THE MeerKAT GRAPHICAL USER INTERFACE TECHNOLOGY STACK

M. Alberts*, SKA SA, Cape Town, South Africa
F. Joubert#, SKA SA, Cape Town, South Africa

Abstract

The South African MeerKAT radio telescope, currently being built some 90 km outside the small Northern Cape town of Carnarvon, is a precursor to the Square Kilometre Array (SKA) telescope and will be integrated into the mid-frequency component of SKA Phase 1. Providing the graphical user interface (GUI) for MeerKAT required a reassessment of currently employed technologies with a strong focus on leveraging modern user interface technologies and design techniques. An extensive investigation was performed to evaluate and assess potential GUI technologies and frameworks. The result of this investigative study identified a responsive web application for the frontend and asynchronous web server for the backend. In particular the AngularJS framework used in combination with Material Design principles, Websockets and other popular javascript layout and imaging libraries, such as D3.js, proved an ideal fit for the requirements of the MeerKAT GUI frontend. This paper will provide a summary of the user interface technology investigation and further expound on the whole technology stack adopted to provide a modern user interface with real time capabilities.

INTRODUCTION

MeerKAT is a mid-frequency "pathfinder" radio telescope and precursor to building the world's largest and most sensitive radio telescope, the Square Kilometre Array (SKA). MeerKAT builds upon its own precursor, KAT-7 (Karoo Array Telescope), a seven-dish array currently being used as an engineering and science prototype.

During the preliminary stages of the MeerKAT control and monitoring design it became evident that the current KAT-7 user interface and its underlying technologies have various shortcomings and will not scale well. This prompted an investigation into modern user interface technologies, especially web frameworks with all its accompanied benefits.

This paper starts with a brief description of the design methodology adopted. Next, this paper provides a summary of the user interface technology investigation. As main focus, this paper presents the new MeerKAT GUI architecture with detail on the architecture and technology stack.

METHOD

High level requirements for the MeerKAT user interface were defined by System Engineering and further refined through bi-monthly discussions with the relevant stakeholders, especially the telescope operators and commissioners. During these meetings all requirements were clarified, additional operator requirements were defined, and mock-up displays were drawn to ensure the resulting interface will fulfill all the needs of the end users.

Following an iterative development approach, combined with monthly demonstrations of prototype displays to relevant stakeholders, we were able to obtain valuable feedback that we could include in the development cycle.

TECHNOLOGY INVESTIGATION

After conducting a research exercise into GUI technologies for Responsive Web Design (RWD), various frameworks and libraries were identified. Categorising these GUI technologies according to their main function showed that few were all-in-one solutions for a user interface development platform. Most of them focus on a specific area of user interface design and should be used in conjunction with other libraries to construct a complete frontend development platform solution.

To overcome analysis paralysis we've limited our evaluation to only a select few technology solutions based on industry popularity and peer recommendations, namely AngularJS [1], EmberJS [2] and CS-Studio BOY [3].

With the help of the Control and Monitoring (CAM) team we identified criteria to evaluate the chosen technologies against while building prototypes using each technology as a frontend design platform. The prototypes were based on a typical antenna control and monitoring use case. The evaluation criteria included open source licensing, MVW (Model View Whatever) framework, good documentation, large example base, large active developer community, large scientific user community, pre-built standard widgets, in-house knowledge, user-defined widgets, rapid prototyping toolset, template system, support for testing frameworks, deployment/server side simplicity, easy learning curve, security support (authentication/authorisation), Python support, flexible data-bindings and flexible layouts.

Final scoring of the three prototyped user interface technologies made it clear that web technologies are best suited to our needs, with AngularJS coming out as the favourite.

MEERKAT GUI ARCHITECTURE

Architecture Design Overview

On a high level, the MeerKAT user interface implements a client-server architecture. A frontend component provides the client-side functionality and a

* talberts@ska.ac.za
# fjoubert@ska.ac.za

Copyright © 2015 CC-BY-3.0 and by the respective authors

ISBN 978-3-95450-148-9

backend component provides the server-side functions, as illustrated in Figure 1.

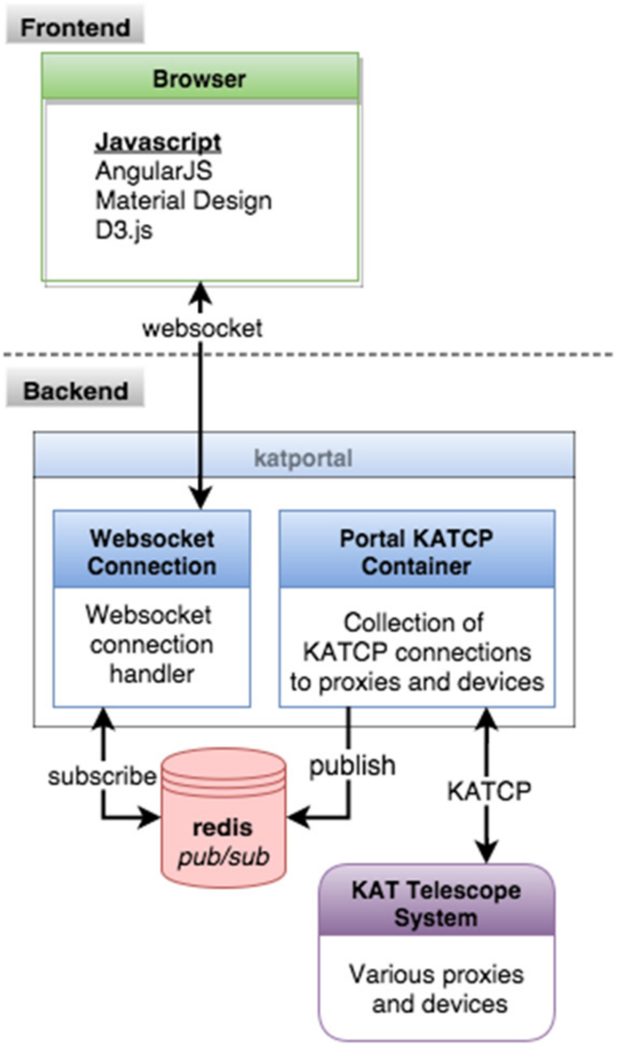

Figure 1: MeerKAT GUI architecture overview.

MeerKAT GUI Backend

Many of the MeerKAT GUI displays require real-time updates of monitored data. Therefore, careful consideration was given to the delivery of a continuous data stream to the frontend.

To enable an interactive session with real-time flow of data between a browser and server, the modern web communication technology of choice is the Websocket [4]. This full-duplex, single socket connection allows a client to send messages to a server and receive event-driven responses without having to poll the server for a reply.

Combining Websockets with a publish-subscribe design pattern (PubSub) allows a GUI display to subscribe to all monitoring points of interest and handle real-time updates to those monitoring points as published by the server.

The Backend Technology Stack comprises:

- Ubuntu 14.04 LTS Server [5] as operating system
- Nginx [6] as Hypertext Transfer Protocol (HTTP) server and reverse proxy
- Redis server [7] for PubSub
- Python 2.7 [8] as the main programming language
- Tornado web framework [9] for the various webservers supporting both normal HTTP requests and Websockets

Katportal is the Python package providing all the backend functionality. It consists of webservers for authentication and authorisation, monitoring, control, querying system configuration and to read data from storage [10].

The monitor webserver exposes a PubSub interface through a Websocket connection. The protocol used for communicating over the Websocket is JSONRPC [11], which enables the server to expose methods as remote procedure calls.

Typically, a client opens a Websocket connection to the monitor webserver and subscribe to a set of monitoring points on the telescope system. The monitor webserver maintains a collection of KATCP (Karoo Array Telescope Control Protocol) connections to the proxies and devices making up the telescope system.

Whenever a subscribe request is received from a client, the monitoring point(s) of interest are registered on the KATCP container and also on Redis. Should the value of any of subscribed monitoring point change on the telescope system, the container will notice the value update event almost immediately and publish the update to Redis. In turn, the existing session for the client's Websocket connection will be notified by Redis and the updated value will be sent to the client.

Default sampling strategies for monitoring points - such as periodic, event-based, event-based-with-rate - are defined through configuration on the server, but can also be set to a custom update strategy by the client through the PubSub interface.

The control webserver exposes a RESTful [12] application programming interface (API) for all control related tasks. Control tasks can only be performed with the proper authentication and authorisation.

The authentication webserver enforces user authentication and authorisation depending on user roles. All web requests influencing the telescope system state (i.e. control related) have to be authorised with a login-unique session identifier as per the Javascript Object Notation (JSON) Web Token standard [13]. If the authorisation information inside the web token matches the information stored at the server for the authenticated user making the request, the action is allowed and executed on the server.

MeerKAT GUI Frontend

The frontend is implemented as a Single-Page Application (SPA), which downloads all necessary HTML - and JavaScript files when the browser loads the frontend's Uniform Resource Identifier (URI). The frontend then communicates with the backend via a RESTful API and Websockets.

Copyright © 2015 CC-BY-3.0 and by the respective authors

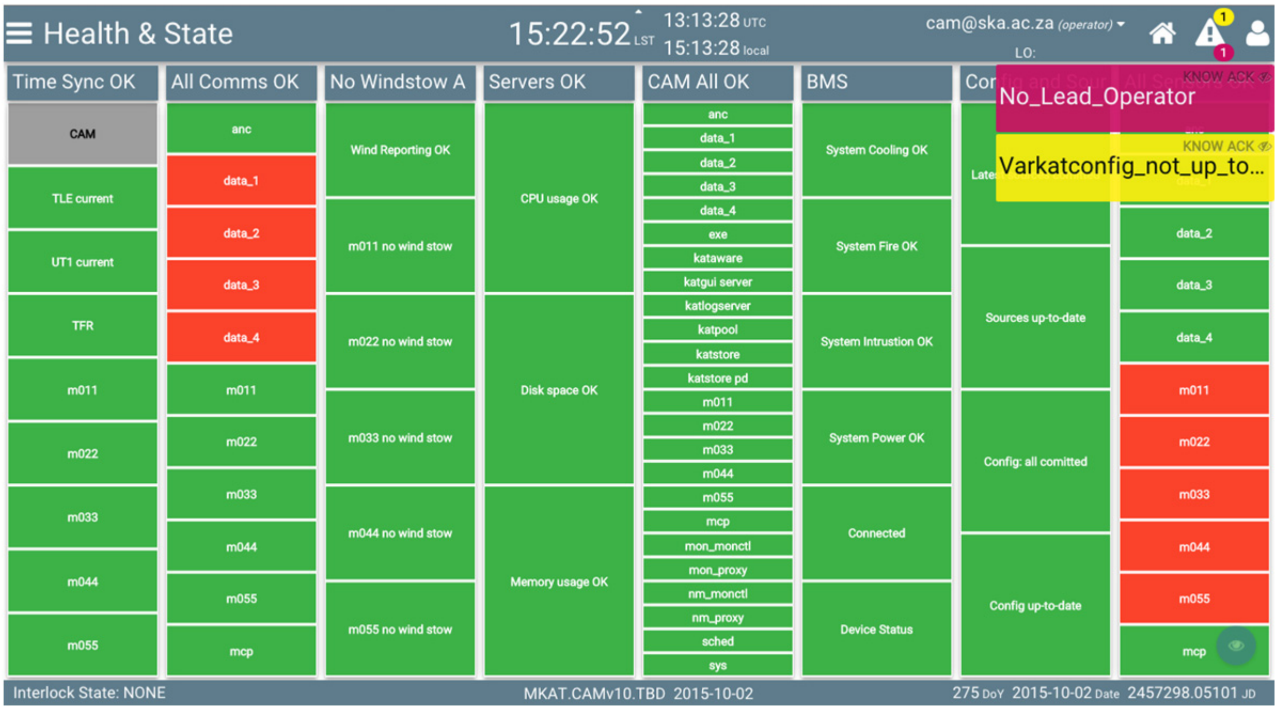

Figure 2: The telescope system health overview display. The system alarms are shown as an overlay at the top right. The main toolbar is displayed at the top.

The Frontend technology stack comprises of:

- The Google Chrome web browser [14]
- AngularJS 1.4.x [1]
- Document Driven Data (D3.js) [15]
- Angular Material [16]
- Numerous JavaScript utility libraries

Each browser window or tab connects to the backend and can make concurrent modifications. Users go through an authentication process that assigns user roles, which in turn limits access and modifications to certain parts of the frontend.

MEERKAT GUI FEATURES

Main Display

The frontend layout contains the following:

- Main toolbar
- Bottom toolbar
- Side navigation
- Alarm notification overlay
- Selected display's content

The main toolbar contains items like navigation links, current UTC, local solar and sidereal time, alarm counter badges, and the logged-in user's information. The bottom toolbar shows the current system interlock status, current version information and date information, including Julian date. The side navigation bar helps with quick navigation between displays. The selected display content occupies the rest of the space.

Landing Page

After a successful login, the user sees the landing page. The landing page is a dashboard that can be configured with widgets depending on the user's needs. Currently, there is a navigation widget and a NASA Astronomy Picture of the Day [17] widget.

Health Displays

Graphical health displays were developed to assist in fast and efficient fault-finding. These can be placed on large, heads-up displays in the operator control rooms. These include the telescope system health overview (see Figure 2), represented as columns of coloured blocks and interactive, customisable, tree-views to show the health of the antennas.

The sensor list display shows all the monitor points of a selected resource as a scrollable list, using colours to emphasise the status of the monitor points.

The user can use regular expressions in the custom health view, in order to filter sensor names and build custom health views, displayed as blocks of colour, and then export these views as a URI for reuse.

Pointing displays show where all the antennas are pointing, in terms of azimuth and elevation as well as right ascension and declination. The pointing displays can be configured to use either the equatorial or the horizontal coordinate system.

A special weather display is used to display the current local weather conditions at the site.

Alarms

The alarm display gives the user the ability to acknowledge and clear alarms. Alarm notifications are shown as counter badges on the main toolbar and as an

Copyright © 2015 CC-BY-3.0 and by the respective authors

overlay on each browser tab, which stays visible until the operator acknowledges the alarm. Every time an alarm is received, a different sound is played, depending on the severity.

Observation Scheduling

Observation scheduling displays allow for the organization and scheduling of observations per subarray. Subarrays are logical collections of receptors and other resources. The workflow starts with setting up the subarray by assigning of resources, setting the frequency band, current configuration, end product and a Control Authority for the observations. The subarray is then activated. Only a user who has a role as a Lead Operator or a Control Authority can modify the subarrays.

After the subarray activation, the designated control authority assigns small units of work, known as schedule blocks, to the subarray. The schedule blocks are then verified and scheduled, which moves them into a list of observations that will be executed automatically or manually, depending on the scheduler mode of the subarray in the telescope system.

Operator Control and Intervention

The operator control display allows the user to execute various emergency operations when needed. This display shows the current status of all the antennas, which can be used to monitor the execution of the emergency operations. Operations include: stop observations, resume operations, shutdown computing etc.

Historical Data

The telescope system has robust data storing and archiving capabilities. The historical data can be queried using the frontend, which plots the data on a chart. Multiple data lines of the same type can be drawn on the same chart. Changes in discrete data (e.g. an antennas mode, which switches between ‘STOP’, ‘STOW’ and ‘POINT’) can also be plotted.

User Logs

Users can create different types of logs on the user log and reporting display. These would typically include shift logs, observation logs, time-loss logs, maintenance logs etc. The display allows the user to specify a start and end time for the selected log type, a text log message and functionality to upload files associated with the log. The user can, at any time, generate a report for a log type and include system activity logs in the report for later review.

Configuration and Theming

Many of the features in the frontend can be customised and configured in configuration display. Configuration options include hiding alarm notifications, disabling alarm sounds, and configuring a colour theme for the frontend. The theming in the frontend allows the user to specify the colour of toolbars and buttons as well as selecting a dark background colour for night operations.

CONCLUSION

Employing a framework based on of web technologies allows even a complicated system to reap the various benefits of web based applications. The growth of computing power enables complicated user interface processing to be handed over to thick clients, reducing unnecessary overhead on the backend servers.

PubSub messaging provides an excellent isolation between consumers and producers of data and allows for easy scalability.

ACKNOWLEDGEMENTS

We wish to thank all SKA-SA operators, engineers and commissioners who participated in discussions and provided us with insightful comments and ideas.

REFERENCES

[1] AngularJS website: https://angularjs.org/
[2] Ember website: http://emberjs.com/
[3] CS-Studio BOY website: http://sourceforge.net/projects/cs-studio/
[4] Websocket website: https://www.websocket.org/
[5] Ubuntu website: http://www.ubuntu.com/
[6] Nginx website: http://nginx.org/
[7] Redis website: http://redis.io/
[8] Python website: https://www.python.org/
[9] Tornado web framework website: http://www.tornadoweb.org/en/stable/
[10] M. Slabber, “Overview of the monitoring data archive used on MeerKAT”, these proceedings, ICALEPCS, Melbourne, Australia (2015).
[11] JSONRPC standard website: http://www.jsonrpc.org/specification
[12] R.T. Fielding, “Architectural Styles and the Design of Network-based Software Architectures”, Ph.D dissertation, Dept. Inf. and CompSc. , Univ. California, Irvine (2000).
[13] JSON Web Token (JWT) standard website: http://jwt.io/
[14] Chrome web browser website: https://www.google.com/chrome/
[15] Document Driven Data website: http://d3js.org/
[16] Angular Material website: http://material.angularjs.org/
[17] NASA Astronomy Picture of the Day website: http://apod.nasa.gov/apod/astropix.html

Copyright © 2015 CC-BY-3.0 and by the respective authors