ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEPHA093

CODE GENERATION BASED ON IFML FOR THE USER INTERFACES OF THE SQUARE KILOMETRE ARRAY (SKA)

M. Brambilla, S. Pavanetto, M. Gasparini, Politecnico di Milano, Milano, Italy
A. Marassi†, R. Cirami, INAF-Astronomical Observatory of Trieste, Trieste, Italy

Abstract

The Square Kilometre Array (SKA) project is responsible for developing the SKA Observatory, the world's largest radiotelescope ever built. In this context, a number of Graphical User Interfaces (GUI) have to be designed and built to be used for monitoring and control, testing, simulation, integration, commissioning and maintenance. The Tango framework and its UI tools, selected for SKA in 2015, support the types of basic control interfaces currently used at both radio telescopes and within high energy physics experiments. This paper reports on the development of a Qt/Taurus code generator prototype based on the IFML (Interaction Flow Modeling Language) standard and respective modeling tools, that are extended for supporting the platform-specific code generation. The purpose of this work is to enable the use of low-code development in SKA GUI design, thus enabling increased efficiency, reliability and coherency of the produced UI. We present a simple GUI use case as complete example of software development cycle starting from requirements and including IFML modelling, Qt/Taurus automatic coding, interface evaluation and validation.

INTRODUCTION

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 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 [1] and remotely orchestrated by the SKA Telescope Manager (TM) system.

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 as described in [1].

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.

The performed technological prototyping of Qt and Taurus based upon Python and PyQt has shown they fulfill the basic SKA.DISH UI requirements and could be used to implement desktop UIs like SKA.DISH UIs.

Therefore, we focused on the development of a Qt/Taurus code generator prototype based on the IFML (Interaction Flow Modeling Language) standard, with the aim of automating the user interface implementation production.

The purpose of this work is to enable the use of low-code development in SKA GUI design, thus enabling increased efficiency, reliability and coherency of the produced UI. We present a simple GUI use case as complete example of software development cycle starting from requirements and including IFML modelling, Qt/Taurus automatic coding, interface evaluation and validation.

The paper is organized as follows: we first introduce the features required in the SKA.DISH LMC user interface, then we discuss the background concepts and technologies that are foundational for our approach, spanning usability, accessibility, user-centered design and Tango control. We describe our user-centered design activities and then we describe the model-driven development process for graphical user interfaces using IFML.

DISH USER INTERFACES

In the present paper we are considering SKA.DISH engineering user interfaces to be used by engineers for test, diagnostic, maintenance of DSH sub-elements, already identified and described in [1].

In particular, we have chosen to design and model LMC Engineering Interface. DSH sub-elements engineering interfaces will be accessible either directly from LMC (to be connected with keyboard/mouse and a screen) as desktop application.

LMC will provide GUIs to be used for testing and DISH control in stand-alone mode for testing, commissioning and maintenance, offering basic functionalities of DSH control & monitoring, set-up, control and testing, health monitoring, alarm management, lifecycle support, direct access monitoring in case of TM failure.

BACKGROUND CONCEPTS

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

† marassi@oats.inaf.it

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEPHA093

- *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.

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 [2-5].

Tango Controls

Tango Controls is a **free** open source device-oriented controls **toolkit for controlling any kind of hardware or software** and building SCADA (supervisory control and data acquisition) systems. Tango Controls is operating system and hardware independent and supports C++, Java and Python for all the components. Tango Controls has proven itself as a mature and reliable toolkit with many users, and as an open source is always being improved. It is used in synchrotron as well as in industrial and other scientific facilities. Tango Controls has been enriched with **many applications** (desktop and web based), among which GUI building tools such as Taurus and Qtango.

QTango is based on C++ and Qt and consists of classes and widgets that interact with the Tango control system, while providing an easy API to the programmer and full integration with the Qt4 designer.

Taurus is a Python framework, based on Python and PyQt or PySide, for control and data acquisition CLIs and GUIs in scientific/industrial environments. It supports multiple control systems or data sources, among which Tango Controls itself and EPICS.

Taurus provides a set of basic widgets (labels, LEDs, editors, forms, plots, tables, buttons, synoptics,…) that extend related Qt widgets with the capability of attaching to Taurus core models in order to display and/or change their data in pre-defined ways. Taurus allows the creation of fully-featured GUI (with forms, plots, synoptics, etc.) from scratch in a few minutes using a "wizard" (*Taurus Qt Designer*), which can also be customized and expanded by drag-and-dropping elements around at execution time. It also gives full control to more 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". The Qt designer will produce a .ui file that is an XML representation of the application/widget that you designed. The Qt designer will produce a .ui file that is an XML representation of the application/widget that you designed. The resulting .ui file is automatically transformed into python code with a specific command.

USER CENTERED DESIGN ACTIVITIES

Considering the lessons learned by SKA precursors and the inherent complexity of SKA systems and interactions, a user-centered design approach has been adopted.

Starting from the set of requirements on LMC GUIs objectives, users and tasks, the analysis identified users (engineers, software maintainers) together with their roles and activities, together with the main usage scenarios of the interfaces by means of *use case* diagrams. Based on this, the method derives *design objectives* and validates a possible user interface (implemented through sketches with Balsamiq Mockups), thus allowing to explore several design ideas such as the interaction models and the features to implement.

All the above DISH LMC GUIs Usage Centered Design activities have been carried out as part of the tasks performed in the so-called SKA pre-construction phase by the SKA.DISH consortium (SKADC).

INAF – Catania Astrophysical Observatory, as member of the SKA.DISH consortium, had the responsibility of DISH LMC design, prototyping, testing and validation.

Figure 1 shows one of the interactive sketches that were produced during the design process of the UI. It shows a panel dedicated to 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.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEPHA093

Figure 1: Sketch of a screen of DISH LMC engineering interface showing the alarm management UI.

MODEL DRIVEN DEVELOPMENT OF THE GUI

A conceptual tool that can be used in user-centered design of user interfaces 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. Conceptual modeling supports 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; and 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), an international standard proposed by the OMG (www.ifml.org) [6]. IFML has been designed for expressing the content, user interaction and control behaviour of the front-end of software applications. IFML does not cover the modeling of the presentation issues (e.g., layout, style and look-and-feel) of an application front-end. IFML integrates with other mainstream software modeling languages like UML and BPMN, and covers the following design perspectives: the view structure and content specification; the events and their effects; the input-output parameter bindings between elements; and the reference to business actions triggered by the user's events. Several extensions of IFML exist covering Web, mobile, and IoT-based 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.

Figure 2 shows a (partial) conceptual model designed with IFML for the user interface reported in Fig. 1, where the model highlights the structure of the interfaces in terms of main windows and, for the LMC main menu window, also the organization in panels, which are represented as XOR sub-screens, because they will always be shown once at a time.

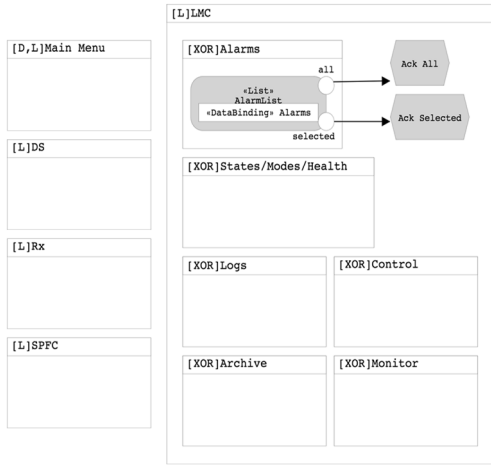

Figure 2: Partial IFML conceptual model of DISH LMC engineering interface.

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.

Various implementation frameworks and tools exist for IFML, which also feature code generators that can produce running applications out of IFML models, such as WebRatio Web Platform (*http://www.webratio.com*), a model-driven development tool which implements the Web-extended version of IFML, and IFMLEdit (*http://info.ifmledit.org*), an online environment for the specification of IFML models, the investigation of their properties by means of a mapping to Place Chart Nets, and the generation of code for web and mobile architecture, based on continuous deployment and agile methods. 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 [7]. 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.

Generation of Taurus UIs from IFML

In this work we exploit IFML as a conceptual modeling language, and IFMLEdit.org as implementation platform for specifying a full model-driven development process with automated code generation for Qt/Taurus.

To this end, we apply the practices of model-driven software development (MDD), which entails automation of some of the steps of the development process. Software development automation consists of starting from a high level conceptual representation of the desired software features (typically described as conceptual models) and

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEPHA093

deriving a running application out of it, possibly through a set of intermediate steps to enable some degree of customization of the generation process.

Typically, when following an MDD approach, the running application can be obtained through one or more model transformations that subsequently produce a more and more refined version of the software description, until an executable version of it is reached, as shown in Fig. 3. As one can see in the figure, models are (semi)-automatically generated using model-to-model transformations taking as input the models obtained in the previous phase. In particular, one can start from conceptual models of requirements, and then transform them into design models through a model-to-model (M2M) transformation, while in the last step, the final code is generated by means of a model-to-text (M2T) transformation from the design models. Analogously, also test cases can be generated automatically.

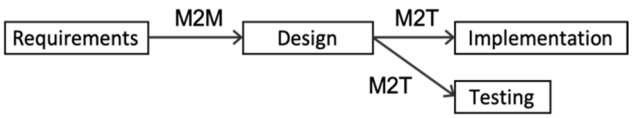

Figure 3: The model-driven development process.

More specifically, in our work we focus on the Design and Implementation phases. Figure 4 describes the MDD framework implemented in our specific case. The designer of IFML models specifies the conceptual interaction models in a visual model editor, like IFMLedit. Automatic code generation can encompass two different scenarios:

- ***One click interface generation***: in this case the model-to-text transformation generated directly QT/Taurus code, ready to be deployed and executed, so that end users can directly work on the running interfaces;
- ***Human-in-the-loop generation***: in this case the transformation produces an intermediate artifact to be fed into QT/Taurus editors (like Taurus Qt Designer), so that interface designers can update the generated interfaces through manual refinements and extensions. In turn, the edited project is then used for generating the final Taurus code to be deployed, through the native model-to-text transformation implemented by Taurus Qt Designer.

The advantages of the latter approach stand in a much broader configurability of interfaces, while on the negative side the effect of this is a misalignment between the customized versions edited in the editor with respect to the conceptual models described in IFML.

Based on this assessment, and considering that the first objective we want to achieve in control systems for large-scale research infrastructures like SKA is self-coherency of the application, we opted for direct QT/Taurus code generation.

Starting from IFML models, our prototype implements a full code generator for Qt/Taurus. The implementation is written in Javascript, so as to be fully integrated with IFMLedit.org. The implementation of the generator is integrated with the IFMLedit.org editor, and is available as open source code on Github under the Apache 2.0 license[1].

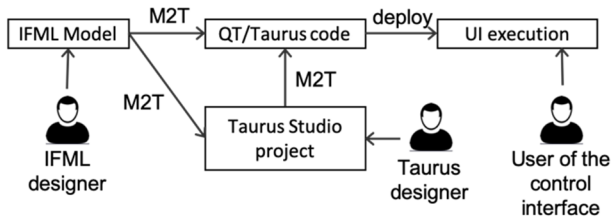

Figure 4: The model-driven process for generating QT/Taurus code for the LMC SKA Dish interface.

The generation process considers in input a simple set of IFML items and builds a mapping to QT/Taurus implementation elements in a straightforward way. Table 1 shows an excerpt of the most important element mappings.

Table 1: Excerpt of Important Mappings between IFML Concepts and QT / Taurus Elements

IFML Concept	QT Implementation
IFML Model	Qt Gui
View Container	Qt Window
Nested View Container	Qt TabWidget
View Component: Form (with Fields)	Qt Form (with fields and text labels)
View Component: List	Qt Table (with columns)

As an example, Fig. 5 reports a simple excerpt of IFML model and Fig. 6 describes a (partial) listing of the corresponding generated code.

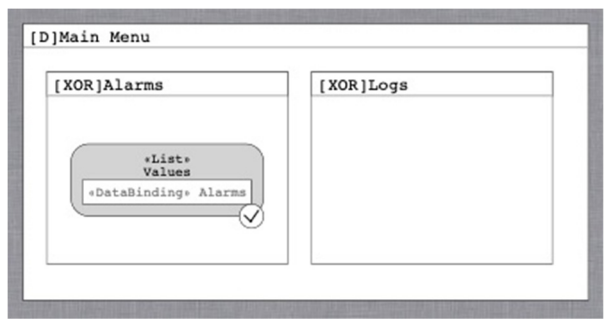

Figure 5: IFML model with one main screen and two internal tabs, to be shown in mutual exclusion.

[1] https://github.com/DataSciencePolimi/IFML-to-QT-Taurus-SKA

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEPHA093

```
class Ui_Form(object):
    def setupUi(self, Form):
        Form.setObjectName(_fromUtf8("Form: Main Menu"))
        Form.resize(680, 739)
        self.tabWidget = QtGui.QTabWidget(Form)
        self.tabWidget.setGeometry(QtCore.QRect(30, 180, 621, 531))
        self.tabWidget.setObjectName(_fromUtf8("tabWidget"))
        self.tab = QtGui.QWidget()
        self.tab.setObjectName(_fromUtf8("tab:Alarms"))
        self.taurusForm = TaurusForm(self.tab)
        self.taurusForm.setGeometry(QtCore.QRect(10, 10, 601, 471))
        self.taurusForm.setObjectName(_fromUtf8("Alarm List"))
        self.tabWidget.addTab(self.tab, _fromUtf8(""))
        self.Tab_2 = QtGui.QWidget()
        self.Tab_2.setObjectName(_fromUtf8("Tab:Logs"))
        [...]
        self.retranslateUi(Form)
        self.tabWidget.setCurrentIndex(4)
        QtCore.QMetaObject.connectSlotsByName(Form)

    def retranslateUi(self, Form):
        Form.setWindowTitle(_translate("Main Menu", "Main Menu", None))
        self.tab.setToolTip(_translate("Form", "Alarms", None))
        self.tab.setWhatsThis(_translate("Form", "Alarms", None))
        [...]

from taurus.qt.qtgui.panel import TaurusForm

if __name__ == "__main__":
    import sys
    app = QtGui.QApplication(sys.argv)
    Form = QtGui.QWidget()
    ui = Ui_Form()
    ui.setupUi(Form)
    Form.show()
    sys.exit(app.exec_())
```

Figure 6: Excerpt of generated code for the interface modelled in Figure 5.

CONCLUSIONS

Control of large-scale scientific infrastructures like SKA requires coherent and effective user interfaces that can be specified only through on usage-centered development practices. UIs implemented through model-driven development and automation of the code generation process can obtain highly configurable and yet standardized interfaces as requested. In this work we demonstrated the feasibility of the approach and we reported on the implementation of a prototype of code generator. Future work will include the extension of the generator and field-testing of the generated interfaces.

Since code generation allows for very quick rounds of development of new versions of the interface, we plan to integrate the current approach with intelligent semi-automatic methods (possibly implemented through machine learning techniques), that will try to automatically optimize the quality of the interfaces based on some quantitative metrics of user performance on the interface.

REFERENCES

[1] Marassi A., Brambilla M. *et al.*, "The SKA Dish Local Monitoring and Control System User Interface", in *Proc. SPIE Astronomical Telescopes + Instrumentation, 2018*, Austin, Texas, United States doi.org/10.1117/12.2313822

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

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

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

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

[6] Marco Brambilla and Piero Fraternali, *Interaction Flow Modeling Language: Model-Driven UI Engineering of Web and Mobile Apps with IFML*, Morgan Kaufmann, 2014.

[7] Carlo Bernaschina, Marco Brambilla, Andrea Mauri, and Eric Umuhoza "A Big Data Analysis Framework for Model-Based Web User Behavior Analytics" in *Proc. International Conference on Web Engineering*, Rome, Italy, Springer, 2017, pp. 98-114.