A user interface framework for the Square Kilometre Array: concepts and responsibilities

Marassi, Alessandro, Brajnik, Giorgio, Nicol, Mark, Alberti, Valentina, Le Roux, Gerhard

Alessandro Marassi, Giorgio Brajnik, Mark Nicol, Valentina Alberti, Gerhard Le Roux, "A user interface framework for the Square Kilometre Array: concepts and responsibilities," Proc. SPIE 9913, Software and Cyberinfrastructure for Astronomy IV, 991347 (26 July 2016); doi: 10.1117/12.2232903

SPIE.

Event: SPIE Astronomical Telescopes + Instrumentation, 2016, Edinburgh, United Kingdom

A user interface framework for the Square Kilometre Array: concepts and responsibilities

Alessandro Marassi[a], Giorgio Brajnik[b], Mark Nicol[c], Valentina Alberti[a],Gerhard Le Roux[d]
[a]INAF - Osservatorio Astronomico di Trieste (Italy); [b]Univ. degli Studi di Udine (Italy); [c]UK Astronomy Technology Centre (United Kingdom); [d]SKA South Africa (South Africa)

ABSTRACT

The Square Kilometre Array (SKA) project is responsible for developing the SKA Observatory, the world's largest radio telescope, with eventually over a square kilometre of collecting area and including a general headquarters as well as two radio telescopes: SKA1-Mid in South Africa and SKA1-Low in Australia. The SKA project consists of a number of subsystems (elements) among which the Telescope Manager (TM) is the one involved in controlling and monitoring the SKA telescopes.
The TM element has three primary responsibilities:

- Management of astronomical observations
- Management of telescope hardware and software subsystems
- Management of data to support system operations and all stakeholders (operators, maintainers, engineers and science users) in achieving operational, maintenance and engineering goals.

Operators, maintainers, engineers and science users will interact with TM via appropriate user interfaces (UI).
The TM UI framework envisaged is a complete set of general technical solutions (components, technologies and design information) for implementing a generic computing system (UI platform). Such a system will enable UI components to be instantiated to allow for human interaction via screens, keyboards, mouse and to implement the necessary logic for acquiring or deriving the information needed for interaction. It will provide libraries and specific Application Programming Interfaces (APIs) to implement operator and engineer interactive interfaces.
This paper will provide a status update of the TM UI framework, UI platform and UI components design effort, including the technology choices, and discuss key challenges in the TM UI architecture, as well as our approaches to addressing them.

Keywords: human-machine interface, user interface, usage centered design, control room, alarm handling

1. INTRODUCTION

The SKA is a project with a large up-front design effort that will take several years to develop and deliver. We are thinking about architecture for an observatory that has not yet been built, supporting two instruments that are not yet fully defined and with a potential lifespan of 20-50 years. It seems sensible therefore to expect that many new features and ways of viewing and working with the underlying systems will emerge during that time and the code needed will change, quite often in unexpected ways.
We need an architecture that is both flexible and robust. It should allow changes to be made easily, and also be highly decoupled. We need it to be flexible because we need to adapt to better technological solutions and changing requirements; we need to be robust, so it provides a trustworthy interface to complex and variable instruments; and we need it to be highly decoupled so that when changes are made, they have a minimum impact on the rest of the SKA operations.
Because of the diverse nature of the user interactions with TM (covering different locations, type and stage of interaction) it was decided that the Telescope Manager would not have a single monolithic UI system. To simplify the definition of the various user interactions with the TM system, the activities can be grouped based on the stages of observation execution:

Software and Cyberinfrastructure for Astronomy IV, edited by Gianluca Chiozzi, Juan C. Guzman, Proc. of SPIE Vol. 9913, 991347 · © 2016 SPIE
CCC code: 0277-786X/16/$18 · doi: 10.1117/12.2232903

Offline preparation: covers all the preparation (proposals submissions and reviewing, scheduling block (SB) creation and planning etc) activities that occur before a set of related SBs are executed on the Telescope.
Online observation execution: includes all the interactions with an active telescope to operate the instrument (including handling of failures and alarms but also scheduling and configuring SBs) so that observations can be executed effectively.
Offline support: is any activity not directly involved in the operation of TM but necessary to maintain effective operations. This will include updates to software, failure diagnosis on the telescope systems and remote deployment of software items.
The architecture design defines a UI console component (Online UI Manager) as a portal or virtual container of UI components. Its main function is to implement a control mechanism whereby different UI components invoke other components and collaborate together. Because of its structure, the Online UI Manager will imply navigation between all its "children" UI components.

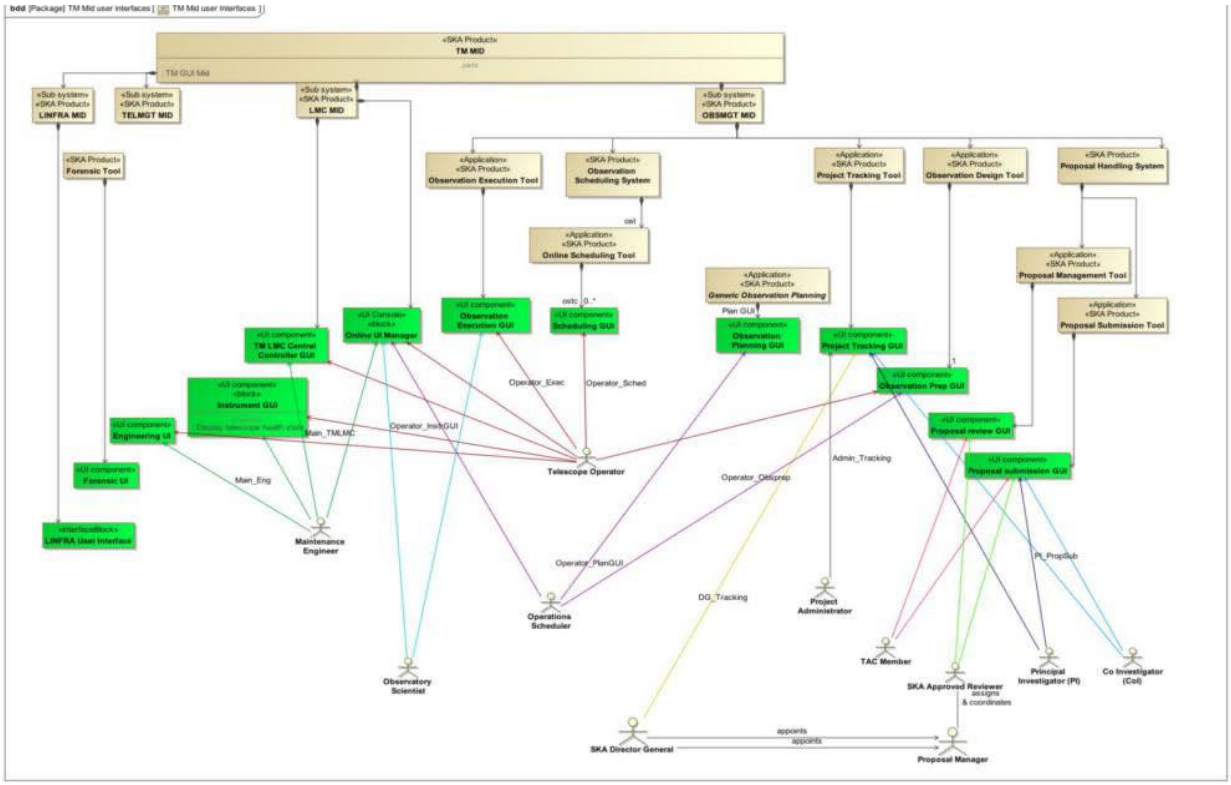

Figure 1. SKA Telescope Manager User Interfaces

Architectural constraints The interface presented to a user of the SKA is likely to consist of a mix of browser-based User Interfaces, desktop applications and script based applications. These various UIs coexist and run on a user's physical (or virtual) desktop. For browser-based applications, the code is executed within the browser, and for other UI applications this implies the code executes on the user's physical machine to present the view to the user.

Browser-based UIs The assumption is that browser-based UIs would be developed as web applications with a front-end consisting of JavaScript, HTML and CSS. Reuse of elements between these UIs would primarily be by means of shared libraries. Components could be broadly defined as UI elements, or collections of UI elements and their associated properties. Shared stylesheets would be used to ensure consistency of look and feel across UIs. Advanced UI visualisations would be provided by using or extending UI libraries such as Domain Driven Documents (D3.js).

Having an agreed default strategy for client server communication between the TM UIs and their supporting applications will simplify the deployment and configuration of the UIs, and make it potentially easier to extend UIs or to develop new ones. For SKA this is likely to be a combination of ReST services over HTTP (using JSON or XML) for control messaging between all UIs and the server, with reception of events and dynamic data being pushed from publisher sockets based on an agreed publish/subscribe model.

Desktop applications are counted as any packaged applications executed on the desktop machine. This covers both compiled applications and Java applications run on the desktop machine using a locally installed Java Virtual Machine (JVM). Where desktop applications share a common architecture and similar concerns, libraries of reusable UI elements could be developed to ensure a consistency of look and feel. This would depend on the number of desktop technologies chosen and the commonality between applications.

Script based applications are both the automated execution of observing scripts, and also any ad hoc tasks executed manually. The primary use of the UI framework would be providing UI elements for creating, managing and validating these scripts and for creating the required interface points for these scripts to interact with other components or services.

A key challenge for the SKA with regard to the UI design, is the sheer size of the telescopes and as a consequence the number of elements and their related data that users will need to manipulate. One example is handling the large number of alarms that may be raised in a given time window. SKA supports a flexible configuration of up to sixteen different sub-arrays, each of which may be scheduling different observations. The interface has to present all the devices, their processes and any alarms raised in a comprehensible way. In order to complete their tasks, operators working in the control room will have to sift information from this vast array of data, checking the health status of the entire system, determining the validity of the scientific data produced and checking that the quality of the results is sufficiently high. Our stance is that a usage-centered approach to the design, analysis and development of UIs is important, as it will reduce the risk of developing UIs that are suboptimal for such a situation.

This paper provides a status update of the TM UI framework, UI platform and UI components design effort, including the technology choices and discusses key challenges in the TM UI architecture, and our approaches to addressing them.

2. BACKGROUND

A usage-centered design (UCD) approach[1] for interactive software applications is based on the early involvement of users of the application from its conception. In practical terms, it means that **feedback offered by users** is considered in analysis phases, as well as design and evaluation (the first characteristic of a UCD process). Also because building a usable UI requires those involved in its construction to understand, and actually conceive, the mental model that users will have of the application, the design process has, in practice, to be **iterative**. Finally, each iteration is based on design-prototype-evaluate activities, where the **evaluation is based on usability** criteria.

Several techniques can be applied to establish such a kind of process, each aimed at informing the design process so that better and better design decisions can be made as the partially specified UI is extended. Often designers proceed through what is called “parallel design”, meaning that they conceive not only a single UI based on a few interaction ideas, but develop a small set of alternative conceptual ideas[2]. Each of these “concepts” are gradually refined and extended, and eventually inferior ones eliminated through evaluation. Such organization of the work guarantees that design proceeds not only through gradual refinement, in an evolutionary way, but also through qualitative discontinuities, in a revolutionary way.

In order to be effective (ie, produce results that are valid and useful) and efficient (and therefore be sustainable), users need to be involved in structured ways, not simply by asking them casual questions and looking for their opinions.

Techniques that can be put in place to follow a UCD approach include structured interviews, contextual enquiries, sketching, storyboarding, user testing, writing scenarios and personas, among others[1,3,4,5,6]. To some extent even in situations where the user population is not well-known or well-defined the use of personas has proven to be a helpful technique in focusing design thinking around the needs that the interface is being designed to support[7].

3. UIs OF SOME SKA PRECURSORS

An important aspect of understanding the needs of the SKA UIs is an appreciation of how these challenges have been tackled by similar modern observatories and precursors projects for the SKA. The following summaries give a broad outline of the heritage on which the SKA is building.

3.1 LOFAR

LOFAR[8] is a large interconnected radio telescope using a vast array of omnidirectional antennas. The electronic signals from the antennas are digitised, transported to a central digital processor, and combined in software to emulate a conventional antenna. The project is based on a vast interferometric array of about 20,000 small antennas concentrated in 60 stations distributed across Europe. The total effective collecting area is approximately 300,000 square meters, depending on frequency and antenna configuration. The data processing is performed by a Blue Gene/P supercomputer situated in the Netherlands at the University of Groningen.
LOFAR makes observations in the 10 MHz to 240 MHz frequency range with two types of antennas: Low Band Antenna (LBA) and High Band Antenna (HBA), optimized for 10-80 MHz and 120-240 MHz respectively.

LOFAR User Interfaces
The entire observation process is overseen by Operators, Science Support and Software Support groups.
A certain number of user interfaces are provided for:

- *Scheduling, Administration and Specification (SAS)*: given the specification, the main responsibility of SAS is to schedule and configure the system in the right mode. Additionally SAS facilitates the possibility to store metadata of the system for a long term and make that information accessible for the user.
- *Monitoring And Control (MAC)*: the main responsibility of MAC is to control the system (in real time) based upon the actual configuration at that moment. Additionally, MAC facilitates the (real time) monitoring of the present state of the system.
- *System Health Management (SHM)*: SHM is identified as an autonomous block that predicts and acts on failures of the hardware before it actually fails. Ideally it should even pinpoint which system component is the cause of a failure. The reason for considering this block as separate from MAC is because of the scale of the system and the percentage of time the system should be effectively operational.

There is a central control room, located at the ASTRON headquarters in Dwingeloo, with several operator screens. These screens display a vast range of different sources of information including: correlator information, administration data such as messages and active processes, IT infrastructure components monitoring based on Zabbix. There are windows showing pipelines and ingestion processes of end products into the Long Term Archive (LTA), Short Term Archive (STA) status.

The scheduler user interface shows information related to observations (times of the observations, whether they were halted and whether there are internal tests or system warm-ups). The correlator user interface shows the log of the correlator (packet loss statistics and log output of observations and tests). Zabbix UI produces HTML plots with averages of CPU, memory usage, load, disk usage in nodes of clusters and within single stations. Zabbix allows authorised users, to configure alarms and set up targeted email alerts for each item being monitored. LOFAR Zabbix monitors 24,000 items, 340 items sampled / second.

The query console (also called MoM, Management Of Measurements) is a UI for managing observations. It contains all the information to configure the parameters for an observation. This is presented as a tree of projects, outlining which images have to be merged, PIs, states, CPUs and other key parameters. MOM also contains detailed information about processing pipelines and verification details for the observation status.

The Navigator windows shows the current status of all available hardware and running software, scheduler, observation management tool (MoM). Navigator supports activities for managing alarms; it supports both geographical and device-based drill-down, enabling the control-room operator system navigation to see devices status and detail operational and

measurement data down to the level of an individual board. The Navigator UI shows an alarm summary at the end of the page. Navigator log levels can be configured; look and feel can be customized; UI can be configured in order to show different panels for different user roles (local approach). Many instances of the Navigator UI can run simultaneously. The navigator is based upon PVSS II, a SCADA (Supervisory Control And Data Acquisition) system. PVSS is used to connect to hardware (or software) devices, acquire the data they produce and use it for their supervision, i.e. to monitor their behaviour and to initialize, configure and operate them.

3.2 MeerKAT

MeerKAT is a mid-frequency "pathfinder" radio telescope being developed as a precursor to the Square Kilometre Array (SKA). The full MeerKAT extends its own precursor project, KAT-7 (Karoo Array Telescope), a seven-dish array which is currently being used as an engineering and science prototype.

MeerKAT User Interfaces

High-level requirements for the MeerKAT user interface were defined by System Engineering and further refined following an iterative development approach, combined with monthly demonstrations of prototype displays to relevant stakeholders to obtain feedback which was included in the development cycle.

The MeerKAT UI[9] main display is a landing page from which more detailed displays can be easily presented. These include health, alarms, observation scheduling, operator control and intervention, historical data, user logs, configuration and theming. This main display consists of a main toolbar (with items like navigation links, current UTC, local solar and sidereal time, alarm counter badges, and the logged-in user's information), a bottom toolbar showing the current system interlock status, version and date information, and a side navigation bar for quick navigation between displays. The selected display content occupies the rest of the space. The landing page acts as a flexible dashboard. The widgets displayed can be customised to suit the needs of an individual authenticated user.

Graphical health displays were developed to assist in fast and efficient fault-finding and for viewing on large, heads-up displays in the operator control rooms. These include the telescope system health overview display with system alarms shown as an overlay at the top right and the main toolbar displayed at the top, the sensor list display showing all the monitor points of a selected resource as a scrollable list, the custom health display in which the user can build custom health views to be then exported as a URI for reuse, pointing displays showing where all the antennas are pointing, in terms of azimuth and elevation as well as right ascension and declination and a special weather display to show the current local weather conditions.

The user can acknowledge and clear alarms via an alarm display, while alarm notifications are shown as counter badges on the main toolbar and as an overlay on each browser tab, visible until the operator acknowledges the alarm. Every time an alarm is received, a different severity related sound is played.

MeerKAT UI[9] designers identified a subset of frameworks and libraries among UI technologies based on industry popularity and peer recommendations. From this shortlist of AngularJS, EmberJS and CS-Studio BOY, they selected AngularJS as their preferred web architecture based on a number of criteria including open source licensing, documentation, active developers and users community, pre-built standard and user-defined widgets, rapid prototyping tools, learning curve, security, Python and testing support having evaluated the chosen technologies by building prototypes using each technology as a front-end design platform.

The MeerKAT user interface implements a client-server architecture. A front-end component provides the client-side functionality and a back-end component provides the server-side functions. Websocket technology is used to enable interactive sessions with real-time flow of data between a browser and server. 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.

The front-end technology stack is built using the AngularJS framework and makes use of Angular Material (Design) and the D3.js (document driven data) JavaScript libraries. The front-end communicates with the back-end via a RESTful API and websockets. The back-end technology stack includes an Nginx HTTP server and reverse proxy and a Redis database

as the Publish/Subscribe provider for the socket level services. Python is the main programming language: the Tornado web framework providing the web servers to support both normal HTTP requests and websockets. A Python package (Katportal) provides all the back-end services for authentication and authorisation, monitoring, control, querying system configuration and to read data from storage.

3.3 ALMA

The Atacama Large Millimeter/submillimeter Array (ALMA) is an astronomical interferometer located in the Atacama desert of northern Chile observing at millimeter and submillimeter wavelengths. It consists of 66 radio antennas, 54 of them with 12 metre diameter dishes, and twelve smaller ones with a diameter of 7 metres. Most of the 12 m antennas can be relocated within the ALMA site to form arrays with different distributions of baseline lengths. More extended arrays will give high spatial resolution, more compact arrays give better sensitivity for extended sources. The telescope is primarily divided into a main array of 12 m antennas, and the smaller Atacama Compact Array (ACA), which consists of the twelve 7 m antennas and four of the 12 m antennas. ALMA has been fully operational since March 2013.

ALMA User Interfaces
Science lifecycle includes different phases such as proposal preparation and management, observation preparation, scheduling and execution, observation monitoring, project tracking, data processing and archiving, research involving archived data access and post-processing.

Different UIs support each of the lifecycle phases as shown in Table 1.

Table 1. ALMA user interfaces and tools

Interface	**Tool**	**Technology**
Proposal Preparation	ALMA Observing Tool	Java desktop “fat client”
Proposal Management	Ph1M (Phase 1 Manager)	Web application
Observation Preparation	ALMA Observing Tool	Java desktop “fat client”
Schedule Planning	Schedule Planning Tool	Web application
Dynamic Scheduling	Dynamic Scheduling Assistant	Java desktop
Observation Monitoring		Java desktop
Telescope Operations	Dashboard/OMC	Web application/Java desktop
Quality Assurance Monitoring	AQUA	
Project Tracking	Project Tracker	Web application

ALMA developed a rich set of frameworks which underpin the design of many of the UIs. ALMA Common Software (ACS) was originally developed to extend CORBA and provide a common infrastructure for all ALMA development. Application code has access to strong common MVC framework for developing UIs and a container component model for storing data in a way that is accessible to all applications. Data is stored in the form of XML documents in an Oracle Database and can be accessed using a common set of services and automatically generated object bindings.

The ALMA Observing Tool is a Java Swing based desktop client that interacts with services hosted on a Tomcat server. Ph1M (Phase 1 Manager), Schedule Planning, Dynamic Scheduling and Observation Execution are all Ajax based web

applications built using the ZK framework, Hibernate and Spring and hosted on Tomcat servers. The Planning and Scheduling applications have a shared business layer. AQUA and Project Tracker are also JavaScript web applications built using the same technologies.

The Operator Management Control (OMC) is a Java-based UI plugin framework while Dashboard is a web application whose front-end technology stack includes Bootstrap, JQuery, D3, AngularJS, the back-end being provided by Django (Python). The front-end communicates with the back-end via JSON messages over a RESTful API.

The case of ALMA OMC: a usage-centered design approach

The operators of the ALMA Observatory monitor and control more than 50 mm/submm radio antennas and their associated instrumentation from an operations site that is separated from this hardware by 35–50 km.

Early commissioning of ALMA used an operator interface implemented with a standard window, icon, menu, pointing device (WIMP) toolkit[10]. Early experience indicated that this paradigm would not scale well as the number of antennas increased towards its full complement. The WIMP model reaches a limit when there is so much information to present to users that they cannot focus on details while maintaining a view from above. On ALMA, operators lost time as they manipulated overlapping or tabbed windows to drill-down to detailed diagnostic data, losing a feeling for their position in the process. The interface designers proposed a solution that replaced the top-level interface with a new multi-scale interface that could take advantage of semantic zooming, dynamic network visualization and other advanced filtering, navigation and visualization features. This solution would simplify the tasks performed by operators and let them concentrate on the real issues at hand rather than continually re-organizing their use of screen space.

To address the challenge, the ALMA software teams adopted a user-centered design approach[11]. For two years, astronomers, operators, software engineers and human-computer interaction researchers have been involved in participatory design workshops, with the aim of designing better user interfaces based on state-of-the-art visualization techniques. This process led to the development of those interface components and to a proposal for the science and operations console setup that was based upon brainstorming sessions, rapid prototyping, joint implementation work involving software engineers and human-computer interaction researchers, feedback collection from a broader range of users, further iterations and testing.

Cycles of design and implementation coupled with active user feedback characterize this project up through deployment. Their design and development method is based on a user-centered design approach involving[11]:

- in situ interviews with operators and astronomers, who are the domain expert end-users of the ALMA Operator Monitoring and Control (OMC) software
- participatory design workshops organized by the HCI researchers, involving a small group of end-users (operators, astronomers, etc.) and software engineers strongly committed to the project
- rapid software prototyping of ideas resulting from these workshops
- demonstration of these prototypes to a larger group of end-users, gathering feedback and new ideas
- high-quality implementation of the ideas identified through this process as software components shipped as OMC plugins, integrated in the general development lifecycle.

This is a model for a user-centered design approach that could be usefully applied to the SKA design activities. Key features are the use of brainstorming, low-fidelity prototyping, feasibility assessment, intensive prototyping, demonstration of prototyped ideas to a larger group of end-users and continually gathering feedback. The cooperative development continued throughout the whole of the development. Teams using virtual machines and prototypes simulating the environment, iterations, testing and debugging tools to present evolving designs and interfaces. Importantly the ALMA OMC experience specifically links the approach adopted with an outcome that:

- provides users with a clear mental map of the instrument
- minimizes window management operations, mouse pointing and clicking to access a given piece of information

- supports users in their primary tasks: monitoring the system, evaluating the quality of observations underway, identifying and possibly anticipating potential problems, troubleshooting them, coordinating the actions of the different actors
- enables fluid and efficient navigation, drilling down into the data with less pointing and clicking

Zoomable User Interfaces (ZUI), coupled with semantic zooming capabilities, have been used in order to pass from the OMC's WIMP original user interface design and implementation to a post WIMP solution.

ZVTM Java toolkit has been used, since it provides off-the-shelf components and features such as smooth panning and zooming, magnification lenses, 9 pie menus and other post-WIMP features that require only limited effort to implement. Implementing such features using WIMP toolkits such as Java Swing is not possible, and usually requires using lower-level drawing APIs such as Java2D, making the development and code maintenance effort much more costly.

The toolkit also makes it easy to create graphical animations of most visual variables that define graphical objects. This helps to reduce the cognitive load on users by providing a high-level of perceptual continuity between the different states of the interface and the system.

In a pie menu the items are placed along the circumference of a circle at equal radial distances from the centre. Pie menus gain over traditional linear menus by reducing target seek time, lowering error rates by fixing the distance factor and increasing the target size. Visual filtering and layout adjustment have been implemented to visualize either all antennas, or the antennas assigned to a specific array rendered with low-contrast colours. Adjacency matrices have been implemented to visualize baselines and treemaps to represent hierarchies. Most user interface toolkits make it relatively straightforward to implement such tree components (e.g. Java Swing's Jtree). Time series and chart visualization have been implemented via ChronoLens framework and QuickLook relying upon JfreeChart.
Coordinated and Multiple Views have been implemented to simultaneously visualize in a coordinated manner (via a synchronization mechanism of interaction events) antenna control, event logs and alarm panels, block diagram of the hardware in each antenna, information about the correlator, software components, and related information.

3.4 ASKAP

The Australian SKA Pathfinder (ASKAP) is a Square Kilometre Array (SKA) pathfinder radio telescope, comprising of 36 small-diameter (12-metre) reflector antennas forming a radio synthesis array with maximum baseline of 6 km, each with a Focal Plane Array consisting of approximately 100 dual-polarised elements operating at centimetre wavelengths and yielding a wide field-of-view (FOV) on the sky of about 30 square degrees. It is located in the radio-quiet desert Midwest region of Western Australia.

ASKAP is designed to be a mostly automated instrument and will spend much of its time carrying out surveys with little human interaction. Its software architecture[12] is based upon EPICS framework for the Telescope Operating System (TOS) and the Internet Communications Engine (ICE) middleware is used for the high-level service bus. All the components of the ASKAP architecture are explicitly loosely coupled, communicating with each other via the service bus. Any component can be developed in any language that has bindings for the Interface Definition Language (IDL) provided by ICE. This currently includes Java, C++ and Python.

ASKAP User Interfaces

TOS and Data Processing are two major subsystems defined in the overall ASKAP System Architecture. TOS links to various key components: the Operator's Display provides the interface for control and monitoring of the instrument by an operator; the Observation Management Portal provides the interface for the input, modification and monitoring of observing programs by the astronomer or science team and the Scheduler User Interface for the scheduling of observations.
Two more interfaces have been implemented. The first is the OSL (Ops Scripting Library) a Python script and reporting framework for engineering, commissioning and initialisation and also command-line tools. It is built on the same interfaces as the observation management portal as such tools are used in commissioning to determine usage patterns for user interaction and batch operations. The second is the engineering UIs developed using EPICS EDM. These were

initially developed in Qt 4.6 + epicsqt library (Australia Synchrotron), which later evolved into Control System Studio (CSS)[13] .

EDM is an interactive UI builder and execution engine maintained by the ORNL EPICS community while Control System Studio is an Eclipse-based collection of tools to monitor and operate large scale control systems, such as the ones in the accelerator community. In particular CSS BOY, which is an Operator Interface (OPI) development and runtime environment for building UIs with drag and drop functionality and instant connection to data. It also allows using JavaScript or Jython to manipulate the UI in a very similar way to using JavaScript in HTML.
In BOY, OPI Editor is a What You See Is What You Get (WYSIWYG) editor. The OPI Runtime works in a similar way to modern web browsers. The user of the system can display the OPIs either in tabs, windows or views and navigate OPIs forward or backward. An OPI is a regular XML file that can be edited in OPI editor or text editor and run in OPI Runtime. No compilation is needed. BOY is a set of Eclipse plugins written in Java and has been tested to run well on Windows, Unix and X OS platforms.

4. USER CENTERED DESIGN ACTIVITIES FOR SKA UIs

Considering the lessons learned by SKA precursors and the inherent complexity of SKA systems and interactions, a user-centered design approach has been adopted in a pilot project deemed particularly significant among all SKA TM user interfaces; in the following part of the paper we give an overview of these activities in the context of the design of a UI for **alarm management**.

User interface design is an iterative process that involves close liaisons between users and designers. It covers topics ranging from usage-centered design during analysis and design, through to testing and validation in later application lifecycle phases. This involves deriving shared principles**,** practices**,** methods and the tools to support the effective participation of different development teams in the design of a well-integrated UI to the SKA systems.
Currently the three core activities in this process are:

- User and task analysis: understanding what the users will do with the system
- System prototyping: developing a series of prototypes for experimentation
- Interface evaluation: allowing the users to experiment and explore these prototypes

The TM design activity relies on this usability prototyping as a core part of defining the design needs for the wider system. It is important to have early users' feedback in the software design and development life cycle to elicit new requirements, validate existing requirements, and highlight possible critical interactions.

To understanding the possible **operational context** of alarm management for SKA users we collected information from one of the precursors (LOFAR) and conducted several *structured interviews* during a field trip. Over the three days a team of three interviewers interviewed seven key staff to gather information about how alarm management is carried out, what kind of UIs exist and how they are used. The interviews were initially guided by an underlying structure of concepts and several hypotheses that interviewers had formed based on existing sources of information before meeting the interviewees.

Hand-written notes were later on transferred to a digital medium (a spreadsheet) and printed on cards attached to *Post-It* notes. After removing duplicates and obvious mistakes, the remaining 550 notes were gradually added to a wall display allowing time to consider how each set added changed the overall patterns. The notes were repeatedly clustered and re-clustered by the interviewers. During this activity some notes were clarified, through discussion between the interviewers, with new notes created to represent intermediate summaries. In the end these clusters were transferred to a digital medium (a *mind map*), which contained 250 categories (i.e. intermediate concepts, interpretations and summaries), covering these topics: structure of the telescope, types of alarms, alarm policies, procedures followed by operators and on-duty scientists, UIs used by operators, broad usability problems of UIs, lessons learnt by the original stakeholders. This is an example of the KJ method (or *affinity diagrams*): a qualitative analysis method aimed at interpreting and comparing facts collected during interviews[14]. See Figures 2 and 3.

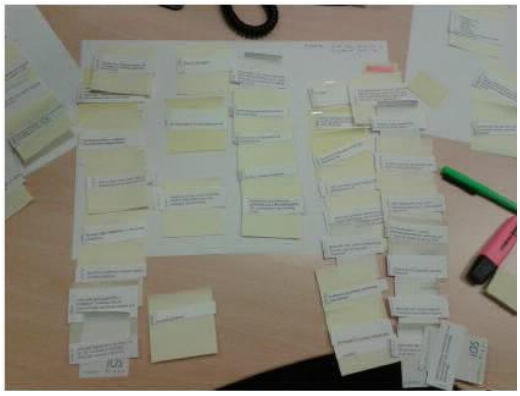

Figure 2: Example of some of the Post-It notes that were manipulated when creating the affinity diagram.

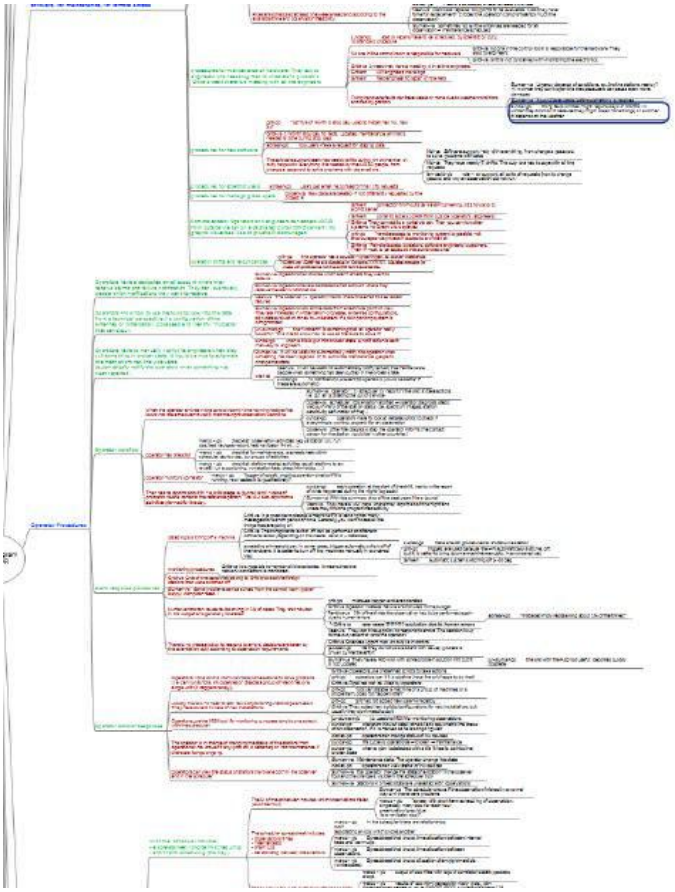

Figure 3: A snapshot of the mind map that was produced for the affinity diagram, highlighting the width and depth of the classification.

The **analysis of the users** of the system is a crucial point in the application of the UCD approach, because it helps to address the user characteristics that can impact on the design of the interface. Its aim is to describe what are the activities that the user performs to achieve his/her goals and how they would use the available technology to do it.

Structured interviews of users of similar systems, conducted both at LOFAR and at Sincrotrone, resulted in the definition of a set of **user roles** that take into account both the context within which the role is played and the characteristics of the performance. This depicts the user's distinctive behavior, values and knowledge revealing the nature of his/her work in relation to the system that has to be designed.
In general, information collected in a profile describing a user role can be used to derive design objectives and to validate a user interface. For example, a profile describes the **operational model** and risks under which the UI will be used; risks refer to what is at stake if the user and the system fail in the correct completion of tasks.
Design objectives that can be derived from given operational models can be: operator speed and accuracy, ease of learning and of retention, ease of interpreting high volume or high complexity of information and similar measures. If the operational model is such that users are highly trained and interaction is very frequent and prolonged, efficiency and accuracy become more critical factors than learnability and retention.
The **profile** of the LOFAR control room operator is shown in the following as an example of the kind of information that can be obtained performing user analysis. We followed the excellent suggestions of Constantine and Lockwood[1] of representing each role according to the Context in which it is played, the Characteristics of its performance and the Criteria that the design should meet in order to support the role to successfully reach its objectives.

The **Context** in which the Operator role is played includes the description of the physical environment where the Operator works, his/her social situation and his/her background and knowledge of the system. The environment is a quite silent control room with a high rate of visual inputs where the Operator monitors the observation progress, the telescope health, and deals with alarms. The social situation of the role is based on the collaboration with the other member on duty during the same shift and eventually with people working for third parties (such as telecom providers), with reference people at the station sites and at universities where data is archived. The Operator's background expressed in term of training, education or experience, should include engineering knowledge equivalent to a higher secondary education, followed by practical experience of engineering work, or an engineering-based degree. Moreover, prior knowledge of Linux and proven aptitude with technology associated with an analytical mindset are important. The Operator is usually an expert user of the system and is expected to be highly proficient in alarm management and to have some scripting signal processing skills.
The **Characteristics** of the performances highlight the Operator's emotional state and the uniqueness of his/her interaction with the system. The Operator performs high responsibility tasks devoted to identifying situations that can potentially affect the quality of the observations. During his/her shift the Operator continuously monitors a huge amount of information that describes the state of the system. In case of critical situation the Operator has to collaborate with the scientist on duty helping in the decision of the reaction method.
Given this description of the Context and of the Characteristics of the role the Operator has to trust the information he/she receives. They need the ability to configure the UI according to individual preferences and for the shown data to be comprehensible. The most essential functional facilities, or criteria to support the role and accomplish these objectives is: a trustworthy and an efficient interaction (for example not needing to perform many clicks to get to the desired data/action).

Findings so far collected, classified and interpreted, were "projected" to the SKA situation. Differences between SKA and the examined precursors were highlighted and considered in other design activities. These were the definition of **scenarios**[3] and definition of **essential use cases**[1]. The purpose is to hypothesize tasks that users of SKA are likely to perform, and conceive functionalities of the UI that could support them.

A scenario is a brief narrative description of what a typical user (for example an operator) would do with an hypothetical UI of SKA in order to carry out some tasks (such as responding to some alarm). We adopted two types of scenarios: a **usage scenario**, which does not refer to specific features of the UI, but simply highlights that the UI would support some activity; the second type of scenario, an **interaction scenario,** emphasizes the specific features that the UI offers to the user. For example, a usage scenario could say that:

> The UI tells the operator that [the alarm] is a serious problem that could require putting the antenna out of service.

While an interaction scenario would say that:

The operator types “o” “d” (“open detailed view”) to open the Detailed Alarm View and in the Procedures field they see that this type of failure could require the shutdown of the antenna.

Essential use cases are used to represent goals that users might want to achieve. More specific goals may be derived from more general ones. In the use case map, essential use cases are linked through generalization, inclusion and extension relationships[1]. This coarse-grained representation can be used as a task model (i.e. a representation of how designers expect that users would carry out relatively complex activities).
As an example, the task CheckHealthStatus could be represented as in Figure 4 which shows that to check the health status of the telescope an operator could perform the task ExamineSubsystemsWithKnownProblems or AnalizeAlarms. AnalizeAlarms could in turn be a generalization of VerifyPresenceOfActiveAlarms which includes ViewAlarmSummary.

Essential use cases differ from traditional UML use cases because they are not structured in main and alternative scenarios (beware that the meaning of the word “scenario” in the context of UML use cases is different from what was meant above, with usage or interaction scenarios), but their detailed description is given in terms of purpose (why a particular user is involved in that use case) and in terms of user intents and corresponding system responsibilities. See Constantine and Lockwood[1] for more details.

Such an essential use case model allowed us to refine SKA requirements - which are mainly requirements from the point of view of what the system has to do - into UI requirements - what human tasks should the UI support.

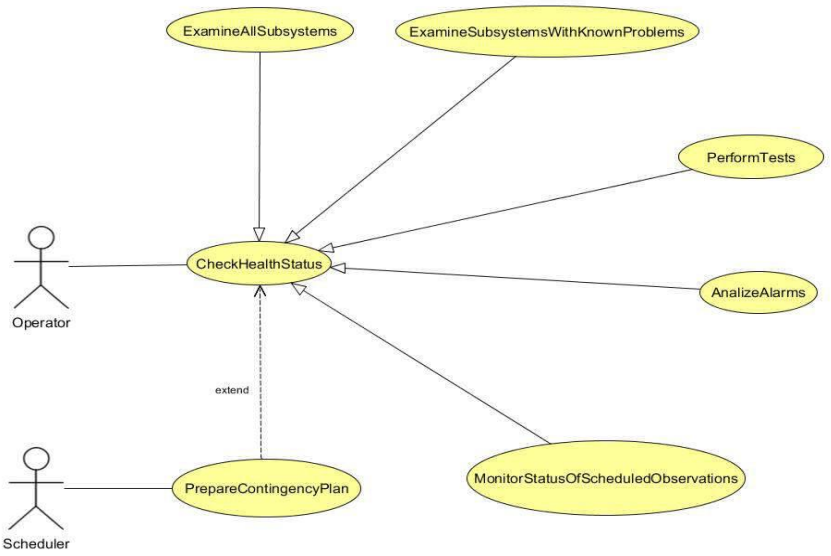

Figure 4 : CheckHealthStatus Essential Use Case

Sketching and storyboarding are techniques to materialize design ideas in such a way that any stakeholder, regardless of his/her design experience, is able to decide if a given UI is appropriate or not for some task[2]. Sketches are evocative tools: they leave lots of details to imagination, and are indeed very valuable for eliciting new requirements from the feedback that users or other stakeholders provide. Furthermore, because a sketch can be very rough, it is usually relatively cheap to produce and to revise, and supports a parallel design process. A storyboard is essentially a scenario described through sketches.
Figure 5 shows one of the sketches that were produced during the design process of the UI. It shows a panel dedicated to alarm handling. The panel consists of a set of highly configurable subpanels that are intended to be resizable, dockable and linkable between each other. Their position can be changed by the user so that they may visualize the desired information in the way they find most comprehensible or easiest to work with. The main part of the panel can be divided

into three sections, one dedicated to the filtering options, one showing the Alarm List with the list of the alarms raised by the system and one, at the bottom, called Alarm Summary.

The Alarm List panel can be linked to the Filter panel so that the former contains only alarms that satisfy the filtering criteria. This feature can also be disabled. The Alarm Summary shows only the reported unacknowledged alarms in decreasing time order. Buttons on the right side of the tables allow the user the ability to perform actions on the alarms, i.e. acknowledge, mute, shelve, etc. Overlaid to the main panel, the detailed description of the selected alarm (highlighted with light-blue contour) is shown. It consists of three parts, one with extensive information on the alarm, including a description of the procedures to be adopted to solve the problem, the second with the available actions that can be performed both on the alarm and on the sub-system and the third one with a summary of the status of the observations in which the alarmed sub-system is involved. The UI should support keyboard navigation and shortcuts.

A storyboard can be used as a discussion document, with the aim of eliciting opinions of other stakeholders. We decided to implement these sketches with one of the several existing tools (WireframeSketcher). And with them we explored several design ideas such as what kind of filters to use. This was an important decision, since focusing on a manageable subset of data was one of the clearly identified challenges for SKA. Design tools such as the one mentioned could also used to implement limited interactive features, such as the definition of clickable hot spots that are linked to other sketches to illustrate interactively the intended dynamics of the UI.

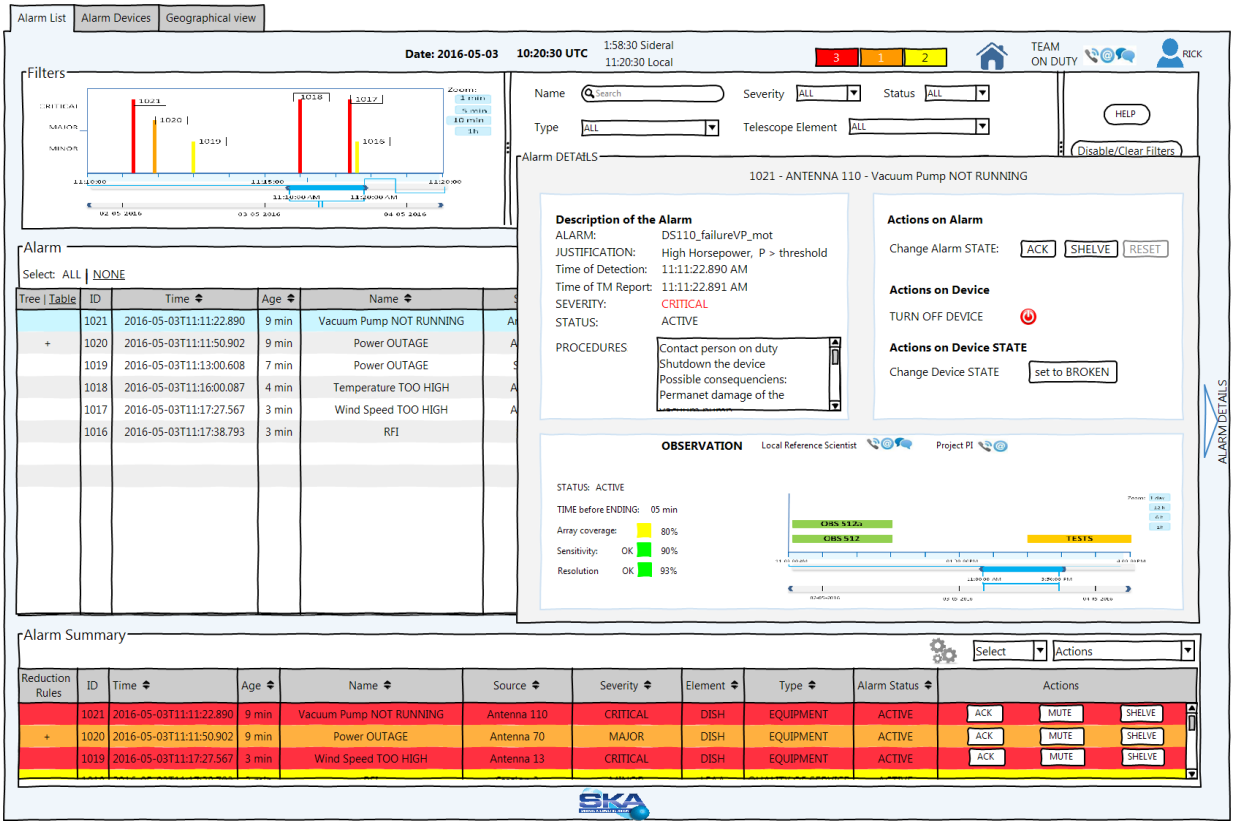

Figure 5: A sketch of a screen of the alarm management UI showing the detailed view of an alarm

Formative user testing is a method, based on the think-aloud protocol, that is often used with sketches and storyboards to better understand how suitable a design idea is, and possibly help to form a better one. The method is based on sessions with a single user at the time, who is asked to describe how he or she would perform a task when given the UI depicted in a sketch. Behind the scenes, a facilitator acts as a computer, and switches to another sketch when the user

pretends to act in a certain way (such as pressing the acknowledge button). This is often called the "Wizard of Oz" technique.

The real value of the method comes from application of the think-aloud protocol: the user is asked to explain how the screen is interpreted, what cues lead to what conclusions, why, what actions are available, what actions should be available, etc. A skilled facilitator is able to acquire a lot of information about the mental model of that particular user trying to do a particular task.
When a formative test with a few (i.e. 3-5) users is carried out, usually a lot of information is acquired which allows deep revisions of the UI (in terms of look and feel, in terms of labels, in terms of interaction structure).

The first formative tests on SKA storyboards are scheduled for June 2016.

5. TECHNOLOGICAL PROTOTYPING

Within the TM UI work-stream, prototyping is also a key tool for the evaluation of technologies. Based on the documented lessons learned by precursors (MeerKAT, ASKAP, ALMA, LOFAR) and specific UI analysis conducted by precursor sites (LOFAR), a set UI tools has been selected and they are being analyzed against SKA TM requirements. The set includes TANGO tools (e.g. Taurus and Sardana) and general UI frameworks (such as AngularJS, Django, PyQT, PyTango, and TurboGears).

The TANGO UI tools have been examined and can be divided into Desktop TANGO UI tools and those that provide a web-based interface to a TANGO environment. The options for TANGO desktop development includes ATK based on Java Swing, QTango based on C++ and Qt and Taurus based upon Python and PyQt. These all fulfill the basic SKA.TM requirements and could be used to implement desktop UIs. However SKA.TM envisages the use of web user interfaces as well and here there is not a well-established solution. TANGO actually offers two tool to implement web interfaces: Canone, based on PHP and Giga. The Canone project was mostly abandoned in 2007, while Giga is still under development and results to be not mature enough to be used in the production. This means that at the moment adapting an established web UI framework appears to be the best option for these interfaces unless additional resources are invested in further development.

The focus of the prototyping at this stage is not on developing a particular interface but rather to consider for each selected frameworks how the tools and facilities might constrain the design of the UIs. At this early stage of software design we want to avoid the risk that the technology selection, rather than the needs of our customers would limit the interfaces developed.

The most appropriate design for any interactive system can not typically be achieved without some degree of iteration. SKA is in the privileged position that good precursor systems exist which tackle some similar operational issues. However many of these control systems operate at a much smaller scale than is envisaged for SKA in full operation. It is extremely difficult, if not impossible, for users to explain what they want from a system they have not seen in order to operate an observatory that does not yet exist. So we would expect that the first design of many of the UIs will identify numerous areas where having seen the interface, the users are in a far better position to describe the improvements or changes required. Based on the experience of observatories such as LOFAR and ALMA this may well take several cycles of iteration and evolution.
From previous studies, there are a small number of technological features that determine the likely success of this type of iterative development:

- An architecture that allows a loosely-coupled clean separation of concerns so that it is easy to replace an interface or an interface component with a more usable functional equivalent, or to adapt to technological change.
- Favouring "narrow" interfaces with clearly defined responsibilities, supported by an architecture that isolates them from wider change yet, makes it easy for them to interact when needed.
- A common framework or set of framework choices that makes it easy for knowledge and components to be shared between teams working on different interfaces.

- Consistency between interfaces so that users spend as little time as possible learning new skills or having to switch between different models for the way interfaces work.

Specifically in the field of Control Room interface redevelopment for a large radio telescope, there are a number of excellent papers produced over the last few years relating to the redesign of the ALMA Operations Monitoring and Control (OMC) system[11,15] mentioned earlier. The OMC paper also provides a useful description of the types of HCI interface elements that they had found useful. We used this to identify some sample experimental tasks for this stage of SKA UI prototyping.

- **Zoomable map/contextual pie chart**: in which information is presented spatially in geographical or topographical arrangement showing a highly summarised perspective. At the highest level this may simply be a colour indication of overall health. As the user zooms into the interface and the element of interest becomes larger more information is displayed. Clicking on an element of interest displays a set of options centered on the object itself.

- **Treemap:** a hierarchy which represents as a 2D space using colour to indicate any issues. Selecting a component 'zooms' focus on that area of the grid and reveals the next level of the hierarchy.

- **Efficient multi-scale and multi-focus visualization**: the ability to combine graphical elements in a navigable way. This includes, stacking and overlaying plots together (for instance to see performance of a group of antennas), displaying plots over other display elements (plotting observation targets over a scaled image of a particular source), or to showing the position of key spectral lines against and observing bands. For series data this includes the ability to zoom in to look in more depth at a narrow part of a time series and the ability to filter data to remove specific elements that are not of interest.

This set of interface elements is similar to some of the interface elements developed by the MeerKAT SKA Pathfinder team.[9] Both the ALMA Dashboard[15] and MeerKAT solutions used very similar technology choices. - a server-based architecture supporting a browser-based user interface developed using a JavaScript framework. We were interested to answer the following questions:

- Were there any strong technical or usability reasons for selecting one framework over another?
- Was it possible to develop the necessary rich graphical and interactive interfaces using a purely browser-based solution?

So far the time spent on this aspect of the prototyping has been limited. From developing basic UI elements similar to the types of controls described above using React, Angular/Angular 2 frameworks and combining these with the Data Driven Documents (D3.js) visualisation libraries, the following seem reasonable conclusions:

- JavaScript frameworks are a useful and practical way to develop the front-end of web-based applications. We would still need to do further work to be certain that they offer a fully scalable solution.
- There are no strong arguments currently for adopting a particular JavaScript framework. Having tried several frameworks, each is slightly different but they all support a separation of concerns into Model, View and Control responsibilities, and accepted patterns for developing encapsulated and sharable UI components.
- The D3 library copes well with large numbers of graphical elements and is flexible in the types of controls and displays that can be created. The ability to create and manipulate richly interactive displays is possible in a web-based solution.
- If a web-based approach is adopted for user interfaces outside the control room, there are the two areas where non browser-based solutions still offer some advantages. These are the ability to store data locally on the client, and to perform intensive processing locally to the user.
- JSON structures exposed via a ReSTful interface offer a solid option for structured messages. Websockets offer an equally practical approach for high capacity or near real-time data feeds. This approach is compatible with most current technologies for developing UIs. The best way of managing socket subscriptions and testing these

interfaces at anything close to the expected scale of SKA traffic is something that may be explored more fully as part of the ongoing alarms prototyping work.

Another strand of the prototyping work was to review a possible architectural approach for providing interactivity between UIs developed using differing technologies. Was it practical to coordinate activity between interfaces and maintain a coherent focus?

A small test prototype was created based on PyTango with a simple ZeroMQ messaging hub. It tested the basic UI Architecture principle that a central client-based hub was a practical option, and could support the type of client to client communication needed for managing opening and closing of Uis and coordinated change of focus.

Given a similar existing architecture is already widely adopted by a range of astronomy applications. Extending the prototype to explore using the Simple Message Application Protocol (SAMP)[16] for this type of coordination would be be a useful next step.

More prototyping activity is focusing on the alarm-handling functionality, both applying existing specific Tango systems and new UI proposals as outcome from the preceding precursors analysis.

On the practical aspects of the prototyping, the technology decisions all seem reasonable. The Tango framework and tools support the types of basic control interfaces that are currently used at both radio telescopes and within high energy physics experiments. The current technologies available for developing front-end web solutions are capable of supporting the types of sophisticated displays envisaged, at least based on an analysis of the needs of precursors.

6. CONCLUSIONS

Considering the lessons learned from precursors, it is clear that proper analysis and design activities are needed. These should be focused on what combination of UIs will best support SKA users (operators, scientists on duty, schedulers, PIs, etc) and based on usage-centered development practices. Such a usage-centered development approach can mitigate product risks (i.e. those concerning with what will be developed and whether it will be the *right* solution) and consists of several key steps:

1. To elicit new requirements in terms of activities that have to be supported (through techniques such as user analysis, precursors analysis, affinity diagrams, brainstorming and focus group sessions among stakeholders)
2. To study the tasks that SKA users would have to carry out (through task analysis, use case modeling, scenario definition, sketching and storyboarding)
3. To design and validate appropriate UIs (through refinement of sketches, storyboards and low-fidelity prototypes and user testing).

On the one hand this activity will help refine requirements. To the extent that the UI is sketched in sufficient detail, the UI can also be used in exploratory usability investigations (also involving end-users) to validate such requirements. On the other hand, the activity will provide enough details to clearly understand assumed or unexplored architectural requirements for the underlying system, for instance that the operator while managing alarms might need to be able see observations scheduled in a view filtered by specific resources.

The output from such a process will be a far better understanding of the user's detailed requirements than would otherwise be available. A set of artifacts that clearly describe these needs and requirements for all the users of TM is an important input for defining or the architecture of UIs for the TM. This work is necessary for the implementation level, investigation into which frameworks can best achieve these needs. It also provides the context for investigating, outlining the advantages and disadvantages of possible frameworks and making informed decisions.
For SKA, we are still very much at the start of this process. Already we can see that a consistent, structured approach to keeping the user experience at the heart of our design efforts is the best way to guarantee that this exceptional

observatory develops and maintains the interfaces that will guarantee an equally exceptional experience to all its users throughout the whole of its operational life.

REFERENCES

[1] Constantine, L. and Lockwood, L., [Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design], Addison-Wesley Professional, (1999).

[2] Greenberg S., Carpendale S., Marquardt N. and Buxton B., [Sketching User Experiences: The Workbook], Morgan Kaufmann, (2011).

[3] Rosson M.B. and Carroll J.M., [Usability Engineering: Scenario-Based Development of Human-Computer Interaction], Morgan Kaufmann, (2001).

[4] Holzblatt K., Wendell J. and S. Wood, [Rapid Contextual Design: A How-to Guide to Key Techniques for User-Centered Design], Morgan Kaufmann, (2004)

[5] Rubin J. and Chiswell D., [Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests], Wiley, (2008).

[6] Preece J., Sharp H. and Rogers Y., [Interaction Design: Beyond Human-Computer Interaction], Wiley, (2015).

[7] Hjalmarsson, A., Gustafsson E., Cronholm S., "Exploring the Use of Personas in User-Centered Design of Web-based e-services", Proc. iConference 2015, (2015).

[8] Van Haarlem M.P. et al., "LOFAR: The LOw-Frequency ARray", Astronomy & Astrophysics, vol. 556, A2 (2013).

[9] Alberts M., Joubert F., "The MeerKAT Graphical User Interface Technology Stack", Proc. ICALEPCS 2015 User Interfaces and Tools, 1134-1137 (2015).

[10] Schwarz J., Pietriga E., Schilling M. and Grosbol P., "Goodbye to WIMPs: A Scalable Interface for ALMA Operations", Proc. Astronomical Data Analysis Software and Systems XX ASP Conference, Vol. 442, 247-250 (2011).

[11] Pietriga E., Cubaud P., Schwarz J., Primet R., Schilling M., Barkats D., Barrios E., Vila Vilaro B., "Interaction Design Challenges and Solutions for ALMA Operations Monitoring and Control", Proc. SPIE 8451 (2012).

[12] Guzman J.C., Humphreys B., "The Australian SKA Pathfinder (ASKAP) Software Architecture", Proc. SPIE 7740 77401J-1 (2010).

[13] Guzman J.C., "Status of the ASKAP Monitoring and Control System", Proc. ICALEPCS 2011 Status reports 1349-1352 (2011).

[14] Lucero A., "Using affinity diagrams to evaluate interactive prototypes", Proc. INTERACT 2015, Lecture Notes Computer Science vol. 9297, 231-248 (2015).

[15] Pietriga E., Filippi G., Véliz L., del Campo F. and Ibsen J., "A web-based dashboard for the high-level monitoring of ALMA", Proc. SPIE 9152 91521B (2014).

[16] Taylor M. B., Boch T., Taylor J., "SAMP, the Simple Application Messaging Protocol: Letting applications talk to each other", Astronomy and Computing vol.11(B), 81-90 (2015).