ESA-HMI STANDARDIZED FRAMEWORK FOR DESIGNING HUMAN-MACHINE INTERFACES

Joanna Modławska(1,2), Krzysztof Samp(1), Michał Tanaś(1,2), Dariusz Walczak(1,3) , Nieves Salor Moral(4)

ITTI Sp. Z o.o.(1)
ul. Rubież 46, 61-612 Poznań (PL), +48(0)61 6226985
Email: joanna.modlawska@itti.com.pl

Adam Mickiewicz University, Faculty of Physics(2)
ul. Umultowska 85, 61-6142 Poznań (PL), +48(0)61 8294000

Poznań University of Economics, Faculty of Informatics and Electronic Economy (3)
Al. Niepodległości 10, 61-875 Poznań +48(0)61 856 90 00

Vitrociset Belgium, permanent establishment in the Netherlands(4)
'sGravendijkseweg, 53 - 2201CZ Noordwijk (NL), +31(0)71 3649770

ABSTRACT

The evolution in IT (Information Technology) systems has been immense in recent years and countless innovative solutions have been successfully implemented. However, wherever there is a fast and frantic development evolution, there is always also a chaos associated with involving simultaneously many incompatible or disjoint solutions. Such variety of solutions significantly raises complexity of the developed systems, as well as increase risk of human operators mistakes. Therefore, the need for widely accepted standards and reference solutions is becoming a rule instead of an option. Such standardization is especially important for human-machine interfaces (HMI), which are the only part of an IT system directly exposed to its users and thus their design flaws or quirks can be neither corrected nor hidden by any other layer of software.

This paper presents the results of the "The technology framework for the development of modular, portable and adaptive Human-Machine Interfaces in ground segment software products" (ESA-HMI) project. The main objective of the project is the development of a standard methodology and framework for design and development of multi-platform human-machine interfaces for the ground-segment IT systems used within the space sector by the definition in a textual format (i.e. XML – eXtensible Markup Language). Such framework can significantly reduce both costs and development time of human-machine interfaces by providing a collection of reusable building blocks for every space-related graphical control. Such approach is expected to significantly reduce training costs of its human operators and simultaneously reduce the risk of making trivial errors. Another significant advantage of the ESA-HMI framework is the separation of the human-machine interface from the associated business logic, thus allowing applying design patterns directly to the development. Such separation allows the human-machine interface to conform to the appropriate industrial standards regardless of specifics of the target system. As proof of concept and due to its importance in ESA environment, in the ESA-HMI project graphical controls ensuring access to the data models described in the ESA standards [1], [2], [3] and [4] will be provided.

1. INTRODUCTION

The "The technology framework for the development of modular, portable and adaptive Human-Machine Interfaces in ground segment software products" (hereinafter called ESA-HMI) is an ESA project, realized in the first Call of the ESA Polish Industry Incentive Scheme. The project is realized by a consortium of two companies. The prime contractor is ITTI sp. z o. o. from Poland and the subcontractor is Vitrociset from Belgium.

The main objective of the ESA-HMI project is to develop a TRL-4 (Technology Readiness Level 4) demonstrator of the idea for automatically generating reusable, system and device independent, HMI controls. The methods and software developed within the project may become a baseline for the standardization of HMI for the ground segment systems.

2. ENGINEERING APPROACH

2.1 The Idea of Single Definition Multiple Compilations Controls

The main idea of the HMI framework is to create UI (User Interface) control definitions independent from the source code, which would allow not only defining the required UI controls but also use them in several applications (i.e. systems and hardware architectures) regardless of their implementation details. Technical feasibility of such independency at the current technological level was analyzed in [5]. To achieve this independency, the UI controls definitions will be translated and compiled by the middleware called HMI Presentation Engine. This engine is target device specific, so the project approach allows using standardized UI definitions without having to manually rely on target optimization utilities of software source code. Moreover, the selected approach allows for easy implementation of UI optimizations for particular target devices. Especially, UI functionalities which may be impractical on a particular target device can easily be re-configured.

For example, a block diagram can be presented on tablets in read-only mode (because the edition of diagrams with imprecise finger driven touchscreen is not a necessity) while on computers it can be presented in editable mode. The translator customizes the UI definition for a particular device where the UI is going to be integrated. The compilation process of the UI controls' definition is presented in the following Fig.1.

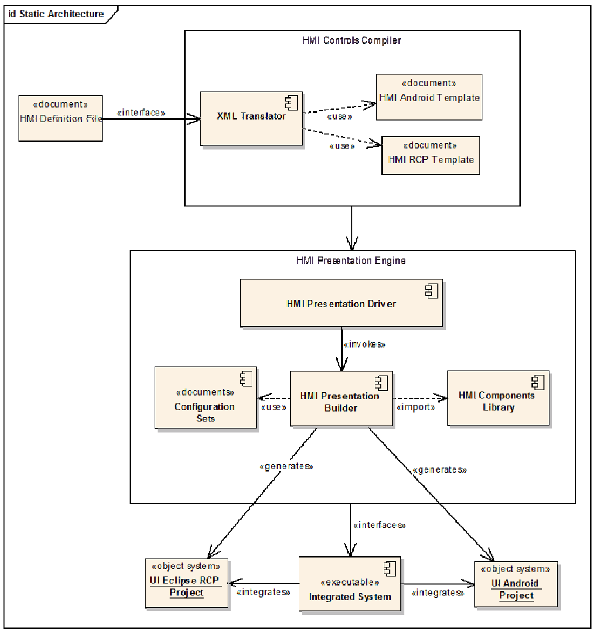

Fig. 1. The process of compilation of an UI control definition

Application frameworks use different methods to provide their user interface definition: source code, template languages, XMLs or even CSS. In order to reduce the complexity and time required to create harmonized UIs, the definition of such graphical interfaces needs to be externalized in a simple and universal way. Due to its extended use as an exchange format, XML has been selected to be used for defining the graphical interface as an input to the framework. Thus, XML is also to be used in the development of Android and Desktop applications through the application of different libraries. One can use Android SDK for mobile phones/tablets while XWT (XML Windowing Toolkit) is applied for Java desktop applications.

However, although both, mobile and desktop applications, require an XML for their configuration, the structure of the files and the runtime environment are completely different. Thus, modifications have to be performed to allow the runtime environments to display the interfaces correctly. These modifications are time consuming, so an automatic method to provide them is proposed as a part of the architecture.

HMI framework enables a user to define the graphical interface through a common XML file for both platforms. The input XML is compiled into its mobile or RCP desktop formats.

Within the system and in the Presentation Engine, the specific formats are used to generate the source/binary Java code projects containing the UI classes with their default behaviour associated. These projects have to be integrated into the Integrated Target System. If no further modification is performed, the defined graphical interface will be displayed but it will lack any data.

In order to close the loop with the integrated system, data shall be imported to the system and their specific business logic shall be specified. In order to achieve this, the framework provides an interface to add the business logic to different controls.

2.2 Exemplary Controls Implemented - Simple control example gauge

In order to test the framework a few new HMI controls were implemented. An example is the "Gauge control", which shows a monitored value on a scale with minimum and maximum thresholds for the value. The control alerts the user when the value is outside of the thresholds and throws an exception in case the value is out of the defined scope.

An example views of the gauge control in their two states (i.e. normal one and alert one) are presented in the following figures:

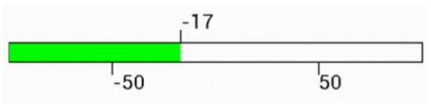

Fig. 2. The gauge control - value between alert thresholds

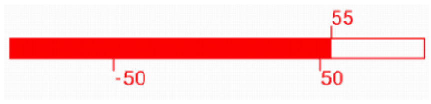

Fig. 3. The gauge control - value above alert thresholds

3. PROVIDED INNOVATIONS IN SOFTWARE DEVELOPMENT PROCESS

3.1 General Idea

The idea of defining control definitions independent of their implementation details allows for roles separation:

- HMI User – the person who defines the final user interfaces with available controls in XML format,
- HMI Business Logic Developer – person who integrates the HMI framework and adds logical behaviours to the controls.

Note that the above roles are users of the framework, not its developers. Therefore, they use the controls that the framework provides, but the development of new controls is not part of their responsibilities.

HMI User

HMI User's role is to design an application's user interface. Users have to know XML syntax and the HMI API. However, knowledge of Java, Android API or any other target programming language is not required. Their work is to compose an XML which contains the controls' definition and run the system application. The application will create bundle projects for desktop and mobile applications. In both cases a demo bundle will be generated as well. User can run this demo project and check the appearance of the user interface as a mockup. Consequently, users' development cycle consists of 3 steps:

1. Defining the UI in XML format,
2. Running the application for generating the UI bundles,
3. Testing the UI via demo project.

The demo project contains only the user interface. There is no business logic so the output interface might look slightly different than the final one. This means, the controls still need to be set up with proper data sets and constraints checks.

HMI Business Logic Developer

The Business Logic Developer's role is to provide the business logic behind the controls and to create the final application. They use the application's output bundles and integrate them within the final system development environment.

Business users do not need to know the low-level details or the implementation of the UI controls. They interact with the controls via Control Manager, which basically presents only the controls data model. All controls should sensitive to model changes and react when a change occurs. HMI is neutral to business logic; therefore there is no explicit limitation for business logic.

In most cases, the business logic developers will have to provide platform specific source code. The manner in which the user handles the reuse of the business logic is a design decision of the business user and it is outside the scope of the HMI project.

3.2 HMI Design and Implementation

HMI framework requires some additional steps for developer at each stage of the development process. In the following section a brief description will be provided, presenting what should be done in order to utilize the HMI framework starting with the requirements.

XML Preparation and Transformation

In HMI there are three different types of XML description of the UI:

- User oriented HMI XML following the HMI Controls API,
- Desktop user interface definition in XWT format, following the Eclipse RCP (Rich Client Platform) UI format,
- Mobile user interface definition in Android layout files.

HMI users are only aware of the first XML format. The other two are important for current and future developers of the HMI framework.

In case new controls need to be supported by the HMI framework or current ones modified, HMI control developers should define user friendly representation of the control applying the following rules:

- Design XML syntax for the control definition,
- Let users define attributes of the control via XML,
- Define default values for optional attributes.

For the gauge example, displayed in Fig. 2, there are five attributes that set the initial state of the control. Therefore, HMI users need to prepare the following common XML code:

```
<Panel> <Gauge highValue="100"  lowValue="0" highThreshold="50" lowThreshold="30" value="35" /> </Panel>
```

Then, the XML translator (see Fig. 1) generates the target specific code (e.g. XWT or Android XML) by applying simple XSLT (Extensible Stylesheet Language Transformation) translations.

For example, the desktop version of the gauge XML will look as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<Shell xmlns:x="http://www.eclipse.org/xwt"
                    xmlns:hmi="clr-
namespace:pl.com.itti.hmi.desktop"
```

```
        xmlns="http://www.eclipse.org/xwt/presentatio
n">
        <Shell.layout> <RowLayout/> </Shell.layout>
        <hmi:HGauge
                highValue="100"
                lowValue="0"
                highThreshold="50"
                lowThreshold="30"
                value="35" />
</Shell>
```

The Android version, generated from the same basic XML will look differently:

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/an
droid"
  xmlns:hmi="http://schemas.android.com/apk/res-
auto"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical"
  android:paddingBottom="@dimen/activity_vertical_
margin"
  android:paddingLeft="@dimen/activity_horizontal_m
argin"
  android:paddingRight="@dimen/activity_horizontal_
margin"
  android:paddingTop="@dimen/activity_vertical_marg
in" >
  <pl.com.itti.hmi.mobile.controls.HGauge
    android:id="@+id/gauge1"
    android:layout_width="fill_parent"
    android:layout_height="150px"
    hmi:label="HMI Gauge"
    hmi:max="100"
    hmi:maxThreshold="50"
    hmi:min="-100"
    hmi:minThreshold="-50" />
</LinearLayout>
```

If the default behaviour of the XSLT translation is not enough for new controls, the HMI developer will have to modify the “Compiler” module of the framework.

Model and business logic implementation

To implement a new control within the HMI framework, one should start with the data and business model for the control. This part should be written for both platforms simultaneously, so no platform specific dependencies are allowed.

1. Define the control interface in *pl.com.itti.hmi.api.interfaces*. The interface should extend one of the existing Control interfaces (either *BasicControl* or *ComplexControl* ones). The interface should focus on the control behaviour and the needed data and not on the presentation aspects.
2. Develop default implementation for the interfaces in *pl.com.itti.hmi.api.controls* package. This class will act as a model for control.
3. All simple controls should extend *BasicControls* class and call default constructor of the class. This will register the control in the singleton *ControlManager* instance. It holds all controls by their name or id. HMI Business Logic Developers use it to retrieve and access control objects.
4. Check *pl.com.itti.hmi.api.events* package for needed listener and event classes. Entirely new class must be created only if no suitable ready to use one is found. At least one listener is required to notify view classes about control state change.
5. The *pl.com.itti.hmi.api.exceptions* package is responsible for managing exceptions mechanisms. Again, entirely new exception class must be created only if no ready to use one is found..

Presentation Engine Development

The process of developing of a platform specific control is different for both platforms and requires knowledge of each of them. On the one hand, desktop developers will extend the Composite class and hold reference to the widget class or build a new widget class based on an existing simpler widget. On the other hand, mobile developers will inherit the most similar widget and modify its behaviour. They will hold reference to the data model.

On both platforms the presentation classes should implement the control interface defined in the previous step (contained in *pl.com.itti.hmi.api.interfaces* package). All methods should delegate responsibility to the model class.

In HMI framework data model should hold the state and the control view class should only react to changes of the model class. Business Logic Developer uses only data model class so this is the only way to be notified that the control should change its view.

The following table compares control development process on desktop and mobile devices:

Step	Desktop	Mobile
1	Define possible attributes for the XML widget representation.	
2	Create class which extends Composite class (typical SWT (Standard Widget Toolkit) step).	Create class extending the most similar widget.
3	Create and store data model for the control.	
4	Parse input parameters.	
5	Implement the same interface as the data model. Delegate all methods to the data model class.	
6	Register your class as a listener for data model change. Add proper behaviour for the control.	Register listeners to react on model changes.

4. CONCLUSION

The project proved that splitting the HMI design from application development provides significant advantages. Using externally provided, ready to use and well-tested control sets instead of hand-coding every control for each new application obviously boosts the development process, at the same time making applications much more error-resistant. Moreover, such a framework automatically enforces similarities in HMIs of various applications, both in the sense of HMI look and behaviour. Such similarities help end-users to decrease the learning curve on how to use them; because they are already familiar with interface look and behaviour.

5. REFERENCES

[1] ECSS-E-ST-70-32C, “Test and operations procedure language”, 31 July 2008

[2] ECSS-E-ST-70-31C, “Ground systems and operations – Monitoring and control data definition”, 31 July 2008

[3] ECSS-E-ST-40C, “Space engineering – Software”, 6 March 2009

[4] ECSS-E-70-32C, “Procedure Language for Users in Test and Operations”

[5] Dalmasso, I.; Datta, S.K.; Bonnet, C.; Nikaein, N., "Survey, comparison and evaluation of cross platform mobile application development tools," Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International , vol., no., pp.323,328

[6] Research2guidance, The “Cross Platform Tool Benchmark 2013” report, 16 October 2013