www.czasopisma.pan.pl PAN POLSKA AKADEMIA NAUK www.journals.pan.pl

Metrol. Meas. Syst., Vol. XVII (2010), No. 1, pp. 27–38

PAN POLSKA AKADEMIA NAUK

METROLOGY AND MEASUREMENT SYSTEMS

Index 330930, ISSN 0860-8229

www.metrology.pg.gda.pl

M&MS

ADVANCED USER INTERFACE GENERATION IN THE SOFTWARE FRAMEWORK FOR MAGNETIC MEASUREMENTS AT CERN

Pasquale Arpaia[1,2], Lucio Fiscarelli[1,2], Giuseppe La Commara[1]

1) University of Sannio, Faculty of Engineering, Piazza Roma, 21–I 82100 Benevento, Italy (arpaia@unisannio.it, g.lacommara@gmail.com)

2) CERN, Department of Technology, Group of Magnets Superconductors Cryostats, CH 1211 Geneva 23, Switzerland (✉ lucio.fiscarelli@cern.ch, +41 22 767 1031)

Abstract

A model-based approach, the Model-View-Interactor Paradigm, for automatic generation of user interfaces in software frameworks for measurement systems is proposed. The Model-View-Interactor Paradigm is focused on the "interaction" typical in a software framework for measurement applications: the final user interacts with the automatic measurement system executing a suitable high-level script previously written by a test engineer. According to the main design goal of frameworks, the proposed approach allows the user interfaces to be separated easily from the application logic for enhancing the flexibility and reusability of the software. As a practical case study, this approach has been applied to the flexible software framework for magnetic measurements at the European Organization for Nuclear research (CERN). In particular, experimental results about the scenario of permeability measurements are reported.

Keywords: software measurement systems, magnetic measurements, automatic test equipment (ATE).

1. Introduction

Main advantages of automatic measurement systems are the improvement of the performance/cost ratio and more accurate quality control. In the last years, their high growth rate has pointed out the need for reusability of existing measurement software. Organizing all the software applications for measurement systems in a framework [1] is very useful to decrease the software development cost by increasing the reuse of existing elements and the flexibility of running applications. However, traditional systems built as monolithic entities often have difficulty to exhibit a flexible behaviour.

In a measurement software framework, first of all, the roles of the test engineer and application user have to be highlighted [2]. In the first phase, after designing a measurement process, the test engineer prepares a script where the corresponding procedure is expressed formally, but in a user-friendly language [3]. Then, the framework processes the script suitably by generating an executable measurement software application. In the second phase, the final user executes the software application, interacts with the measurement system by providing the required input as well as by configuring the instruments, and finally starts the measurement process on the devices. The application user needs to interact with the software application through a convenient graphic interface to carry out easily and quickly the measurement procedure. Therefore, the test engineer should deal with its implementation.

Such as most interactive applications, producing an attractive Graphical User Interface (GUI) for a measurement software framework is not an easy task [4].The powerful GUI libraries offered by the operating system can be used of course, but the offered level of abstraction is, in general, rather low. Therefore, a visual editor, such as many commercial

Article history: received on Jan. 31, 2009; received in revised form on Jan. 25, 2010; accepted on Feb. 3, 2010, available online on Feb. 26, 2010; DOI: 10.2478/v10178-010-0003-y.

programming environments, should be used. Such tools turn out to be very user-friendly, even though merging the provided code with the existing one is in general not easy. Moreover, graphical representations depending on run-time data cannot be drawn in advance.

Summarizing, a visual editor is a useful tool for simple applications, but for more complicated GUIs, the test engineer has to struggle still with a low-level programming code. In addition, the quality of manual GUI development depends strongly on the experience of the designers as well as on their skills in the platform and development tools.

For these reasons, user interface generation has been the subject of research for many years, sometimes under the diction of "model-based user interfaces" [4], because interfaces are generated by dividing the application domain in models. The original contribution of this research field was to allow programmers, as test engineers, not typically trained to design interfaces, to produce user interfaces customized to their own applications.

On the other hand, the main feature of automatic techniques for generating interfaces is to allow the designer to specify them at a very high level, with the details of the implementation provided by the system [4]. Nevertheless, this approach is very unspecific and further effort is required to tailor the model to a definite context such as the frameworks for measurement software products. Designing interaction rather than interfaces attempts to enhance the quality of the interaction between user and computer, according to the main paradigm: "user interfaces are the means, not the end" [5].

In the context of measurement software products, LabView [6] is very popular. It is a graphical programming environment used to develop measurement and test systems by using graphical icons and wires well symbolizing the data flow. The approach of the G programming language [7], on which LabView is based, is to emphasize the objects involved in the application and the data exchange among them with less care for the temporal sequence of the actions to be executed. Conversely, in a scientific and academic community, scripting languages are commonly used [8]. They point out the operation's order and allow the temporal constraint in the measurement application to be managed easily [9].

In this paper, an evolution of model-based user interface generation in a measurement software framework based on domain specific language for scripting [3], the Model-View-Interactor paradigm, shifting the test engineer from designing interfaces to designing conceptual interactions, is proposed. In particular, in Section 2, a brief state of art is introduced; in Sections 3 and 4, the proposed paradigm and the corresponding architecture will be treated and, in Section 5, some experimental results and a case study will be shown.

2. Model-based approach

In the model-based approach to GUI generation [4], analysis, design, and implementation are based on a common repository of models. A model is a declarative specification of some single coherent aspects of a user interface, such as the appearance, the interaction with the user and /or the interface with the underlying application feature. By focusing the attention on a single aspect of an interactive system, a model can be expressed in a highly-specialized notation [5]. This property makes systems developed using the model-based approach to be implemented and maintained more easily. Unlike conventional software engineering, where designers construct artefacts whose meaning and relevance can diverge from the delivered code, in the model-based approach designers build models of critical system attributes and then analyze, refine, and synthesize these models into running systems.

Early examples of model-based tools include Cousin [10] and HP/Apollo's Open-Dialogue [11], providing the designer with a declarative language for listing the input and output requirements of the user interface. The system then generated the dialogs to display and request the data. These evolved into model-based systems, such as Mike [12], Jade [13],

Metrol. Meas. Syst., Vol. XVII (2010), No. 1, pp. 27–38

UIDE [14], ITS [15], and Humanoid [16]. These systems exploit specific techniques, such as heuristic rules, to automatically select interactive components, layouts, and other details of the interface.

Generating interfaces automatically is a very difficult task, because automatic and model-based systems constrain significantly the kinds of interfaces they can produce. A related problem is that the generated user interfaces are generally not as good as those created by conventional programming techniques. However, in specific domains with very few particular graphical requirements, such as the measurement domain, automatic techniques can be used.

Model-based GUI generation relies on the principle that development and support environments may be built around declarative models of a system [17]. Developers using this paradigm build the interface by specifying declarative models, rather than writing a program. Generally, for any interactive system, three kinds of models can be derived [18]:

- Presentation models, specifying the appearance of user interfaces in terms of their widgets and the related behavior.
- Application models, specifying the parts (functions and data) of applications accessible from the user interface.
- Dialogue models, specifying end-user interactions, their order, and how they affect the presentation and application.

3. Model-View-Interactor paradigm

In the following, (i) the basic concepts, (ii) the view, (iii) the interactor and (iv) the model of the proposed Model-View-Interactor paradigm are illustrated.

3.1. Basic concepts

The proposed approach to generate interfaces in measurement system frameworks starts from a fundamental consideration: usually test engineers are not trained to design interfaces, but at the same time they would like to maintain a high level of usability in measurement applications.

In this case, test engineers are responsible for preparing test scripts [19], where the interaction between measurement application and final user are described at high level [3], without any indication of GUI aspects.

Therefore, the main concept underlying the proposed approach is the interaction between the user and the GUI. Interaction is a kind of action occurring when two or more objects have an effect upon one another. Examples of simple interactions in measurement software are reading a user input or displaying a value.

Test engineers are prevented from dealing with raw graphical characteristic of software measurement system, by separating functional from look aspects of the interface. Accordingly the architecture is organized by a three-way decomposition: (1) the parts representing the model of the underlying application domain, (2) the way the model is presented to the user, and (3) the way the user interacts with it.

This proposal is called the Model-View-Interactor (MVI) approach [20] (Fig. 1), derived as an evolution of the MVC (Model View Controller) and MVP (Model View Presenter) [21]. The Model represents the model domain, and, in case of measurement software framework, is constituted by the core classes. The views consist in the aspect of the generated GUI, defined by a GUI expert, completely transparent to the test engineer using the framework. In particular, the GUI expert defines a set of presentation models used to generate the final user interface. The interactor represents the tie between model and view, by making available a different component specifying the GUI desired behaviour.

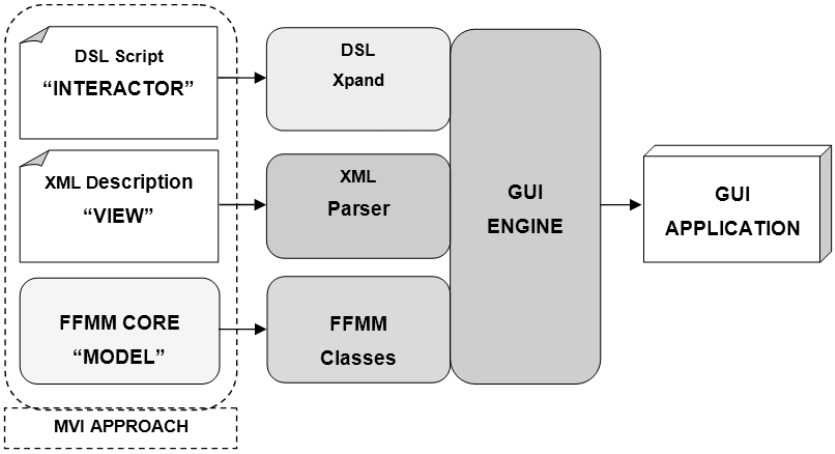

Fig. 1. MVI Architecture.

In this way, the test engineer can define the interaction between the measurement application and the user by means of a set of specific objects: the Graphic Interactor Component (GIC) (Section 3.3).

3.2. View

The view description is a XML-file containing all the presentation features of the GUI. XML stands as a solution for the standardization of the interoperability between applications. Therefore, XML-based languages can be employed to define user interfaces. They are the XML-compliant user interface definition languages (XML-UIDL) [22] and their advantage is to be transparent to different interface technologies and to provide a homogeneous resource for heterogeneous ways of interaction.

Generally, at graphical level, the user interface content can be organized in areas, represented usually by a rectangular bounding shape. These rectangular areas are referred as box. Graphical user interface layouts can be seen as a container subdivided in boxes, where the graphic components (text editor component, buttons, menu item, and so on) can be placed. One box can contain others boxes, and so on.

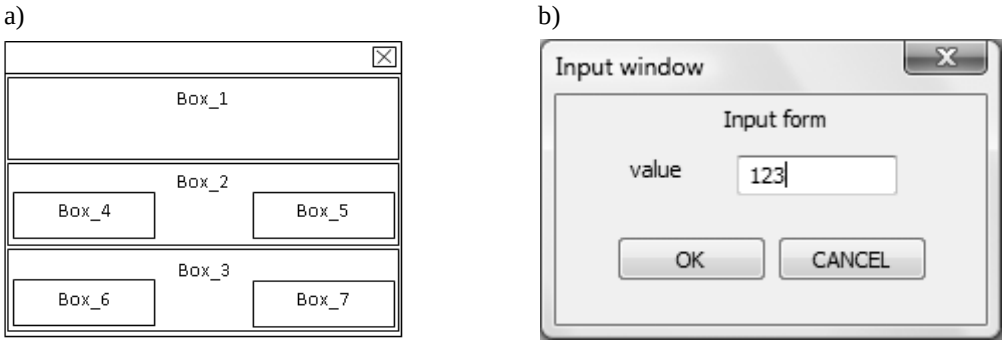

Fig. 2. Example of a view model (a), and the related final form aspect (b).

Two types of boxes can be distinguished: (i) horizontal boxes (HBox), with elements aligned horizontally; and (ii) vertical boxes (VBox): with elements aligned vertically.

As an example (Fig. 2a), the View model used in a form asking for an input value to the application final user, is considered. The layout as a whole is formed by 3 VBox (Box_1, Box_2 and Box_3):

– Box_1 will contain a text component for the title.

Metrol. Meas. Syst., Vol. XVII (2010), No. 1, pp. 27–38

– Box_2 is formed by two HBox: Box_4 and Box_5. The former will contain a text component for a description, and the latter will contain a text editor component to read an input value.
– Box_3 is formed by two HBox: Box_6 and Box_7. Both will contain a button component. Button in Box_6 will command an action to confirm inserted input value, while button in Box_7 will command an action to discard the operation.

End user will see a form with the aspect shown in Fig. 2b.

The View model in Fig. 2a is stored as a declarative model, containing also the information about the GUI component properties (character font, text component colour and text editor component positions, background and foreground colours, as well as further information related to the GUI aspect).

Basically the proposed approach is based on a database of different View models, associated to one or more interactive components, such as shown in the next Section.

```
<?xml version="1.0"?>
<window id="box example" title="Input Form"
      <vbox>
            <staticText id=”description”/>
      </vbox>
      <vbox>
            <hbox>
            <staticText id=”label”/>
            <ctrlText id="control"/>
            </hbox>

      </vbox>
      <vbox>
            <hbox>
            <button id="ok" label="OK"/>
            <button id="cancel" label="CANCEL"/>
            </hbox>
      </vbox>
</window>
```

Fig. 3. View XML description example.

A view description example for a simple window is depicted in Fig. 3. The view description file is written by the application engineer during the software development phase in order to fix the GUI presentation look. Then, at run time, this file is read by the XML-Parser and the information is used by the framework to generate the graphical elements.

3.3. Interactor

The main aim of the proposed Model-View-Interactor paradigm is to permit the test engineer to develop complicated GUI applications, with a minimal effort and moreover without graphical knowledge. This aim is achieved mainly through the Graphic Interactive Component (GIC), from which any customizable graphical component is derived. This component encapsulates all common aspects of graphical components [23, 24]. It is generated automatically for any type T, *e.g.* int, float, double, or more complex data types.

Namely, the GIC_T, defined for the type T, can:

– be used by test engineer to display automatically any value of type T;
– be used by test engineer to plot on screen an array of type T;
– be used by application user to view and edit some value of type T;

– communicate any value change made by the user or by the program to any other depending component.

For any concrete type T, the compiler is able to derive automatically an instance function of this meta-description for the given type.

```
BEGIN SCRIPT " DSL example"
      Def  FDI "FDI1" with(2);
      Def GIC "InputParam";
      Capture InputParam with (2,"Parameter request","FDI bus:");
      BEGIN MTASK "Task1(start_procedure)"
      ...
      END MTASK
      BEGIN MTASK "Task2(flux_measurement)"
      ...
      END MTASK
END SCRIPT
```

Fig. 4. DSL script example.

The test engineer, in the Domain Specific Language (DSL) script writing phase [3] (Fig. 4), defines the component contained in the GUI and their input/output data by using the GIC components. Then, after building the script by means of the DSL-Xpand component, the framework can generate the application with the desired GUI.

3.4. Model

The model is composed by the data structures and the classes of the framework involved in the GUI generation and subject to change by them. A typical example is offered by the device classes concerned to the configuration step of the measurement procedure. During this phase, a broad interaction whit the user is required to set up the devices, the data needed to the configuration are structured in the class definition where the data variables are preset for type and number by the application developers.

4. The GUI engine

The classes architecture allowing the automatic user interface generation is named GUIengine and is shown in Fig. 5.

The GUI engine is composed by several classes: (i) GIC, providing to TestManager the input/output features without graphical details; (ii) GenericWindow, giving the interface for all the frames; (iii) InputWindow, OutputWindow, and PlotWindow, the concrete windows; and (4) LayoutManager, responsible for instantiating concrete windows defining the graphical features parsing the view description file and computing the dimension and position parameters [25]. The View is kept clear-cut from the Interactor by implementing the GUIengine complying with the abstract factory pattern, often employed to separate the details of GUI implementation from its general use.

As an example, if the test engineer needs for asking as input an integer value at runtime, he will use the capture method of a GIC object in the measurement script [19]:

– Def GIC "InputParam".

Metrol. Meas. Syst., Vol. XVII (2010), No. 1, pp. 27–38

– Capture InputParam with (param,1, “Input form”, “value”).

By inserting in the script only these instructions, a form is displayed, and the value entered by the user is stored in the variable pointed.

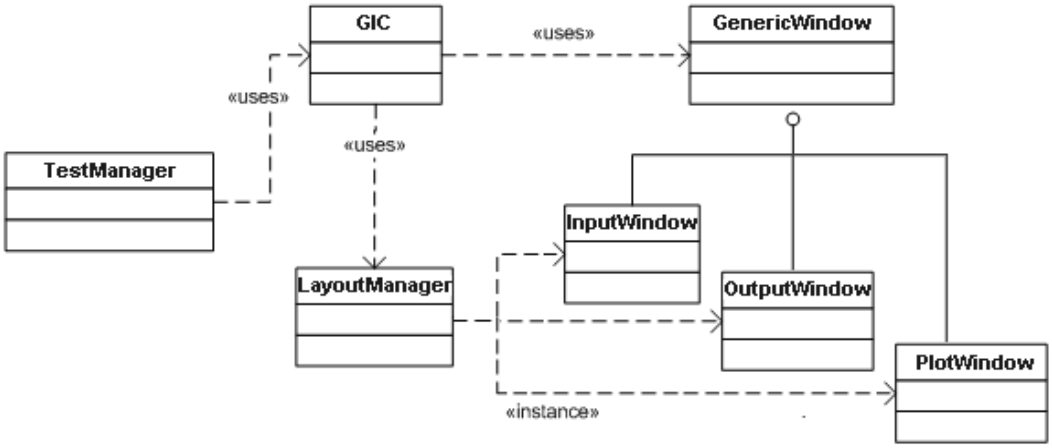

Fig. 5. Abstract Factory Pattern for the GUI engine.

5. Experimental results

At CERN, the European Organization for Nuclear Research, the design and the implementation of the LHC (Large Hadron Collider) required a big effort in all the engineering fields. In particular, the test of the about 8,000 LHC superconducting magnets working at 8.3 T and 1.2 K, stimulated new stricter requirements for magnetic measurement software. The Flexible Framework for Magnetic Measurements (FFMM) was designed at CERN in cooperation with the University of Sannio [19] to satisfy a wide range of measurement requirements and to integrate more performing flexible hardware [26]. FFMM is a software platform under development aimed at generating in a systematic way all the measurement software applications for testing the particle accelerator magnets.

FFMM software applications can command several devices, such as encoder boards, digital integrators, motor controllers, transducers, and so on [2]; and synchronize and coordinate different measurement tasks and actions [27, 28]. Now, such as the new generation of measurement frameworks, FFMM in addition to satisfy all the functional requirements can provide means to generate the graphical user interface.

In the following, (i) the case study of magnetic permeability measurement, and (ii) the measurement results are reported.

5.1. Case study of magnetic permeability measurement

The proposed case study is aimed at illustrating how the proposed approach supports test engineers in generating the GUI automatically for a measurement procedure based on the methods of the split-coil permeameter [29].

The split-coil permeameter is composed by two coils wound in a toroidal shape, that can be opened allowing to wrap a toroidal specimen. One coil is to excite the field and the other one to capture the flux.

A PC, hosting the FFMM Automatically-generated User Interface (AUI), is linked to a DAQ [30], in order to control the voltage controlled power supply of the excitation coil by the analog output (Fig. 6).

A PXI crate, containing:

– two CERN FDI (Fast Digital Integrator), a CERN proprietary PXI board general-purpose digitalization board, configured for the coil signal acquisition and numerical integration [26];
– a CERN encoder board, a CERN proprietary PXI board, for managing the encoder pulses and feeding the trigger input of a digital integrator,

is also linked to the PC and used to acquire, through the FDIs, the value of the excitation current, the relative flux, and to generate the trigger signal by the encoder board.

The measurement algorithm is composed by the following steps:

1. setup of all the devices needed in the procedure;
2. demagnetization of the specimen [29];
3. start the acquisition of flux and current;
4. start the generation of the signal controlling the power supply;
5. wait for the reaching of the selected maximum current value;
6. stop the acquisition.

This procedure is codified in the application script and processed by the FFMM framework in order to produce an executable file.

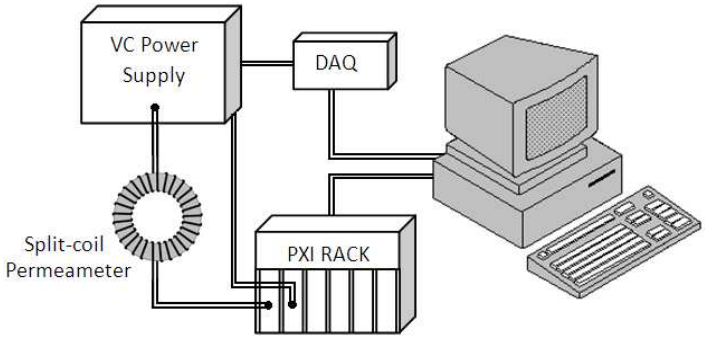

Fig. 6. The split-coil permeameter measurement setup.

5.2. Measurement results

To set up the devices involved in the measurement procedure, the AUI features of FFMM are used.

As an example, at the beginning of the measurement script, the test engineer needs to configure the FDIs: the number of FDI and their bus are required to start the acquisition. Thus, the test engineer puts in the script the following statements:

– Def GIC “InputParam”.
– Capture InputParam with (numFDI 1, “Parameter Request:”, “number of FDI”).
– Capture InputParam with (bus, numFDI, “Parameter Request:”, “FDI bus”).

Then, during the application execution, the forms are generated (Fig. 7).

As a further example, during the measurement, the test engineer can program the application to show to the user the current flow by using the plot feature of the GIC object. Thus the following statements have to be placed in the script:

– Def GIC “CurrentPlot”.
– Plot CurrentPlot with (currentData).

During the application execution, the window with the plot is generated as shown in Fig. 8.

A steel specimen was tested and, according to the procedure explained in [29], the permeability characteristic curve may be obtained by analyzing the data.

For validating the AUI by Model-View-Interactor paradigm, in Fig. 9, the relative permeability magnetic versus the magnetic field curve is reported.

Metrol. Meas. Syst., Vol. XVII (2010), No. 1, pp. 27–38

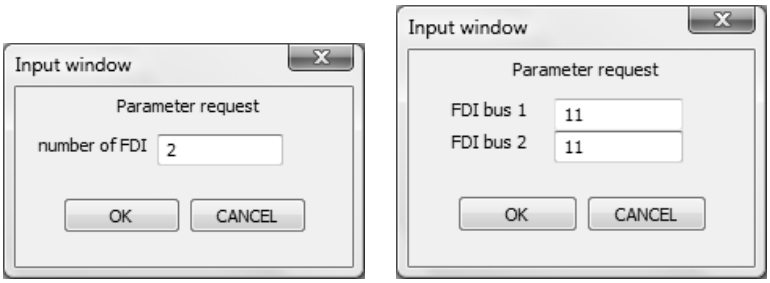

Fig. 7. FDI configuring forms.

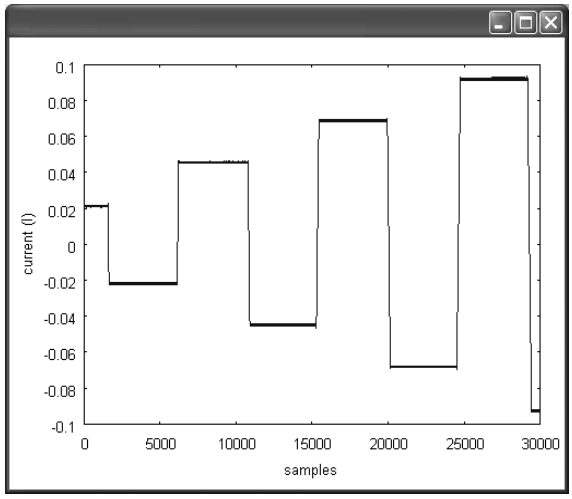

Fig. 8. A window plotting some current cycles.

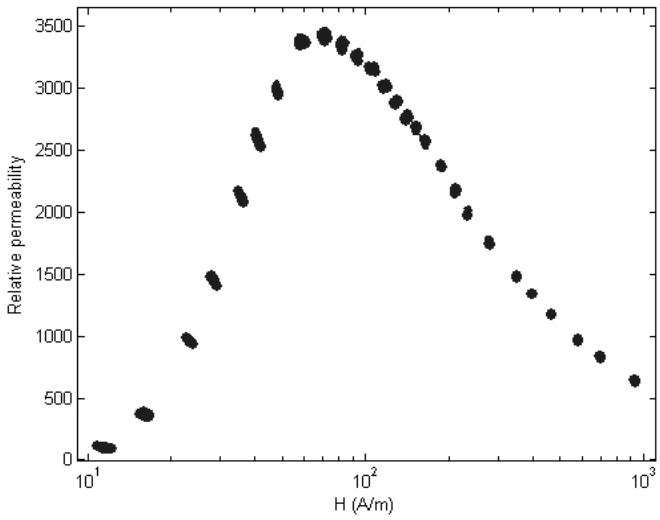

Fig. 9. Relative permeability vs. magnetic field curve.

6. Conclusions

The Model-View-Interactor paradigm for Automatically-generated User Interface (AUI) was proposed. The main purpose of this technique was to allow test engineers using the Flexible Framework for Magnetic Measurements (FFMM) to produce easily the GUI for their measurement applications. The architecture of the proposed method was extensively

explained by describing each component. A complex and realistic case study, the magnetic permeability measurement, was treated and both the graphical and the measurement results were shown. The advantages of the proposed technique meet the requirements of software framework for measurements systems and furthermore agree with their basic idea, primarily by decreasing the performance/cost ratio of the application development even with a graphical interface.

References

[1] Bosch, J. (1999). Design of an Object-Oriented Framework for Measurement Systems. In Fayad, M., Schmidt, D., Johnson, R. (eds.), *Domain-Specific Application Frameworks*, 177–205, John Wiley.

[2] Arpaia, P., Bottura, L., Inglese, V., Spiezia, G. (2009). On-field validation of the new platform for magnetic measurements at CERN. *Measurement*, 42(1), 97–106.

[3] Arpaia, P., Buzio, M., Fiscarelli, L., Inglese, V., La Commara, G. (2009) Measurement-Domain Specific Language for Magnetic Test Specifications at CERN. In *Proceedings of I2 MTC 09*. Singapore.

[4] Mayers, B., Hudson, S.E., Pausch, R. (2000). Past, Present and Future of User Interface Software Tools. *ACM T. Comput-Hum. Int.*, 7(1). 3–28.

[5] Beaudouin-Lafon, M. (2005). Interactions as First-class Objects. In *Proceedings of the ACM CHI 2005 Workshop on the Future of User Interface Design Tools*. ACM Press.

[6] *http://www.ni.com/labview/whatis/*

[7] *http://www.ni.com/pdf/manuals/321296b.pdf*

[8] *ftp://ftp.ni.com/pub/devzone/pdf/tut_7671.pdf*

[9] Arpaia, P., Fiscarelli, L., La Commara, G., Romano, F. (2010). A Petri net-based software synchronizer for automatic measurement systems. In press on *IEEE Trans. Instr. Measur*.

[10] Hayes, P.J., Szekely, P., Richard, A. (1985). Design Alternatives for User Interface Management Systems Based on Experience with COUSIN. In *Proceedings of the ACM CHI 85 Human Factors in Computing Systems Conference*. San Francisco, California, 169–175.

[11] Schulert, A.J., Rogers, G.T., Hamilton, J.A. (1985). ADM-A Dialogue Manager. In *Proceedings of SIGCHI' 85, CA, Human Factors in Computing Systems*. San Francisco, 177–183.

[12] Olsen Jr., D.R. (1986). The Menu Interaction Kontrol Environment. *ACM T. Graphic.*, 5(4), 318–344.

[13] Vander Zanden, B., Myers, B.A. (1990). Automatic, Look-and-Feel Independent Dialog Creation for Graphical User Interfaces. In *Proceedings SIGCHI'90, Human Factors in Computing Systems*. Seattle, 27–34.

[14] Sukaviriya, P., Foley, J.D., Griffith, T. (1993). A Second Generation User Interface Design Environment: The Model and The Runtime Architecture. In *Proceedings INTERCHI'93, Human Factors in Computing Systems*. Amsterdam, The Netherlands, 375–382.

[15] Wiecha, C. *et al.*(1990). ITS: A Tool for Rapidly Developing Interactive Applications. *ACM T. Inform. Syst.*, 8(3), 204–236.

[16] Szekely, P., Luo, P., Neches, R. (1993). Beyond Interface Builders: Model-Based Interface Tools. In *Proceedings INTERCHI'93, Human Factors in Computing Systems*. Amsterdam, The Netherlands, 383–390.

[17] Browne, T.P., *et al*. (1997). Using declarative descriptions to model user interfaces with MASTERMIND. Paternò, F., Palanque, P. (eds.), *Formal Methods in Human Computer Interactions*. Springer-Verlag.

[18] Stirewalt, K., Rugaber, S. (1998). Automating UI Generation by Model Composition. Submitted to *Automated Software Engineering, ASE '98, 13th IEEE International Conference.*

[19] Arpaia, P., Bottura, L., Buzio, M., Della Ratta, D., Deniau, L., Inglese, V., Spiezia, G., Tiso, S., Walckiers, L. (2007). A software framework for flexible magnetic measurements at CERN. In *Proceedings of IEEE IMTC 07*. Warsaw, Poland.

Metrol. Meas. Syst., Vol. XVII (2010), No. 1, pp. 27–38

[20] Arpaia, P., Buzio, M., Fiscarelli, L., Inglese, V., La Commara, G. (2009). Automatically-generated user interfaces for measurement software frameworks: a case study on magnetic permeability at CERN. In *Proceedings of XIX IMEKO World Congress, Fundamental and Applied Metrology*. Lisbon, Portugal.

[21] Krasner, G., Pope, S. (1988). A cookbook for using the Model-View-Controller user interface paradigm in Smalltalk-80. *Journal of Object-Oriented Programming*, 1(3). 26–49.

[22] Abrams, M., Phanouriou, C., Batongbacal, A.L., Williams, S., Shuster, J.E. (1999). UIML: An Appliance-Independent XML User Interface Language. In *Proceedings of the Eighth International WWW Conference*. Toronto, Canada.

[23] Achten, P., van Eekelen, M., Plasmeijer, R. (2004). Compositional Model-Views with Generic Graphical User Interfaces. *Practical Aspects of Declarative Programming, PADL04*, 3057 of LNCS.

[24] Achten, P., van Eekelen, M., Plasmeijer, R. (2003). Generic Graphical User Interfaces. *Selected Papers of the 15th Int. Workshop on the Implementation of Functional Languages, IFL03*, 145 of LNCS, Edinburgh, UK: Springer.

[25] Lutteroth, C., Weber, G. (2008). Modular Specification of GUI Layout Using Constraints. In *Proceedings of ASWEC 2008 - 19th Australian Conference on Software Engineering, IEEE Press*.

[26] Arpaia, P., Masi, A., Spiezia, G. (2007). A Digital Integrator for Fast Accurate Measurement of Magnetic Flux by Rotating Coils. *IEEE Trans. Instr. Measur.*, 56(2).

[27] Arpaia, P., Bernardi, M., Di Lucca, G., Inglese, V., Spiezia, G. (2010). An Aspect Oriented Programming-based approach to software development for measurement system fault detection. In press on *Comput. Stand. Inter*.

[28] Arpaia, P., Bernardi, M., Di Lucca, G., Inglese, V., Spiezia, G. (2008). Aspect Oriented-based Software Synchronization in Automatic Measurement Systems. *Instrumentation and Measurement Technology Conference Proceedings*, 1718–1721.

[29] Henrichsen, K.N. (1967). Permeameter. *Proc. 2nd Int. Conf. on Magnet Technology*. Oxford.

[30] *http://sine.ni.com/nips/cds/view/p/lang/en/nid/1037*