ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUDPL03

CONTROL SYSTEM SIMULATION USING DSEE HIGH LEVEL INSTRUMENT INTERFACE AND BEHAVIOURAL DESCRIPTION*

A. J. T. Ramaila†, K. Madisa, N. Marais, SKA SA, Cape Town, South Africa
A.M. Banerjee, S.R. Chaudhuri, P. Patwari, TCS Research and Innovation, Pune, India
Y. Gupta, NCRA, Pune, India

Abstract

Development of Karoo Array Telescope Control Protocol (KATCP) based control systems for the KAT-7 and MeerKAT radio telescopes proved the value of a fully simulated telescope system. Control interface simulators of all telescope subsystems were developed or sourced from the subsystems. SKA SA created libraries to ease creation of simulated KATCP devices. The planned SKA radio telescope chose the TANGO controls framework. To benefit from simulation-driven development tango-simlib, an OSS Python library for data-driven development of TANGO device simulators, is presented. Interface simulation with attributes only requires a POGO XMI file; more complex behaviour requires a simple JSON SIMDD (Simulator Description Datafile). Arbitrary behaviour is implemented selectively using Python code. A simulation-control interface for back-channel manipulation of the simulator for e.g. failure conditions is also generated. For the SKA Telescope Manager system an Eclipse DSEE (Domain Specific Engineering Environment) capturing the behaviour and interfaces of all telescope subsystems is being developed. The DSEE produces tango-simlib SIMDD files, ensuring that the generated simulators match their formal specification.

INTRODUCTION

The current MeerKAT Control And Monitoring (CAM) system is developed against a fully simulated telescope system. In development environments, all the subsystems that would make up the real telescope are simulated at the level of their KATCP interfaces. KATCP is a communications protocol based on top of the TCP/IP (Transfer Control Protocol/Internet Protocol) layer. It is a syntax specification for controlling devices over a TCP link. The full, actual, MeerKAT CAM is run against the simulated devices, thus CAM functionality to be tested without the need of real telescope hardware. This is exploited by CAM developers in their own development environments and also allows automated functional integration tests to be run daily.

The testing of SKA Telescope Manager (TM) will be started in the absence of other Elements since not all of them will be available when TM is ready to be tested, it would be beneficial to have a mechanism that allows TM testing without depending on other element's Local Monitoring and Controls (LMCs). A useful tool for developing an evolutionary TM prototype is a data-driven TANGO simulation framework that is used to develop Element LMC simulators that can be used in the TM test environment. The goal is to develop a simulation framework for Element LMC Simulators that can be used in the TM test environment. Furthermore, the TM interface to LMCs can itself can be simulated using this framework. The simulation framework was presented to Element Consortia to keep them abreast of developments in this respect. Element Consortia are being encouraged to make use of the Simulation Framework to develop LMC Simulators where required. The following risk reductions have been identified:

- Risk reduction for early Assembly Integration and Verification (AIV) support;
- Risk reduction for TM product development, by providing early LMC simulators;
- Risk reduction for Element development by producing LMC Simulator Framework to aid them in the development of element simulators;
- Risk reduction by producing an early scriptable TM Simulator that can aid Element LMC development and integration efforts.

The early AIV integration would demand complete development of some components. Unavailability of hardware or incomplete development of element can be a hurdle for such AIV integration. The simulation framework reduces such risk by generating simulators that could be used in place of LMC's which are not fully developed or unavailable due to hardware dependencies. The test framework provides an approach to create test cases and points out areas where it can reduce manual effort in writing test cases through some amount of automation. It was shown that a basic LMC simulator can be produced using the information provided by the Element LMC Interface Control Document (ICD) through the simulation data-description file. The approach also enhanced our understanding of how domain specific simulators can be integrated into the testing and simulation framework as and when they become available. The simulation framework demonstrates how this risk can be mitigated for the TM product development by auto generating the simulators for the LMC's to a great extent based on the Self Description-Data (SDD) data that captures the information typically captured using ICD's in a structured and machine processable manner. Initially this was an exploratory prototype with the aim to develop it further into an evolutionary prototype during 2017 in the period towards Critical Design Review [1].

* Work supported by NRF.
† aramaila@ska.ac.za

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUDPL03

APPROACH AND STRATEGY

As TANGO has been selected as the LMC common framework this prototype can be developed using TANGO while incorporating concepts from the MeerKAT fully simulated framework as used in the MeerKAT CAM development and qualification. The usage of simulators during the development of MeerKAT telescope proved useful. Hence the aim was to enable the reuse of the same simulators for TANGO based environment. Based on this idea, a generic TANGO simulator open source library [2] was implemented in the Python language. The simulation library (simlib) eases the development of simulators exposing a desired TANGO interface. The initial step explored the technological feasibility of implementing a development environment which is aligned with the proposed Telescope Management (TELMGT) design. Using a Domain Specific Language (DSL) to generate the SDD was explored by the TCS-R&I team showcasing the use of aDSL developed using the Model Driven Engineering (MDE) methodology, implemented as an Eclipse Plugin. The DSL enabled capturing SDD information that was proposed in the TELMGT design. The first version of the SDD template to capture the self-description data was released as a part of LMC Interface Guideline (LIG) document by the TELMGT team. In this phase the DSL was used and compared against the SKA schema to see if there was any problem generating an instance of the template using the DSL. The prototyping plan identified the following iterations for the LMC Interface Simulator Framework prototype:

- **Iteration 1** - First demonstration with MeerKAT framework using Tango simulator(s) and TANGO -> KATCP translators;
- **Iteration 2** - Evaluation of TANGO tool capabilities in the context of a MeerKAT-like radio telescope;
- **Iteration 3** - Improved Simulators, including behaviour extension;
- **Iteration 4** - Demonstration of the exploratory LMC Simulation.

The main focus of the initial iteration was on: training and familiarisation with the TANGO framework; implementing simple TANGO based simulators (i.e. simulated "devices" exposing TANGO interfaces; and investigating how TANGO devices may be incorporated in the existing, fully functional KATCP based MeerKAT Control and Monitoring (CAM) environment. The MeerKAT CAM performs a function equivalent to the SKA Telescope Manager for MeerKAT. The main focus of the second iteration was on investigating Tango tool capabilities in the context of a real telescope (MeerKAT) TM system; progress towards data-driven TANGO device simulators with richer behaviour; making simlib more dynamic; investigating the Eclipse based Domain Specific Engineering Environment prototype (DSEE) as a source of data for data-driven simulators; and translating between KATCP and TANGO devices and clients. The primary use case was to allow the use/evaluation of TANGO tools alongside the existing KATCP based MeerKAT CAM. Another future application could be interfacing KATCP devices (potentially MeerKAT subsystems) to a TANGO based TM.

TOOLS

Since we were a distributed collaboration team based out of South Africa and India various tools were used to facilitate the development of the Control System Simulation Framework, such JIRA© for issue tracking and project management and Google Drive© and WEBEX web conferencing for association purposes. GitHub© is mainly used as version control of project code repository.

PROJECT OUTPUT

First Demonstration with MeerKAT Framework using Tango Simulators

During the initial stage, simple TANGO based simulators were implemented, and we investigated how those device simulator can actually be incorporated in the existing, fully functional KATCP based MeerKAT CAM environment. As a proof of concept, a weather device simulator was implemented, i.e., Fig. 1. This simulator was configured to match the functionality of the existing MeerKAT weather station. Each TANGO attribute of the simulated weather station is backed by a simlib simulated quantity. Only the data parameters need to be specified; simlib automatically creates a simulated TANGO attribute that varies according to the specified data. During phase 1, slew-limited Gaussian random variable quantities and constant quantities are supported. Simulation parameters (e.g. min/max/mean) can be specified, as can the metadata (e.g. TANGO attribute label, description text, unit) that is used to configure the TANGO attribute backed by the simulated quantity. Each simulator created with simlib creates two TANGO interfaces. The simulated device interface [Fig. 1] mimics the TANGO interface that a real device would have presented to the TM. The simulation control interface [Fig. 2] is used to control the behaviour of the simulator. It can be used to change simulation parameters and values in real time. It could e.g. be used to simulate an error condition to test the TM's reaction. The ATKPanel view of the two TANGO interfaces of the weather simulator is presented [Fig. 1 & 2]. To demonstrate the effect of the simulation control interface, the wind speed is simulated with the default values that were entered in the simulator code. Around 17:33:30 the standard deviation of the simulated wind speed quantity was reduced [Fig. 3]. At around 17:36:30 the simulation was paused [Fig. 3]. At around 17:37:30 the simulation was resumed [Fig. 3]. As another proof of concept, an existing advanced MeerKAT simulator was adapted to present a TANGO interface rather than a KATCP interface. The MeerKAT Antenna Positioner (AP) simulator presents the same KATCP interface as the

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUDPL03

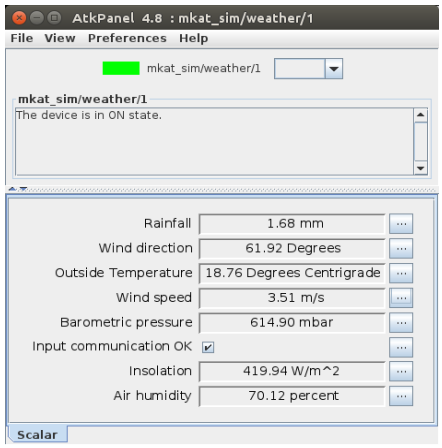

Figure 1: A TANGO implemented weather device simulator.

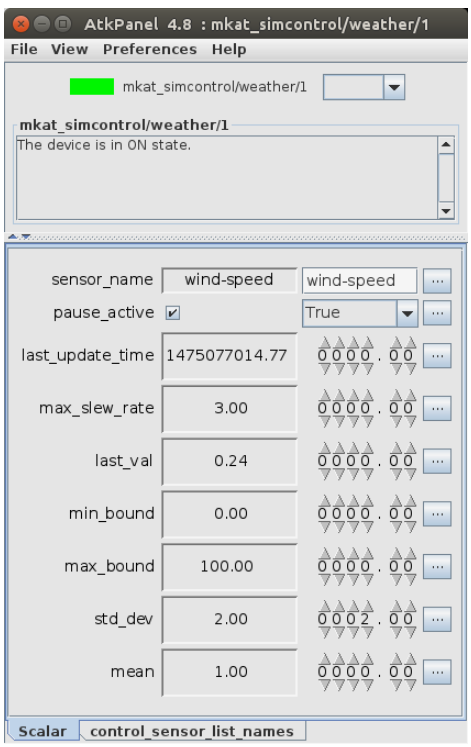

Figure 2: Simulation control interface with adjusted std_dev and paused simulation.

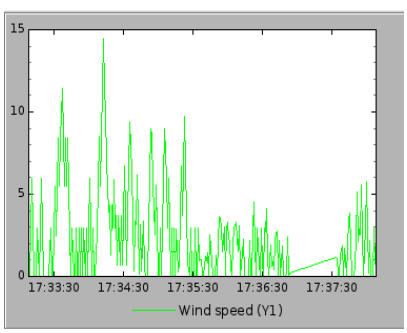

Figure 3: Simulated wind speed response to control interface updates.

actual hardware AP as delivered by the antenna motion control subcontractor. A TANGO based simulator was created by using the original AP simulator model class unchanged, and implementing a TANGO interface class to replace the KATCP interface class as the view and controller. For each TANGO device that needs to be incorporated a translator instance is created [Fig. 4]. The translators are generic, and have no precoded idea of the TANGO device it is translating. Each translator connects to its TANGO device as a client and inspects the device for TANGO commands and attributes and exposes them over a KATCP server interface as KATCP requests (TANGO commands) and sensors (TANGO attributes).

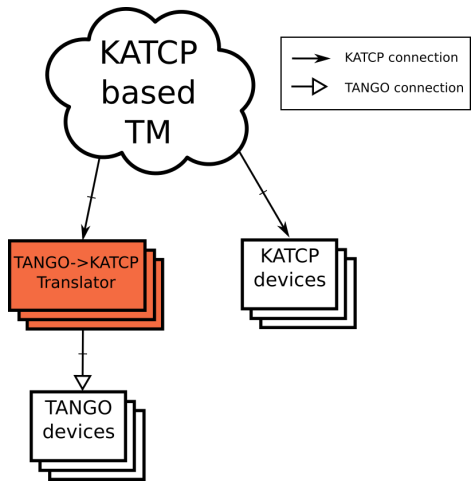

Figure 4: KATCP based TM controlling TANGO devices.

Evaluation of TANGO Tool Capabilities in the Context of a MeerKAT-like Radio Telescope

The main focus of this iteration was investigating TANGO tool capabilities in the context of a real telescope (MeerKAT) TM system. We also wanted to explore the usage of DSEE so that much of the simulation requirements could be specified using DSL's rather than having to hardcode them using programming languages. The DSEE provides a DSL for specifying the details of a system that needs to be simulated through a controller. From the specified description, it provides capability to derive the TANGO implementation of the same. It further allows to refine the input with specific details about the types of simulation needed e.g. algorithm to be used with parameters. Based on this the DSEE configures tango-simlib library as necessary. Using the KATCP device to TANGO translator [3] to expose MeerKAT devices as TANGO devices, the TANGO HDB++ [4] and PANIC [5] tools were evaluated. We leverage Python's dynamic language features to allow the simulator to be defined by an input data file, while providing a mechanism for behavioural extension using external Python code. In support of this approach, the generic simulator library described in TANGO Simulator devices is extended to be more dynamic with an eye on data driven simulator gen-

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUDPL03

eration. The data derived from a device description (such as the DSEE DSL) or simulator description data, should have been derived from a device's ICD in the first place. For this reason, it could be useful to testing device simulators, and potentially also comparing real devices to their ICDs. In light of this use case, the test generation mechanism currently implemented in the DSEE have been evaluated [6].

Improved Simulators, including Behaviour Extension

Here we mainly focused on updating the simulator library to allow the creation of functional, fully data-driven TANGO device simulators without having to code. The updated simlib component was released as an open source package on github [2]. Support for model actions that back TANGO commands have been introduced. By default a do-nothing action is generated that simply produces a successful return value, but support for model state manipulation and Python code-based action overrides are supported and are described below. Thus a useful basic device simulator can be created using only the TANGO POGO interface generation tool. The XMI file generated by POGO is interpreted at runtime, and used to dynamically set up a simlib device model. TANGO attributes defined in the XMI file are mapped to model quantities like those demonstrated in the first stage of the project, but without the need to write code. TANGO commands defined in the XMI are mapped to default no-op model actions. These simulators expose the same functionality as described in Iteration 1, including the simulation control interface. A provisional Simulated Device Description (SIMDD) format was introduced to allow more complicated data-driven simulators to be generated. This is a fairly simple JSON based format. The SIMDD can be used to specify attribute quantity simulation types and parameters. Table 1 shows a SIMDD code snippet example for specifying attribute simulation parameters. Moreover extended TANGO device command action simulations were implemented; the behaviour of the default command actions can be modified using the SIMDD as shown in Table 2; SIMDD example snippet for specifying command simulation. The aim is not to support arbitrarily complex command behaviour through the SIMDD (see command simulation overrides below, Table 3), but to allow the most common, relatively simple command behaviours to be specified via the SIMDD. Current SIMDD specification allows command input parameters to be transformed (currently "copy" is the only transform, but mathematical and other basic operation could be added) and stored to a named temporary variable. These temporary variables can be used to define side effects that update attribute simulation quantities in the simulation model. Furthermore the command can return either a temporary variable or a model quantity. Custom action handler overrides can be coded in Python (**APPENDIX**). The override class is specified as shown in Table 4; SIMDD snippet for specifying a command action override class.

Demonstration of the Exploratory LMC Simulation

The main focus of this iteration was to utilize tango-simlib in an SKA-mid representative LMC configuration and to implement interoperation between the DSEE and tango-simlib. To explore the concept of using tango-simlib to generate LMC interface simulators for SKA telescope elements, a simulator was generated for the LMC interface of a real SKA telescope element. The DSH element was chosen for this exercise since we have easily obtained the developed LMC ICD of the element at the time of writing. Since the first DSH prototype is to be qualified using MeerKAT in the near future, it would also be beneficial to have a DSH simulator allow integration with the MeerKAT TM system before the physical DSH hardware is on site. The existing DSH ICD was analysed to generate a basic device interface using the TANGO POGO tool. The .xmi file generated by POGO was then passed to tango-simlib to generate the basic simulator functionality. The MeerKAT AP simulator code was ported to use the tango-simlib override class API in order to extend the basic tango-simlib simulator to provide more detailed physical modeling of a moving DSH. To incorporate the DSH simulator in the MeerKAT system, the TANGO <-> KATCP translator [3] developed during earlier iterations was used. Since the DSEE device DSL already captures much of the information that would be needed to generate a device simulator, the approach of generating tango-simlib input files for a device described using the DSL was investigated. In principle this would allow a simulator to be derived from the same description as a real device, ensuring consistency between the two. The DSEE DSL describes a device at a somewhat abstract level, and does not include simulation specific information such as the expected statistical parameters of simulated attributes. To address this, the original DSL is enriched with two additional DSL layers [6], viz. the Pogo DSL and the Tango SimLib DSL, Fig. 5. The Pogo DSL layer adds TANGO-specific metadata that can refine the TANGO representation generated from the Monitoring and Control Modelling Language. The Pogo DSL layer is applicable both to the real device and to a simulated representation, ensuring that both have the same TANGO interface. The Tango SimLib DSL adds extra simulation parameters, such as statistical parameters for simulated attributes and expected behaviour of simulated commands. This information can be exported in the tango-simlib SIMDD format, allowing a simulator to be created without having to write any extra code. Regarding the Domain Specific Engineering Environment prototype (DSEE) for driving simulator data generation and test generation: The approach was found to have some promise, but the DSEE in its current form could not be used for e.g. testing an actual TANGO device against its ICD. After feedback was provided to the DSEE developers, it was extended to support the generation of tango-simlib simulator data files. This would allow both a real device and a simulated device to be generated from the same DSL description.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUDPL03

Table 1: SIMDD for Attribute Simulation Parameters

```
"dynamicAttributes": [
    {
      "basicAttributeData": {
         "name": "an attribute",
         // Other "standard" attribute parameters that
         // can also be specified in the XMI file has
         // been removed for brevity
         "dataSimulationParameters": {
            "gaussianSlewLimited": {
              "min_bound": "minimum bound for randomly
                   varied simulator quantity",
              "max_bound": "maximum bound for randomly
                   varied simulator quantity",
              "mean": "mean value",
              "max_slew_rate": "maximum slew rate",
              "update_period": "update period"}
    }   }   },
]
```

Table 2: SIMDD for Command Simulation Behaviour

```
"basicCommandData": {
     "name": "command name",
     "description": "command description",
     "actions": [{
     // Transform the command input value and store it
        in a named
     // simulation temporary variable (scope is the
        command handler)
     "behaviour": "input_transform",
     // Name of a temporary variable to store the
        transformed value into
     "destination_variable": "temporary_variable_name"
     },
     // Side effects
     {
     "behaviour": "side_effect",
     // Temporary variable that provides side
        effect data
     "source_variable": "temporary_variable"
     // Model quantity that is updated
     "destination_quantity": "temperature"
     }
     // Output return values
     {
     "behaviour": "output_return",
     // either a temporary variable
     "source_variable": "temporary_variable"
     // or a model quantity, but not both, only one of
        source_variable or
     // source_quantity may be specified.
     "source_quantity": "model_quantity_name"
     }
]}
```

Table 3: SIMDD for a Command Action in Override Class

```
"override_class": {
    "name": "unique_override_identifier",
    // "module_directory": "Locate the override module
       in this directory",
    "module_name": "mkat.simulators.weather_sim"
    "class_name": "WeatherActionOverrides"
}
```

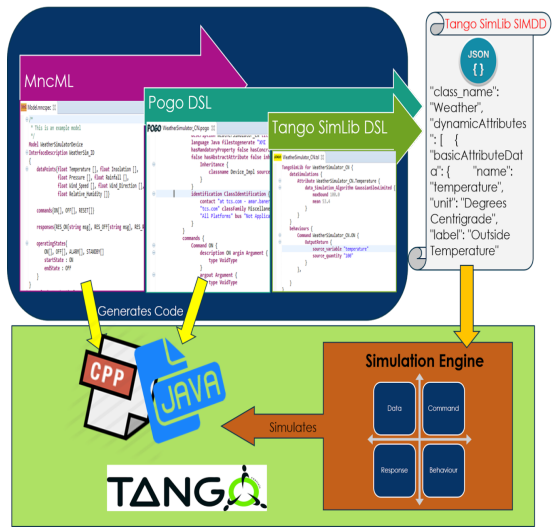

Figure 5: Using DSEE to generate tango-simlib data files.

CONCLUSION

This exercise has shown that KATCP and TANGO based devices are similar enough in principle that the concept of translators could be utilised when integrating MeerKAT into SKA Mid Phase 1. The development of a generic, data-driven TANGO simulator generation library has progressed well and can now be used to create moderately complex simulators with TANGO interfaces without having to write any code. This current functionality should be sufficient to develop a basic simulated SKA telescope environment against which TM could be developed. This was proven by implementing an early DSH LMC simulator and integrating it with MeerKAT TM using a TANGO <-> KATCP simulator to support software integration of the SKA Dish element (DSH) prototype into MeerKAT before the physical prototype arrives on site.

ACKNOWLEDGEMENT

We would like to acknowledge the National Research Foundation (NRF) and Department of Science and Technology (DST), India, for funding the project. And we gratefully thank the SKA TM Consortium partners responsible for the development of the TM, namely NCRA (India), SKA SA (South Africa), and TCS Research and Innovation (India).

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUDPL03

APPENDIX

An override class example is shown:

Table 4: Simulation Command Action Override Class

```
class WeatherActionOverrides(object):
    """An example of the override class for the TANGO device
    class 'Weather'.  It provides implementations of the
    command handler functions for the commands specified
    in the POGO generated XMI data description file.
    """
    def action_stoprainfall(self, model, tango_dev=None,
                            data_input=None):
        """Totally sets the simulated quantity rainfall to a
        constant value of zero.
        """
        try:
            quant_rainfall = model.sim_quantities['rainfall']
        except KeyError:
            MODULE_LOGGER.debug("Quantity rainfall not in "
                                "the model")
        else:
            quant_rainfall.max_bound = 0.0
```

REFERENCES

[1] V. Mohile, SKA1 TM Prototyping Plan, Apr. 2017, unpublished.

[2] tango-simlib, https://github.com/ska-sa/tango-simlib

[3] K. Madisa, L. van den Heever, N. Marais and A. J. T. Ramaila, "Integration of MeerKAT and SKA Telescopes using The KATCP/TANGO Translators", presented at ICALEPCS'17, Barcelona, Spain, Oct. 2017, paper THSH201, this conference.

[4] Tango Evaluation: HDB++, https://docs.google.com/document/d/1qgpn54w3Igs-lmFWsv2Nusuo3NTcfclAM-3Q17yXeYQ/edit#heading=h.h722b3a2ksep

[5] Tango Evaluation: PANIC, https://docs.google.com/document/d/1X3UB5-zLQKBnWjJ-WEQnXyaW-ggtgPz0I8fAGZraZII/edit#heading=h.hhg6djgg60h8

[6] DSL: DSEE, https://gitlab.com/patwari.puneet.ska/MAC-SEEN