International Journal of Recent Technology and Engineering (IJRTE)
ISSN: 2277-3878 (Online), Volume-8 Issue-2, July 2019

Multiplatform Application User Interface Design Based on Spreadsheet

Noprianto, Benfano Soewito, Ford Lumban Gaol, Bahtiar Saleh Abbas

Abstract*: *User interface plays an important role in an application: it is what users directly see and feel. It is also what programmers spend much time on. Therefore, many tools to ease the user interface development are available. Many of them are part of integrated development environment (IDE) of programming languages. Some of them are more generic and available for a few user interface toolkits or programming languages. However, these tools usually assume their users are targeting specific platform: desktop, web, or mobile application. While generic tools are sometimes generous enough, they usually cannot make it totally independent of underlying toolkit or programming languages. Modeling tools exist, but it means learning something new. This is one of the backgrounds for what we propose in this paper. An application user interface model based on spreadsheet: software packages that have been around for almost four decades. We emphasize multiplatform output: design once and output in several platforms (we provide two: desktop and web; adaptable to more). We do not assume a specific toolkit or programming language, but we still care about styles. Our experiments prove this method is usable and results in quicker user interface prototyping—even for professional programmers—with development time comparison ranging from 26.7% (best) to 80%.

Index Terms*: *user interface, user interface modeling, code generation, spreadsheet, programming language.

I. INTRODUCTION

For applications that are used by humans, the user interface is the user's first impression. This is because users interact with an application through its user interface [1], for example, command-line interface (CLI), text-based user interface (TUI), or graphical user interface (GUI). As the name suggests, it provides an interface between the user and the functionality of the application. For a command-line user interface, users type the command and its argument directly on the command prompt. It may offer auto-completion, but commands still have to be typed. For a text-based user interface, users work with menus, buttons, and other components that are character-based drawn/emulated. For a graphical user interface, those menus, buttons, etc. are drawn graphically. GUI also refers to the usage of graphics, keyboard, and mouse for the interface [1]. It first appeared in 1968 [2] and has been available since about 1985 [3].

As an interface, it hides the complexities from users. The user clicks on a button. What happens internally can be very complex. And the user interface is not just the buttons or the menus. How a component behaves, for example, the disabled and enabled state of a button, is also important. Styles and colors can be applied to put some emphases. The layout of the user interface components is also important. Placing too many buttons on a window may confuse users. After all, one of the goals of GUI is to match the display to the user's semantics [1].

Therefore, programmers working on the user interface usually are aware of user interface guidelines provided for the platform on which the application will run. So they do not have to perform too many experiments only to produce a weird-looking application on that platform. For example, there are GNOME Human Interface Guidelines [4], KDE Human Interface Guidelines [5], and User Interface Guidelines for Android [6].

Still, the user interface needs to be coded. Traditionally, decades ago, programmers drew the components character-by-character or pixel-by-pixel along with their behavior. Then a set of user interface components was available and ready to use. All the programmers needed to do was put the components in their codes. No more custom-drawn buttons.

To the next level. It seemed that writing codes to put some buttons on a window or a frame wasted too much time. So, there were graphical user interface builders—usually part of Integrated Development Environments—where all user interface components are just dragged and dropped. No programming was needed [7]. Designing a user interface was like drawing—but much easier.

However, as technologies advanced, new platforms were born. There were web pages, and then there were web applications. There were mobile-specific web pages, and then there were mobile applications. Programmers are busier when they target more platforms. Single-code base—at the user interface—is sometimes hard, or it is not applicable to produce an application that runs on many platforms. At least, applications would not be able to take advantage of all features offered [8].

Revised Manuscript Received on 30 July 2019.
***** Correspondence Author
Noprianto*, Doctor of Computer Science, Bina Nusantara University, Indonesia.
Benfano Soewito, Computer Science Department, BINUS Graduate Programs, Bina Nusantara University, Jakarta, Indonesia.
Ford Lumban Gaol, Doctor of Computer Science, Bina Nusantara University, Indonesia.
Bahtiar Saleh Abbas, Doctor of Computer Science, Bina Nusantara University, Indonesia.

Certainly, there were many efforts to ease the multiplatform user interface design. That way, a single design can be applied to more than one platform [9]. However, we see this field is still open wide to research.

What we propose is a user interface model design that emphasizes a multiplatform output. Design only once, and codes for multiple platforms are easy to generate because of the simplicity offered. It is simple because we assume no specific user interface toolkit. We never talk about buttons on GTK+, Qt, Windows, or Android. We only talk about buttons—period—and we use spreadsheets.

II. SPREADSHEET

Spreadsheet applications have been around for almost four decades [10]. They are installed on many computers. Spreadsheets are also used in almost all businesses, from simple calculations to complex financial models [11].

What makes spreadsheets interesting? First, spreadsheets provide computational techniques that match users' tasks and shields users from low-level details of programming [12]. When we talk about the content of a cell, for example, we never directly talk about its data type, such as integer or floating number. We just work with numbers as needed. Then we format the numbers to look better and appropriate without caring too much about the details. Similar to programming activities, we can use formulas and call some functions. However, in some spreadsheet applications, many functions are already there and ready to use.

Spreadsheets also are interesting for their table-oriented user interface [12], which is what most interests us. Each cell in a table may contain a value and may have some styles applied. They make a spreadsheet document function as a data source and—at the same time—a simple kind of report styled with colors, borders, and text decorations. Later, in the proposed method, we will discuss why this table-oriented user interface and cell characteristics provide a good foundation for user interface modeling.

In some spreadsheet applications, their functionalities do not stop at the user interface and available functions. They are even programmable via macros. Some spreadsheet applications support embedded programs written in specific programming languages [11]. Test-driven development may also be applied to spreadsheets, for example, using spreadsheet test case application [13].

III. RELATED WORKS

There are many researches on spreadsheets, from spreadsheet usage to spreadsheet as a model for application development. The latter is particularly interesting for us: how a spreadsheet document is used as a model for a non-spreadsheet related application, for example, web applications.

The usage of spreadsheets as models for user interface specification was found in a paper published more than two decades ago. We find it still relevant and interesting because Penguims—techniques described in the paper made use of spreadsheet properties, as much as possible (such as spreadsheet cell value and optional formula)—to model a user interface [14]. Penguims also allowed spreadsheet cells to be collected together into objects. However, we learned that the method used an enhanced spreadsheet model, while our proposed method tried to use a spreadsheet as pure as possible (probably because our case is simpler).

Spreadsheets also are used to model web applications. In fact, in a recent research, a spreadsheet was used to drive a read-write-compute web page. To describe the client user interface, a spreadsheet model was used. The paper presented a system named Quilt, for rapid prototyping of applications [15]. However, as our target is on multiple platform outputs, and not only web pages, we try to use spreadsheets without any specific web-related technologies. Certainly, we do agree that spreadsheets offer models with benefits [15].

Gummy is probably closer to what we try to achieve. It offers multiplatform code generation using a platform-independent representation of the user interface [9], but it is not based on a spreadsheet. As a consequence, it looks closer to real user interface design.

IV. PROPOSED METHOD

We would like to propose the usage of a table-oriented user interface of a spreadsheet in user interface modeling. We will start with spreadsheet documents. A document—also known as a workbook—represents a part of an application. In the context of a web application, it represents a single web page. In a desktop application, it represents a single top-level window. As a file format, we work with an Office Open XML (ISO/IEC 29500) spreadsheet with .xlsx file name extension.

A workbook may consist of many worksheets, but only a few of them are used in this proposed method. The first worksheet represents a user interface layout. The second worksheet holds properties of each user interface component in the layout. The third worksheet should contain related or helper code if any.

A worksheet consists of rows and columns. Columns are named with alphabet characters starting from "A." After "Z" column, the name continues with "AA," and so on. Rows are identified using numbers, starting from 1. The intersection of a column and a row is a cell, named as a combination of column and row name, such as A1. Together they virtually represent a big table.

For the user interface layout, we map this table into a grid-like layout. Therefore, if we have a 10 column x 5 row table, it will map an identical grid on a user interface container. This is not a new concept because almost all modern user interface toolkits support this layout. For example, GTK+ with GtkGrid [16] and Qt with QGridLayout [17]. In that table, we put user interface components into cells.

This way, we never put a component into a fixed, coordinated-based location. Neither the size. That way, if a container, e.g., top-level window ever gets resized, all of the components will also proportionally resize. This method is particularly useful when we talk about how devices vary in screen size or screen resolution. At a higher level, it could also be useful on systems varying in default font or font size.

How do we define a user interface component? We propose a simple, descriptive, text-based definition. For example, to define a label, we simply put the content of a label, such as Hello World text in the A1 cell. No quotes, no curly braces, only plain text. To define a button or another components, we propose the usage of <component_type>, followed with a colon and then followed by a label. For example, to define a button with "save" label, the definition would be button:save, put in a cell and plain text. As we can see, there is no assumption on the underlying toolkit.

We never talk about a button in GTK+, Qt, Tk, Android, or HTML. Some spreadsheet applications support the drawing of user interface components, such as buttons, but our current proposal prefers text-based definition.

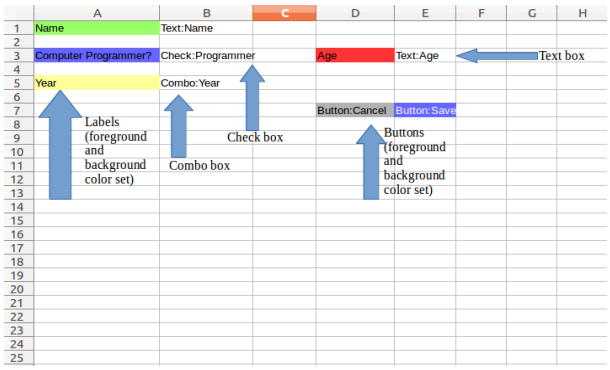

Fig. 1 User Interface Layout (Defined in First Worksheet)

How do we style a component? For example, how do we set the background and foreground color of a button or a label? This is our proposal: because a cell may be styled with colors, we just make use of this feature. To make a button red, just set the background of a cell to red. To color a text green, just set the font color to green. After all, we would like to have a design that is close to the real application. An example of a user interface layout along with styles of components is shown in Fig. 1.

Done with first user interface layout on the first worksheet, we move to the second worksheet where all of the properties are defined. We may easily define the colors of a component—a cell—because it is already supported by the document standard. In our opinion, there is no easy and spreadsheet-like way to define how to fill a combo box dynamically, or how to set the max length of a text entry. This is the reason we propose the usage of a dedicated worksheet.

Let us start with how to identify the component. Because we prefer not to make it too programming-like, we never assign a component to a variable. Rather, we reuse the value of a cell. If, for example, A1 contains button:save, then we reuse this button:save value in the second worksheet. As a spreadsheet offers visibility of intermediate results [14], we may use optional formulas (as a next improvement).

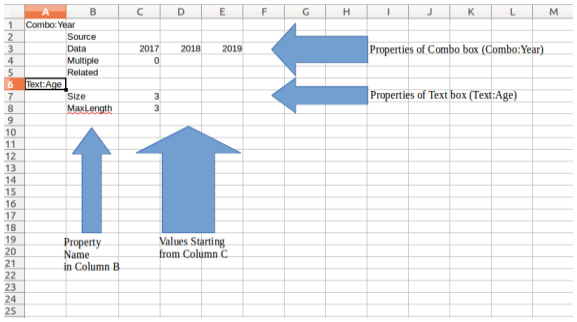

Fig. 2 User Interface Properties (Defined in Second Workseet)

We propose that some rules are needed in this second worksheet. All references to components must be put in column A. Then, all the properties of a component must be put on consecutive rows, with property name put in column B. Values for each property must be put in consecutive columns in its current row. For example, we have a text entry component in the first sheet in cell A1, whose value is text:name. Then, to set the maximum length property, in the second sheet in cell A1 (column A), we reuse the text:name. We then put maxlength (property name) in cell B2. If we want to set the length to the maximum of five characters, then we put 5 (property value) in cell C2. If we ever need to put another properties, then we will start at cell B3, and so on. An example of user interface properties definition is shown in Fig. 2.

In our default implementation, all of these properties are read when we read the layout. Therefore, we may freely put the properties definition not in order they are defined in the layout. If we define a component and put it in A1 cell in first worksheet, we do not have to put the properties starting in A1 cell in second worksheet. As long as the properties are put in consecutive rows, then we are fine.

Before coming up with these rules, we initially think about adding another colon in component definition. That is, kind of text:name:5 value for a cell. In our experiments, however, we found that this was relatively confusing and some of our respondents found this too technical. We agreed, because of the lack of meaning for such "5" value. To make it worse, if we continue with this syntax, we may end up with another colon.

Finally, when designing a user interface, one may add some comments. What is great about spreadsheet-based modeling is that we can add comments in form of drawings, as shown in Fig. 1 and Fig. 2. We added some arrows and text boxes as comments without affecting the validity of user interface model and generated codes.

Proposed method may also be useful for end-user programmers who write codes primarily for personal use [18]. This is particularly true when it is combined with a declarative programming language. In such language, programmers specify the properties of the problem and the expected solution, not how to obtain the solution [19].

V. IMPLEMENTATION

Our implementation provides two outputs: a HTML page and a graphical python application using Tk toolkit (runnable in multiple operating systems). Tk is a user interface toolkit that is part of Tcl programming language. Python standard library comes with interface to Tk in module Tkinter (for Python version 2.x). The code generator implementation itself is also in Python (using Openpyxl library to work with an Office Open XML spreadsheet), released as free/open source software and may be downloaded from http://noprianto.com.

Currently, we support only few user interface components: label, button, check box, combo box, and text entry. For the latter two components, we support few properties also: static values for combo box and maximum length for text entry (more are prepared, but were not interpreted).

Below are steps performed in the implementation at the outer most level:

- Independently of output target, we get the boundaries for layout table in the first worksheet. To make it flexible, we do not restrict the size of the layout table. Assuming that there are some data in the worksheet, we get the highest non-empty column and row.
- Then, for each row in the table:
  - For each column in that row:
    - We get the reference to the cell, and interpret both the value and the styles. The interpretation will vary depending on the output.
- Write the generated codes.

In this implementation, HTML layout is done using table, for the sake of compatibility and simplicity. There is alternative, based on cascading style sheet [20], but at this version, we prefer table to make it displayable in more web browsers, including text-based ones. Labels are mapped to HTML text labels, each enclosed in . Buttons, check boxes, and entries are mapped to <input>. Combo boxes are mapped to <select> and <option>. All styles are mapped to in-line cascading style sheet using color and/or background-color. Example of HTML output for layout defined in Fig. 1 is shown in Fig. 3.

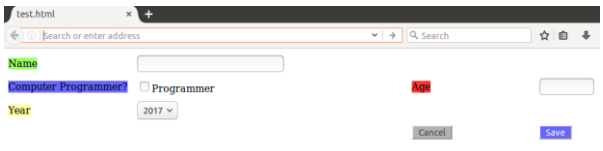

Fig. 3 Output as HTML (test.html)

For Python GUI application, more challenges were faced due to the lack of variable names, as stated in the proposed method. Therefore, we maintain a list of symbols, use generic variable name with a counter (var_0, var_1, and so on), and map them with each user interface component. We put some comments in the source code for clarity. For styling in Tk, we use fg, activeforeground, bg, and activebackground keys. For layout, we use grid layout management. This is perfectly aligned with our spreadsheet-based model. Example of Python application output for layout defined in Fig. 1 is shown in Fig. 4.

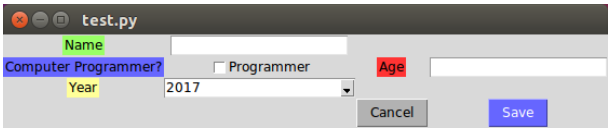

Fig. 4 Output as Python Application (Using Tk Toolkit)

VI. EXPERIMENTS AND RESULT

We would like to measure the time needed to develop applications for at least two platforms—web and desktop—and we focused only at user interface. We argue that our proposed method provides much quicker time for prototyping user interfaces. A simple, trivial user interface with some components was provided (as shown in Fig. 3 or Fig. 4) and respondents were free to choose any tool or method, as long as able to produce similar user interfaces and based on programming activities.

There were eight respondents with various backgrounds on software development. To make it simpler, we grouped the respondents into two groups: (A) those who familiar with user interface development for multiple platforms and (B) those who are not. However, all of them were familiar with at least one modern spreadsheet application, i.e., spreadsheet application that is runnable on current available and supported operating systems (as of this writing). Finally, no specific toolkits, tools, or programming languages were assumed.

Experiments were done in two rounds. First, they were using their own preferred tool or method, then continued with our proposed method. Details are listed in Table 1.

Table 1. Time Needed to Develop Multiplatform User Interfaces

Respondent	*Respondent Group*	*Total time needed (minutes)*		*Comparison (percent)*
		Their own method or tool	*Proposed User Interface Model*	
1	(A) Familiar	25	20	80.0%
2	(A) Familiar	40	23	57.5%
3	(A) Familiar	38	15	39.8%
4	(A) Familiar	50	24	48.0%
5	(B) Not familiar	105	28	26.7%
6	(B) Not familiar	65	30	46.1%
7	(B) Not familiar	90	29	32.2%
8	(B) Not familiar	48	30	62.5%

From the result, we concluded that all respondents spent fewer minutes when using our proposed method to develop two user interfaces,

for web and desktop applications, compared with their own preferred tool or method. There were no differences between two respondent groups. All of them showed similar results, with comparison ranged from 26.7% (best) to 80%.

VII. CONCLUSIONS

Based on the experiments and result, our proposed method is usable and useful in at least user interface prototyping, for multiplatform applications. However, we do aware of its limitations.

First, the limitation would be cell merging. Most spreadsheet applications support cell merging feature. Merged cell should correspond with cell merging or spanning in user interface grid. Then, we never discuss gravity of user interface components (when their container is resized). We were thinking about text alignment—both horizontal and vertical—in a spreadsheet cell, but have no definitive proposal. In addition, an initial disabled or enabled state would be nice if can be emulated using styles, but what kind of styles then?

By releasing our implementation as free/open source software, we hope that it may be useful. We also will be happy to see any further improvements and/or developments.

REFERENCES

1. J. W. Chen and J. Zhang, "Comparing text-based and graphic user interfaces for novice and expert users," *AMIA Annual Symposium Proceedings* (pp. 125). *American Medical Informatics Association.* 2007.
2. J. Reimer, "A History of the GUI," *Ars Technica*, *5*. 2005.
3. B. V. Schooten, "Development and specification of multimodal and multi-user virtual environments". 2002.
4. GNOME Human Interface Guidelines. Available: https://developer.gnome.org/hig/stable/
5. KDE Visual Design Group/HIG. Available: https://community.kde.org/KDE_Visual_Design_Group/HIG
6. User Interface Guidelines. Available: https://developer.android.com/guide/practices/ui_guidelines/index.html
7. V. Aleven, B. McLaren, J. Sewall, and K. Koedinger, "The cognitive tutor authoring tools (CTAT): Preliminary evaluation of efficiency gains," *Intelligent Tutoring Systems* (pp. 61–70). Springer Berlin/Heidelberg. 2006.
8. A. I. Wasserman, "Software engineering issues for mobile application development, " *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research* (pp. 397–400). ACM. 2010.
9. J. Meskens, J. Vermeulen, K. Luyten, and K. Coninx, "Gummy for multi-platform user interface designs: shape me, multiply me, fix me, use me," *Proceedings of the working conference on Advanced visual interfaces* (pp. 233–240). ACM. 2008.
10. D. J. Power, "A brief history of spreadsheets," Available: DSSResources.Com. 2004.
11. R. Abraham, M. Burnett, and M. Erwig, "Spreadsheet programming, " *Wiley Encyclopedia of Computer Science and Engineering*. Redmond, Washington: Microsoft Corporation. 2009.
12. B. A. Nardi and J. R. Miller, "*The spreadsheet interface: A basis for end user programming, "* Hewlett-Packard Laboratories. 1990.
13. Noprianto, B. Soewito, F. L. Gaol, and S. W. H. L. Hendric, "Simple spreadsheet test case application to test spreadsheet formula in end-user software engineering." *The Eleventh 2016 International Conference on Knowledge, Information and Creativity Support Systems* (KICSS). 2016.
14. S. E. Hudson, "User interface specification using an enhanced spreadsheet model," *ACM Transactions on Graphics (TOG)*, pp. 209–239. 1994.
15. E. Benson, A. X. Zhang, and D. R. Karger, "Spreadsheet driven web applications," *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology* (pp. 97–106). ACM. 2014.
16. GtkGrid. Available: https://developer.gnome.org/gtk3/stable/GtkGrid.html
17. QGridLayout Class. Available: http://doc.qt.io/qt-5/qgridlayout.html
18. A. J. Ko, R. Abraham, L. Beckwith, A. Blackwell, M. Burnett, M. Erwig, and M. B. Rosson, "The state of the art in end-user software engineering," *ACM Computing Surveys (CSUR)*. *43*(3). 2011.
19. M. Hanus, "Multi-paradigm declarative languages," *International Conference on Logic Programming* (pp. 45–75). Springer, Berlin, Heidelberg. 2007.
20. CSS/Properties/float. Available: https://www.w3.org/wiki/CSS/Properties/float

AUTHORS PROFILE

Noprianto is a software engineer in Jakarta, Indonesia. He has written several computer books including Python (2002), Debian GNU/Linux (2006), OpenOffice.org (2006), Java (2018), wxWidgets, SQLiteBoy, and OpenERP. Noprianto has a bachelor's degree in Computer Science (2003) and a master's degree in Management of Information Systems (2015), both from Bina Nusantara University. He is currently a doctoral student in computer science, with research interest in programming language. He has certifications in Java (OCP).

Benfano Soewito received B.Sc. degree on Physics from Airlangga University, Surabaya, Indonesia on January 1991. He received M.Sc. and Ph.D. degree from the Electrical and Computer Engineering, Southern Illinois University, Carbondale, USA in 2005 and 2009 respectively. He is currently an Associate Professor at Computer Science Department, Bina Nusantara University, Indonesia. His research interests include Information Security, Sensor Network, and Mobile Applications.

Dr. Ford Lumban Gaol is currently Associate Professor Informatics Engineering and Information System, Bina Nusantara University. He is the Deputy Chair of Bina Nusantara University Doctorate Program in Computer Science and Research Interest Group Leader "Advanced System in Computational Intelligence & Knowledge Engineering" (IntelSys). Dr. Ford is the Vice Chair of IEEE Indonesia section on the period of 2013-2016. He is the Director of Region Office, Co-Chair, IIAI Southeast region (Indonesia Office). Dr. Ford was the ACM Indonesia Chapter Chair on year 2012. Dr. Ford already involved with some projects related with Technology Alignment in some of multinational companies as well as some government projects. For international highlights, Dr. Ford is the recipient of Visiting Professor in Kazan Federal University, Russia 2014 and 2015, Visiting Professor in Vladimir State University, Russia 2016, Visiting Professor in Financial State University 2017, and Southern Federal University, Russia 2018. He is also Visiting Researcher in Advanced Institute of Industrial Technology, Tokyo Metropolitan University, Japan 2018 and 2019. He was Invited Scholar in Aligarh Muslim University, keynote speaker in ICCNT 2014 and Invited Scholar in ICTP Trieste Italy. He has 164 papers that indexed in SCOPUS and 12 books that published by Springer-Verlag German. He was General Chairs or Co-Chairs for some International conferences and IEEE conferences including ACIIDS, Tensymp, and Tencon. Dr. Ford is member of Indonesian Mathematical Society (IndoMS), The Association for Computing Machinery (ACM) Professional, The International Association of Engineers (IAENG), and the Indonesia Society for Bioinformatics. He holds the B.Sc. In Mathematics, Master of Computer Science, and the Doctor in Computer Science from the University of Indonesia, Indonesia in 1997, 2001, and 2009, respectively.

Prof. Ir. Bahtiar Saleh Abbas, M.Sc., Ph.D is a Professor of Operation Research at Bina Nusantara University. He is currently the Head of Doctor of Research in Management. He has a Ph.D in Industrial Engineering and an M.Sc in Statistics, both from Iowa State University. Before he joined Bina Nusantara University, he was President Director of PT Apperindo Agro Konsul, Operations Director of PT Aneka Pionir Pertanian Indonesia, and a researcher at Indonesian Planters Association for Research and Development (IPARD). He has published 15 Scopus indexed joint papers.