BUILDING EFFECTIVE GUI PANELS

G. Pajor*, J. Dovc, M. Kadunc, J. Kamenik, I. Kriznar, N. Oman, M. Plesko, G. Tkacik, I. Verstovsek, J. Stefan Institute and Cosylab Ltd.

Abstract

In this article we present our basic GUI components and how we build GUI panels using visual and conventional programming. The new additions to the GUI components are discussed in light of their flexible design.

Building an application as a front-end for control system is a responsible task that merges end-users' expectations of easy-to-use tool and developers' skill to find the right solutions. The most efficient way of writing applications that have certain similarities is obviously to create re-usable components. These components should be generic enough to fit into wide array of possible application specifications. On the other hand it should be possible to meet the specifications completely with very little additional coding.

Re-usable components in ANKA accelerator project case were designed at the time of the project and were a success by itself. About two dozens applications were built and virtually no application-based adjustments have been made to the components due to their generic nature.

The slightly different aspect of our past work was presented to us as we began working on another project, beamline control application for Japanese commercial customer. We built the graphical user interface for the application and upon received screenshot they developed a quite clear idea of what the application should look and behave like. Their specifications went beyond the implemented functionality of components at that time. Due to good engineering of the components we added the requested functionality without problems.

Decision should be made carefully which of the new features can be implemented generically enough to put them into the re-usable components instead of implementing it in application itself.

1 RE-USABLE GUI COMPONENTS

Reuse of GUI components is vital for practically any kind of GUI building. They help achieve the consistency of GUI look and feel across the whole project and considerably shorten development time. By providing a set of GUI components, the code maintenance is centralized and, with little forethought and skill, completely application independent.

Some of these components (buttons, text areas, etc.) are typically already available by the programming language vendor. If they are written in an extensible way (such as for example Swing Java Beans components), they provide an excellent foundation for development of new components.

*gasper.pajor@cosylab.com

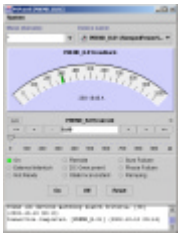

Figure 1: Simple Power Supply Panel

1.1 Java is Hot

Java is great in respect of previous paragraph. We built our *CosyBeans* [4] components (integrated in panels on Figures 1, 2, 3) on the primitive Java components. CosyBeans are re-usable, customizable and very convenient for control system applications. Among the most used (but far from being the only ones) are *Gauger* (Figure 1, second from top), *Slider* (Figure 1, below Gauger) and *Ledder* (Figure 1, below Slider).

Java also provides a modification of MVC (model-view-controller) pattern [7]. This pattern provides separation of visual representation (view), managing of data the component uses (model) and the behavior on user interaction (controller). This results in complete de-coupling the visual and non-visual parts of component. Consequently more than component can use the same model and one component can use more than one model.

Java also features layout managers that manage the correct resizing and positioning of components when resizing the panel and when adding components in runtime. The position of components is therefore not absolute (as is in Visual Basic).

1.1 Need for New GUI Component

There are reasons for starting the development of a new GUI component. Firstly, the idea may arise from the immediate need, since the component has to be used in an application currently being developed. Secondly, a new component may be created because the developers feel that there is a part of functionality in GUI libraries that is missing, although there is no immediate need for the component. We would like to stress this second case, because a lot of our components started out as either prototypes, toy-models or simply exercises for programmers. However, when the first version was developed, it typically soon found its way into the applications.

In the first case, where components are developed for immediate use, care has to be taken in two respects. First and foremost, we must meet the demands of the customer. Before actually developing something new, we investigate if existing components can be extended and used. If not, a new work is started, which leads to the second

requirement: to produce reusable components. In other words, a new GUI component must, apart from satisfying the customer at hand, be appropriate for a range of envisioned uses in the future. It is at this point that the component design and engineering really gets tested...

1.2 Meeting Customer Demands, GUI-wise

The customer is always more or less active participant of the GUI development for ordered applications. If the customer is satisfied with the range and functionality of the existing GUI components, their role in GUI development is minimal, as most work is already done and the components only need to be placed, tuned and connected to whatever runs underneath the GUI.

On the other hand, customer can develop his own ideas for GUI. This normally leads to upgrading and extending the functionality of existing components as well as developing completely new components.

The upgrading of existing re-usable components is always a test of components' engineering design, meaning that the components with added functionality should be similar enough to their previous versions without compromising their functionality in older applications.

Process of enhancing of components on customer demand has to be led firmly on both sides; inside the development team to assure good engineering of the upgrades and towards the customer to prevent excessive work as a result of too loose specification.

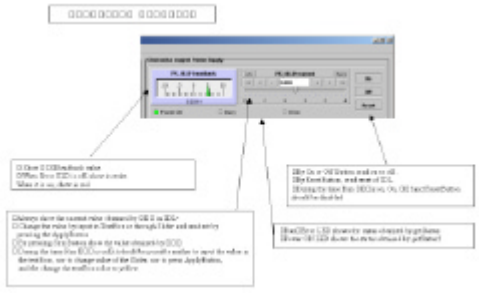

Figure 2: Demo feedback with relevant comments

Demo applications proved to be very useful in process of developing new or extending existing components as the customer can easily determine his preferred way of GUI behavior and functionality (Figure 2). However the excessive use of demos can lead to development deadlock where developers too often wait for customer's feedback. Therefore the right timing of releasing demos is required.

As demos cannot be run with real control system underneath, we developed simulator [6]. *Abeans* [4] layer (our lowest client-side layer) is completely unaware of simulator as simulator behaves just like the real control system.

2 PANEL COMPOSING

In general, GUI can be composed either by visual building, conventional programming or combination of both techniques.

2.1 Visual Building

Visual editor can provide great assistance when composing a GUI panel, as the placing and most of setting and tuning of components is done with a mouse click. Speed of panel development, automation of some processes and ease of use are the chief attributes of GUI visual building.

Visual editors function as WYSIWYG editors and as such they tend to have some deficiencies. Among the most bothering are:

- ?? When using fast-developing programming languages (such as Java™, where new versions are available practically every quarter) the visual builders as quite complex tools cannot always keep up. Being stuck with an old version of any program, when a new version provides just the right solutions, can of course be very frustrating.
- ?? Code maintenance of visually built panels is often limited to specific visual builder in order to preserve visual builder data (as some extra data for WYSIWYG editing is stored together with the source code).
- ?? All actions can be defined with mouse-drag but excessive use of this can lead to connection chaos (Figure 3), which makes the code maintenance considerably harder.
- ?? Components need to be written by certain specifications to ensure visual builder recognizes them correctly.
- ?? Visual builders usually need a lot of computer resources (RAM and CPU).

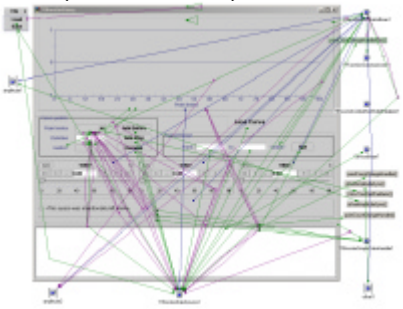

Figure 3: Connection chaos in visual composition

2.2 Conventional Programming

When writing source code in a text editor, none of the visual builder weaknesses affects the developer. However, handwriting GUI takes a lot of time, as the code that would be generated and maintained by the visual composition has to be set-up by hand.

Handwriting the GUI also demands more skilled and experienced developers. As no code is generated by visual builder the developer has full control over the code and the problems can be eliminated more thoroughly if not faster.

Design of the GUI is often better when done by hand as all positioning has to be planned in advance to prevent excessive work when repositioning the components on the panel.

2.3 Best of Both Worlds

Obviously the best way to compose a GUI panel is to use a healthy deal of both methods. Speed and visual representation of visual builders is effective way to position the components when the details and connections to underlying code should be done manually to avoid visual builder dependency and most of other visual builder related problems. Code maintenance can also be much easier when code is not generated by visual builder.

For fast development of relatively simple application panels the visual editor alone is sometimes sufficient.

3 REAL WORLD EXAMPLE OF A GUI PANEL DEVELOPMENT

At Cosylab Ltd. we prepared the presentation of our work, both GUI and lower layers of control system, for potential customers in the Japanese market. Attending representatives of one of the Japanese companies showed immediate interest and their first order was a couple of applications along with GUI and the interface layer (Abeans) for communication with control system.

The applications we built went trough the normal process of the following stages: meeting specifications, demo, meeting additional specifications and feature requests, testing, bugifx, writing documentation and final release, but it was the way we handled the process and the tools we used that made the difference.

3.1 Specifications and Feature Requests

All specifications and feature requests were sent in form of screenshots application with feature request text connected to the respective components (Figure 1). This was one of the best ways to specify all the GUI requirements clearly and without ambiguity.

3.2 Coding

Applications were built on Abeans platform. Re-usable components were placed and tuned using the visual editor of IBM Visual Age while the non-visual part was done completely by-hand.

3.3 Testing, Simulator and Bugfix

The real thorough tests were done using our simulator. The testing was completed considerably faster and was very efficient. The behavior of application was exactly the same as in real control system and we were able to tune all the responses of the simulator to test different scenarios.

3.4 Documentation and Release

Documentation was written using our XML-XSLT schema, which provides all the necessary functions for documentation writing. Strict XML rules prevent any unwanted declination of documents' formatting and are format-wise quality assurance by themselves.

Release was announced when everything was tested and documented. The release package was published on project's web page on our server.

Note that all communication (from our offer to final release) was done by email on two mailing lists, one internal and one for discussion with customer. No phone calls or meetings were necessary.

4 CONCLUSION

Whereas not much skill is required to make a functional visual application, much skill indeed is required to create a maintainable visual application. If the code is not maintainable (or maintainable enough), the duplication of code presents a considerable if not critical overhead. To assure maintainable code and fully functional products, the following requirements should be met:

- ?? Application should consist of smaller units where re-usable components are basic elements.
- ?? Even if re-usable components have shared functionality, it should be extracted to avoid code duplication.
- ?? Writing documentation for components and applications is essential.
- ?? Quality should be assured by thorough testing. Powerful testing tools are of great assistance at this task.

5 ACKNOWLEDGEMENTS

We would like to thank colleagues at KGB team (J. Stefan Institute) [1] and Cosylab Ltd. [2] for help at developing my programming skills, Mr. Koji Natakani, Mr. Takeshi Nakamura and Mr. Yoshitaka Yamamoto of Hitachi Zosen Corporation, Japan for providing exemplary feedback on GUI part of our collaboration.

Additional word of acknowledgement and recognition goes to developers at eclipse.org [8] for providing exceptional free tool for Java development.

6 REFERENCES

[1] http://kgb.ijs.si/

[2] http://www.cosylab.com/

[3] M. Kadunc et al. "Control System Application Look and Feel Guidelines", PCaPAC 2002, Frascati, October 2002

[4] I. Verstovsek et al. "The New Abeans and CosyBeans: Cutting Edge Application and User Interface Framework", PCaPAC 2002, Frascati, October 2002

[5] G. Milcinski et al. "Developing a Control System from a Divan Bed", PCaPAC 2002, Frascati, October 2002

[6] D. Vitas et al. "A Generic Simulator of Control Systems For Application Development and Testing", PCaPAC 2002, Frascati, October 2002

[7] http://java.sun.com/products/jfc/tsc/articles/getting_started/getting_started2.html

[8] http://www.eclipse.org/