VMer – Visualized Mobile Designer for Applications on Small Devices

Chia-Feng Lin , Tzu-Han Kao and Shyan-Ming Yuan
Department of Computer Science
National Chaio Tung University
1001 Ta Hsueh Road, Hsinchu, Taiwan 300, ROC
Taiwan

Abstract: - Currently, the mobile application can be classified into two types (mobile application and mobile Web application). The former can be directly executed in the devices. The later is mobile Web application which is executed through an embedded mobile Web browser. At this time, there are no toolkits capable of developing both types of applications through authoring a single generic application interface. So we propose a visualized toolkit within an integrated development environment and discuss the design issues. The toolkit can create the generic application interface simply through intuitive drag-and-drop generate to two types of applications through transformation style sheet. Developers can utilize this integrated toolkit to compile or test both types of applications to save development time and reduce development efforts

Key-Words: - Mobile PUML XML Transformation Toolkit Authoring

1 Introduction

In order to provides the ability to write once and generate both offline mobile applications and online mobile Web applications. Fig.1. draws a picture about this concept. The single generic application interface is based on the XML-based mobile application development kit from our laboratory, it proposed [1] PUML (Pervasive User-interface Markup Language) as the user-interface transformation matrix, our toolkit can therefore adopt this language to achieve the goal of writing once, and generating both types of applications.

PUML is an XML-based language which describes a generic user-interface for the mobile application in the abstract level. It can be transformed into various languages by using multiple XSLT style sheets[2]. The target languages currently are manipulated in [3] XHTML-MP, WML, and J2ME MIDP, and they are all executable in the mobile environment. Based on the PUML transformation framework, visualizing the user-interface presentation of PUML and providing an integrated development environment will be targeted in this paper.

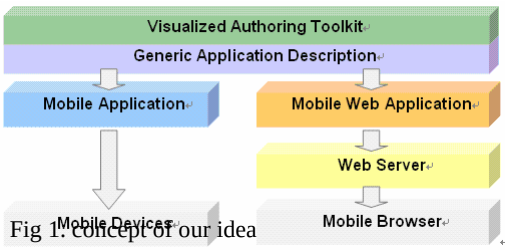

Fig 1. concept of our idea

2 Problem Description

The mobile application development has already been evolved in our laboratory more than three years. One of the development kit proposed by Liu [4] is based on the thin-client platform called ART (Adaptive Remote Terminal). The other kit proposed by [1] is based on the PUML and PGML. However, both kits lack a visualized development toolkit to develop the application user-interface easily and quickly.

As for the academic researches, there are many researches

related to the user-interface transformation mechanism design, however, few researches put their focus on authoring multi-device applications. One of these researches is published in the paper[5]; a Platform-Independent Model for Applications (PIMA) is proposed to adapt the user-interface in both design-time and runtime. It contains a generalization mechanism for extracting a model from device-specific interfaces such as HTML. A specialization mechanism that adapts the application to various target devices automatically is also included.

The research objectives can be categorized into the following four categories:

Rapidly development: Developers therefore do not need to hand-write the PUML source code. The visualized toolkit tries to give developers a “What You See Is What You Get (WYSIWYG)” interface which can be used to generate corresponding PUML source code.

Extensibility: once the PUML specification revises in the future, the extensible toolkit can be updated simply through replacing some components. Moreover, once a new transformation style sheet is released, the extensible toolkit can also add it to generate a new language.

Write once; generate multiple application user-interfaces: Authoring multiple applications with the same functionalities is annoying and

time-consuming, we try to save development time by writing a single generic application based on PUML and then generating the user-interface and logic skeleton of both mobile application and mobile Web application.

3 Problem Solution

Authoring Framework

The workflow of authoring PUML-based applications can be separated into three parts; one is designing the user-interface and defining the usage of the logic objects in PUML. Another is transforming PUML files and generating multiple applications with user-interface files and logic files separated. The other is writing the application logic for each applications being generated. The whole development process is shown in Fig2.

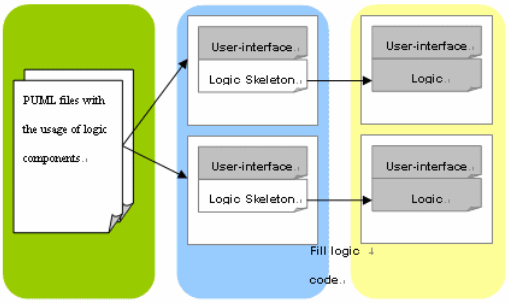

Fig.2. Workflow of building PUML-based applications.

3.1 Overview

Developers can simply drag-and-drop to create a generic user-interface. Moreover, developers can add any simulator to the configuration file and then select one of the simulators to simulate and test the transformed files.

The authoring framework of our toolkit contains designing user-interface, defining logic usage, transforming, programming application logic, and simulating, all of the details will be revealed in the following sections.

3.2 Designing User-interface

The application user-interface is presented in PUML, an XML-based text file. Since it takes a lot of time to hand-write PUML source code, we propose a visualized toolkit to automatically generate PUML source code from editing a generic user-interface to reduce the efforts. A generic user-interface is composed of many widgets which can be dragged from the toolbox and dropped to the canvas. Fig. 3.3. demonstrate the user-interface design environment in JBuilder, the circled part is the proposed toolkit with visualized editing functionality.

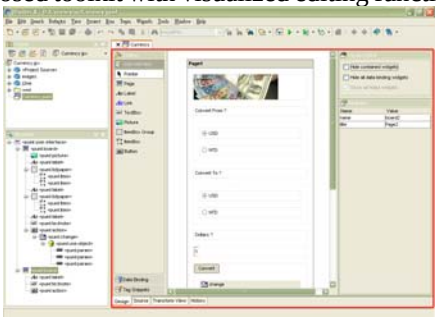

Fig. 3. The user-interface design environment in JBuilder.

3.3 Toolkit Composition

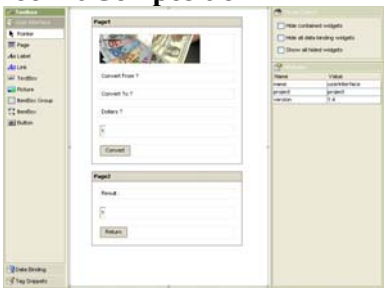

Fig. 4.: The visualized editing environment of the toolkit.

The visualized toolkit depicted in Fig. 4.. is mainly composed of four parts. At the left side is a toolbox containing three different categories as depicted in Fig. . The first category contains widgets for constructing user-interface. The second category contains widgets for binding the user-interface and the data objects. The last category initially contains no widgets; it is designed for developers to add some edited widgets tags. At the center is a canvas where widgets can be added to construct a generic user-interface. At the top-right corner is a visual control pane which contains some controls related to the widget visibility on the canvas. At the bottom-right corner is an attributes pane, it shows editable attributes of the selected widget and organize them in a table for quickly referencing and editing.

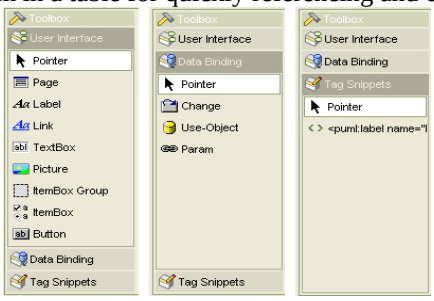

Fig. 5. Three categories of the toolbox.

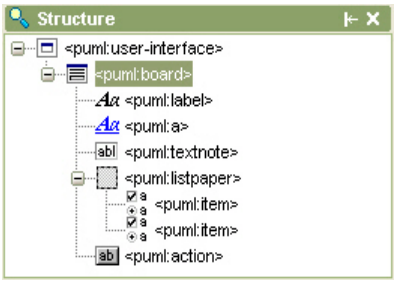

Fig. 6. The structure pane which represents the structure of the source PUML file.

The visualized toolkit has another part resided in the down-left side of Fig.5; it is enlarged in Fig. 6. This part is constructed from PUML source file in a tree structure; this is due to the natural tag-based tree structure of an XML document. Developers who familiar with the PUML tags can quickly reference corresponding widget on the canvas by selecting the node in this tree structure pane.

Visualized Editing

There are plenty of features that we added to facilitate developers to build a user-interface quickly and easily. These features are categorized following based on the control operations:

Drag-and-Drop / Click-and-Drop

Through the mouse actions, developers can easily drag a widget from the toolbox and drop it onto the canvas. Furthermore, if developers want to add a certain widget continuously, the first step is to click a widget he preferred in the toolbox, the second step is to move the cursor to a position in the canvas and click again to add the selected widget on the canvas. Developers can then repeat the second step to add the same widget continuously.

An additional feature in the drag-and-drop is in the opposite way, developers can drag a widget from the canvas to the toolbox. The toolbox contains a category called “Tag Snippets” which is used to hold widgets in the form of PUML tags. Since developers may want to store a widget with modified attributes for using in the next time, tags in the “Tag Snippets” can be saved and manipulated through a popup menu showed by clicking the right mouse button.

Widget Selection

Developers can select a widget on the canvas simply by clicking on the widget. All feasible visual control items will then become selectable at the top-right corner, and all editable attributes of the selected widget will be shown in the attribute editing table at the bottom-right corner.

Visualized Control

The visualized control includes three basic controls over showing/hiding widgets on the canvas. The first control controls if a selected widget should display all of the widgets it contains. The second one controls if all of the data binding widgets on the canvas should be displayed. Since the data binding widgets will not be showed after transforming to various formats, developers can make these widgets invisible to get a closer look to the final user-interface after transformation. The last control is responsible for showing all of the hided widgets at one time, no matter it is hided from selecting the first or the second control.

Attribute Editing

All of the editable attributes will be showed in the attribute table. In this way, developers can get instant visual feedback of the editing result.

Source Editing

If developers want to modify the source code of a PUML file, he can click on the bottom tab to change to the source view.

Defining Logic Usage

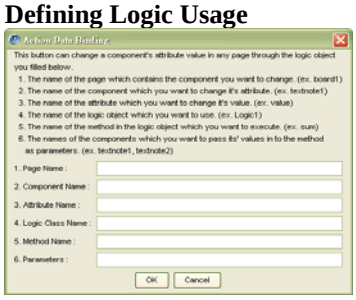

Fig. 7. The helper dialog for defining logic usage.

The defining logic usage operation is used to bind a user-interface component to a logic object behind; this operation is also related to the data binding widgets in the toolbox. Developers can drag-and-drop data binding widgets to the canvas as well, however, in order to let developers who do not familiar with the usage of the data binding widgets can also define the logic usage, another helper dialog is created to provide the information about defining logic usage as depicted in Fig. 7. The logic usage is defined in a way of “changing what attribute of what component in what page through using what method of what logic object?”, developers only need to fill the five “what” statement and then the logic skeleton can be generated in the defined way.

3.4 Transforming

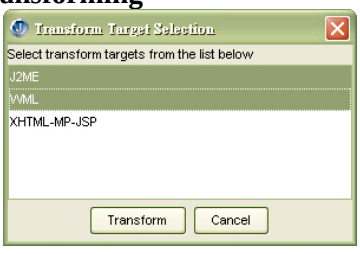

Fig. 8. The transform target selection dialog for choosing either one or multiple targets.

The transformation process as shown in Fig. 8 is accomplished by using multiple style sheets.

3.5 Programming Application Logic

After multiple applications being generated, developers can write application logic inside the generated logic skeleton. Currently, three formats of logic skeletons are generated. One is J2ME MIDP which is used for user-interface generated in J2ME MIDP, another is WML Script [6]which is used for user-interface generated in WML, and the other is Java which is used for user-interface generated in XHTML-MP embraced in JSP.

3.6 Simulating

However, developers may want to preview the final outlook on the mobile simulator, Fig 9. shows the snapshot of simulating the generated WML files through NOKIA Mobile Browser Simulator[7].

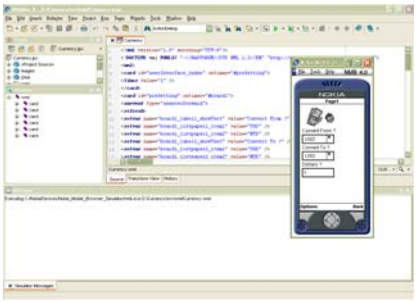

Fig. 9. The simulating result in the NOKIA Mobile Browser Simulator

4 Toolkit Architecture

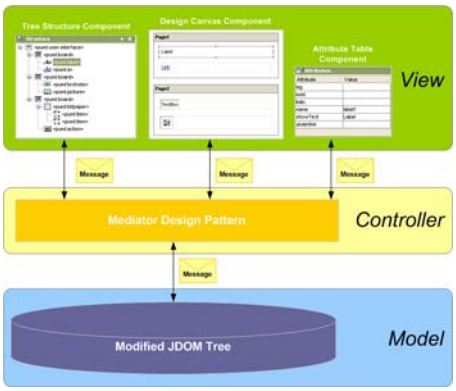

Fig. 10. The toolkit architecture in the form of Model-View-Control design pattern.

View-Control design pattern.

The architecture of our toolkit follows the design principle of the MVC (Model-View-Controller) design pattern as depicted in
Fig. In this chapter, each part of the architecture will be detailed in this chapter; furthermore, the way of integrating the toolkit into JBuilder will be detailed as well.

4.1 Model

A model is a structure which maintains the application data under the surface. The toolkit basically contains two kinds of models inside, one is the document model which represents a single PUML document, and the other is the view model which manipulates the data inside each view.

Document Model Design

For the sake of abstracting a PUML document into visualized widgets on the screen, an internal model is needed to represent the structure of the source PUML document. Since PUML is an XML-based language, it can be easily parsed and built into a DOM (Document Object Model) tree. DOM tree is composed of various kinds of nodes such as element node, text node, entity node, etc.; however, the only node we concerned about is the element node. Since each element node corresponds to a tag in a PUML document, what left behind is to make the connection between a DOM element node and a corresponding widget on the screen.

The link between the DOM element node and the widget can be connected through a unique identification. In this way, both of them need to add an identification field for mapping between each other. Unfortunately, in the Java reference implementation of DOM, it is not allowed to make an extension to the DOM element node to add an identification field. For this reason, we use another third-party API called JDOM instead.

View Model Design

As shown in
Fig10, there are mainly three view components in the toolkit; each of them has its own view model inside.

Tree Structure Model

A tree structure model provides not only the tag text but also an icon corresponding to the widget icon in the toolbox; in addition, the unique identification is retrieved from the element node in the document model and added to each tree node.

Design Canvas Model

The design canvas contains various visualized widgets on it; each widget maintains its own data model behind. In this way, the only thing the design canvas model does is to maintain a map mapping between the unique identification and the widget, therefore, accessing the widget can be done through directly retrieving instead of searching through entire widget structure.

Attribute Table Model

The attribute table model changes while the selected widget on the canvas changes. Each attribute of the selected widget is corresponded to an element attribute inside the document model.

4.2 View

A view is a visualized representation of the internal data model. The views related to the document model in the toolkit are described below.

Tree Structure View

The tree structure view is composed of multiple tree nodes, and each tree node represents a tag element in the document model.

Design Canvas View

A design canvas on the screen is composed of various widgets which are visualized with its own outlook. For the sake of customizing each outlook of the widget, the widget structure is designed as a base container containing a visualized component. The base container is responsible for drawing the selection border of the widget while the visualized component is in charge of drawing the outlook and holding the data inside.

Attribute Table View

The attribute table view provides an organized presentation of the element attributes, the table is formed in two columns and many rows according to the number of attributes.

4.3 Controller

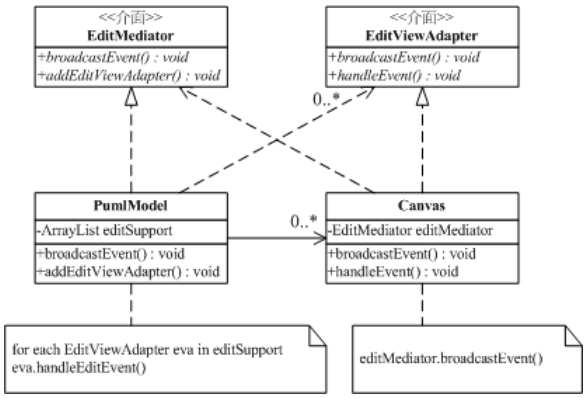

Figure 11: The UML representation of the mediator design pattern of the toolkit.

A controller is a mechanism that handles the interaction between the views and the model. The controller designs in the toolkit utilize the mediator design pattern as depicted in Figure 11. The mediator design pattern in the toolkit is mainly composed of two interfaces; one is EditMediator while the other is EditViewAdapter. The EditMediator maintains a list of EditViewAdapters and is in charge of broadcasting incoming events to every EditViewAdapters in the list. Each EditViewAdapter represents the view which intends to receive or broadcast the change event, in this way, changes on each view can be reflected to another view for keeping the views display the data in consistent.

Through the mediator design pattern, the document model can work as a central coordinator between different views, furthermore, the correctness of the data model can be ensured while the data is changed in any views.

4.4 Other Mechanisms

There are many components to form the whole toolkit architecture; following is some remarkable designs in the toolkit which make the toolkit more flexible, friendly, and easy-to-use.

4.4.1 Extensible Widgets

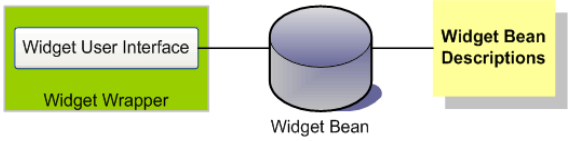

Figure 12: The widget architecture with regard to the JavaBeans.

Since the version of PUML specification changes over time, the design of a widget which represents a PUML tag element must be extensible to increase the software usability. For this reason, JavaBeans component framework is conducted into the widget design, each widget is regarded as a pluggable component in the toolkit and the composition of an extensible widget is illustrated in Figure 2.

4.4.2 Widget-Generation Framework

On the one hand, the created PUML document can be saved for next time usage; on the other hand, generating visualized widgets back to design canvas while opening a saved PUML document is required as well. The widget generation framework recursively generates all of the widgets from each tag element, a widget-generation process is not only used in opening a document, the actions such as move, paste, and drag-and-drop widgets are also utilize this process for code-reusing purpose.

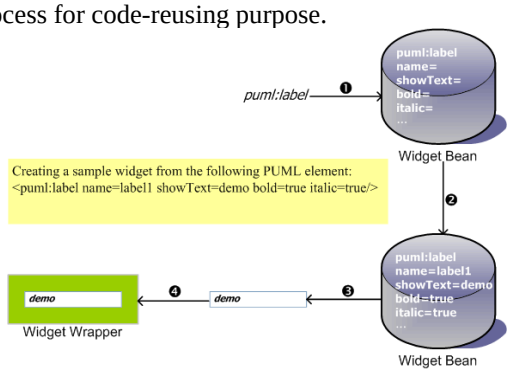

Figure 13: The process of generating a widget from the PUML tag element.

A widget-generation process example is depicted in Figure13; the first step is to generate a widget bean according to the tag element name. The second step is to set fields of the widget bean according to all available attributes in the tag element. Following step is to generate a widget user-interface component according to the widget bean and wrap the component inside a widget wrapper at last. The example widget can then be added to its legal parent widget in the design canvas.

4.4.3 Smart Position Inference

The layout of a generic user-interface is in a vertical way for fitting into the small screen size of mobile devices. In this vertical layout scenario, smart position inference mechanism is added to facilitate the layout arrangement.

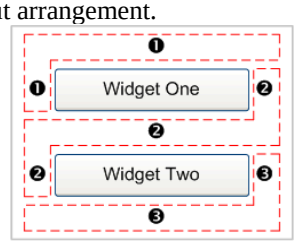

Figure 14: The smart position inference area.

While adding a widget onto the design canvas, there are mainly three areas which could be put on. Figure 14 depicts these areas with a dashed line circled around and a number marked on each area. Once the developer drag-and-drop a widget to the area marked with ❶, the widget will be added above "Widget One" in Figure 14. In the same way, the widget will be added between "Widget One" and "Widget Two" while drag-and-drop it to the area marked with ❷. Obviously, the widget will show up under "Widget Two" while drag-and-drop it to the area marked with ❸. Although it seems simple to add a widget according to the position above, before, under, or after a certain widget, the insertion action actually requires finding out what widget to insert before and then insert the widget with size and position adjustments. Developers therefore do not need to know what happened in the background, simply drag-and-drop to a position and get an intuitive result is the goal we have achieved.

5 Conclusion

In this paper, we propose a visualized toolkit within an integrated development environment for authoring a single generic application to generate both mobile application and mobile Web application. Base on the existing PUML transformation technology from our laboratory, this paper focuses on the design issues in crafting a visualized toolkit in JBuilder instead of detailing the transformation of both application types. The main purpose of crafting this toolkit is to reduce the efforts of developing applications for multiple mobile devices. For this reason, we discuss them in four criteria: rapidly development, extensibility, integration, and write once, generate multiple application user-interfaces. From the rapidly development aspect, it is important to reduce the efforts of learning how to use a new toolkit for developers.

From the extensibility aspect, widgets are designed in an extensible fashion based on JavaBeans technology, in this way; new widgets can be added easily to reduce development efforts in the future. From the integration aspect, providing an integrated development environment for developers is quite essential since they can perform all the editing or testing works inside a single environment. From the write once, generate multiple application user-interfaces aspect; it is completely feasible through our toolkit by using the PUML technology.

Future Works

There are mainly two directions for improving the toolkit in the future, one is in adding some more transformation target languages, the other is in completing the logic content generation instead of logic skeleton generation.

6 References

1. Tzu-Han Kao, Yung-Yu Chen, Tsung-Han Tsai, Hung-Jen Chou, Wei-Hsuan Lin, Shyan-Ming Yuan, PUML and PGML: Device-independent UI and Logic Markup Languages on Small and Mobile Appliances Lecture Notes in Computer Science (LNCS) of Springer-Verlag, The 2005 IFIP International Conference on Embedded And Ubiquitous Computing (EUC-05), Nagasaki, Japan, 6-9 December 2005.
2. Tzu-Han Kao and Shyan-Ming Yuan, Designing an XML-based context-aware transformation framework for mobile execution environments using CC/PP and XSLT, Computer Standards & Interfaces September 2004
3. OMA, XHTML Mobile Profile (XHTML-MP) Specification, http://www.openmobilealliance.org/tech/affiliates/wap/wap-277-xhtmlmp-20011029-a.pdf
4. Yun-sheng Liu, Shyan-Ming Yuan, "ART-based Mobile Application Development Kit", Master. Dissertation, DcsLab, Dept, CIS, NCTU. 2005
5. Guruduth Banavar, Lawrence D. Bergman, Yves Gaeremynck, Danny Soroker, Jeremy Sussman, "Tooling and system support for authoring multi-device applications", The Journal of System and Software, vol. 69, 2004, 227-242
6. OMA, WMLScript Language Specification, http://www.openmobilealliance.org/tech/affiliates/wap/wap-193-wmlscript-20001025-a.pdf
7. Nokia, Nokia Mobile Browser Simulator, http://www.forum.nokia.com/main/0,6566,034-13,00.html