The SKA dish local monitoring and control system user interface

Alessandro Marassi[a], Marco Brambilla[b], Simone Riggi[c], Adriano Ingallinera[c],Corrado Trigilio[c], Paolo Di Marcantonio[a]
[a]INAF - Osservatorio Astronomico di Trieste (Italy); [b]Politecnico di Milano (Italy); [c]INAF - Osservatorio Astrofisico di Catania (Italy)

ABSTRACT

The Square Kilometre Array (SKA) project is responsible for developing the SKA Observatory, the world's largest radiotelescope ever built: eventually two arrays of radio antennas - SKA1-Mid and SKA1-Low - will be installed in the South Africa's Karoo region and Western Australia's Murchison Shire, each covering a different range of radio frequencies. In particular SKA1-Mid array will comprise 133 15m diameter dish antennas observing in the 350 MHz-14 GHz range, each locally managed by a Local Monitoring and Control (LMC) system and remotely orchestrated by the SKA Telescope Manager (TM) system. Dish LMC will provide a Graphical User Interface (GUI) to be used for monitoring and Dish control in standalone mode for testing, TM simulation, integration, commissioning and maintenance. This paper gives a status update of the LMC GUI design involving users and tasks analysis, system prototyping, interface evaluation, provides details on the GUI prototypes being developed and technological choices and discuss key challenges in the LMC UI architecture, as well as our approaches to addressing them. In the GUI design task we have adopted a Usage-Centered Design (UCD) approach based on the early involvement of users whose feedback is being iteratively considered in analysis phases, as well as in design and evaluation. An IFML based user interaction modeling approach has been adopted.

Keywords: human-machine interface, user interface, usage centered design, modeling

1. INTRODUCTION

SKA-MID1 Dish array is composed of 15-m Gregorian offset antennas (see Figure 1) with a feed-down configuration equipped with wide-band single pixel feeds (SPFs) for the bands 1 (0.35-1.05 GHz), 2 (0.95-1.76 GHz) and 5 (4.6-13.8 GHz) of SKA frequency. The array will consist of 133 dishes plus the 64 MeerKAT dishes, arranged in a dense core with quasi-random distribution, and spiral arms going out to create the long baselines that go up to 200km.

Four sub-elements can be identified in the SKA-Mid1 dish element: the *Dish Structure* (DS), the *Single Pixel Feed* (SPF), the *Receiver* (Rx) and the *Local Monitoring and Control* (LMC).

The *Dish structure* features the following components: an offset Gregorian reflector system with a feed-down configuration to optimise system noise performance, a fan-type feed indexer at the focal position which allows for changing between the 5 frequency bands by moving the appropriate feed into position, a pedestal providing a RFI shielded cabinet for housing digital electronics and computing equipment hosting other sub-elements' controllers, hardware for antenna movement control and monitoring (Antenna Control Unit or ACU), power distribution to all sub-elements, networking equipment, lightning protection and earthing, cooling ventilation for all the equipment mounted in the RFI shielded compartment itself.

Single Pixel Feed receivers include feed packages for the bands 1 (0.35-1.05 GHz), 2 (0.95-1.76 GHz) and 5 (4.6-13.8 GHz) of SKA frequency, three cryostat assemblies (respectively for band 1, band 2 and band 3,4,5) housing each a Gifford McMahon (GM) cryogenic cooler to cool the LNAs at a set point of approximately 20K, a second amplification stage and a calibration noise source, both temperature stabilised inside the vacuum, a common shared Helium System, a Vacuum System and a SPF controller, i.e. a single controller located in the pedestal which controls and monitors all three feed packages, helium system and vacuum system, and interfaces with the Dish LMC for external control and monitoring.

The *Receiver* includes the following components: RF over fibre transport to the antenna pedestal where the digitisers are located, Digitizers performing some RF conditioning (filtering and level control), digitisation, packetizing and

transmission to SKA Central Signal Processor (CSP), the Master Clock timer which receives time and frequency reference inputs externally and generates timing and frequency references and a Central controller that acts as single point of control and monitoring to the LMC sub-element.

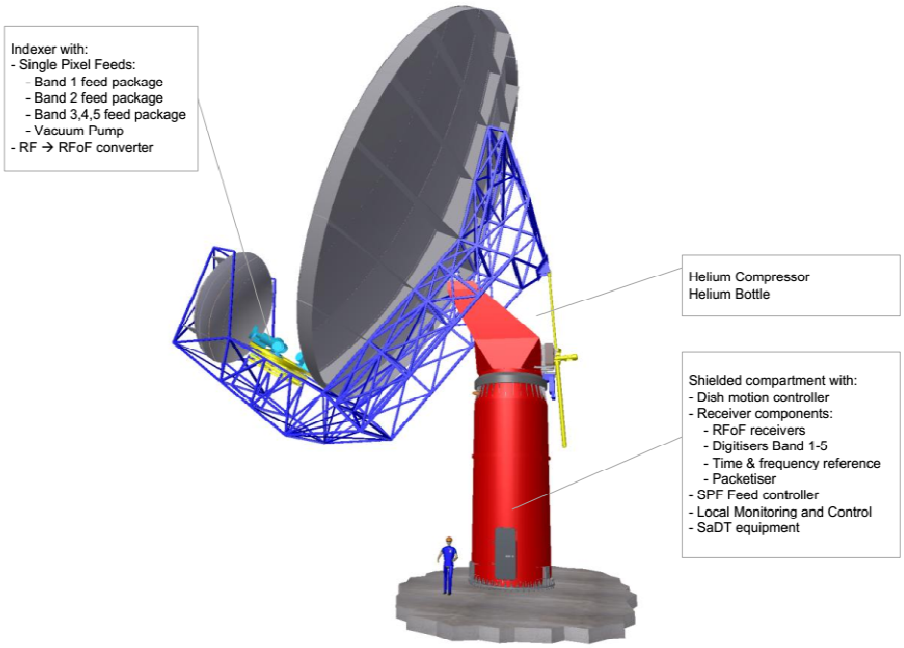


Figure 1: SKA DISH overview

Local Monitor and Control is the subsystem for each dish antenna that deals with the management, monitoring and control of the operation as orchestrated by the Telescope Manager (TM). It consists of a commercial off the shelf controller that serves as a single point of entry for all control and monitoring messages to the outside (see figure 2). Besides configuring the static configurations of the various sub-elements, it also relays the real-time pointing control and applies local pointing corrections. For the monitoring, it aggregates and filters monitoring data as set up from the external (central) controller. The LMC allows for a drill-down capability for maintainers to access detailed diagnostic information of sub-elements on request. The LMC implements also a circular buffer of detailed monitoring information that can be downloaded remotely for diagnostics purposes after a system failure.

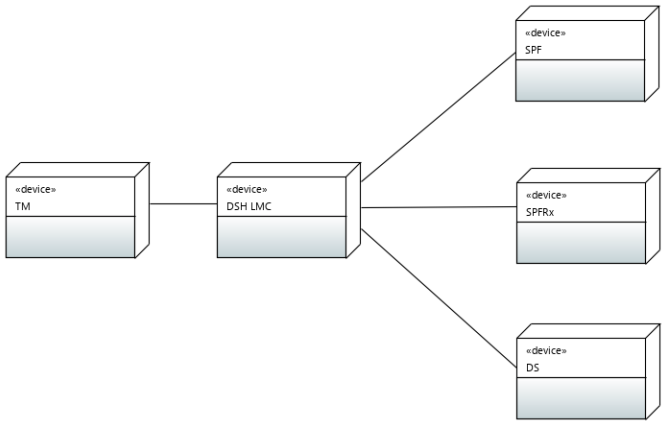


Figure 2: SKA DISH sub-elements deployment diagram

2. SKA DISH USER INTERFACES

Two user interface types (with different user roles) are assumed for the Dish element: Engineering interfaces, to be used by engineers/maintainers for test, diagnostic, maintenance of Dish sub-elements (see Figure 3) and a Navigation/Operation interface to be used by the control room operator/scientist on-duty for operations and observation supervision purposes.

LMC is in charge of the design and implementation of its own engineering interface, which should also enable the access to the other sub-elements UIs, while the navigation interface is TM direct responsibility.

Engineering interfaces will be accessible either directly from LMC (to be connected with keyboard/mouse and a screen) as desktop application or remotely (from the control room or via some other application (e.g. a TM simulator)) using a tunnelling mechanism (e.g. SSH tunnelling and/or Tango Access Control) and are supposed to support authorization and authentication for specific interactions like changing some configuration parameters or setting mode for a sub-element.

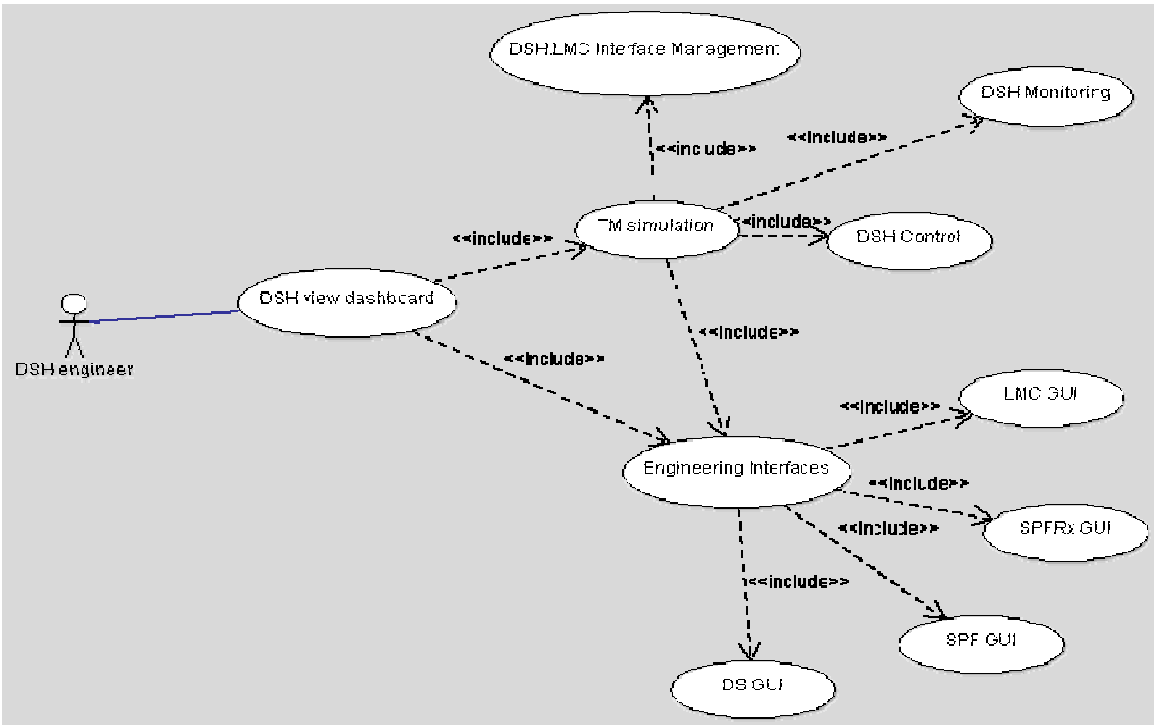


Figure 3: Dish Engineering Interfaces

LMC will provide GUIs (see Figure 3) to be used for testing and DISH control in stand-alone mode for testing, commissioning and maintenance. A TM simulator to be used via GUI is envisaged too.

A main LMC engineering interface is here assumed, offering the following functionalities:

- basic DSH control & monitoring
- set-up, control and testing the integrated (or not/partially integrated) Dish
- launch Element specific UI/tools for configuration, debugging, testing and diagnostics
- health monitoring
- alarm management
- lifecycle support, maintenance
- provide direct access to monitoring data by external operators (engineers) in case of TM failure
- navigation to other DSH sub-elements engineering interfaces via a tunnelling mechanism

In particular, DISH LMC engineering interface will include the control of Dish States and Modes and will also expose an aggregated health monitoring capability. Such GUI will provide to the test engineer/operator the capability of complete configuration, control and visualization of the Dish parameters in the absence of TM during integrated DSH stand-alone site integration and verification. It will be capable of managing both TM (or simulated TM) and stand-alone Dish operation. TM (or simulated TM) operation will be performed according to the TM-LMC Interface Control Document.

3. BASIC CONCEPTS AND METHODS

3.1 Usability and Accessibility

The ISO 9241 standard Ergonomics of Human-System Interaction (ISO, 2008) defines *usability* as "the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use", specifying:

- *Effectiveness*: the accuracy and completeness with which users achieve specified goals.
- *Efficiency*: the resources expended in relation to the accuracy and completeness with which users achieve goals.
- *Satisfaction*: the comfort and acceptability of use

The ISO 9241 standard Ergonomics of Human-System Interaction (ISO, 2008) focuses on important, but rather difficult to measure goals: effectiveness, efficiency, and satisfaction. For a practical evaluation heuristics may be used or direct usability measures, such as: *Time to learn, Speed of performance, Rate of errors by users, Retention over time, Subjective satisfaction*.

Accessibility is the degree to which a product, device, service, or environment is available to as many people as possible. Accessibility can be viewed as the "ability to access" and benefit from some system or entity. The concept often focuses on people with disabilities or special needs (such as the Convention on the Rights of Persons with Disabilities) and their right of access, enabling the use of assistive technology.

3.2 Usage-Centered Design

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, in order to achieve high usability standard, *feedback offered by users* is to be considered in analysis phases, as well as in design and evaluation. The design process has to be *iterative* also because building a usable UI requires all those involved in its construction to understand, and actually conceive, the mental model that users will have of the application. Each iteration is based on design-prototype-evaluate activities, whereas the *evaluation is based on usability* criteria.

In order to be effective (i.e. 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].

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]. A storyboard is essentially a scenario described through sketches and may be implemented also by using interactive sketches. Interactive sketches and storyboards may be used as throw-away mockup UI prototypes (Figure 3) and discussion documents for brainstorming with the aim of eliciting opinions of stakeholders and users.

3.3 Model-driven Development of User Interfaces

A complementary tool that can be used in user-centered design together with storyboarding is *conceptual modeling* of the user interactions. Interaction design focuses on expressing the content, user interaction, and control behavior of the front-end of software applications through visual diagrams that represent the navigation paths of the user. Front-end and user interaction development is a costly and risky process, where manual coding is the predominant development approach, reuse of design artifacts is low, and cross-platform portability remains difficult. The availability of a platform independent modeling language targeting this aspect of software development can bring several benefits to the development process of application front-ends: it permits the formal specification of the different perspectives of the front-end (content, interface composition, interaction and navigation options, and connection with the business logic and

the presentation); it separates the stakeholder concerns by isolating the specification of the front-end from its implementation-specific issues; it improves the development process, by fostering the separation of concerns in the user interaction design, thus granting the maximum efficiency to all the different developer roles; it enables the communication of interface and interaction design to non-technical stakeholders, permitting early validation of requirements.

Various languages, approaches and tools exist to support this task. Among them, we select the Interaction Flow Modeling Language (IFML)[9], an international standard proposed by the OMG (*www.ifml.org*). IFML has been designed for expressing the content, user interaction and control behaviour of the front-end of applications belonging to the following domains:

- Traditional, HTML+HTTP based Web applications
- Rich Internet Applications, as supported by the forthcoming HTML 5 standard
- Mobile applications
- Client-server applications
- Desktop applications
- Embedded Human Machine Interfaces for control applications
- Multichannel and context-aware applications

It's worth noting again that IFML does not cover the modeling of the presentation issues (e.g., layout, style and look-and-feel) of an application front-end and does not cater for the specification of bi-dimensional and tri-dimensional computer based graphics, videogames, and other highly interactive applications. It is mainly aimed at business-oriented, data-intensive applications instead.

IFML integrates with other mainstream software modeling languages like UML and BPMN, and covers the following design perspectives:

- The view structure specification, which consists of the definition of view containers, their nesting relationships, their visibility, and their reachability;
- The view content specification, which consists of the definition of view components, i.e., content and data entry elements contained within view containers;
- The events specification, which consists of the definition of events that may affect the state of the user interface. Events can be produced by the user's interaction, by the application, or by an external system;
- The event transition specification, which consists of the definition of the effect of an event on the user interface;
- The parameter binding specification, which consists of the definition of the input-output dependencies between view components and between view components and actions; and
- The reference to actions triggered by the user's events.

Furthermore, IFML concepts can be stereotyped to describe more precise behaviours. For instance, one could define specific stereotypes for describing web pages (a specific kind of view container); forms, lists and details (specific kinds of view component); submission or selection events; and so on. For instance, several extensions exist covering Web[16], mobile[15] and IoT-based[17] applications.

Interaction modeling through IFML is instrumental to provide a clear conceptual view of the user interfaces, as well as to provide a formal representation of it, which can lead to automatic validation and quick prototype generation on the target platform of choice. Indeed, some tools exist that check the validity of IFML models and compute their properties, based on Petri-Nets or LTL formalization.

Various implementation frameworks and tools exist for IFML, which also feature code generators that can produce running applications out of IFML models.

For instance, the WebRatio Web Platform (*http://www.webratio.com*) is a model-driven development tool which implements the Web-extended version of IFML. The tool supports developers in the specification of the domain model and of the interaction flow model for web applications. The tool features model checking and full code generation that

produces application code executable on top of any platform conforming to the J2EE specifications. In particular, at design time WebRatio provides four main integrated modeling and development environments, supporting respectively: the modeling of IFML diagrams for the specification of the user interaction; the modeling of UML class diagrams (or ER diagrams) for the content design; the graphical layout template and style design environment; and the specification and development of the modeling level and execution level of custom IFML elements defined as extensions of IFML. The WebRatio models are saved as XML documents. Based on the input provided through the modeling and development environments, WebRatio provides code generators which transform the specifications of the application into concrete, executable implementations. The generated code consists of a Java EE code covering both front-end of back-end of web applications including: The configuration file of the Controller which contains the navigation control flow logic; The action classes which are invoked by the Controller and, in turn, invoke the runtime services; The XML configuration files (called runtime descriptors) of the runtime services; and the server-side templates for dynamically building the actual pages of the application. At execution time, the WebRatio run-time framework consists of object oriented components and services for organizing the business tier, clustered in three main layers: the service layer, the application layer, and the logging layer.

Another example of implementation is IFMLEdit (*http://info.ifmledit.org*) is an online environment for the specification of IFML models, the investigation of their properties by means of a mapping to Place Chart Nets[14], and the generation of code for web and mobile architecture, based on continuous deployment and agile methods. IFMLEdit supports the following workflow: (1) the developer edits the IFML model of the application in the online editor, possibly providing hints for the generation of the fast prototype (e.g., sample data); (2) he (optionally) maps the model into a PCN and simulates the network to understand the dynamics of the application in response to events; (3) he generates the code of a fast prototype, for the web or for a cross-platform mobile language, executes and validates the prototype; (4) he turns the validated prototype into a real app, by customizing look-and-feel and replacing mock-up data access and operational APIs calls with real ones.

These tools and environment can be combined with runtime behaviour analysis, through a model-driven engineering approach that combines user interaction models with user tracking information and details about the visualized content in the pages. The integration of modeling languages for user interaction development and usage log analytics approaches has high potential of delivering valuable insights to designers and decision makers on the continuous improvement process of applications[10].

4. USAGE CENTERED DESIGN ACTIVITIES FOR DISH LMC GUIs

A usage-centered design approach has been adopted, considering also the lessons learned by SKA precursors and the inherent complexity of SKA systems and interactions.

As already outlined, user interface design is an iterative process that involves close liaisons between users and designers. 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. It covers topics ranging from usage-centered design during analysis and design, through to testing and validation in later application life cycle phases.

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

Starting from the set of requirements on LMC GUIs objectives, users and tasks, we began the analysis by identifying users (engineers, software maintainers) together with their roles and activities and by describing the main usage sceneries of the interfaces by means of *use case* diagrams, detailing the tasks to be performed by the users textually and via *swimlane* interaction diagrams.

Use cases are used to represent activities and goals that users might want to achieve (see Figure 3). More specific goals may be derived from more general ones. In use case diagrams use cases may be 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).

The *analysis of the users* of the system is another 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. In general, information collected in a profile describing a user role can be used to derive *design objectives* and to validate a user interface. A profile of the Dish element/sub-element engineer has been obtained performing user analysis by using sketches and storyboards as discussion documents for brainstorming, with the aim of eliciting opinions of stakeholders.

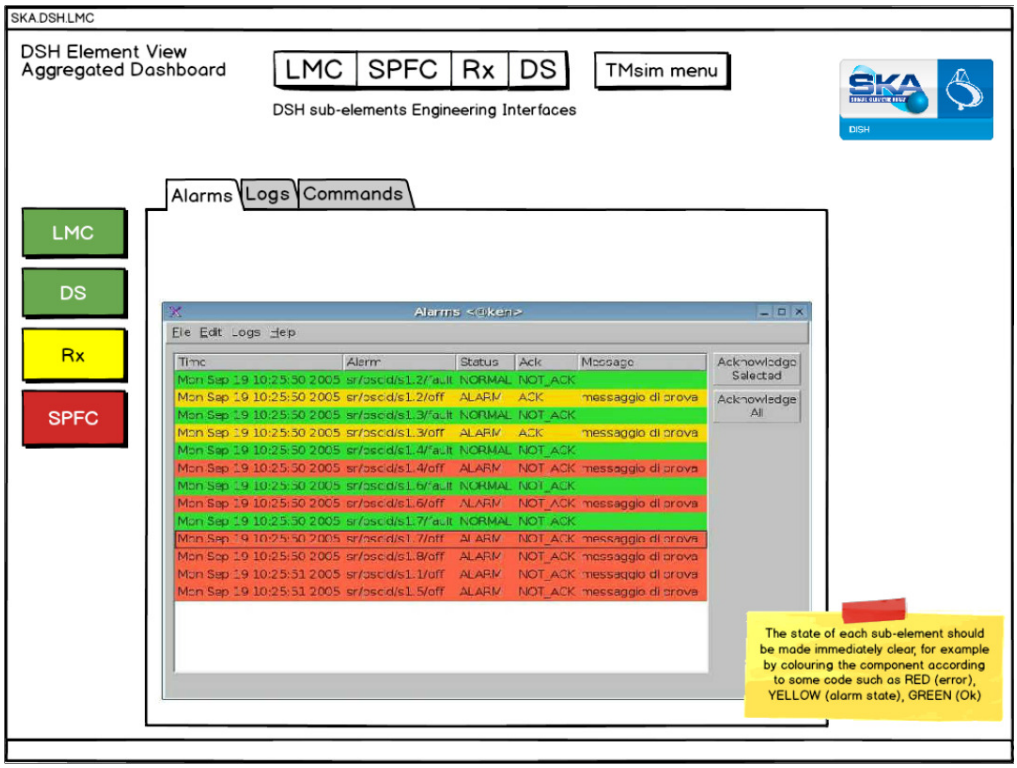


Figure 4: DSH element view (operator) aggregated dashboard

We decided to implement these sketches with one of the several existing tools (Balsamic Mockups) also able 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. In this way we explored several design ideas such as the interaction models and the features to implement. Stakeholders have then been presented with a discussion document integrating and refining the initial set of requirements with a new set of tentative scenarios proposals illustrated by means of an interactive mock-up simulating the interface. The discussion with stakeholders resulted in the definition of a set of *user roles* and *tasks* that take into account both the *context* within which the role is played and the characteristics of the performance.

Figure 4 shows one of the interactive sketches: a tentative DSH element view (operator) aggregated dashboard, where all the most important information regarding the whole DSH element is displayed drawing the attention to element/sub-element states also by the use of colours (green = OK, yellow = warning, red = alarm).

The main operator dashboard offers the user a drill-down navigation function to navigate sub-elements and monitor low level components detail data. Moreover it allows the user to open also the Dish sub-elements engineering interfaces and access the TM simulator menu.

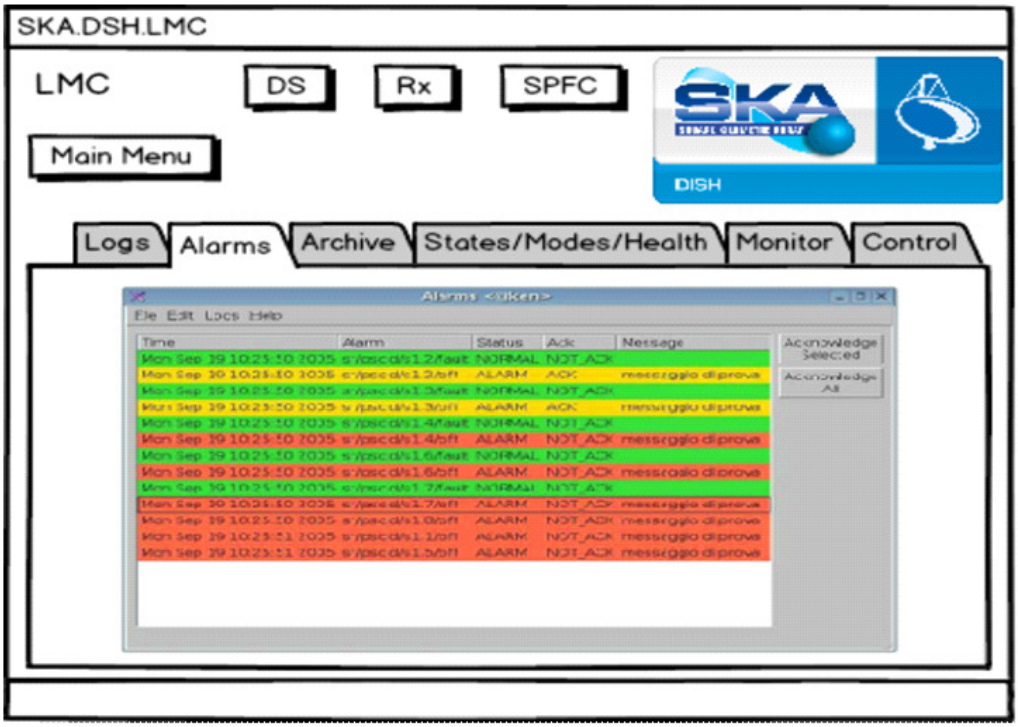


Figure 5: sketch of a screen of DISH LMC engineering interface showing the alarm management UI

Figure 5 shows another of the interactive sketches that were produced during the design process of the UI. It shows a panel of the Dish LMC engineering interface. It allows the user to open also the other Dish sub-elements UIs and to get back to a main menu in which further high level selections are available. The main part of the panel consists of a simple tabs bar implementing a flat menu of options and controls windows to be chosen and opened by the user via a simple click. The open window actually shows the Tango Alarms Managements GUI designed and implemented at Elettra.

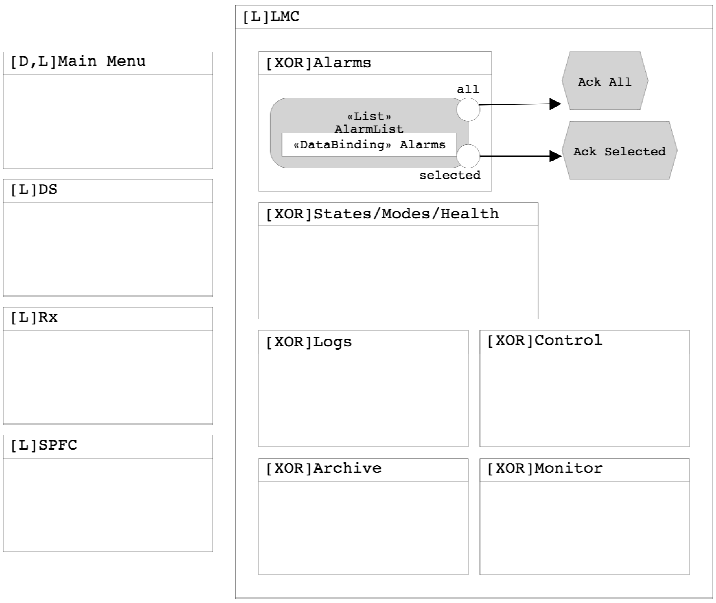


Figure 6: partial conceptual model of DISH LMC engineering interface

Figure 6 shows a (partial) conceptual model designed with IFML for the same user interface, where the model highlights the structure of the interfaces in terms of main windows and, for the LMC window, also the organization in panels, which are represented as XOR sub-screens, because they will always be shown once at a time. The opened panel (Alarms) contains the list of alarms and the two options available, i.e. the possibility of acknowledging all alarms at once, or only a selected subset of them.

From the same main operator dashboard shown in figure 4 it is possible to open a TM simulator window (Figure 7) from which TM (or simulated TM) operation can be performed according to the TM-DISH Interface Control document. Similarly to the case reported in Figure 6, IFML models are designed for all the other sketches (not reported here for space reasons).

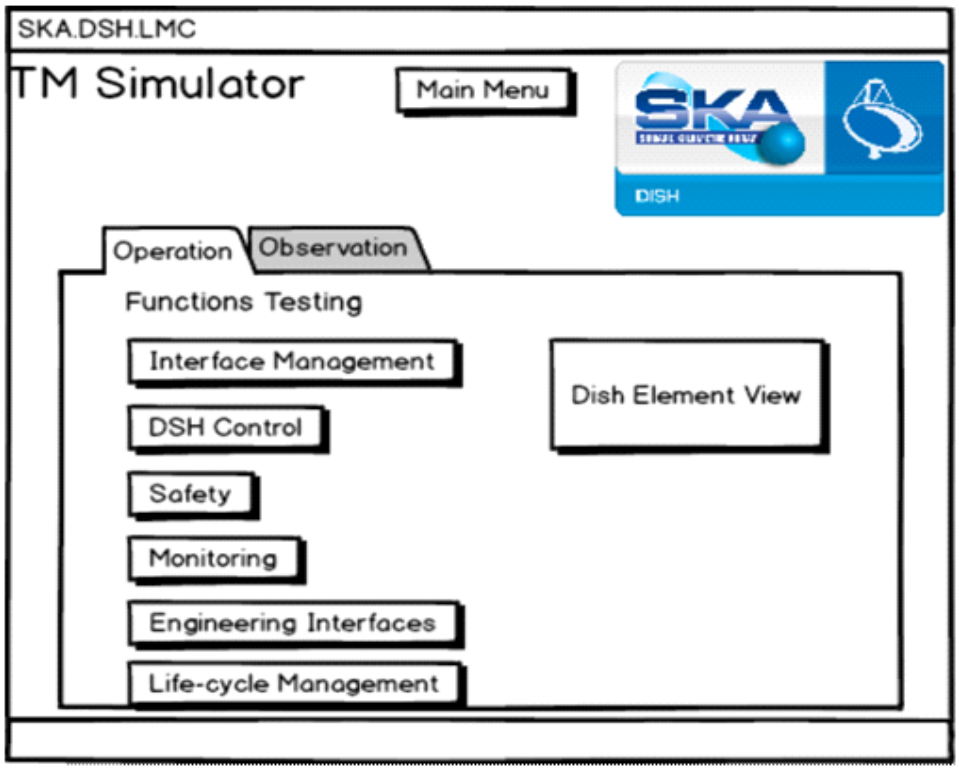


Figure 7: TM simulator window

5. TECHNOLOGICAL PROTOTYPING

Within the SKA Dish UI workstream, prototyping is also a key tool for the evaluation of technologies. Based on the documented lessons learned by precursors (LOFAR[5], MeerKAT[6], ALMA[7], ASKAP[8]) and specific UI analysis conducted by precursor sites (LOFAR), a set of UI tools has been selected to be analyzed against SKA Dish UI requirements. The set includes TANGO tools (e.g. Taurus[12] and Sardana) and general UI frameworks (such as AngularJS, Django, PyQT, PyTango[13], and TurboGears).

The Tango Control System was selected as the SKA framework in March 2015. The TANGO framework and its UI tools support the types of basic control interfaces that are currently used at both radio telescopes and within high energy physics experiments. TANGO UI tools can be divided into Desktop 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.DISH UI requirements and could be used to implement desktop UIs like SKA.DISH UIs.

In particular Taurus[12], a Python framework for control and data acquisition CLIs and GUIs in scientific/industrial environments, has been used and has proved valuable to implement SKA.DISH UIs prototypes.

LMC, SPFRx, SPF are built upon TANGO device servers[11]. Control and configuration operations requested by TM in the Dish are performed via a single Tango device server acting as a dish master/interface device.

The interfaces between LMC and other sub-elements, except DS, for which a Tango device proxy is envisaged, are defined in terms of Tango commands and attributes exposed by each device server.

Taurus implemented GUIs behave as clients following a client-server paradigm interacting with LMC, SPFRx, SPF and external hardware equipment Tango device servers.

Taurus allows the creation of fully-featured GUI (with forms, plots, synoptics, etc) from scratch in a few minutes using a "wizard" (such as *Taurusform* and *TaurusGUI*), which can also be customized and expanded by drag-and-dropping elements around at execution time. Moreover it gives full control to advanced users to create and customize CLIs and GUIs programmatically using Python and a very simple and economical API which abstracts data sources as "models". For convenience, Taurus provides the *Taurusdesigner* command that launches the standard Qt designer application extended to show also the widgets provided by Taurus.

Several panels have been prototyped using the above Taurus tools.

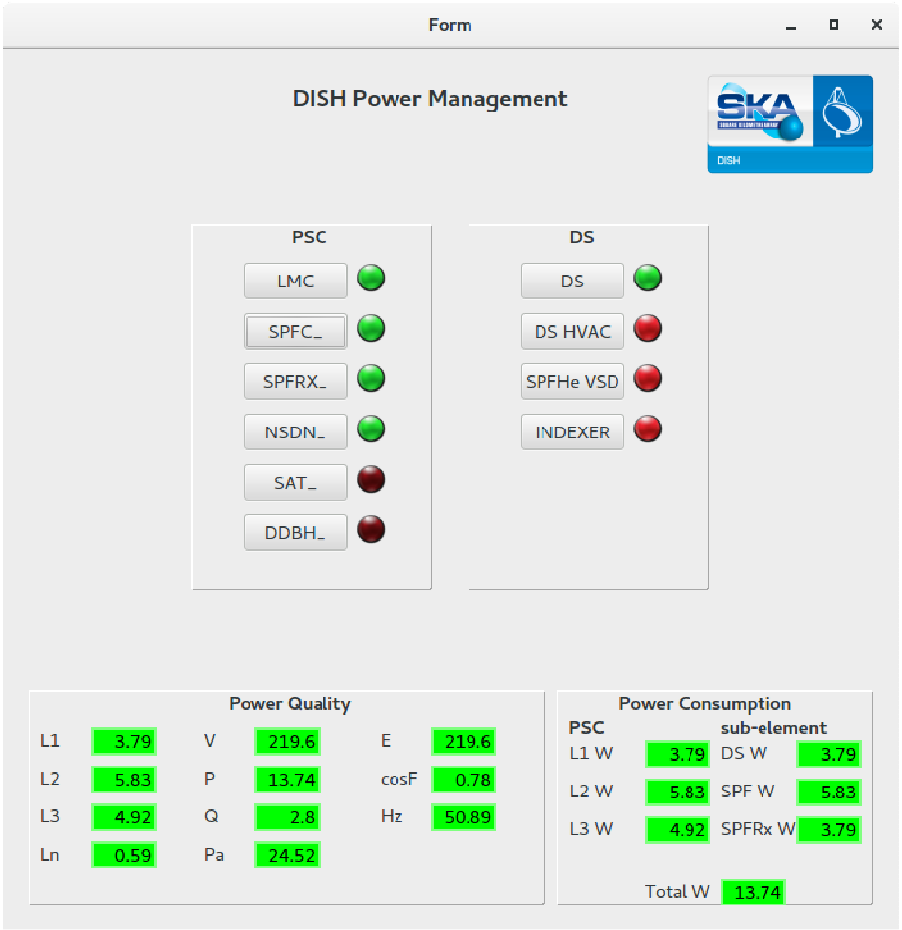


Figure 8: DISH Power Management

Figure 8 shows the general DISH Power Management monitoring and control GUI (implemented via *Taurusdesigner*) letting the user know at a glance all the power consumption and power quality data related to a single dish (low panels). Command buttons are used to power cycle dish devices located in the dish pedestal shielded compartment (PSC) directly controlled by LMC or to make the DS power cycle all the other dish devices outside of the PSC. Such GUI has been used also to perform LMC power management testing procedures.

For this purpose, a testing setup, as described in the scheme of figure 9, has been used: a software Test Unit connects itself as client to DS, SPF, SPFRx simulators (or possibly true sub-element devices) implemented as Tango device servers, in order to monitor their attributes. The Test Unit connect itself also to a TM simulator module, implemented as a Tango device according to TM-DSH.LMC ICD. Such TM simulator module includes a Power Management GUI by which an operator can manually control and monitor DISH power (see figure 8).

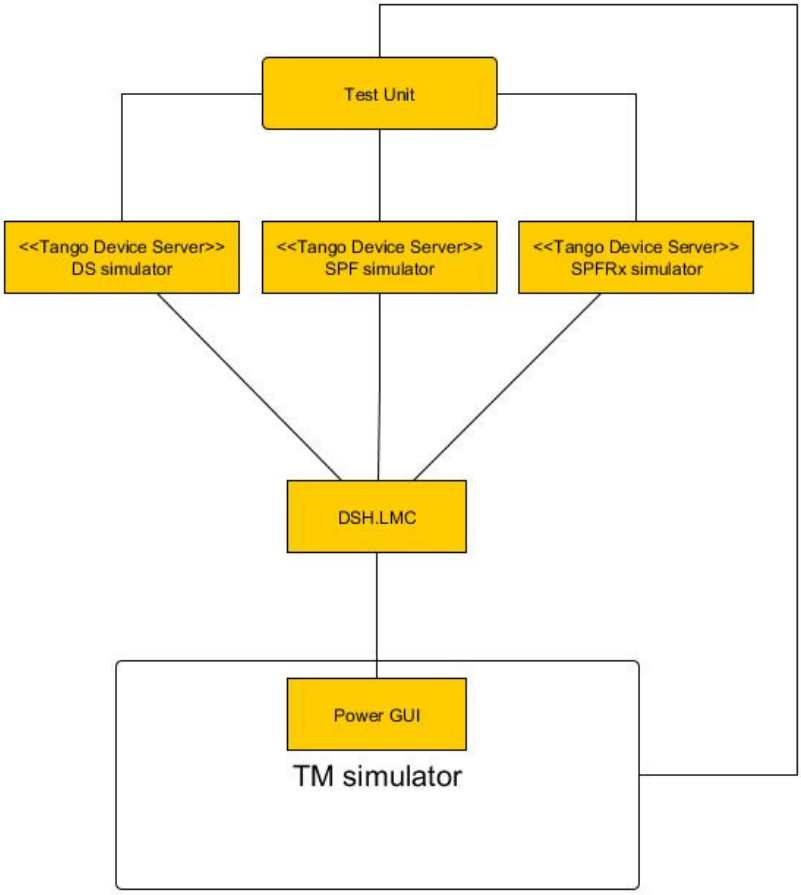


Figure 9: DISH Power Management testing setup

The Power Management GUI lets the user command the DISH LMC to control DISH devices power cycling and monitor DISH power quality and consumption data.

Power cycling and Power quality and consumption data reporting are to be performed as described in the Dish project documents: “SKA Dish States and Modes” and “SKA1 Dishes Element Power Management”.

Once agreed a proper manual testing procedure (as defined in the Dish LMC Qualification document), it has been automated so that the Test Unit launches a *PyTango*[13] testing script which acts, as an operator would do, in parallel with the TM simulator module (Power Management GUI), in order to perform proper power cycling and monitoring. The testing script, based upon the Python *Nose* framework, may act both step-by-step, in order to let testing personnel

visually control the progress of the testing procedure via the Power Management GUI and possibly interact with the system, and/or in a fully automated way, generating a results file to be automatically checked.

The software Test Unit connects itself as a client both to DS, SPF, SPFRx simulators (or possibly to true sub-element devices) in order to directly monitor their attributes and to the TM simulator module which interacts with DSH.LMC. In this way it may check monitoring data generated from the different sources.

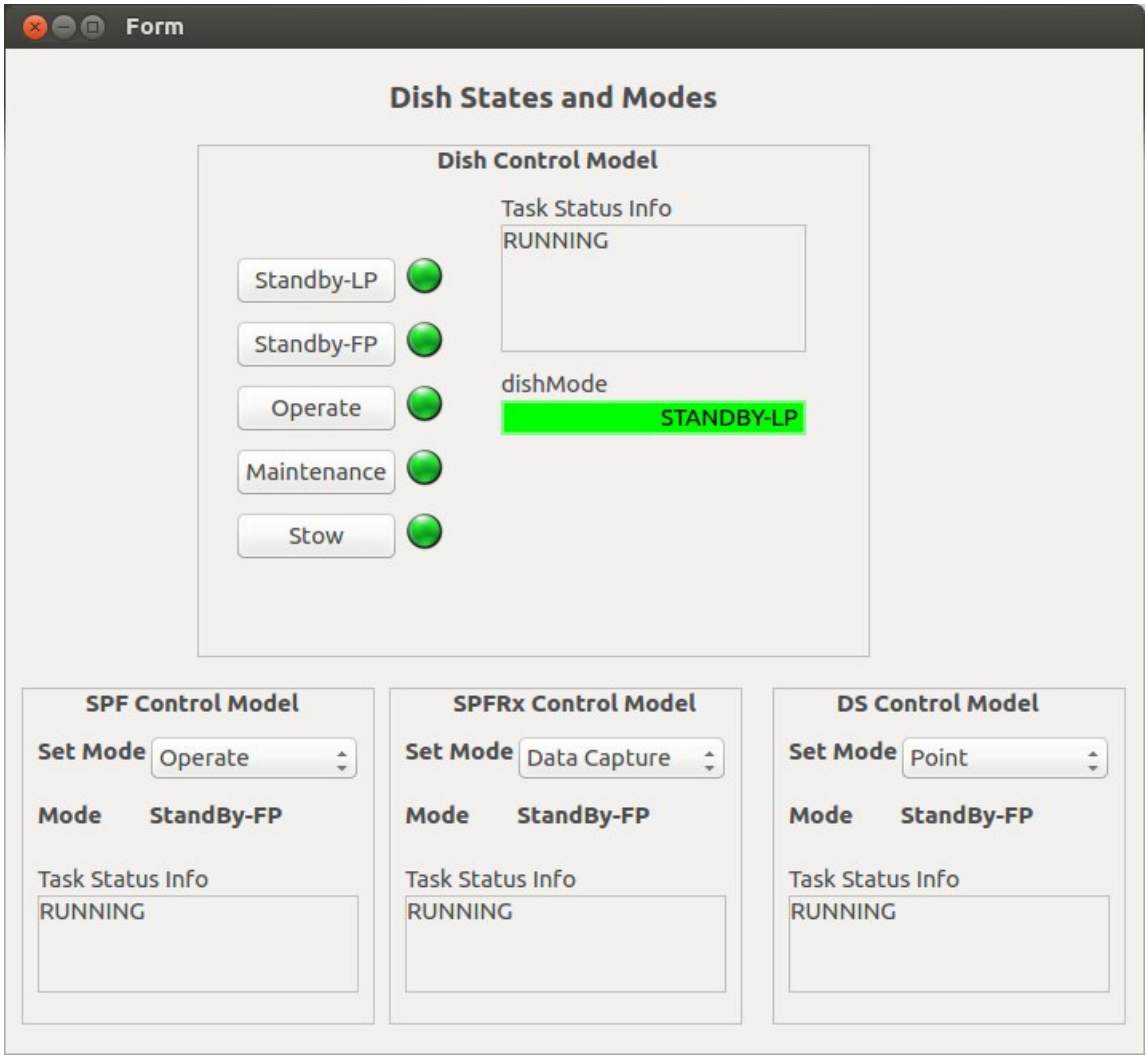


Figure 10: DISH States and Modes interface

Figure 10 shows a user interface specifically requested by end users and thought to keep an eye on Dish system states and modes, providing to the test engineer/operator the capability of complete configuration, control and visualization of the Dish parameters in the absence of TM during testing and integrated DSH stand-alone site integration and verification.

The DISH States and Modes interface has been implemented using Taurus tool *Taurusdesigner*.

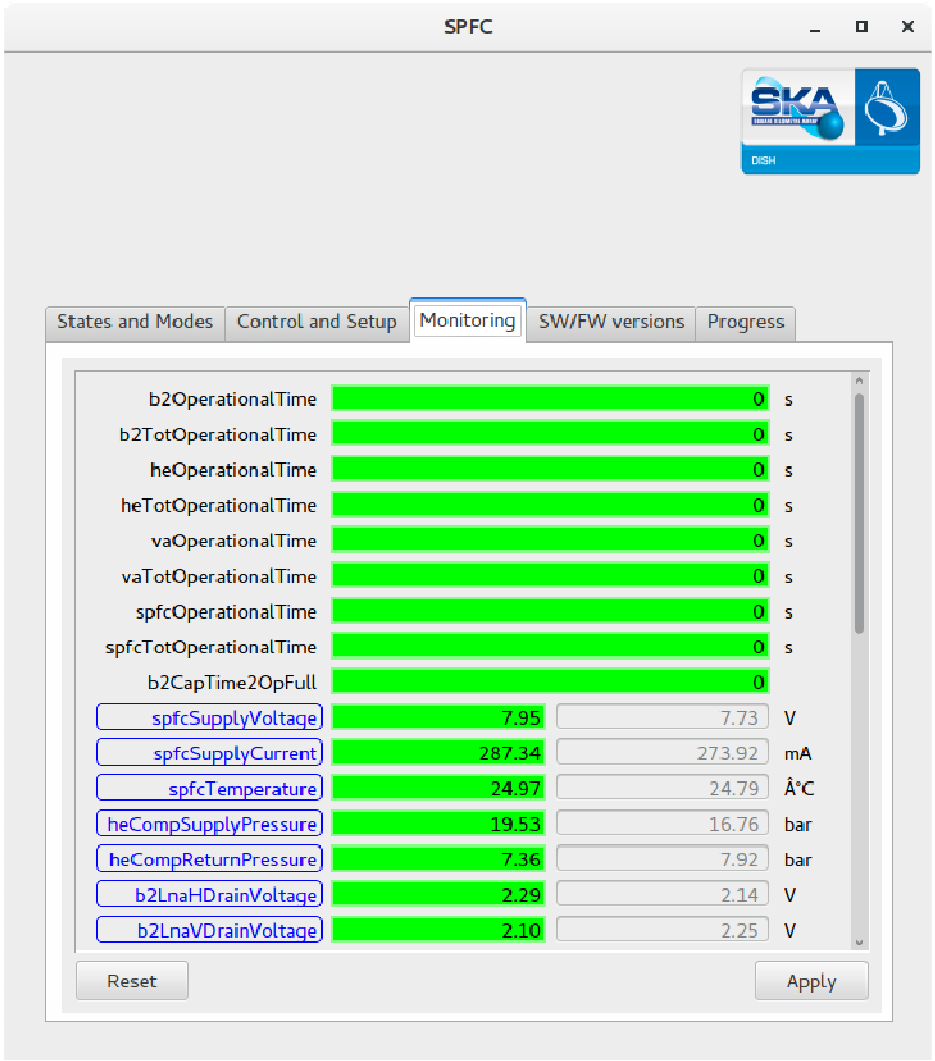


Figure 11: SPFC Engineering Interface

Figure 11 and 12 show respectively a glimpse of the monitoring panel of the SPFC Engineering Interface and of the SPFRx Engineering Interface organized as envisaged by the SPFC-LMC and SPFRx Interface Control Documents (ICDs) which details interface requirements (as regards configuration and setup, states and modes, sub-element control, fault reporting, diagnostics, alarms and events reporting, remote support), implementation (interaction model, naming conventions, facility context, control model, configuration parameters, commands (as regards control and configuration, logging and maintenance), Tango device attributes monitoring (as regards states and modes, control and setup, fault and diagnostics, alarms and events handling, archiving), remote support interfacing for software update and engineering GUI) and different use cases views (startup, shutdown, power cut and power restore events, SPF mode control, SPF errors handling, engineering GUI). The SPFC Engineering Interface has been implemented using *Taurusdesigner* and *Taurusform* tools, integrated and customized programmatically using Python language.

Taurusdesigner command launches the standard Qt designer application extended to show also the widgets provided by Taurus, so that the application/widget may be designed using not only the standard Qt widgets but also the Taurus ones. It is possible to use the Taurus Qt Designer to define a full GUI, or the TaurusGUI framework to create the GUIs and then the Taurus Qt Designer just for creating widgets to be inserted as panels in a TaurusGUI-based GUI.

Taurusform allows the user to interact with the data represented by its attached models. The actual association of a view (widget) with a model is done by providing the model name to the widget.

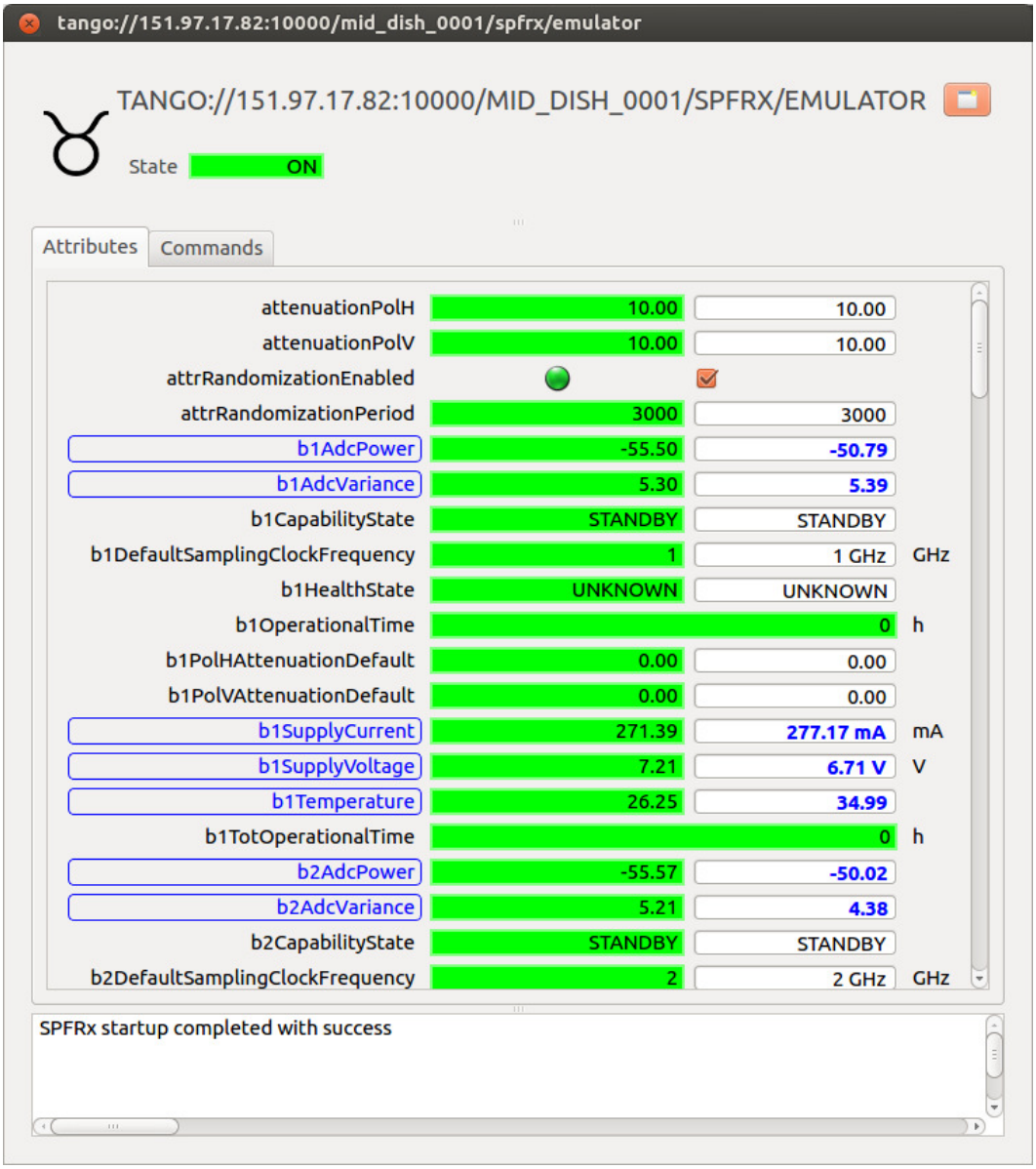


Figure 12: SPFRx Engineering Interface

The SPFRx Engineering Interface has been implemented using *Tauruform* tool, as a fast implementation prototyping tool.

Another example of fast prototyping is shown in figure 13, displaying the DS interface, automatically generated by *Taurusform* interacting with an emulated DS device server.

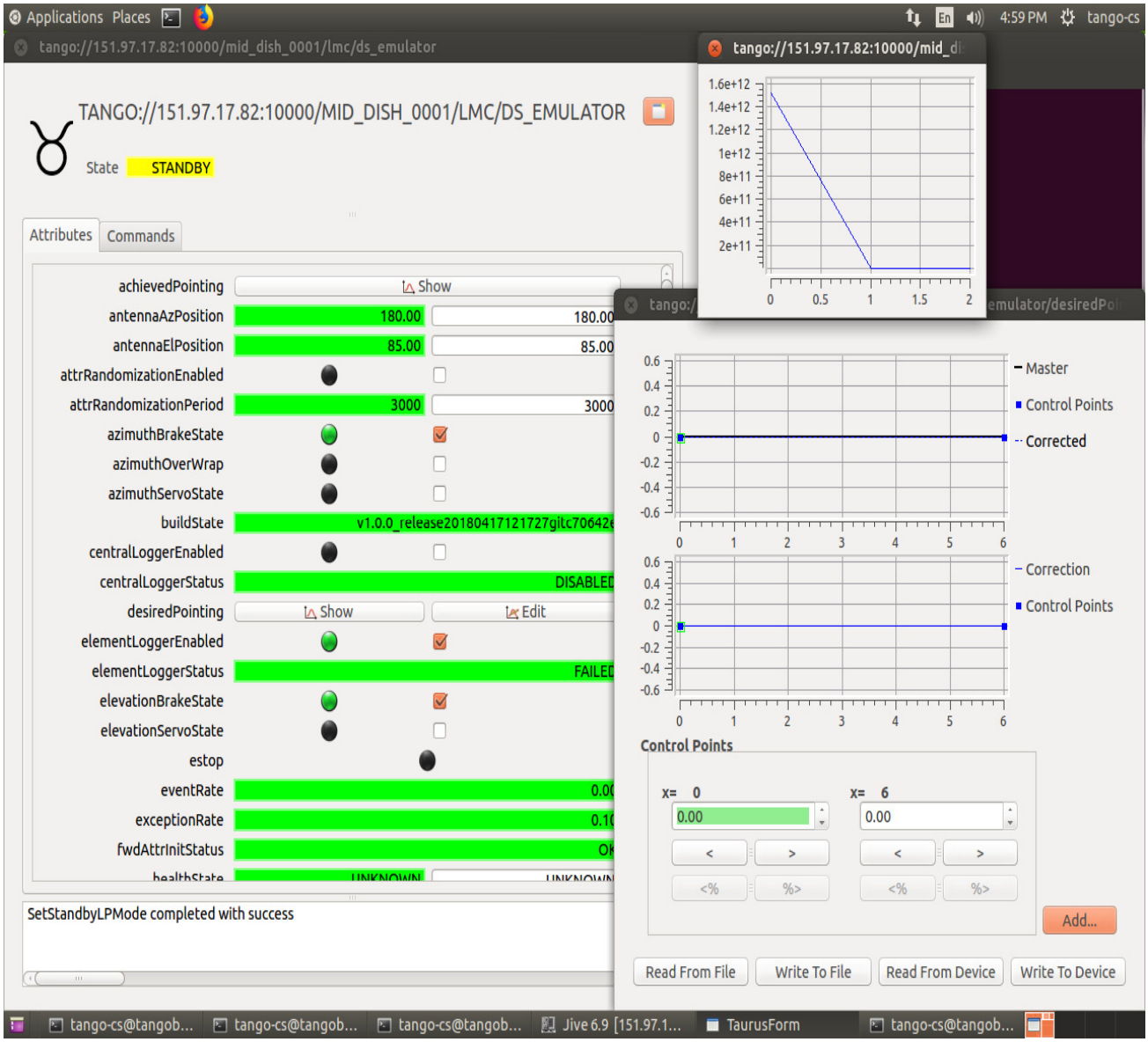


Figure 13: DS Engineering Interface

6. CONCLUSIONS

Considering the lessons learned from SKA precursors, proper GUI analysis and design activities are needed. These are focused on what combination of UIs will best support SKA users (in this case engineers and maintainers) and are based on usage-centered development practices. Such a UCD 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 iterative key steps:

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

The SKA Dish LMC Engineering User Interfaces UCD approach has given some important hints both from the methodological side and as regards concrete outputs.

It has helped to:

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

The use of sketches and storyboards as discussion documents for brainstorming, with the aim of eliciting opinions of stakeholders, has been appreciated and yielded concrete results. In particular, the use of interactive sketches with clickable hot spots linking to other sketches to simulate GUIs, thus illustrating interactively their intended dynamics, has proven to be a valuable instrument to get users' feedback. In this way we explored several design ideas such as the interaction models and the features to implement. The discussion with stakeholders resulted in the definition of a set of user roles and tasks to be performed and in the integration and refining of the initial set of requirements.

As regards the technological evaluation, the options for TANGO desktop development (including ATK based on Java Swing, QTango based on C++ and Qt and Taurus based upon Python and PyQt) all fulfill the basic SKA.TM requirements and could be used to implement desktop UIs like SKA.DISH engineering Uis.

Several working panels have been prototyped using the available Taurus tools: *Taurusform*, *TaurusGUI*, *Taurusdesigner,* when necessary integrated and customized programmatically using Python language and *PyTango*.

Further work is envisaged as regards look-and-feel (colours, shapes, layout, widgets), user interaction and context information visualization.

IFML[7], an international standard proposed by the OMG, has been tested in the user interaction modeling to provide a conceptual view of the UIs. Further steps are envisaged towards quick prototyping on the target platform of choice and automatic verification and validation: in particular, integration with the Taurus toolsuite, the exploitation of formal models derived from IFML (such as Petri nets) and automatic generation of test cases of the models will be explored.

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

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

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

[7] 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).

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

[9] Brambilla M. and Fraternali P., [Interaction Flow Modeling Language: Model-driven UI engineering of web and mobile apps with IFML], Morgan Kaufmann, (2014).

[10] Bernaschina C., Brambilla M., Mauri A., and Umuhoza E. A big data analysis framework for model- based web user behavior analytics. In International Conference on Web Engineering, pages 98–114. Springer, (2017).

[11] The TANGO Control System Manual, Version 9.2.5, (2018) http://www.tango-controls.org/ (16 May 2018).

[12] Taurus Documentation, Release 4.3.2-alpha, (2018) http://taurus-scada.org/ (16 May 2018).

[13] PyTango Documentation, Release 9.2.2., (2017) http://www.tango-controls.org/ (16 May 2018).

[14] Kishinevsky, M., Cortadella, J., Kondratyev, A., Lavagno, L., Taubin, A., Yakovlev, A.: "Coupling asynchrony and interrupts: place chart nets". In: Azéma, P., Balbo, G. (eds.) ICATPN 1997. LNCS, vol. 1248, pp. 328–347. Springer, Heidelberg (1997). doi: 10.1007/3-540-63139-9_44

[15] Brambilla, M., Mauri, A., Umuhoza, E.: "Extending the interaction flow modeling language (IFML) for model driven development of mobile applications front end". In: Awan, I., Younas, M., Franch, X., Quer, C. (eds.) MobiWIS 2014. LNCS, vol. 8640, pp. 176–191. Springer, Cham (2014). doi: 10.1007/978-3-319-10359-4_15

[16] Acerbis, R., Bongio, A., Brambilla, M., Butti, S.: "Model-driven development based on omg's ifml with webratio web and mobile platform". In: Cimiano, P., Frasincar, F., Houben, G.-J., Schwabe, D. (eds.) ICWE 2015. LNCS, vol. 9114, pp. 605–608. Springer, Cham (2015). doi: 10.1007/978-3-319-19890-3_39

[17] Brambilla M., Umuhoza E., Acerbis R.: "Model-driven development of user interfaces for IoT systems via domain-specific components and patterns". J. Internet Services and Applications 8(1): 14:1-14:21 (2017).