DATA DRIVEN SIMULATION FRAMEWORK

Amar Banerjee, Subhrojyoti Roy Chaudhuri, Puneet Patwari, TRDDC, Pune, India
Lize Van Den Heever, SKA South Africa, Cape Town, South Africa

Abstract
Control systems for radio astronomy projects such as MeerKAT[1] require testing functionality of different parts of the Telescope even when the system is not fully developed. Usage of software simulators in such scenarios is customary. Projects build simulators for subsystems such as dishes, beam-formers and so on to ensure the correctness of a)their interface to the control system b)logic written to coordinate and configure them. However, such simulators are developed as one-offs, even when they implement similar functionality. This leads to duplicated effort impacting large projects such as Square Kilometer Array[2]. To mitigate this we leverage the idea of data driven software development and conceptualize a simulation framework that reduces the simulator development effort to: 1)capturing all the necessary information through instantiation of a well-defined simulation specification model 2)configuring a reusable engine that performs the required simulation functions based on the instantiated and populated model provided to it as input. We discuss the results of a PoC for such a simulation framework implemented in the context of Giant Meter-wave Radio Telescope[3] in this paper.

INTRODUCTION

Large projects that involve implementation of large hierarchy of control systems generate dependencies across the controllers to be developed and as a result on the teams developing them. This inter-dependency creates problems in the verification of controllers developed across teams since the individual teams follow their own time-lines which often are not well synchronized.

A general solutions to this problem is replacing the missing components with simulators to aid the verification of the dependent components. This however increases the effort to manually develop the simulators which also at times results in duplication of efforts.

Although the need for simulators in such scenarios seem to be essential, it is desired to reduce the cost of building such simulators since it might impact the overall cost of projects such as Square Kilometer Array(SKA) significantly, as such projects have large number of modules.

With the proposed simulation and testing framework it becomes possible to achieve the goal of verification of the module with simulation, following the model driven approach. The framework incorporates meta models of the controller node, the simulator and testing based on which it automatically generates the simulators. Much of this information is derived from the Self Description Data (SDD) which contains description of the Controller to be developed.

In this paper we discuss the simulator and test framework, the architecture and the working of the framework.

The paper starts with a discussion on standard practice involved in control system testing and verification using MeerKAT as the Case study. It is followed by the architecture section where we describe our understanding and design of the framework. In the final section we present a proof of concept showing the use of our simulation framework. This is followed by a conclusions section where we describe the conclusions of our work.

STANDARD PRACTICE

MeerKAT Case Study

One of the development practices of the MeerKAT CAM (Control and Monitoring) team was to use a fully simulated system at all times. The MeerKAT CAM team has been using simulators extensively and continuously for development, testing and qualification of the CAM subsystem functionality throughout the MeerKAT project life-cycle, since the early days of Fringe Finder for the very first two antennas in the Karoo, through KAT-7[4] to this day for MeerKAT with each array release. It is possible to run a MeerKAT CAM configuration including only simulated devices, or any combination of real and simulated devices combined. This allows full software development, unit testing, integration testing, and CAM subsystem qualification without any dependency on the hardware being available.

While the CAM team was responsible for developing most of the simulators, some of these device simulators were contractually delivered by the subsystem contractor to ensure that, given their knowledge of the device, the behaviour of the device is reflected with sufficient accuracy by the device simulator. In cases where the subsystem contractor did not deliver such a simulator, the CAM team developed a software simulator for the KATCP interface. Preparing the simulators gave the CAM team a valuable opportunity to gather information about the behaviour of the other subsystems even before those subsystems have been fully developed and are ready for integration.

Each simulator represents the specific messages on KATCP (KAT Control Protocol) interface for a subsystem (commands/requests and monitoring points/sensors), as

ISBN 978-3-95450-148-9

well as simulating the expected behaviour of the module when commanded through the control-and-monitoring interface, including maintaining state and mode and reflecting current state in monitoring points. Each simulator also provides a test interface that is used to stimulate the simulator to affect responses and outcomes, alarms and failure conditions.

This approach has been extremely beneficial for the MeerKAT CAM development, but the simulators can be improved by providing a simulation framework that can be personalised for each specific interface by using data-driven specification of the interface instead of writing each simulator manually and keeping it up to date separately. An additional improvement would be to extend the simulation framework with a standard behaviour extension module that can also be personalised through a data-driven behaviour specification. This could include specifying the timing taken by commands before responding, specifying interrelations between commands and reflecting the outcome in monitoring points, and also to relate commands to one another. To explore the benefits of these improvements a POC (Proof of Concept) has been developed by TRDDC on the GMRT with a possible approach to a data-driven simulation framework.

PROPOSED ARCHITECTURE

Simulation Specifications Model

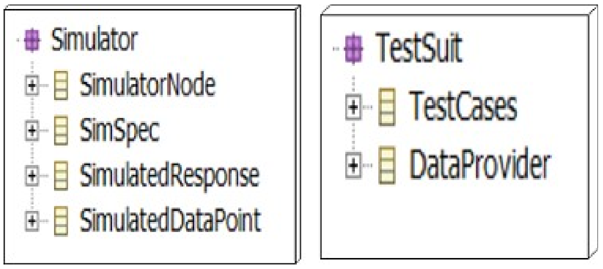


Figure 1: Simulator Model and Test Suite Model.

As Fig. 1 explains the abstract controller simulator model comprises of 2 parts :-
1)Software Emulator and
2)Hardware Simulator.

The analysis of this model leads to gathering requirements for the simulation and testing framework.

From the figure, the information that form the part of the controller simulator are grouped as

a) details such as commands, responses, events, alarms and data
b) communication protocols and protocol translation rules,
c) behavioral specifications such as state machines
d) specific input related to simulation such as simulated responses, data stream, simulation of the behavior of the underlying infrastructure such as unexpected responses, streams
e) skeleton structure of the implementation of the code.

The idea is to capture all these items as a part of the simulator model so that an environment can be provided to capture these information in a structured manner. This provides all the key requirements that the simulation and testing framework needed to support.

Architecture

Using the Model Driven Engineering philosophy, we provide as part of this framework aDomain Specific Language (DSL) MnC&ML[5] to capture all the above information. We term an instance of such information captured for a controller node as the Self-Description Data (SDD) for the controller node. Hence an SDD instance can capture all the key information pertaining to a controller that is not ready yet but is required to be simulated.

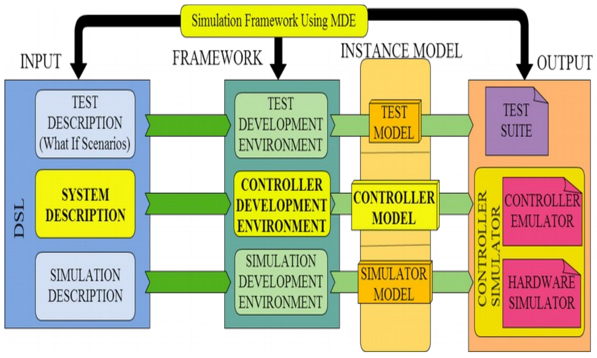


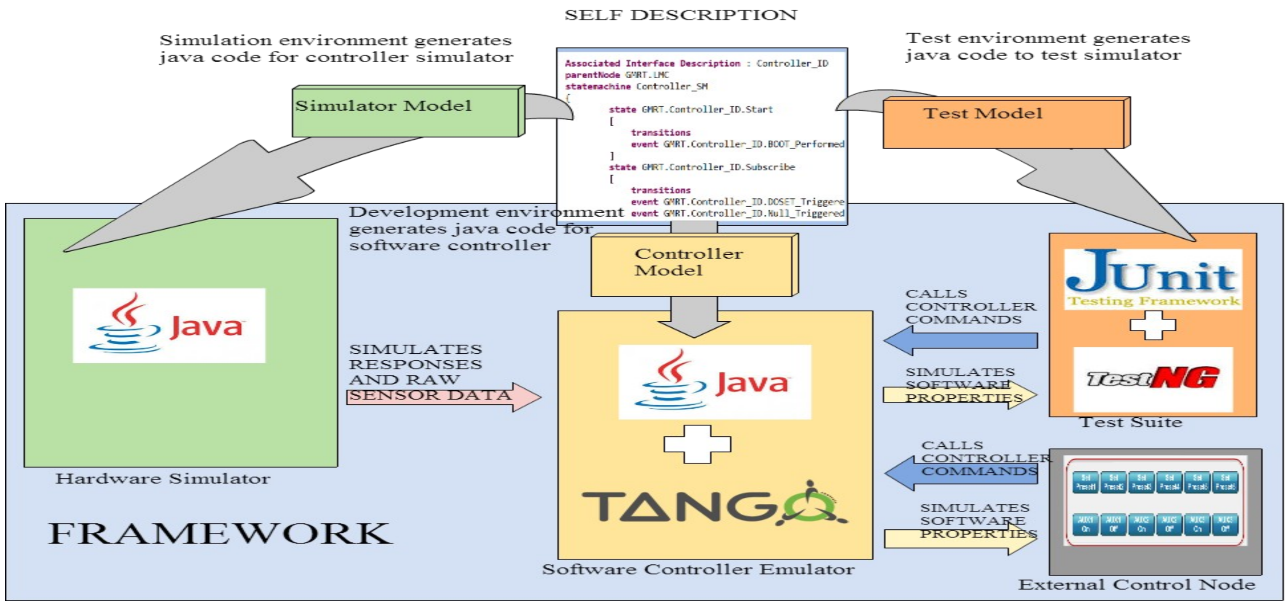
Figure 2: Role of Environments.

The high level architecture of the framework is provided in Fig. 3 and the information related to the implementation of the same such as the technology stack used and so on can be seen from Fig. 4.

From the instantiated model or SDD the framework then code generates the controller/software simulator, hardware simulator and the test suit to test the simulator itself.

The test suit acts as a driver and calls the functionality of the controller and the simulator provides with a suitable simulated reply. The controller node which needs to be tested can then be connected to the simulator to then carry on with the testing of the controller node.

As can be seen from the figure below, the environments help to create the different parts of the models:-


SELF DESCRIPTION
Simulation environment generates java code for controller simulator
Simulator Model
Test environment generates java code to test simulator
Test Model
Development environment generates java code for software controller
Controller Model
Java
SIMULATES RESPONSES AND RAW SENSOR DATA
Hardware Simulator
FRAMEWORK
Java
TANGO
Software Controller Emulator
CALLS CONTROLLER COMMANDS
SIMULATES SOFTWARE PROPERTIES
JUnit Testing Framework
TestNG
Test Suite
CALLS CONTROLLER COMMANDS
SIMULATES SOFTWARE PROPERTIES
External Control Node

```
Associated Interface Description : Controller_ID
parentNode GMRT.LMC
statemachine Controller_SM
{
        state GMRT.Controller_ID.Start
        [
            transitions
            event GMRT.Controller_ID.BOOT_Performed
        ]
        state GMRT.Controller_ID.Subscribe
        [
            transitions
            event GMRT.Controller_ID.DOSET_Triggere
            event GMRT.Controller_ID.Null_Triggered
```

Figure 3 : Technology Stack and Internal Details.

- **Development Environment** – A controller model is instantiated by the development environment and captures the behavioral and skeletal properties of a controller node like commands, events, alarms, state machine logic etc. These properties are desired to create the software emulator for the controller node. This environment is developed as a plug-in for Eclipse using modeling tools such as EMF[6] and XText[7].
- **Simulation Specification Environment** – A simulator model is captured by the simulation specification environment and captures the specification required to simulate the behavior of the underlying hardware. The environment gathers specifications such as expected and unexpected responses to corresponding commands, selection of patterns to generate the sensor data such as sine wave and so on, the time lag to generate two consecutive sensor data values, the number of values to be skipped before generating the next sensor data value. These details inject realistic behavior into the generated simulator. Since these specification items do not form a part of the SDD of a controller they are captured separately through this environment This environment is implemented as a plugin for Eclipse using EMF framework and with building custom UI.
- **Test Specification Environment** – A test model gets instantiated by the test environment to capture the test scenarios involved to test the simulator itself. It reuses the information from the SDD such as expected response to commands, response validation rules and so on to automatically generate the test cases. It also injects some specific test conditions which force the simulator to act in a specified manner to obtain specific test results. The test environment is generic in the sense that it could also be used to test an actual controller and not necessarily only a controller simulator. This environment could incorporate algorithms to make sure that it generates all the exhaustive test cases to test a particular controller or hierarchy of controllers. This environment is implemented as a plug-in for Eclipse using EMF framework and with building custom UI.

After populating these models the environments automatically generates the code based on the instantiated models.

- **Controller/Software Emulator** – The controller emulator is generated by the development environment utilizing the details specified in the SDD. This controller simulator implements the functionality described in the SDD in form of an executable JAVA file utilizing the APIS' from the TANGO[8] framework.
- **Hardware Simulator** – The hardware simulator generated by the Simulator environment, incorporates the details captured by the Simulation model through the simulation specification environment. This too generates a JAVA file which contains mapping of the desired response to a command and the functionality to generate a raw sensor data as per the rules provided in the simulation model.
- **Test Suite** – The test suite is generated by the test environment implementing the test scenarios based on the populated test model. The generated Java code uses the Junit[9] framework API's and consists of multiple test cases for the simulator testing. Each test case is mapped to five different test conditions provided using the Data Providers. This makes the test cases run 5 times and tests the simulator for 5 different conditions. The test case tests the command of the controller by invoking it with 5 different input data and checks the corresponding response against an expected response collected as part of the test model. Hence for N commands of a controller the total tests performed will be 5N times.

The envetual goal for this test suite is to perform exhaustive testing of all features not just limited to commands. The test case also injects a specific test

ISBN 978-3-95450-148-9

condition where it forces the simulator to generate that specific response which is pre-populated by the test model, hence ensuring that every test case has at least one test condition which gets the desired response and passes the test case.

PROOF OF CONCEPTS

Giant Meter-wave Radio Telescope

As a PoC we have used the simulation and testing framework to simulate the controller nodes for one structural hierarchy of the GMRT as follows:-

In the GMRT control system, an Antenna Node controls an IF Controller Node, which in turn directly interacts with the underlying hardware device [provide reference to GMRT architecture]. In order to try out our framework, we assumed that the Antenna Node has been developed and needs testing of its functionality but the IF Controller is not yet developed With our framework, we could still create the self descriptions (SDD) of the IF Controller using our DSL. This file contained the behavioral and skeletal properties of the IF controller without having the implementation details and hence could be created in less than an hour.

Once the DSL is create we generated the controller software emulator for the IF Controller followed by generating the hardware simulator. The simulator model reuses some details in the controller SDD like Commands, Responses, States, Events etc. and also uses simulation specific input. We also generated the test cases for IF Controller Simulator which generates a report upon the execution of the test cases. . We needed to study the GMRT system a bit to come up with the required specifications. But once we had good understating of the features of the antenna node and the IF controller, the creation of the individual specs using our framework took less than an hour.

Using the simulation and test framework for creating a simulator for an IF Controller led us to the following observations:-

- Existing knowledge of the control system is required before creating the simulation specification.
- The use of the framework made it easy to generate the simulator as creating the specification required very less time (around 30 mins).
- Manually coding the simulator would have taken approximately 1-2 hrs, while with the framework it took only around 15-30 mins to do it, hence a 400 – 800 % efficiency in time consumption
- Test cases made it possible to test the simulation before actually plugging it in a real developed Antenna Controller.
- The execution of the test cases results into testing the dynamic behaviour of the simulator before it could be made to real use.

CONCLUSION

The MDE approach for implementing the simulation and test framework can definitely prove more efficient than the traditional way of manually developing individual simulators. However, the simulators generated will still need to be enhanced with appropriate domain logic Although the generated simulators currently are standalone and statically configured during compile time. we want to create an integrated approach which could make changes to the specification possible during its execution as well. We also look ahead to create better methodologies to generate test conditions which would target the acute cases for simulator testing. The simulation model now contains minimal necessary features, in future we wish to provide it with more features so that it would behave more realistically and show better performance.

ACKNOWLEDGEMENT

We would like to thank Dr. Swaminathan Natarajan, Dr. Alan Bridger and Dr. Yashwant Gupta for their continuous support in shaping our thoughts. Also thanks to the GMRT and SKA South Africa teams for providing their support.

REFERENCES

[1] MeerKAT CAM Design Description, DNo M1500-0000-006, Rev 2, 2013.
[2] A.R. Taylor, "The Square Kilometre Array", Proceedings IAU Symposium No. 291, 2012.
[3] Giant Metrewave Radio Telescope, www.gmrt.ncra.tifr.res.in
[4] KAT-7 (seven dish MeerKAT precursor array) http://www.ska.ac.za/meerkat/kat7.php
[5] Puneet Patwari, Subhrojyoti Roy Chaudhuri, Swaminathan Natarajan , G Muralikrishna : M&C ML: A Modeling Language for Monitoring and Control Systems.
[6] Eclipse Modeling Framework: http://www.eclipse.org/modeling/emf/
[7] Lorenzo Bettini, “Implementing Domain-Specific Language with XText and XTend”, (August 2013)
[8] The Tango Control System Manual Version 8.1 : www.esrf.eu/computing/cs/tango/tango_doc/kernel _doc/ds_prog
[9] JUnit : https://www.junit.org/