leadership

There was a strong perception from the growing community that a High Level Control System Architect and common architecture is missing. From the analysis of the use cases, it was apparent to the experts and to the attendants (particularly to SE) that, from the architectural point of view, there wasn't a common guidance for the development of the control subsystems.

5.7 Workflows

In order to manage sequences of commands and/or initialisations, enforce order of execution, etcetera, the different groups made use of non-TANGO tools. It has been noted that in the TANGO distribution a SEQUENCER Device already exists. Before introducing tools not part of the TANGO ecosystem, it must be shown that alternatives do not exist.

5.8 Logging and Historical Information:

The meant of “logging” it is sometime confused with “history of informations”. In TANGO these have a well defined meaning and utilities to manage both in adequate manner. In some use cases this different meaning is not so clear or explicit.

5.9 Security

Since TANGO have been developed to run in local self-contained systems, strong security features, like encryption have not been implemented. However, usually in demanding processes, encryption reduces performances and has to be taken into account. It has to be evaluated if is more effective develop an encryption feature or identify which parts, in the distributed control system, could be more exposed to attacks and look after them properly. A more extended discussion regarding security Policies, rather than a security features, is necessary.

6 LMC Harmonisation Process Description

The open discussion has identified three main aspects:

- Perception that both high-level control system architect and architecture is missed.
- Implementation issues.
- Need to keep momentum going on to proceed in the harmonisation

To resolve the first aspect, it have been suggested that, as action, SKAO should nominate or ask support for a Control System architect. The architect function of SKA telescope even in the field of Monitoring and Control will remain under the domain of SKAO. This architect shall have the responsibility to draw and guide the high-level control architecture. It shall be recognised as an outstanding experienced figure in TANGO control community.

The point of contact between the MC architect and consortia will be the ANT Team. Name ANT has been suggested not as an acronym but related to the insect. It lives in extremely well organised colonies. Strong and devoted to their roles, they work together to achieve a common goal. This restricted team (about 6 people) represents the elements. ANT Team shall have a leader that coordinates the activities. The team responsibilities are to provide to the architect all the information he requests in order to design the architecture and to address specific issues identified by the architect or as outcome of the peer review meetings. It also shall guarantee the presence of consortia TANGO developer at the peer review meeting. During the Trieste meeting, first sets of issues have been identified. These have to be addressed by the ANT Team and/or by the architect.

- Take responsibility to update LIG accordingly with the outcome of the workshop
- Naming convention and Definition
- Hierarchy levels
- Best practices
- State and modes
- "Standard" tools
- Survey of tools and practices
- Prioritization of activities
- Tango pattern for SKA
- Archiving logging alarm policy and GUI
- Security: Agreements, implementations, policies.

The community agreed that team have to be built soon. Team members and team lead have to be identified and signed up by **31st March 2016**.

In order to respond to the need of keeping the momentum and improve the ability to implement Command and Monitor system under the TANGO framework, the proposed peer review scheme has been discussed. Has been clarified that a peer review is not an Earn Value Milestone. The purpose of the process is to:

- Share the implementation solutions
- Encourage reuse of implementation solution
- Same Problem == Same Solution
- Increase progress curve of less mature devices

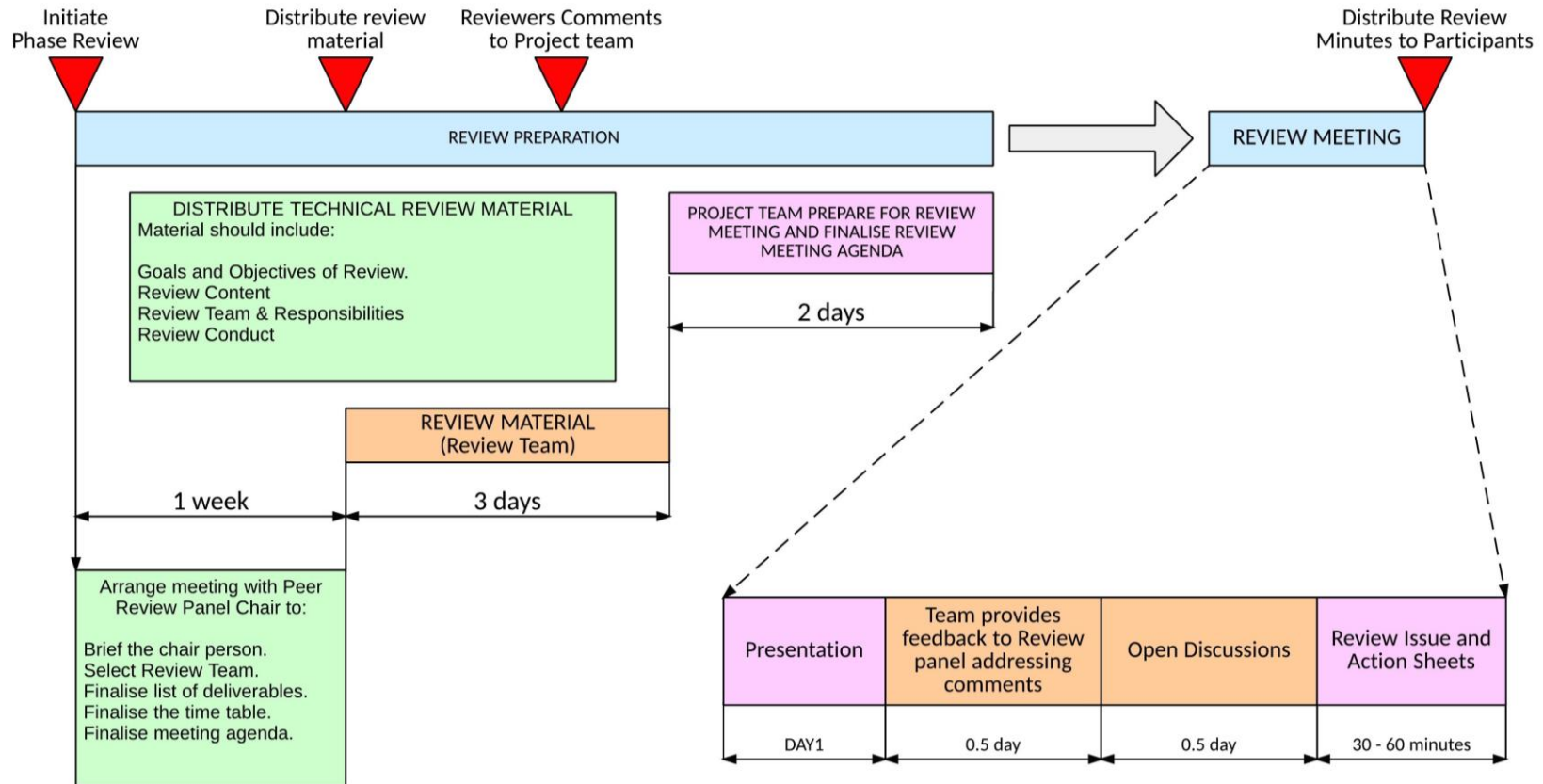
- Absorb used implementation to fill gaps
- Compare different implementations and agree an optimal solution
- Use the power of the community in order to respect TANGO Paradigm
- Identify SKA essential needs for future contribute in developing

The proposed Peer Review Process is described in Figure 1 and it is provisional (Thanks H. Schnetler). Since our review is a collective effort could only partially follow this scheme. The way it will be conducted and the question that collectively have to addressed will be finalised later on. The elements under review should provide document two weeks before the meeting, however design team can decide to present the material at the review meeting. Consortia Element SE SHALL nominates the most relevant participants under the consortia Domain. They shall review the material and shall attend the peer review meetings. Dates and venue of the peer reviews are listed in Table 1. Elements under review are still provisional. During the same meeting, ANT Team shall address the assigned actions or agree how to address them.

Date	Venue	Element under reviewed
11-13/4/16	Madrid (After CSP TIM#5)	MID.CSP.LMC MID.DSH.LMC
4-6/7/16	Edinburgh (After SPIE 2016)	LFAA LMC SADT.NMGR
2-6/10/16	Stellenbosch (During SKA Engineering Meeting)	LOW.TM.LMC MID.TM.LMC SAT.LMC Harmonization status
jan/feb 2017	TBD	SDP.LMC AIV.LMC INFRA.LMC

Table 1: Proposed Peer review Meeting dates and Venues

Figure 1: Conceptual Peer review process description (H. Schnetler)



7 Agreements, issues and actions

At the end of each workshop a list of agreement, issue, actions will be listed in order to monitor the progress.

Table 2: Workshop Template for List of Agreement, Issues and Actions

Trieste Harmonization Workshop		
Agreements		
Has been agreed that a collective effort will improve the quality of design and development.		
High Level architect and architecture is missed		
A unique transport BUS could avoid the creation of unwanted constraints and unnecessary interfaces		
TANGO parading have not to be violated		
Naming Convention and definition need to be addressed		
General recognition that the presence of TANGO experts during the workshop was essential in order to clarifies several crucial points		
Attendant agree of the coordination role of SKAO in the harmonization process		
Issues		
The purpose of future meetings will not be actual peer review. The review process will be a common effort to optimize development and design: difficulties to conduct the process		
After 1 year of Choose of TANGO there are still a lot of gaps in implementation		
Propose review schedule is aggressive: it has to be managed.		
Actions	Assignee	Due Date
Establish ANT Team	SKAO	31/3/2016
SKAO Discuss Internally MC architect issue	SKAO	11/4/2016
Setup Indico page for Madrid and Edinburgh Peer review meetings	SKAO	1/3/2016
Naming convention and definition	ANT+architect	
LIG review	ANT+architect	
Code Repository: were hosted and policy	SKAO	

8 Appendix A – Use Cases and Moderator comments

Use Case Title	Best way to let TM access 'lower-level' parameters.
Proposer	Dr. Carlo BAFFA, Dr. GIANI, Elisabetta
Description	
<p>The CSP–LMC, as the other LMCs, will have a layered structure. As a consequence, each level of our layered structure maintains a 'summary' status of the physical conditions of lower level devices. But we need to let higher levels, engineering interface, and TM to get detailed parameters values of those lower devices.</p> <p>We devised a flexible approach to this operation. We propose a new command: getLowerStatus. When a device server receives such a command it sends the same command to lower level device server with its argument, recursionLevel, decreased by one. If recursionLevel is already at zero, the device server doesn't send such a request. After getting the response from lower level devices (or at once, if recursionLevel is at zero), each device server builds a json1 object composed by all its parameters and by the lower level device answers, and sends the resulting blob to the device immediately above, via a Tango Pipe. To limit the amount of data exchanged, we can devise some provision for 'data pruning' or direct device addressing, as, for instance, defining the start point of command execution (e.g. the class which start the real execution of the command). Comments and suggestions?</p>	
Moderators Comments	
<p>This means throwing away the hierarchical approach and the object oriented too. Introduces additional issues to handle recursion, data size. The TANGO database of each LMC, which contains all the device information, can be used as the single point of access to the LMC lower level devices whenever a detailed information is needed.</p> <p>Use the higher level devices to provide a summary of the lower level devices. Access the lower level devices when you need the full information. Why use JSON on top of the TANGO Pipes? Pipes transport a mixture of data types in binary. The same as JSON but more efficient.</p>	

Use Case Title	Best way to set/report multiple parameters?
Proposer	Dr. GIANI, Elisabetta, Dr. Carlo BAFFA
Description	
<p>There are important times when it is necessary to set or to read multiple parameters at the same time. Notably during initialization, when each device driver needs to set up the zero-level safe parameters values, or when TM configures the initial state of the system, or again at the start of a new set of observations, when it is required a large change of state of the instrument. We propose two commands getFullStatus and setFullStatus. The first command instructs the device server to build a json object containing all its parameters and send the resulting blob to the device driver issuing the command via a Tango Pipe. The setFullStatus accepts, as argument, a json object containing many parameters and then executes a “Set Parameter” for each of them. Optionally this command can be followed by the execution of a getFullStatus for confirmation. For more complex situation, we can also devise a recursive setFullStatus, inserting the name of device class to witch a group of parameters applies.</p> <p>Comments on this proposal?</p>	
Moderators Comments	
<p>As above why use JSON on Pipes? This use case sounds like you need to manage a lot of settings. Why not write a TANGO Device Class for doing this? It can be organised as a hierarchy of classes to control</p>	

a large number of devices. Settings could be stored in files or the database.

Example 12.1 – ESRF is developing a Settings device class for managing all settings in the accelerator. The settings are stored in files. The Setting device manages a hierarchy of sub-devices for sending Settings to lower levels. This scales to thousands of devices.

Example 12.1 – ELETTRA developed a dedicated application to restore multiple settings on the accelerator. It's based on the context concept, which is basically a set of devices, which you can save and retrieve values in a single pass. Moreover you can possibly restore the settings for the context back in time, at any desired timestamp, retrieving data from the HDB++ historical archive. More information here:

<http://icalepcs.synchrotron.org.au/papers/wepgf152.pdf>

Use Case Title	How to effectively use TANGO in a hierarchical system.
Proposer	Ms. Sonja VRCIC
Description	
Central Signal Processor consists of three major sub-elements. The 4th sub-element, CSP.LMC (Local Monitor and Control) has been introduced to implement a single point of communication with TM and represent CSP as a single Element. Each of the three sub-elements (namely: Correlator and Beamformer - CBF, Pulsar Search Engine - PSS, and Pulsar Timing Engine - PST) consists of a number of LRUs/components and implements internal monitor and control. Each of the three sub-elements (CBF, PSS and PST) implements a 'sub-element master' which communicates with LMC. At least two sub-elements (CBF and PSS) intend to use TANGO for communication between the Sub-element Master and other LRUs/components. The challenge: How to define TANGO devices / servers so that the operations, via TM, can monitor and control CSP as a single entity, but when needed also at sub-element level, and down to LRU and component. In other words, operations, via TM, when needed, should have access to every parameter of every component.	
Moderators Comments	
Build a TANGO device server hierarchy, starting from low level devices. CSP.LMC TANGO device server will then provide a short/effective summary of the underlying layers, e.g. TANGO device server masters for CBF, PSS, PST which, in turn, provide a summary of the lower layers. When any additional/specific parameter is needed the TM will get them connecting to the lower layer TANGO device servers. The TANGO database for CSP.LMC can be seen as the single point of access that allows a straightforward connection to the specific TANGO device servers.	
TANGO is a hierarchical control system. This means the hierarchies are used to present a summary of the lower layers. This is essential in a big system to control large systems. The lower layers are directly accessible by any client (unless access control has been configured to prevent this).	
TANGO anti-pattern – duplicate the information of the lower layers in the upper layers.	
Example 2.1 – ESRF beam position measurement diagnostic systems consists of 250 lower level sophisticated BPMs which are grouped together and controlled by a single high level device server BpmLiberaAll. Refer to this paper on the ESRF BPM architecture: http://accelconf.web.cern.ch/AccelConf/icalepcs2011/papers/mopks014.pdf	
Figure 4 : Example of architecture of a highlevel device controlling multiple devices i.e. ESRF BPM system	

Example 2.2 – FERMI multiscreen controllers. Few dozens of fluorescent multi-screen controllers are in use at FERMI. Each controller consists of actuators for target screen positioning, filter positioning and a CCD for beam acquisition. Each controller is modeled as a TANGO device, with an additional TANGO device in charge of the CCD. A master TANGO device allows for operating targets definition and simple and effective monitoring of common information.

Use Case Title	Initialization of multiple devices: use of Yat?
Proposer	Dr. Carlo BAFFA
Description	
<p>During the initialization phase it is inconvenient to proceed over many devices in sequence. We have seen Tango community devices sometime parallelizes tasks by means of the multi-threading libraries Yat and Yat4tango. How can a yat thread signal to the main (calling) thread that something has changed, for example the status of a monitored device? Is there a kind of “event handling”? Or the main thread has to poll some attributes whose values are monitored by the yat threads?</p>	
Moderators Comments	
<p>YAT implements a messaging system in the DeviceTask you can use to send messages from task to main thread. See WritingSubsystemManagerDeviceServerWithYAT.pdf for a short introduction and refer to the following links for YAT documentation:</p> <p>http://www2.synchrotron-soleil.fr/controle/docs/yat/yat_html/index.html http://www2.synchrotron-soleil.fr/controle/docs/yat4tango/yat4tango_html</p> <p>For simple requirements also an omnithread based approach could be considered.</p> <p>If the initialisation phase of a Device takes a long time (>500 ms) it should be delegated to a thread. A standard thread library or Yat4Tango can be used. Same for monitoring. Might be an issue in Python. PyTango implements gevent compliant asynchronism which can be used to delegate slow actions. In the future gevent will be replaced by asynch (new feature in Python3).</p>	

Use Case Title	Real-time updates for on-going observations.
Proposer	Ms. Sonja VRCIC
Description	
<p>Telescope Manager (TM) shall provide to Central Signal Processor (CSP) regular updates for delay tracking and beam-forming parameters (for all on-going observations/scans). Delay Models are specified in a form of coefficients of a polynomial, number of coefficients per model and cadence for the updates is still TBD; some estimates are provided in the CSP to TM ICDs (for mid it may be 3 coefficients per antenna per polarization every 10 seconds). Weights used in beam-forming and Jones Matrices shall be provided by SDP, via TM (cadence TBD). Topic for discussion: Should these parameters be delivered to CSP using the same method and approach as other configuration parameters? In other words, should they be defined as parameters of the TANGO device?</p>	
Moderators Comments	
<p>Yes, Use TANGO attributes to implement values, which are need by (many) multiple clients. Use TANGO events to update the clients efficiently. Either as ON_CHANGE or by pushing USER events.</p>	

Use Case Title	LFAA LMC Overview Use-Case
Proposer	Dr. Andrea DEMARCO
Description (and Comments)	
<p>Introduction</p> <p>This section gives a brief list some of the features for an LMC system, and what kind of control the system should provide to the TM interface layer for generic control of all elements. Primarily, LMC will be concerned with some or all of the following, to different extents:</p> <ul style="list-style-type: none"> Monitoring and control of digital devices and any hardware/software components The detection of any specified alarms/events, to inform appropriate recipients The running of computational/logical jobs/tasks with specified schedules Routine checks and diagnostics Exposing functionality in a logical fashion, or in detailed form for debugging and fault mitigation Interaction with any operators/users Routing, processing, saving of data streams etc. Management of the available computational resources <p>For the purposes of the LFAA LMC, the above requirements could be serviced by various software blocks as follows:</p> <ul style="list-style-type: none"> Monitoring/control of devices/hardware --> TANGO Server/Drivers + Hardware interfaces/Access Layer Alarms/Events --> TANGO Drivers + more generic core Resource management --> Apache Mesos Scheduled jobs --> Apache Aurora (on top of Mesos) and native services Routine checks --> Applications running via Aurora Exposing functionality --> LFAA Interface API Interaction with users/operators --> TANGO wrapper over LFAA Interface API Routing/processing/saving data --> GlusterFS logical volumes + low-level DAQ services + real-time processing services <p>TANGO Device Control - TANGO States</p> <p>All LFAA LMC requirements must fall within the Telescope Operating States, which are assigned manually by the telescope operator, while the operating status can be set by the LMC or by TM. An example of a typical state flow could be as follows:</p> <p>The system starts in the “off” state, where all hardware/software is powered down. Once powered up the state transitions to “standby”, the default operating status. A number of transitions are possible (a) Debug/Maintenance: occurs when telescope is set to maintenance mode, (b) Low-power: occurs when a command from TM sets the telescope to low-power mode, (c) Safe-state: an optional state (similar to low-power), (d) Init: occurs when an observation schedule is received by TM and the telescope needs to be initialized (forming stations, programming boards, setting up workflows etc.), (e) Faulty: occurs in cases where a serious or critical error is detected. “Debug”, “low-power” and “safe” states will remain in those states until TM instructs a change. “Faulty” state can go to “standby” if the fault is mitigated/fixed, or to the “off” state. Errors can occur in all operating states, so every state can transition to the “faulty” state.</p> <p>All of this seems to be very specific to LFAA... how to generalize? Could a set be defined for all LMCs?</p> <p>It is apparent during our prototyping that Tango (v8/v9) does not allow for custom states to be</p>	

added to the set of states in the framework. There are some workarounds e.g. modifying the source code to allow for more states, but this breaks Tango-related packages like Pogo (the device designer). Tango 9 has introduced ENUM attributes, but one can't assign the Tango state machine system to attributes out of the box.

In our current prototyping, we provide a design pattern to write a new state system within a Tango device driver that can replicate the same functionality for Tango states, but with any list of custom states.

When would the State transition check happen? Is some extra code in each method foreseen?

The Python TANGO driver will include:

A dictionary called `state_list`, which contains a list of key/value pairs, the key is the driver function name, and the value is a list of states (represented by a numeric value) of allowed states under which that function is runnable.

An inbuilt function called `check_state_flow` which, given a function name, checks if the current state of the device is in one of the allowed states. This function returns True/False accordingly. Furthermore, with this re-implementation that mirrors the Tango state-flow control, the predefined Tango state/status information is redundant, though it could still be used for some specific cases if needed. We anticipate replacing the "numeric" value in `state_list`, with an ENUM value.

TANGO states should be always implemented. If you need sub-states then your proposal is OK. See comment for use case 3 above.

Till the problem on how to guarantee a consistent management of all the sub-states apply.

TANGO Device Control - TANGO Device Hierarchy

In LFAA, a number of antennas are connected to a single tile (via a Tile Processing Module, or TPM). Based on the Telescope Model information, tiles are configured into logical stations, and some of the operations will be transmitted to stations rather than to individual TPMs.

We can define a TANGO station with the following non-exhaustive capabilities:

Add/Remove TPM to/from Station

Connect/Disconnect TPM in a Station

Set/Get station state (which is an aggregate of TPM states)

Run station command - a command across all TPMs in the Station, and return all results for each TPM

This hierarchy is therefore very straightforward. We shall simply go over the implementation model to send commands to all TPMs within a station, waiting and gathering all results. A number of criteria must be catered for:

The command to be executed must exist on all connected devices.

Each TPM must be in a state where the command in question can be run.

Arguments for the command have to be passed in to the various TPMs.

It is trivial to check if the command exists on all TPMs. In order to pass in parameters to the various TPMs, a pickled list of dictionaries can be utilized. If the list contains only one dictionary, then the same parameters are applied to all TPMs. If there is more than one item in the list, then the number of items in the list must be equal to the number of TPMs in the station.

Why should a command not exist in a TPM? Are TPM different? A common interface would be mandatory...

When passing a command to all TPMs, the Station device maintains an index, called "command_indexes". Each command per TPM is run asynchronously in TANGO via the "command_inout_asynch" TANGO call. This returns a command ID which can be polled for completion, and we store this in "command_indexes".

A while-loop is then able to use the command IDs stored to poll for a result from each individual TPM by utilising the "command_inout_reply" TANGO call, which by default returns an exception when polling a command that has not yet returned. Once all results are accounted for, stored in an array, the Station call can return these replies.

A good pattern to use for this are the grouped commands. Create a group of the stations to be controlled and then send/receive replies from them using a single group call. The group call use the asynchronous call to trigger all devices in the group. This way the reading of all devices is only limited by the slowest device to reply and not the sum of all the replies. Check out the documentation on Groups: in the TANGO manual

http://www.esrf.eu/computing/cs/tango/tango_doc/kernel_doc/ds_prog/node5.html#SECTION00570000000000000000

TANGO Parameter Limitations

Commands in TANGO can, at most, pass in/out a single parameter. The easiest way to circumvent this issue when multiple parameters are required is to use a string representation of a set of key/value pairs. In the case of pyTango, this can be easily achieved by using a pickled representation (giving a string) of a dictionary. For example, from within the TANGO driver, one can "un-pickle" the argin input argument as follows:

```
self.debug_stream("Unpacking arguments...")
arguments = pickle.loads(argin)
commandName = arguments['commandName']
inDesc = arguments['inDesc']
outDesc = arguments['outDesc']
allowedStates = arguments['states']
```

In fact TANGO commands also support arrays of homogeneous TANGO types, so multiple parameters can possibly fit into an array.

Another possibility for multi-parameters is to use the pipes in TANGO V9. PyTango has also implemented a TANGO Object which sends/receives multiple parameters to and from TANGO devices. It uses pickle to do over the DevEncoded interface. But in the future it will use Pipes. More information about PyTango Objects in this presentation:

<http://www.eli-alps.hu/sites/default/files/tangows/20150224-1200-ESRF-Building-TiagoCoutinho.pdf>

In any case, whenever very different type of parameters are needed, consider implementing multiple commands.

TANGO Parameter Limitations - Other Devices

The LFAA LMC will require the control of other devices such as Switches, PDUs. TANGO drivers for these devices will also be written in a similar fashion to those for TPMs, with a state-handling

mechanism as described.

Some TANGO device servers already exist for these devices.

Moving Upwards

The LFAA LMC use-case requires that additional components aside from a TANGO-based control system to be available. In the architecture currently being developed, some of these components are Apache Mesos, Apache Aurora and GlusterFS. We give a brief overview of each.

Moving Upwards - Apache Stack for Resource/Workflow Management

The LFAA will require a set of nodes, the “Monitoring, Control and Calibration Servers”, which is a compute cluster. On this MCCS cluster a number of applications and services will be run (including the TANGO service). Cluster management technologies can be employed to manage the resources of each node as well as submit applications and services as jobs. Traditionally, the resources of a cluster are handled separately for each node, such that the job submission system as well as the administrators need to know how the cluster is configured. An abstraction over this is to aggregate all the available resources as a single entity, such that the entire cluster is viewed as a single large node. These tools implement a level of abstraction on top of operating systems, and can be viewed as a cluster operating system. Apache Mesos is such a tool, a cluster manager on which frameworks can run. These frameworks provide the scheduling and executing logic required to launch jobs and workflows. In this respect, Apache Aurora is being considered. Additionally, large clusters require a level of redundancy and reliability, especially when master nodes fail. Apache ZooKeeper can provide this functionality.

The mix of technologies chosen here may seem daunting. In fact, each block included requires separate configuration and setting up. It is good to minimize the amount of “moving parts” in a design. The stack chosen here can adequately handle the LMC requirements for LFAA. This is not the first design that was considered for LFAA. Earlier designs and test modules included some more/different components that were eventually taken out and or replaced by better packages. The choice was based on some general and some very specific criteria. The products are all open source. They are tailor made to handle the tasks required of them in an LMC setting. They have dedicated teams actively developing the products and fixing bugs/adding features. The TANGO control system was analyzed earlier on for its utility in LMC and was found to be very suitable to device-only control and very scalable for that specific purposes. On the other hand, packages like Apache Mesos, Apache Aurora, GlusterFS etc. are already deployed in very large systems and guarantee scalability and industry standards for their specific purpose as well.

The various units of this Apache stack provide an API/command line tools that can used to run, monitor and control the stack. We anticipate that the required functionality will be wrapped as part of the entire LFAA LMC API. But we do not anticipate that this will be done via a TANGO driver. The reasons are that most of the interaction with some of these units are based on their own UI (for internal use) or through scripts. TANGO does not follow the scripting model, since it is basically a reply-request mechanism meant for simple commands to devices.

It is true that other softwares like Apache Mesos and other messaging systems are better suited for these tasks than TANGO. However careful thought must be given to how much of these systems needs to be archived, logged and monitored together with the TANGO control system. If you need to correlate information from the multiple systems then it is strongly recommended to build a bridge to the sub-system via device servers and then use the TANGO archiving system to store information from both systems.

TANGO anti-pattern – build multiple systems to control, monitor and archive information in a large control system

Example 5.1 – ESRF uses Apache Camel to ingest metadata from TANGO devices into a metadata catalogue. The Camel workflow sends status information back to the TANGO device about the ingestion. This way all status information is visible from one control system.

Moving Upwards - Logical Volumes

GlusterFS is an open source, distributed file system capable of scaling to several exabytes and handling thousands of clients. GlusterFS clusters together storage building blocks over Infiniband RDMA or TCP/IP interconnect, aggregating disk and memory resources and managing data in a single global namespace. It is based on a stackable user space design and can deliver exceptional performance for diverse workloads. Most existing cluster file systems are not mature enough for the enterprise market. They are too complex to deploy and maintain, although they are extremely scalable and cheap since they can be entirely built out of commodity OS and hardware. GlusterFS solves this problem, by offering stability and low-maintenance setup for some speed trade-offs. GlusterFS is an easy to use clustered file system that meets enterprise-level requirements:

GlusterFS can be deployed with the help of commodity hardware servers.

No metadata server is required.

Any number of servers can access a storage that can be scaled up to several exabytes.

Aggregates on top of existing file systems. A user can recover the files and folders even without GlusterFS.

GlusterFS has no single point of failure. It is completely distributed, thanks to not having a centralized meta-data server like Lustre.

It is not tightly coupled with the OS kernel (like Lustre) and therefore any updates to the system as a whole have no effect on GlusterFS.

As a software component of the LFAA LMC system, GlusterFS will manage one logical cluster wide partition to store raw data files and logs. Each node will have a dedicated partition which will make up part of this logical volume. GlusterFS will automatically:

Handle failures in case of drive or node failure

Use the fastest available cluster interconnect (in this case, 40GbE network) to transfer data between nodes

Recover lost data (depending on the type of volume deployed)

The raw data files will use a custom defined file format, which typically takes the form of an HDF5 format. A directory structure within the logical format will be used to organize data generated by the data acquisition application and logs. All application access to the distributed file system will occur through appropriate calls in the API.

Moving Upwards - Functional Applications

Functional applications refer to processes and application which are not related to monitoring and control, but are required for the correct execution of an observation schedule. These include calibration, pointing, diagnostic routines, and data acquisition. They are referred to as 'functional applications' since they are not part of the software infrastructure, but rather use it to execute upon. They will generally be executed by running a workflow submitted to the workflow manager. These applications need to specify how much resources they require to run, and whether they'll be run as services or one-off. Depending on the availability of these resources, as managed by the resource manager, they will be scheduled to run on the cluster.

A non-exhaustive list of possible functional applications is as follows:

Data Acquisition – streaming control data from the TPI to the MCCS cluster, to be processed and stored in HDF5 files.

Pointing – beamforming coefficients are calculated every few seconds by the pointing algorithm. The coefficients are downloaded to each TPM (via the TANGO control system).

Calibration – Antenna signals need routine calibration.

Diagnostics – Integrity checks are performed routinely, to make sure antennas, signal path, firmwares are running within acceptable parameters. Some of the information required by the diagnostic routines can be requested via the TANGO control system.

TM-LFAA LMC Interface

As one can guess, a “core component” is required to manage all the LMC subcomponents themselves, as well as expose the available functionality to external users. The primary user of the LMC is TM. The interface between these two entities is defined in the TM-LFAA ICD. This document only lists a number of high-level requirements which this interface should meet, with no reference to technology preferences (for example, REST, RPC, SOAP). Nevertheless, the interfacing protocol should be unaware of any technology choices adopted within the LMC. It should also be architecturally agnostic, meaning that whatever the underlying architecture of the LMC is, it should be able to satisfy the interface requirements through a standardised API. To accomplish this task, the interface on the LMC side should “flatten-out” the functions provided by all system components and provide an API which is de-coupled from the underlying deployment.

This basically contradicts the hierarchical approach. Hierarchies are in place in order to simplify large systems management; in this perspective higher layers just need a summary of lower layers in day by day operation. Whenever detailed information is required, anyway, high layer devices and/or clients could/should relay to the TANGO database of each LMC subsystem as the single point of access.

Broadly, we can list the generic aims of the API as follows:

Implement the TM-LFAA interface

Manage the telescope configuration

Report errors/alarms through an appropriate notification system

Manage role-based privileges and audits

Monitor and control the diverse set of underlying components (TANGO-based or not)

Logging

One realises quickly that there needs to be a way to amalgamate in an effective way, the method by which each different component is managed and communicated with.

Well, most of these requirements are already available in TANGO.

TM-LFAA LMC Interface - Components and Capabilities

LFAA LMC extends the concept of components and capabilities defined in the TM-LFAA ICD to include any internal hardware, software and logical entities. The primary components type defined in the system are:

Hardware (racks, servers, switches, PDUs, ...)

Software (logger, core, device controller, file manager, cluster manager, ...)

Tiles (and stations)

Workflows (observation modes and related jobs)

Running tasks (beamformer, calibrator, DAQ, ...)

The list of components present in the systems is updated dynamically during the execution lifetime of the LMC. For example, when a new job is launched, it is represented as a new component which exposes a new list of capabilities which can be queried and/or called. Capabilities can be categorized into four types:

Properties which represent values that can be set or retrieved (for example “state”, which is available for all components). Several property types are defined, such as metrics, registers, values. Commands re functions defined in the component which can be called. A command may require a number of arguments, and can return several results. Typical commands include initialise, check_status and configure.

Events generated by a component can be used to notify other components of changes. Components may register to receive events from specific components types or instances, and when the event is fired, it will be forwarded to all components registered to receive it. Several event types can be defined, for example “Component configured” and “Component status changed”

Data epressents output data generated by a component which can be accessed or viewed by other components. This can be used, for example, to access raw antenna data generated by a data acquisition process

Additionally, alarms can be set on any property. A permissible value range can be set, and if the property’s value exceeds this range, the component itself is set in alarm state. Alternatively, an alarm value can be specified on the property, and if the current property value matches the alarm value, then it is set in alarm state. The latter behaviour can be used to automatically set a component in alarm state when its internal state matches a particular value.

All the above is already available in TANGO

Unifying Communication

The LFAA LMC core is built as a hierarchy of components, with the core itself being represented as the root of the component tree. The state of each component is an aggregate of the internal states of the underlying components. Communications between components are handled by a communication channels network with a central broker. Messages are sent to the broker which forwards them to any interested components. For example, if component A wants to read a property value from component B, it will send a request message to the broker specifying component B as the destination. The broker will decide on which channels the message will travel through and the message will eventually arrive at its destination. The same procedure is performed to send the reply from B to A. The communication system is currently based on RabbitMQ, with message formatted using JSON.

When a new component is introduced to the system (for example, during system startup) it exposes all the capabilities which can be used by other components. A software component interface is plugged into the core which is capable of communicating with this instance. The core communicates with this interface, which in turn forwards the JSON-formatted request to the component. For example, if the device controller needs to be initialised, the following steps are performed (all communication between the core and interface are direct function calls, whilst communication between the interface and the component instance happens through JSON-formatted messages over RabbitMQ on a dedicated channel):

The core loads the device controller component interface

The core calls the component initialise method

The interface starts the device controller component (depending on which controller the core is configured to use, for example, the TANGO server or custom scripts). The controller can be started on the same server or another server (say, by using the job scheduling interface).

Once the device controller is initialised it will send a notification to the interface

When the core receives this notification (through the interface), it will call the configure method (configure and initialise are different processes, since a component's configuration can change during its lifetime, however it does not need to be initialised each time)

After configuration, the core will call the get_component_capability method on the interface. This will send the JSON-formatted request to the component instance, which will generate a list of internal components, each having their own capabilities. This list is sent as a reply to the core, which places this list in the database so that other components and external entities (such as TM) can query it through the core's API.

Notes on the above: - The list of components (and associated interfaces) which the core should load will be provided through a configuration file - The list of hardware devices which the core should monitor and control (through the device controller) will be provided through a configuration file - All components have a pre-defined set of capabilities, which they inherit through the generic component interface (for example, the state property and the initialise, configure and get_component_capability commands). Upon initialisation and configuration, the component can expose any other internal components and capabilities it desires.

The LMC core uses a NoSQL database (MongoDB) for internal housekeeping, such as: - Keeping track of loaded components and their capabilities - Storing (and possibly updating) a local copy of the Telescope Model - Keeping track of which components are registered to which events - Keeping track of alarms - Temporarily queue events and command runs which need to be forwarded to external parties such as TM

The component system described above sounds like it fulfills a very similar role to the TANGO control system. Extreme care should be taken about re-inventing the wheel and all the related protocols and generic tools. Avoid having a second network protocol to develop and maintain. The question should be asked if TANGO can fulfill this functionality or not. If it can it will simplify the access for all clients in the system and provide a unified system. Easier to manage and maintain and to which clients can connect to via one protocol. If TANGO cannot do the job or requires lots of extra development and if this is a top level application i.e. not a middleware, then it might be necessary to develop a new solution.

Example n.1 – The new ESRF beamline control sequencer which is being developed is written in Python and uses Redis to store acquisition data in memory. It generates data files in HDF5. It talks to TANGO devices to control, trigger and monitor the data acquisition. Any clients that need to talk to the control sequencer should use a TANGO bridge. The goal is not to develop a new middleware even if the temptation is there.

TANGO anti-pattern – develop or use two or more protocols as middleware in your control system.

Unifying Communication - Interfacing with LMC

The dynamicity of the core must be reflected in the interfacing layer, which exposes all the available functionality to third party clients, most notable of which is Telescope Manager. It is known that each LMC element will communicate with TM via a Tango interface. Beneath this Tango interface will reside a RESTful API. The list of possible URLs is generated dynamically when the core is started

and throughout its lifetime, depending on the entries in the database. The URLs are designed in such a way as to make it easy to drill down, or filter, components and capabilities by specifying IDs, types and other filtering options.

REST stands for Representational State Transfer. It relies on a stateless, client-server, cacheable communication protocol (primarily HTTP). It is an architecture style for designing network applications. The primary aim is that instead of using complex mechanisms such as CORBA, RPC or SOAP to connect between machines, simple HTTP is used to make calls instead. RESTful applications use HTTP requests to post data (create and/or update) read data (for example, to make queries) and delete data. Thus, REST uses HTTP for all four CRUD (Create/Read/Update/Write) operations. These operations are performed through the following HTTP requests:

GET – Query an entity for information or data

POST – Issue a command which changes the state of an entity (for example, to create an observation or write a property value)

PATCH – Update the state of a created entity (for example, to stop an observation)

DELETE – Delete an entity (for example, unsubscribe from receiving an event, which will delete the appropriate entry)

Notes: - The API allows for multiple observations to be running concurrently, however currently the design of the core can only support one. When the core is extended, the API does not need to change. - A single component instance can be specified by its component type, component name and component ID. For example Switch 5 would have component type “hardware”, component name “switch” and ID “5”. - The client can subscribe to any event being exposed by the core and internal components. When an event is generated it will be placed in a queue for a specified time window. The client can poll for generated events using the URL above. Filtering on event type and even source is also possible. - Transient entities like observations, command runs and events are only available for a specified time window after they are completed, after which they will be purged from the database. This timeout will be referred to as the PostTimeout. - All API calls returning more than a pre-defined number of items will be paginated so that there is not risk of hogging the core (and client) if a request generated a large reply. The reply will include a URL which points to the next result collection.

Some usage examples:

Check component state, check the current state of all tiles:

GET /components/tile/properties/state

Check the current state of tile 10:

GET /components/tile/10/properties/state

Check the state of the beamforming job:

GET /components/beamformer/properties/state

Check the temperate of all tiles in station 2:

GET /components/station/2/properties/temperature

Create an alarm which provides a maximum value for tile temperature (all tiles)

POST /components/tiles/alarms

POST will contains the property on which to set the alarm (in this case temperature) and the

maximum value (JSON form).

All events and alarms alert are reported on /events. To get all alarm triggered from tiles, the following request can be used:

GET /events/alarm_triggered?source_type=tiles

To get all events for tiles:

GET /events?source_type=tiles

An observation is defined by a telescope model. The external user must generate a valid model which can then be sent to the LMC by using:

POST /observations

POST contains the telescope model, including any software and firmware binaries which would be required. The reply will include an observation ID, which can then be used to start the observation. If an ID of 1 is received, the following can be used to start the observation:

PATCH /observations/1/run

PATCH includes the command to start the observation. A GET request to this URL will return information relating to the current status of the observation.

Unifying Communication - TANGO Wrapper

Given that the LMC has to interact with TM via Tango, the RESTful API has to be wrapped in a TANGO device. The TANGO community is already proposing ways for the TANGO ecosystem to include a RESTful API in future releases. So the wrapper has to be custom-developed for the time being. We currently have defined an API for LFAA, but we feel that there are ways to have this formalised across many LMCs if required. The real work is in mapping the REST API do the different sub-elements, including, but not limited to Tango. This requirement is taken care of by the LMC Core element. To follow the current development on REST APIs for Tango, go to: link

The TANGO REST-api is available and it is strongly recommended to use it. But it's main use is for interfacing web clients and not for defining a new protocol for TANGO. REST is http based with data transferred in ascii and therefore not very efficient. It is not object oriented and therefore it is not a good idea to use the TANGO REST-api as the main communication between clients and the control system. Website for TANGO REST-api: <https://bitbucket.org/hzgwprn/mtango/wiki/Home>

TANGO anti-pattern – use the TANGO REST-api as the main way to communicate between clients and the TANGO devices.

As the result of the LMC Standardization workshop not just the TM was supposed to adopt and use TANGO. In that perspective all the complexity behind this wrapping can be avoided just allowing the TM and the LMCs to natively “speak” TANGO.

Summary

This use case is a description of the current prototype of our development effort for LFAA LMC. The use case is based on our use of Tango, as well as other software packages that are involved in the LMC infrastructure. Primarily, we discuss the following:

Monitoring and control of digital devices and any hardware components
 The detection of any specified alarms/events, to inform appropriate recipients
 The running of computational/logical jobs/tasks with specified schedules
 Routine checks and diagnostics
 Exposing functionality in a logical fashion, or in detailed form for debugging and fault mitigation
 Interaction with any operators/users
 Routing, processing, saving of data streams etc.
 Management of the available computational resources
 This use case will focus on the use of TANGO within the LFAA LMC architecture, and then move outwards to show how TANGO fits within LFAA LMC, to satisfy the LMC features above.

Use Case Title	LMC-TM Interface
Proposer	Dr. Simone RIGGI
Description	
<p>The Telescope Manager monitors and controls SKA Elements through the interface with their LMCs. Within LMC we made these assumptions:</p> <p>1) Interface realization: The interface is realized by a unique instance of a TANGO device running in the LMC control domain. The interface device contains all the Dish monitoring attributes for TM subscription and commands callable by TM (following the TM-DSH.LMC ICD), plus pipes for defining events/alarms. TM directly communicates only with the interface device not with internal LMC TANGO components. Is this assumption correct? 2) Static vs Dynamic Interface: Is the interface supposed to dynamically change at real-time with respect to the ICD specification (stored in the Self Description Data)? In other words, is LMC requested to support creation/removal of monitoring points and/or commands at run-time by TM? This seems the case from the TANGO LIG, but the exact mechanism and physical use cases justifying this for the Dish are unclear at the moment. For instance, for internal interface definition, Dish sub-elements (SPF, Rx) suggested a static interface. As an exercise, we were able to dynamically generate monitoring points at run-time (and also subscribing and actually monitoring them) on the basis of a parsed SSD config (i.e. an XML modified config file with respect to the provided SSD template) but how to generate commands and their actual behavior (=code to execute actions)?</p>	
Moderators Comments	
<p>1) A clean, hierarchical design should only expose the necessary parameters, meaning a summary of the DSH.LMC, in the highest level interface. The TANGO database for DSH.LMC can be seen as the single point of contact between TM and DSH.LMC. Whenever a deeper level of detail is needed TM contacts the DSH.LMC TANGO database (i.e. the DSH.LMC TANGO domain) in order to connect to the relevant TANGO device server.</p> <p>2) This is mainly matter of discussion for the consortia. Anyhow, how could the ICD change with respect to the real available resources (hardware)?</p> <p>TANGO Pipes should only be used for sending complex data. Attributes are preferred because they are simpler to implement, configure and monitor. Use Device Attributes as much as possible for exchanging data.</p> <p>TANGO anti-pattern – using Pipes when Attributes could be used.</p> <p>TANGO supports dynamic creation of Attributes and Commands (since V9). However in an object oriented design every object belongs to a class which implements the same class behaviour and</p>	

interface for all objects of that class. It is not good OO practice to have the objects of the same class with different interfaces and behaviour unless this is really an integral of the behaviour of devices of that class e.g. an RGA. Even if this is possible with dynamic commands and attributes it should be avoided. Use dynamic commands and attributes to **define the same interface and behaviour for all objects of that class**. If you need different interfaces for objects of the same classes consider implementing multiple classes.

Example 10.1 – ESRF uses dynamic attributes to implement the elements detected and read by each residual gas analyser (RGA) at startup time.

Example 10.2 – FERMI uses dynamic attributes to map a large number of hardware registers on low lever radio frequency (LLRF) sybsystems.

TANGO anti-pattern – use **dynamic commands and attributes to implement different class behaviour for different objects of the same class when different classes could be used**.

Use Case Title	Receive health and monitoring information
Proposer	Ms. Shagita GOUNDEN
Description	
SDP LMC will send health and monitoring information to TM.	
Moderators Comments	
Study and choose the TANGO communication mechanism best suited to your needs i.e. synchronous, asynchronous, events. Events should be used for fast communication with large numbers of clients. Check out this tutorial: http://ftp.esrf.fr/pub/cs/tango/conferences/pcapac2014/tango_communication.pdf	

Use Case Title	Request an observation
Proposer	Ms. Shagita GOUNDEN
Description	
TM sends through requests for observations/capabilities to the SDP via the SDP LMC. The SDP LMC then queries the SDP internally to determine the availability of resources to perform the observation and responds to TM.	
Moderators Comments	
Use TANGO device servers to model the SDP and send commands or set/read attributes.	

Use Case Title	Telescope State Information
Proposer	Ms. Shagita GOUNDEN
Description	
TM send telescope state information/metadata to the SDP LMC. Are we closer to knowing what the specifics of this data set are i.e. data type, values, cadence, etc.? TANGO will send this data using TANGO.	
Moderators Comments	
This is not clear. Why should TM send state information to the lower layers? Anyway, define the data which need to be sent and implement them using TANGO Device Attributes.	

Use Case Title	SKA Distributed Tango Facilities Use Case
Proposer	Mrs. Lize VAN DEN HEEVER
Description	
<p>After attending the Tango workshop at ICALEPCS in October 2015 and some discussions at the conference a very early suggestion for using distributed Tango control systems for the SKA project where described (with very little hands-on experience with Tango at the time).This is in Appendix A.</p> <p>Below is a conceptual description of the use case for distributed Tango facilities for the SKA project. The SKA-Mid telescope is being used as example for this use case.</p> <p>Use case description</p> <p>For this use case, let's assume the SKA-Mid consists of the following functional Elements: TM (Telescope Manager) - central monitoring and control system SDP (Science Data Processor) - science data processor CSP (Central Signal Processor) - correlator, beam former, pulsar timing engine, pulsar search engine DISHes (1 to N, N \pm200 for SKA Phase 1, \pm3000 for SKA Phase 2)</p> <p>TM is the central monitoring and control system for the SKA-Mid telescope and orchestrates the other Elements into a working telescope. Each of the Elements is expected to implement an LMC (Local Monitoring and Control) Tango device server providing a standardised monitoring and control interface to TM.</p> <p>Each of the Elements, including each Dish, has to operate as a stand-alone entity (in terms of starting up, restarting, deployment, upgrading, etc.) and will themselves most probably be a Tango facility with its own Tango host and hierarchy of sub-elements, applications, components and devices.</p> <p>All these distributed Tango facilities then need to be coordinated by TM into a central SKA-Mid Tango facility. TM implements Leaf nodes (Tango Clients) that connect to each of the Element LMCs at the bottom of the TM hierarchy (and the Element LMC typically being at the top of the hierarchy of the Element's Tango facility). A question here would be if there are any aspects of Tango architecture or known limitations that will impact the decision on whether to have a separate standalone TM Tango facility in addition to SKA-Mid Tango facility, vs combining these into a single central SKA-Mid Tango facility only with no TM Tango facility.</p> <p>Remote logging from the distributed Tango facilities is another related use case. The roles and responsibilities specified for the LMC includes support for remote logging. A possible approach to this would be that each Element implements an Element Log Consumer Tango device to provide localised logging for that Tango facility, gathered and stored at the Element. And that the LMC Tango device of the element collaborates with the Element Log Consumer to distribute the logs from a selected component and level (as commanded via the Element LMC interface) to TM via a remote log message on the Element LMC interface.</p>	
Moderator Comments	
<p>Having separate TM TANGO facility and SKA-Mid TANGO facility allows for the largest flexibility. Also, very small changes should, in principle, be requested to TM TANGO facility when moving to SKA-Phase2.</p> <p>Concerning the logging it is not clear if the requested feature deals about message logging or engineering data archiving. For both the requirements, anyhow, TANGO device servers exist. Engineering data archiving can be done with the HDB++ archiving system; in order to guarantee a</p>	

complete standalone operation a dedicated HDB++ archiving system for each LMC TANGO facility is foreseen.

Also, for logging purposes, in addition to the file target and the LogConsumer application, a dedicated TANGO device server exist which allows for remote logging on file (or database). FERMI adopts a multiple TANGO domain approach, mainly because of cleanliness of design than performance. A TANGO domain is dedicated to the accelerator, an additional one to the optical sampling subsystem and one to each of the experimental stations. Each TANGO domain is standalone and can operate without the others. Running the accelerator, however, requires that all the TANGO domains are up and running.

The definition of what constitutes a TANGO system depends on the logical and physical separation required and the size of the system. Systems which need to be logically separate i.e. need to independent of the other systems in terms of management and life cycle, should have their own TANGO database. Communication between TANGO systems is transparent using the fully qualified name. Inter-TANGO control system is supported for all tools including the archiving. This means it is possible to have multiple TANGO control systems archived in the same HDB++ database. Having one big system is possible but it means any failure on the database impacts all sub-systems. The way around this is to have multiple Database device servers pointing to the same mysql database which act as fallback systems. The same approach can be used to distribute the load across multiple databases. MySql can be used to implement redundancy and automatic switchover to avoid failure. One drawback of having only one TANGO control system is the load on the database. Until recently this was an issue for storing set_values in the database. This has been fixed by not storing a history for set_values. But you still need to be careful if many devices are storing set_values with a high frequency in the static database. If different life cycles for the different sub-systems or necessary or possible then the rule of thumb is that they have their own TANGO database. We should discuss the exact size and role of each system to give more detailed advice.

The TANGO Logging system is efficient and should be used. Logging can be extended to have more sophisticated tools for analysing logs. One idea is to use an existing logging manager like Apache Kafka for aggregating and analysing logs and interfacing it to the TANGO logging.

Example 18.1 – ESRF has 42+ TANGO control systems. The largest one is for the accelerator complex. It has one MySQL database with two TANGO Database device servers serving devices. All clients and servers connect to the same one (orion:10000) and will automatically switch to the second one (orion:11000) if the first one fails. The size of this system is roughly 200 hosts, 3000 servers and 17000 devices. Backup of the database is done by replicating the TANGO database in a separate slave database (using mysql technology) and then backing up the readonly replicated database. This guarantees that the main database is never blocked during backup. The database size returned by the command DbInfo:

```
TANGO Database sys/database/2
```

```
Running since 2016-01-18 09:01:08
```

```
Devices defined = 17019
```

```
Devices exported = 15422
```

```
Device servers defined = 2956
```

```
Device servers exported = 2422
```

```
Device properties defined = 319947 [History lgth = 1980406]
```

Class properties defined = 3811 [History lgth = 33186]
Device attribute properties defined = 104563 [History lgth = 361724]
Class attribute properties defined = 76 [History lgth = 522]
Object properties defined = 1048 [History lgth = 5596]

The number of calls and average time to execute (in ms) of the Database device server for the ESRF accelerator are displayed in the table below.

Table : ESRF accelerator database statistics for one month

The limitation on the size of a TANGO database has not really been measured. There could be a limitation if the number of devices, servers and clients were an order of magnitude bigger than the ESRF one (20000+ devices). The limitation will depend on the hardware (cpu+disk) where the MySQL database is running, the network and the number of requests per second e.g. if many set values are being recorded at a high frequency or many imports are being done at a high frequency. The way to measure the limit would be to monitor the number of requests served by the Database device server, the performance of the host + database. Any flattening out of performance would be an indication of a bottleneck. If it is on the host then try to increase the hardware the database is running on. It is best to run the database server and database on a dedicated machine so as not to have interference. In the case of the ESRF the host is a 32 core Linux box with 16 GB RAM. The MySQL daemon has about 10% of the load and the Database device server only 1%. The overall load on the server is < .5. The server runs a number of other active device server processes too.

The other 40 control systems are one for each beamline. A beamline is made up of up to 10 hosts, 25 device servers and 500 devices. The beamlines have their own life cycle and start and stop when they need to. The beamlines gets information from the accelerator control system via a dedicated device per beamline which is fed with information from the accelerator. The beamline has direct access to certain hardware which it needs to set or read.

There is at least one TANGO database for testing. But nothing prevents labs or individuals from starting their own database for test purposes e.g. using the TANGO VM.

Figure : ESRF 42 control systems

Use Case Title	Scheduling and deferring control operations
Proposer	Dr. Simone RIGGI
Description	
<p>It is expected that DSH.LMC needs to perform controlled sequence of actions, possibly requiring durations beyond the default TANGO command response time, interactions with different devices and some kind of dialing interaction (i.e. progress reporting) between communicating parties. The configuration of the antenna for observation, performing known safety or error handling actions (i.e. dish stowing after power cut) in response to alarms, or setting-up the feed vacuum level for operations are among the functionalities of DSH.LMC. What is the suggested approach in TANGO (or available tools) to implement this kind of “workflow” or sequenced operation, beyond the provided client API?</p>	
Moderators Comments	
<p>If it's just matter of long execution time for a single command in a TANGO device use the asynchronous command. Conversely, any complex sequence of operation better to be implemented as a TANGO device server.</p> <p>The common pattern here is to implement a sequencer in a TANGO device class. There are a number of possibilities for sequencer but Python is the one most commonly used today. A TANGO device class exists for sequencing as part of the Sardana system and is called the Macro server. It executes macros as Python sequences which are triggered via TANGO and their state is reflected in the device state.</p> <p>Example 9.1 – ESRF uses the Macro device server from Sardana to implement the sequences which control the state changes for the radio frequency system. More information on the Macro server here : http://www.sardana-controls.org/en/stable/devel/overview/overview_macroserver.html</p> <p>Example macro sequence for switching the synchrotron RF system to off state :</p> <pre> """This module contains macros that demonstrate the usage of macro parameters""" from sardana.macroserver.macro import * from PyTango import * import time from ssa_names import * class ssa_off_standby(Macro): """Bring the SYRF SSA into STANDBY state""" def run(self): self.output("[ssa_off_standby] Start macro") # # import devices # self.power_supply = DeviceProxy (PowerSupply) self.debug("[DEBUG] Connected to device " + PowerSupply) self.waveguide = DeviceProxy (Waveguide) self.debug("[DEBUG] Connected to device " + Waveguide) self.amplifiers = [] for i in (Amplifiers): self.amplifiers.append (DeviceProxy (i)) self.debug("[DEBUG] Connected to device " + i) self.phase_loop = DeviceProxy (PhaseLoop) self.debug("[DEBUG] Connected to device " + PhaseLoop) </pre>	

```

self.cavities = DeviceProxy (Cavities)
self.debug("[DEBUG] Connected to device " + Cavities)
#
# check that the transmitter is connected to the cavities
#
self.debug("[DEBUG] transmitter mode = " + str(self.waveguide.TRA0_mode))
if self.waveguide.TRA0_mode != 1:      # Cavity mode
    raise ValueError("Transmitter is not connected to the cavities, cannot go
further!")
#
# open regulation loop
#
self.phase_loop.Amplitude = 0.0
self.phase_loop.RFAmplitude = 0.0
self.phase_loop.OpenMagLoop()
self.phase_loop.OpenPhaseLoop()
self.phase_loop.On()
#
# Switch on power supply if necessary
#

if self.power_supply.State() != DevState.ON and self.power_supply.State() !=
DevState.ALARM:
    #
    # reset before switching power supply ON
    #
    self.output("Reset amplifiers")
    for i in (self.amplifiers):
        i.Reset()
    time.sleep( 1 )
    self.output("Reset power supply")
    self.power_supply.Reset()
    time.sleep( 1 )
    #
    # switch power supply ON
    #
    self.output("Switch ON the power supply")
    self.power_supply.On()
    time.sleep( 5 ) # Transient fault state!!!! Needs to be corrected.
    while self.power_supply.State() == DevState.MOVING:
        self.output("power supply is ramping-up, please wait")
        time.sleep( 2 )
        self.pausePoint()
    if self.power_supply.State() != DevState.ON and self.power_supply.State() !=
DevState.ALARM:
        raise ValueError("The power supply did not reach the ON state: \n" +
self.power_supply.Status())
    #
    # reset after switching power supply ON
    #

```

```

        self.output("Reset power supply")
        self.power_supply.Reset()
        time.sleep( 1 )
        self.output("Reset amplifiers")
        for i in (self.amplifiers):
            i.Reset()

#
# Switch on the cavities if necessary
#
if self.cavities.State() != DevState.ON and self.cavities.State() != DevState.ALARM:
    #
    # switch ON the cavities
    #
    if self.cavities.State() != DevState.ON and self.cavities.State() !=
DevState.ALARM:
        self.output("Switch ON the SY cavities")
        self.cavities.On()
        time.sleep( 5 )
        while self.cavities.State() == DevState.MOVING:
            self.output("Cavities are starting, please wait")
            time.sleep( 2 )
            self.pausePoint()
        if self.cavities.State() != DevState.ON and self.cavities.State() !=
DevState.ALARM:
            raise ValueError("The cavities did not reach the ON state: \n" +
self.cavities.Status())
        #
        # switch STANDBY the Amplifiers
        #
        for i in (self.amplifiers):
            amp_state = i.State()
            if amp_state != DevState.STANDBY and amp_state != DevState.ON and
amp_state != DevState.ALARM:
                i.Standby()
                self.output("STANDBY requested for amplifier " + i.dev_name())
                time.sleep( 5 )
                self.pausePoint()

        self.moving = True
        while self.moving == True:
            self.output("Amplifiers switching STANDBY in progress, please wait")
            time.sleep( 2 )
            self.pausePoint()
            self.moving = False
            for i in (self.amplifiers):
                amp_state = i.State()
                if amp_state == DevState.MOVING or amp_state ==
DevState.UNKNOWN:
                    self.moving = True
                    break

#

```

```

        # check the STANDBY state
        #
        time.sleep( 3 ) # Amplifier paases to off first!!!!!! Needs to be corrected
        for i in (self.amplifiers):
            amp_state = i.State()
            if amp_state != DevState.STANDBY and amp_state != DevState.ON and
amp_state != DevState.ALARM:
                raise ValueError("Amplifier did not reach the STANDBY state: \n" +
i.Status())
            self.output("[ssa_off_standby] End macro")
            return
        def on_abort(self):
            """Hook executed when an abort occurs. Overwrite as necessary"""
            self.output("[ssa_off_standby] Abort macro")
            pass
        def on_pause(self):
            """Hook executed when an pause occurs. Overwrite as necessary"""
            self.output ("[ssa_off_standby] Macro is in pause mode, wating for resume")

```

Use Case Title	Software system monitor and Tango
Proposer	Mr. Matteo DI CARLO
Description	
<p>One of the main responsibilities of the TM.LMC sub-element is the ability to monitor resources and performance of TM as a computer network and as distributed application. As any other TM sub-element, TM.LMC has been decomposed into different products/applications and one of them has to be a software system monitor (SSM) like Nagios (www.nagios.org) or Zabbix (www.zabbix.com) for the above reason.</p> <p>The use case is about the collaboration model between a SSM and the tango framework. From the SSM point of view, the tango framework is a list of services (the domains) and a list of processes (device servers) and in this way, it is important to monitor the health status of them. On the other hand, many times the control system built with the help of the framework has capabilities of monitoring and control and one can think to collect all the elements from the SSM and from the control system monitor everything. What is the best way to solve this? Are there any standards or best practices to adopt?</p>	
Moderators Comments	
<p>As correctly reported the TANGO control system framework has built in monitoring and reporting capabilities. Nevertheless, in a complex system, some services may not be implemented in the framework, such as large database back-ends, large storage systems, or the network infrastructure itself.</p> <p>In principle TANGO and the SSM have different scopes. Design the architecture in a clean, sharp way; avoid mixing different tools for the same purpose. This means that everything that is “in-band” with respect to the control system should be managed by the TANGO framework, using it's own built-in monitoring and reporting capabilities. Anything that is “out-of-band” could be advantageously monitored with the SSM.</p> <p>Consider the db back-end, the storage systems etc., the ones which won't have TANGO on board and will be monitored by the SSM, as auxiliary systems for the telescope and import any relevant parameter into the TANGO framework (for instance for alert, event correlation, etc...).</p> <p>Refer to the following paper for infrastructure management considerations: http://accelconf.web.cern.ch/AccelConf/ICALEPCS2013/papers/thmib09.pdf</p> <p>TANGO has a manager called Astor. This is used to manage all device servers and hosts declared in a TANGO database. It allows all servers to be started, stopped and monitored from one application. It relies on the Starter device server to manage the starting and stopping of all devices on a host. The host starts the Starter at bootup time. Astor does not monitor the same parameters as nagios. Nagios does this very well. One possible approach is to implement a device server which gets the overall state information from nagios and which can then be logged and displayed as an alarm on the TANGO monitoring applications and databases. Refer to the online documentation on Astor: http://www.esrf.eu/computing/cs/tango/tango_doc/tools_doc/astor_doc/index.html</p> <p>Example 1.1 – ESRF uses Astor+nagios to manage 179 hosts and 2249 device servers for the accelerator control system</p> <p>Example 1.2 – ESRF uses Astor to manage 40 beamline TANGO control systems</p> <p>Figure 2: Astor configured to control the ESRF accelerator control system Figure 1: Astor summary of ESRF accelerator control system info Figure 3 : Astor control window for example host with multiple device servers</p>	

Use Case Title	State/Mode/Capability implementation
Proposer	Dr. Simone RIGGI
Description	
<p>Each LMC system is expected to follow the abstract SKA Control Model when defining operational states, modes and capabilities, in agreement with the prescriptions and naming conventions given in the LMC Guideline Implementation (LIG) document. It is likely that some devices (i.e. interface devices or devices in charge to map from internal to external state model) shall accommodate the state/mode/capability information. In contrast, TANGO allows only a DevState within a given device with a predefined list of state codes. The list of state codes can be extended to cover SKA convention by manually modifying TANGO IDL core components but the information is not propagated to developing tools (i.e. Pogo). During the DSH.LMC internal interface definition we therefore modeled states/modes/capability with Enum attributes (recently introduced in TANGO 9). As far as we understood it seems that the predefined state machine management (ruling allowed and forbidden commands/operation under given mode/states) is lost. Furthermore how to consider the predefined TANGO state and status information? Do we simply ignore them in the device implementation phase?</p>	
Moderators Comments	
<p>No. The predefined TANGO State (and Status) should be addressed anyway. They will carry some of the information concerning the TANGO device and are mandatory for an effective operation of a TANGO control system. The added Enum attributes for state/mode/capability will carry the additional information. Unfortunately with this approach the predefined state machine management is lost for the Enum attributes.</p> <p>Consider adding a configurable State subsystem, managed by the TANGO core, that will allow a number of additional user-defined states? i.e. fixed set of core states plus a number of user defined states whenever needed. How to guarantee a homogeneous approach? Clients?</p> <p>The TANGO states are an essential part of each TANGO device and are used by all the generic tools and the TANGO device model. The TANGO state must be implemented as fully as possible. The limitation of 14 states is the result of our experience with our previous control system where any number of states was allowed. After 10 years we had almost 100 different states (many of them similar) and it was impossible for generic tools to handle all the states. If you really need more than 14 states then consider defining a sub state which returns additional information.</p> <p>If these are really sub-states then they should be limited using an ENUM attribute. One way of ensuring all devices have the same interface for sub-states is to use a superclass device e.g. SKADevice or LMCDevice, which implements the sub-state interface and could have a hook for checking the sub-state state machine.</p> <p>TANGO anti-pattern - changing the TANGO IDL file will break the generic tools and library and will be equivalent to forking TANGO.</p> <p>Example 3.1 – The LIMA framework for 2D detectors at the ESRF uses the TANGO state for the overall state and manages sub-states for finer state transitions e.g. ARM detector, READING file, etc.</p>	

Use Case Title	Support for TM's Data Driven Approach
Proposer	Mrs. Lize VAN DEN HEEVER
Description	
<p>There is a lot of variety contributed by the different types of products that TM will have to interface with in SKA. Hence, it is necessary for TM to optimize the integration cost between TM and product LMC's of SKA. TM takes a data driven approach to solve this and standardizes the way TM integrates with all LMC's. TM implements the idea through the definition of a standard Self-Description (SD) schema that it uses to capture the information about all LMC's in a uniform manner. Each LMC provides their respective information to TM through their respective SD. Following is an initial list of minimum items that will be contained in the self-description:</p> <ol style="list-style-type: none"> a. The logical location details of the Element LMC. b. The various generic operating states and the possible transitions that need to be viewed from TM. c. List of commands along with their parameters and attributes. This should also contain the possible values, along with constraints for validation. d. List of possible responses for the individual commands along with associated parameters and attributes. e. List of monitoring points along with associated details such as their rate of updates, interpretations, allowed sampling frequencies, required to be logged and so on. f. List of various statuses that the Element can resume, such as Health status levels. g. List of events and alarms. This should contain the associated data such as their types, severity levels, recommended handling, frequencies and so on. h. Provide description for suppressing of alarms for Elements and their sub-Elements, such as not-fitted, in-maintenance etc. i. List of configurations for the maintenance and health checking of the Element LMC such as parameters that TM should monitor. j. List of issues that the Element can raise, process to troubleshoot and resolve. k. List of events that the Element will need to subscribe to and be provided by TM, e.g. Dish waits for the beam former from the CSP to be ready and send a ready event. l. Provide description for any control loops required by the Element, including size of data, source of data, calculations to be performed on data by TM, frequency of updates. <p>The above information may also evolve going forward. Hence there should be a mechanism in place to implement the SD approach so that all the above information pertaining to an LMC can be captured efficiently and the process supported by appropriate validation and consistency checking. All these information eventually will be consumed by various aspects in TANGO, such as device server, Panic Tool for alarms, Archiver to store the monitoring data and so on. So the process should ideally also allow translating this information into implementation code automatically.</p> <p>Note:</p> <p>We saw the need for such a tool in the context of the ITER project. ITER CODAC team implemented the SDD Editor to solve this problem for their context. Inspired by this, we proposed the development of a Domain Specific Language (DSL) for SKA to solve the problem of TM SDD implementation. This has already been prototyped and a demonstration of the same will be provided as a video link shortly.</p>	

9 Appendix B - Agreed implementation solutions

This section is Still TBD and will be populated after any peer review part of the process or whenever any salient aspect of the control system will be agreed at community level.