JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 23, 1189-1211 (2007)

Generating User Interface for Mobile Phone Devices Using Template-Based Approach and Generic Software Framework*

MING-JYH TSAI AND DENG-JYI CHEN
Institute of Computer Science and Information Engineering
National Chiao Tung University
Hsinchu, 300 Taiwan

It has been shown that the major effort spent on the design and implementation of the system software for mobile phone devices is the user interfaces (UI) (or man-machine interface, MMI) [15, 16]. If UI can be developed in a short time, it can be a great help to reduce development time for application software system. Therefore, many researchers in software engineering area have been seeking better solutions to aid UI designers to crate UI.

In this paper, we propose a template-based approach to generate UI for mobile phone devices. Specifically, a UI design templates generator is purposed for UI designers to easily and quickly create the UI templates for mobile phone. Furthermore, the developed UI templates can be fine tune with a visual UI authoring tool to generate the UI prototype of the target mobile phone system under consideration. Then, the programmer takes the generated UI prototype as a guider for the program generator to glue the software system architecture and associated functions together to produce the target application system code. Finally, In order to demonstrate the feasibility and applicability of the proposed UI design templates generator, a simulator is designed and implemented for carrying out the software simulation The benefit of the template-based approach is that it enables UI designers to generate UI prototype easily and quickly, and produces automatically the target UI program without writing any textual code. Thus the proposed approach is very suitable for the UI designers (nonprogrammers). In addition, the developed UI templates can be reused by UI designers to generate target UI prototype. Therefore, it can reduce development time.

Currently, UI design and implementation are tightly coupling under the operating system (OS) and hardware specifications; any modifications in either one require UI redesign and re-implementation. We also propose a generic software framework for designing the software system architectures of mobile phone devices, which reduces the need to re-design UI following OS or hardware device changes.

Keywords: user interface, UI design templates generator, generic UI template, UI prototype, visual UI authoring tool, program generator, generic software framework, simulator

1. INTRODUCTION

Developing application software with sophisticated and elegant user interface is a complex and time-consuming task. Studies have shown that near 80 percent of the code of applications is devoted to the user interface (UI) [1], and that about 50 percent of the implementation time is devoted to implementing the UI portion [2, 3]. Thus, UI plays a significant role in the development of application software. For years, researches in soft-

Received May 1, 2006; revised October 11, 2006; accepted November 8, 2006.
Communicated by Kuo-Chin Fan.
* This research was supported in part by the National Science Council of Taiwan, R.O.C., under contract No. NSC 95-2221-E-009-023 and CAISER 2004 project, NCTU.

ware engineering area have been seeking better solutions to aid system developers to build UI [4-7, 19-23].

In general, the process of developing a UI includes the following activities or steps:

(1) UI designers create the UI requirement specification using text, graphic, illustrations, and relevant multimedia representation.
(2) UI programmers then implement it according to the defined specification.
(3) The created UI is verified against the UI requirement specification by UI designers to see if it meets the specification.
(4) If it does not meet the UI requirement specification, one has to modify and re-implement it till the requirements are fully met. This implies that both UI designers and UI programmers have a long iteration process to go in order to meet the target UI requirement specification. Fig. 1 depicts this iterative process.

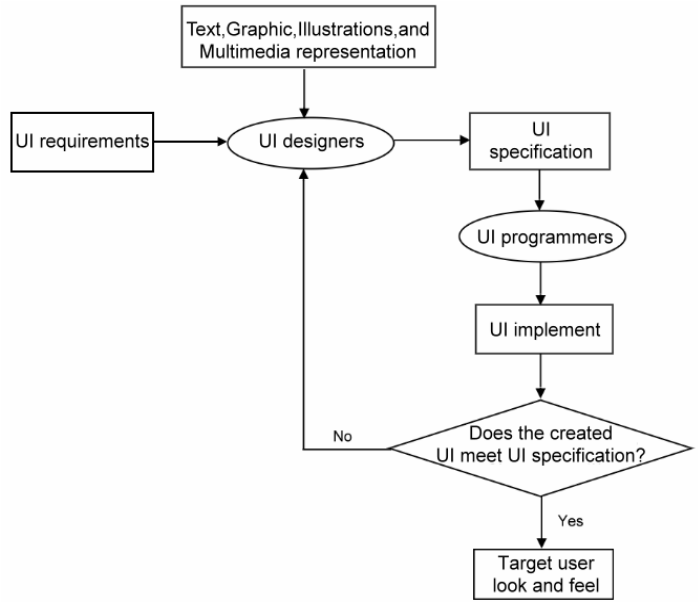


Fig. 1. The general flow diagram for UI development process.

To reduce the UI designer's heavy workload in this long iterative process, we propose a UI design templates generator for UI designers to easily design and author the UI template. These created UI template can be reused easily to create the new UI prototype for a new mobile phone device. To avoid UI programmer's tedious workload in this long iterative process, the UI design templates generator can automatically generate the UI program according to the UI template. Based on this innovative approach, the UI designer alone can complete the UI design and implementation without bothering the UI programmers since the UI program will be generated automatically by using the UI design templates generator. Thus, the UI requirement and functional requirement can be separated as shown in Fig. 2.

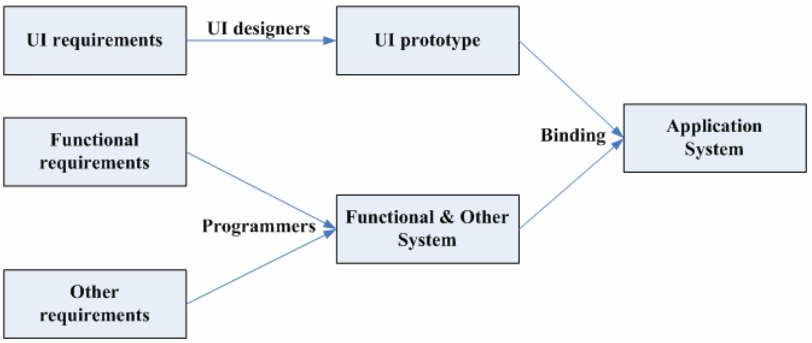


Fig. 2. The new flow diagram for UI development process.

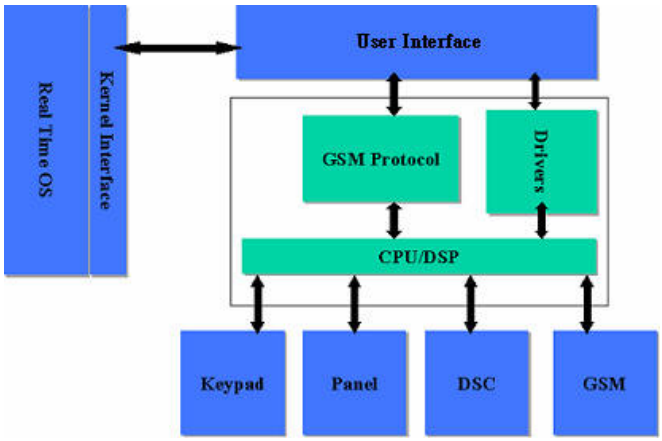


Fig. 3. System architecture of mobile phone devices.

In [8-10], a visual based software construction model has been proposed for supporting this UI development process. Here, we propose a UI design templates generator to generate the UI prototype of mobile phone. The proposed UI design templates generator has integrated into the visual based software construction model. In order to demonstrate the feasibility and applicability of the proposed UI design templates generator, a simulator is designed and implemented for carrying out the software simulation. In section 3, we will give a more detailed treatment on this methodology.

A mobile phone device basically contains a UI system, a real time operating system (RTOS), and several hardware devices (Fig. 3). The UI system accepts input events and executes corresponding functions, reacting to such external events as user input (key presses), connections (outgoing and incoming calls), and changes in handset status (battery, antenna, timer, *etc*.). The RTOS provides resource management services to the application system, including multithreading management, communication, synchronization, and interrupt service routines (ISR). Most mobile phones contain a panel, keypad, digital still camera (DSC), and global system for mobile communication (GSM) system.

As shown in Fig. 3, the UI, OS, and hardware devices must work together to ensure proper mobile phone system operation. Any changes in an OS or hardware device require

immediate UI re-design and re-implementation. This is usually a nightmare for system designers and implementers while such kind of changes must be made. To streamline this process, we proposed a generic software framework for designing mobile phone system architectures.

2. RELATED WORK

There are a few specific tools available for creating the UI prototype for mobile phone devices based on the visual authoring approach. The most common tools found in industrial sectors for the UI development of mobile phone devices are eMbedded Visual C++, Rapid, and Symbian's Eclipse tools.

Microsoft Windows Mobile 2003 Second Edition (mobile phone operating system) not only provides a complete developing tool on this platform such as GUI framework and Visual developing environment, eMbedded Visual C++ [11, 12], but also a simulator to verify the executing result from the machine. On creating the UI, it offers an authoring environment for limited functions such as designing pull down menus. If users want to create a more complicated design such as inserting a picture on the screen, they need to write the script program.

Rapid is a crossed platform system, developed by e-SIM, for embedded system design, implementation, and simulating [12]. It provides object layout during UI development for adding new objects onto the screen or defining the position of objects. It also provides an object editor to modify screen of object, and a simulator to verify executing result.

When one uses Rapid tool to develop the UI of mobile phone, he needs to clearly define the object on screen, all of the states in the entire system, and conditions for transferring in each state. In order to effectively use the Rapid tool to design and implement the target UI prototype for the system, users need to understand all the details of machine state and operations in addition to designing the UI screen. Thus, it is probably only at programming level that users can sort out these details. The Rapid tool is therefore suitable for UI programmers to use (not for UI designers).

Eclipse tools appear for Symbian C++ developers using the open source integrated development environment from Eclipse foundation. Symbian C++ runs under Symbian OS [12-14], its operation is similar as eMbedded Visual C++.

Consequently, the following problems will be faced when using the above mentioned UI developing tools:

(1) Writing textual program is still inevitable.
(2) UI Programmers have to work with UI designer in order to modify the changes of UI. (UI designer alone cannot complete the UI task.)
(3) A long iterative process between UI designers and UI programmers cannot be avoided while using the current approaches to design and implement the UI of the mobile phone devices.
(4) No UI design templates generator supported in current UI developing tools. Thus, a UI designer could not use it to generate various UI template for future reuse.

3. UI DESIGN TEMPLATES GENERATOR AND VISUAL BASED SOFTWARE CONSTRUCTION MODEL

The visual based software construction (VBSC) model supports a visual requirement authoring tool that allows requirement facilitators to produce GUI based requirement scenario and specifications. It also supports a program generator that allows programmer to generate the target application system as specified in the visual requirement scenario. The target application system code can be produced based on the function binding features provided in the program generator to bind each UI component with the associated application function.

3.1 The Framework of UI Design Templates Generator and Visual Based Software Construction Model

In this section, the framework of a VBSC model is recalled and a UI design templates generator including generic UI template, UI template constructor, UI template manager, and UI templates database is proposed as shown in Fig. 4.

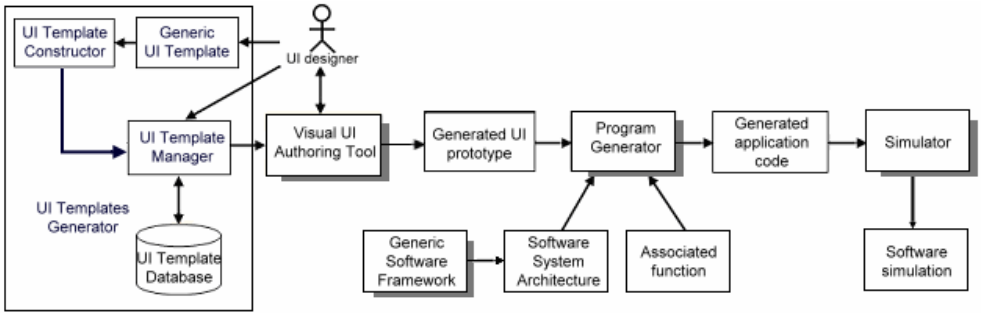


Fig. 4. The framework of UI design templates generator and VBSC model.

The framework includes the following major parts: (1) *The UI design templates generator*, which is used to create the UI templates; (2) *The visual UI authoring tool*, which is used to modify or fine tune UI template (created by UI design templates generator) to generate UI prototype; (3) *The program generator*, which is used to produce the target application system code according to the UI prototype generated; (4) *The generic software framework*, which is used to generate software system architecture for the target application system; and (5) *The simulator*, which is used for software simulation.

- *UI designer*: A person who is responsible for the UI design of the software system. He or she can use UI design templates generator to create a UI template and use the visual UI authoring tool to modify or fine tune the UI template.
- *Generic UI template*: It is consisted of UI structure template, UI layout template, and UI style template which will be elaborated in section 4.2.
- *UI template constructor*: A tool for making new UI Templates. It instantiates the generic UI template to construct the UI template and then stores it into UI templates da-

tabase through the UI template manager.
- *UI template manager*: A database management system for managing UI templates; it provides an interface for adding or deleting UI templates as well as for retrieving an existing UI template.
- *UI template database*: A database for storing UI templates.
- *Visual UI authoring tool*: It can be considered as a multimedia authoring tool.
- *UI prototype*: We can fine tune the created UI template by visual UI authoring tool to generate target UI prototype.
- *Program generator*: Binding software design framework and associated function with each UI component (defined in the generated target UI prototype) to produce the targeted application system code.
- *Software system architecture*: The software system architecture is generated by generic software framework for the target application system.
- *Associated function*: Associated function developed based on application program interface (API) library function developed by the functional programmers according to the hardware specification.
- *Simulator*: It is used to simulate the functionalities of the produced target application system on the target mobile phone.

UI designers use the UI design templates generator to construct an initial UI template, and then use the visual UI authoring tool to modify or fine tune the initial UI template to generate the UI prototype of target mobile phone system. The generated UI prototype is then as a guider for the program generator, the function binding system, to glue the software system architecture and associated functions together to produce the target application system code. Finally, one uses the simulator to do software simulation.

4. GENERATING UI TEMPLATES USING UI DESIGN TEMPLATES GENERATOR

We have discussed the framework of UI design templates generator for mobile phone devices in section 3. In this section, we discuss how to use the UI design templates generator to generate the UI templates for mobile phone. Specifically, a generic UI template is introduced and its corresponding UI template constructor is implemented to construct the UI template.

4.1 UI of Mobile Phone

When we operate a mobile phone, we will see the stand-by screen after power on the mobile phone. Then, there will be several functional buttons ready for pressing to initiate the desired function. To press a specific functional button, it takes us to the corresponding screen associated with the function. These functional buttons are usually organized into a tree style structure as shown in Fig. 5 and consists of two basic elements (1) Node that represents a screen or a function and (2) Link that defines the relationship between screen and screen or screen and function.

A screen (or node), which is not a leaf, can be considered as a container (or scene)

that may contains many actors (or UI components) including, text actor which provides function of text representation, icon actor which provides function of drawing representation, input box actor which allows user to input data during execution of a specific function such as pressing telephone number, and list box actor which represents function of multiple data. A link provides a binding among nodes and functions, and control information for a screen to another screen (node to node) navigation.

A screen or node at leaf level will be considered as a function. There are many common functions in most of the current mobile phones including the essential functions (such as communication, phone books, and conversation records) and value-added functions (such as recording, camera, Java game, infrared transmission). These functions usually are implemented by API programmers.

Thus, the UI of mobile phone can be quite different if the UI navigation structure is different, the layout of actors in a container (or scene) is different, and the style of actors (leave node) is different. In the following section, we investigate the UI of different mobile phone devices to quest for the common presentation template of mobile phone.

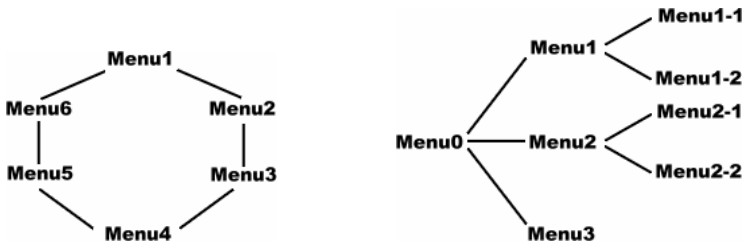
4.2 Generic UI Template

After comparing the UI of different mobile phones, we find that there are some similarities of appearance of user interface when these mobile phones are made by the same manufacturer. Nevertheless, even though the mobile phone comes from different manufacturers, the structure of the UI also has some similarity. Therefore, we factored out the common parts from various UI structures, screen layouts, and actor styles to define the generic UI template. These will be defined as structure template, layout template, and style template.

4.2.1 Structure template

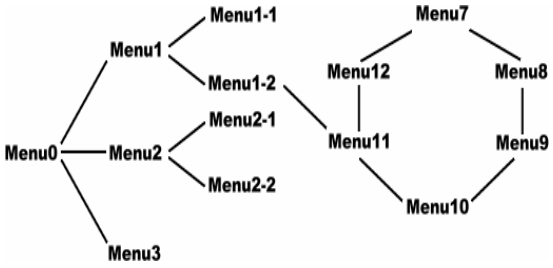
The UI prototype is very dependent on the UI structure for screens to screens navigation. The UI structure, based on topological point of views, can be a ring (or circle list), tree, and ring of tree as shown in Fig. 5.

The complete UI structure data is stored in the structure template, which includes actors in each scene, layout, and style. The layout and style are defined in layout template and style template. Thus, the structure template consisted of (1) the structure of all scenes and (2) actors information in scene.



Ring (or Circle list) structure Tree structure

Fig. 5. Various UI structures.



Ring of tree structure

Fig. 5. (Cont'd) Various UI structures.

4.2.2 Layout template

The UI prototype is also sensitive to each actor's position in a scene. This positioning is called Layout. We could define layout template according to the layout information and then change the actor's position according to a different layout template. For example, in Layout1, the text is placed at the bottom and icons are placed at the top; in Layout2, the text is at the top and icons are at the bottom as is shown in Fig. 6.

In general, the UI positioning arrangement has regularity. Taking an example of tree structure of UI, the layout is almost the same in the same level of scene. We only needed to define few of them then we could describe layout of each scene in the entire UI completely. Therefore, we could use a layout template to create the same layout for each scene under consideration in UI design of a mobile phone. The layout template consisted of layout data of each scene. Layout of scene is made of (1) name of layout and (2) each actor's information in the scene.

Layout1 Layout2

Fig. 6. Two different layouts.

Appearance of icons Appearance icon with text

Fig. 7. Two different styles.

4.2.3 Style template

The other factor that affects the UI prototype of a mobile phone is the UI style. The UI style considers the style of actor's appearance. An actor's appearance is decided by its attributes in a scene. Even though it is the same actor, different attributes may produce different appearances.

As shown in Fig. 7, appearance based on icons (left hand side) could be changed to be appearance based on icon with text (right hand side) by changing its attributes. There-

fore, style template is used to change each actor's appearance in the scene. The style template consisted of each actor's style data which is the relevant attributes of actor's appearance.

4.3 Generic UI Template for Mobile Phone Devices

In previous subsections, we have defined three UI templates (structure template, layout template, and style template). After combination of these three templates, we obtain a generic UI Template. Furthermore, we use MVC models (model, view, control) [17] to implement a generic UI template for the UI template generation of the mobile phone device. The structure template produces model, layout template, and style template provide view. It could generate different UI prototype by different combination of these three templates as shown in Fig. 8. Structure template generates UI structure first, layout template offers layout data of actor on the scene, and then style template provides each actor's style.

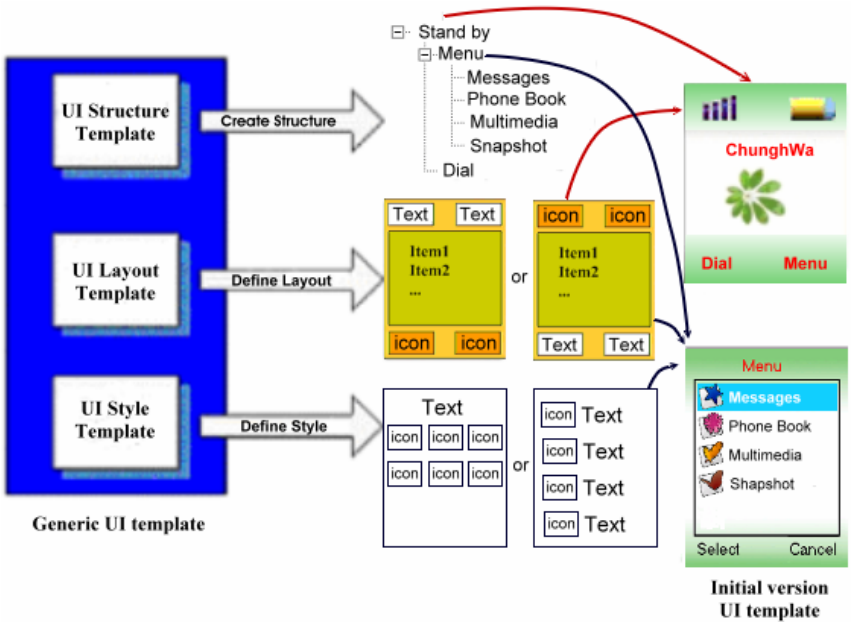


Fig. 8. Generic UI template for mobile phone devices.

Let these three templates be denoted by *a*, *b*, and *c*, then one can produce *a* * *b* * *c* combinations of UI instantiations.

4.4 UI Templates Generation Procedures Using UI Template Constructor

How to use the UI template constructor to create different UI templates is summarized below.

Step 1: Select a generic UI template

The UI designer needs to select the required type of structure template, layout template, and style template.

Step 2: Generate the complete UI structure from the chosen structure template

The system sets up all the scenes and actors according to the structure template chosen by the UI designer and sets up the required file for the visual UI authoring tool such as script file, multimedia file, and so forth. As shown in Fig. 9, we chose the tree structure template. From the "Stand By" scene, the system sets up all scenes, actors, and required files on each scene.

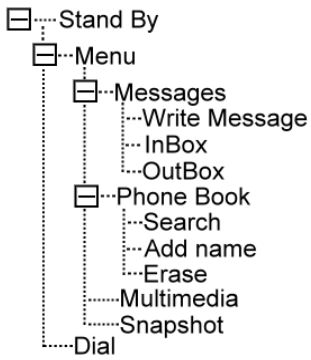


Fig. 9. Tree structure of UI.

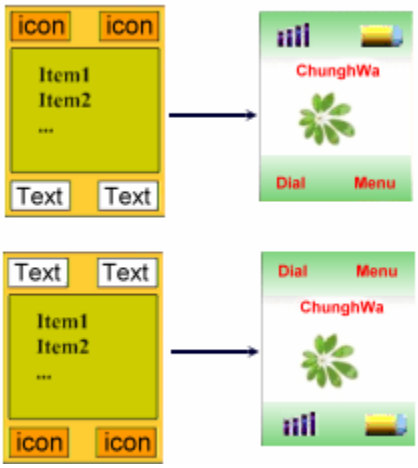


Fig. 10. Layout selection.

Step 3: Generate the desired layout based on the information in layout template for each scene (or node in the tree structure)

According to the information of structure template, the system acquires the corresponding information from the layout template chosen by the UI designer and then inputted into the scene. For the "Stand by" scene shown in Fig. 10, the system chooses a layout from the layout template according to the information from the structure template; it decides the position of each actor on the scene.

Step 4: Generate the desired style based on the information in style template for each actor (or UI button) on each screen (or scene)

According to the information of structure template, the system acquires the corresponding information from the style template chosen by the UI designer and then inputted into the actor. As shown in Fig. 11, the system chooses a style from the style template according to the information from the structure template; it decides the style of each actor on this screen (or scene).

Step 5: Repeat steps 3 and 4 till all the scenes and actors chosen from the structure template are defined

The system continuously repeats the actions in steps 3 and 4 till the entire UI prototype required layout and style information has been filled into the chosen structure template. After setting up the scene of the phone book as shown in Fig. 12, the system continuously repeats steps 3 and 4 till all scenes of the tree structure is completed.

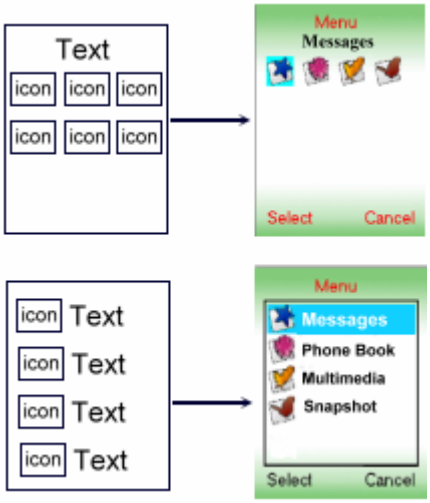


Fig. 11. Style choice.

Fig. 12. UI of phone book generated by UI template constructor.

After executing the above steps, we can construct the UI template for the mobile phone device under consideration. Next, we can fine tune the created UI template by visual UI authoring tool to generate target UI prototype. Then, we use program generator to binding the associated function with UI component for producing the application system code. After complied, the produced object code can be executed on the simulator.

5. THE PROPOSED GENERIC SOFTWARE FRAMEWORK

5.1 Architecture of the Proposed Generic Software Framework

Our proposed generic software framework (GSF) (see Fig. 13) consists of eight parts: a generic kernel interface (GKI), UI kernel task, UI task, GSM task, DSC task, keypad ISR, display application programming interfaces (APIs), and signaling protocol. Generally, there are UI scenarios controlled by UI Task, and RTOS controlled by GKI. Hardware device consists of GSM, DSC, keypad and panel. They are controlled by GSM task, DSC task, keypad ISR, and display APIs respectively. The UI kernel task distributes the received signals to the related tasks (such as UI task or GSM task or DSC task) for execution. The advantage of the GSF is that one just needs to modify the DSC task if the hardware of DSC is changed or to modify GKI if the OS is changed. In both cases, the generated UI system does not need to be modified.

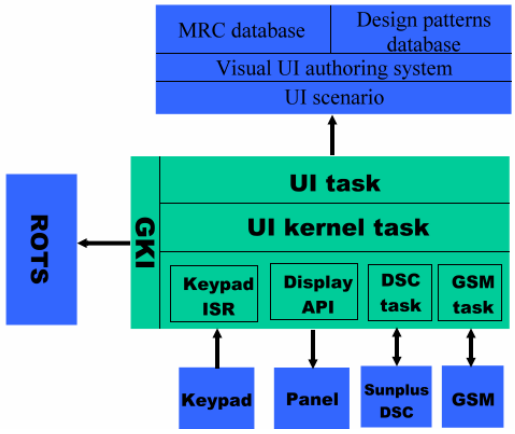


Fig. 13. Generic software framework for mobile phone devices.

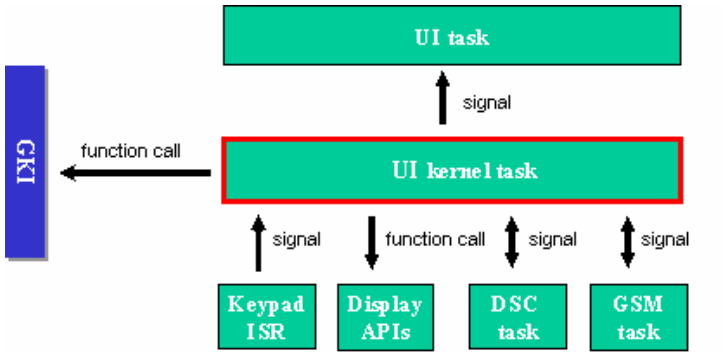


Fig. 14. The relationship between UI kernel task and other components.

The relationship among UI kernel task, GKI, UI task, DSC task, GSM task, and other components are depicted in Fig. 14.

There are eight major sub-components in the proposed framework:

1. *The Signaling protocol.* It is the communication and synchronization mechanism that connects the UI task, UI kernel task, DSC task, and GSM task.
2. *The UI kernel task.* It distributes tasks that request to fulfill the execution of application functions (UI task). The UI kernel task does not directly request driver functions, which allows the generic software framework to serve as a hardware-independent platform for software applications. Each task has a mailbox for storing signals. Signal behavior is task-dependent.
3. *The UI task.* This task displays the current scene content on a handset panel based on the requested signals received from the UI kernel task.
4. *The GSM task.* It is a software component that manages the GSM subsystem; this task provides a stable interface via the signaling protocol. In response to the requested signals from the UI kernel task, the GSM provides information on the current communication status. The AT command interface provides a connection between the

GSM task and GSM sub-system; the GSM task must conform to the signaling protocol to provide GSM sub-system services.

5. *The DSC task*. It is a software component that manages the digital camera hardware; this task controls the DSC module to perform the services of snapshot, image uploading, and image previewing.
6. *The keypad ISR*. It is registered to the interrupt target system; this ISR manages key press events and reports them to the UI kernel task.
7. *The display API.* It is consisted of a set of service routines that execute drawing functions on a mobile phone device panel; the UI task uses the functions to draw pictures, buttons and icons.
8. *The GKI.* This class library encapsulates OS implementations and provides a stable interface for software applications. When programmers port an application to an OS other than the original, they only need to rewrite the GKI library instead of modifying the entire software application system. Each task in our proposed generic software framework is a GKI task sub-class.

5.2 The Design and Implementation of the Proposed Generic Software Framework

5.2.1 GKI

A GKI is a class library that provides the functionalities of multitasking, inter-task communication, and the synchronization of platform-dependent objects such as mutexes, critical sections, locks, and semaphores. The multitasking thread approach for platform-independent consideration is employed for the GKI implementation. When porting program (constructed using the GKI) to another platform, we were required to rewrite synchronized classes instead of completely restructuring our code. Signals, tasks, and mailboxes were the three synchronized objects used to construct the GKI class library. Each GKI task has its own mailbox to store signals, with *sendSignal* and *recvSignal* for each class relevant to the mailboxes.

• Signal

Our proposed generic software framework utilizes signals for task communication and synchronization; the UI kernel task uses signals for coordinating event invocation and execution. External events trigger the construction of a signal that is sent to the UI kernel task, where it is processed and dispatched to other tasks. Signals are stored in task mailboxes, with each task having a public *sendSignal* method and a private *recvSignal* method for respectively transmitting and fetching signals to and from mailboxes.

Different signal types (*e.g.*, *KeypadSignal*, *UISignal*, *UISignal*, and *GSMSignal*) derived from the Signal class represent various external events. These signals overwrite a virtual destructor defined in the base class; the signal class defines the pure virtual destructors for which the derived classes provide concrete implementation. When a signal is deleted from a program, the signal destructor is invoked to release resources contained in the signal.

• Task

The task class provides a blueprint for multitasking, inter-task communication, and

synchronization routines. We used a template method pattern [18] to design the task class and the four tasks in the generic software framework sub-classes (UI, UI kernel, DSC, and GSM).

A task template defines concrete methods that a sub-class can utilize for overwriting and primitive methods that task subclasses are required to implement. Primitive method policies vary according to the OS and hardware device involved in a given situation.

• Mailbox

Each task in the generic software framework has its own mailbox for signal storage; these mailboxes also provide interfaces for pushing and retrieving signals. We used a strategy pattern [18] to design and implement the mailbox class. The mailbox (context) has a reference to a queue object that is part of an abstract class (s*trategy*) that defines interfaces with all strategies. *Win32Queue* and *NucleusQueue* are subclasses of queue (*concrete strategy*) that can be implemented for different operating systems. Queue sub-classes express similar behaviors and provide various implementation strategies for the mailbox class. This design makes it easy to port GKI mailboxes to different operating systems.

The mailbox class (which establishes queues and delegates signal storage duties) provides methods for pushing, fetching, and popping signals. Mailboxes provide "wait" capabilities that prevent queue pooling, thus saving considerable amounts of computing time. They also maintain atomic-access mechanisms for queue access.

5.2.2 UI task

The UI task maintains a set of scenes generated by the visual UI authoring tool, with each scene consisting of icons, buttons, and pictures. The UI kernel task sends signals to the UI task to trigger scene changes, and the UI task applies a state pattern [18] to manage scene transitions. The program generator translates UI scenarios into code snips to be used by the UI task.

The program generator inserts UI and control scenes into the UI task program, which executes the code snips at run time. The UI task processes signals with the identification tag SIG_UI_GOTOSCENE by searching for the next scene, updating the current scene, and redrawing the panel for the current scene.

5.2.3 UI kernel task

In our proposed generic software framework, the UI kernel task (which is based on an active object pattern [18]) serves as both an "event listener" and "event dispatcher." The dispatching process is especially important for a user-response system. The UI system is capable of making an immediate response to any event that is sent, allowing users to monitor the progress of or cancel a current event. Our proposed system uses a responsive listener to increase response speed. The UI kernel task listens for and reacts to external events and manages responses according to the current status of related events. When a signal is fetched from the signal queue, the UI kernel task acknowledges the signal type, uses it to update the current state of, looks for the relevant event handler, and dispatches the signal to the appropriate task.

5.2.4 GSM task

The GSM sub-system handles all communication events and mobile phone communication services — for instance, dialing, ending a call, indicating an incoming call and accepting a call. We designed the GSM sub-system as a modem and the GSM task as a sub-system manager. The GSM task uses the AT command interface to control the GSM sub-system and to simulate GSM communication features. The GSM task receives incoming signals from the UI kernel Task, determines which action needs to be taken, and sends an AT command to the GSM module. The GSM task provides three communication services: originating, answering, and ending calls, triggered by the corresponding signals SIG_GSM_DIAL, SIG_GSM_ANSWER, and SIG_GSM_HANGUP

For example, when the UI kernel task makes a phone call, it sends a signal to the GSM task to perform this service; the outgoing signal contains the phone number and the signal identification is set to SIG_GSM_DIAL. The GSM task receives the signal and executes the request by sending an AT command to the GSM module.

5.2.5 DSC task

The DSC task provides and manages digital camera functions. When a user requests the DSC module to take a snapshot, the UI kernel task processes the request by sending a signal to the DSC task, which processes the request by calling corresponding drivers to perform the service. When the UI kernel task takes a snapshot it creates a signal, sets the identification field to SIG_DSC_SNAPSHOT, and sends the signal to the DSC task.

5.2.6 Keypad ISR

A key press event triggers the keypad ISR to send a signal to an UI kernel task, and the mobile phone device processor executes the keypad ISR on the key press interrupt. Each time a key is pressed; the processor receives a hardware interrupt signal and sends it to the UI kernel task.

6. THE UI AND APPLICATION SYSTEM DESIGN AND DEVELOPMENT PROCESS OF MOBILE PHONE DEVICES

According to framework of UI design templates generator and VBSC model, we organized the system design and development process into following four stages (see Fig. 15):

1. UI templates generation: This stage falls under UI design templates generator section of this model, we can obtain initial version of UI template for mobile phone devices.
2. Visual UI authoring: This stage falls under the visual UI authoring tool section of this model. In this stage, we generate a target UI prototype to meet user's UI requirement.
3. Code generation: This stage falls under the program generator section of this model. In this stage, we produce a target application system code.

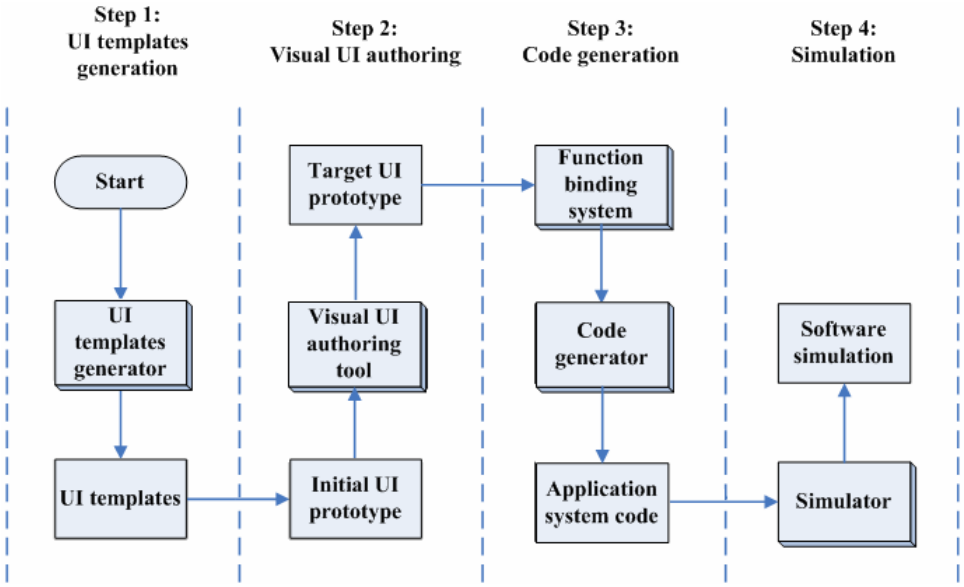


Fig. 15. The UI and application system design and development process of mobile phone devices.

4. Simulation: This stage falls under the simulator section of this model. In this stage, we verify whether the produced target software system code works normally under the simulator.

In the following subsections, we elaborate this development process.

6.1 UI Templates Generation

The first stage is UI templates generation. UI template constructor instantiates the generic UI template to construct the UI template, and then stores it into UI templates database through UI template manager. By the end of this stage, we will obtain an initial version of the UI template for the mobile phone device under consideration.

6.2 Visual UI Authoring

The stage is visual UI authoring. This stage consists of the following two steps:

(1) UI editing: the UI designer uses the visual UI authoring tool to modify and fine tune the UI template to generate the target UI prototype.
(2) The UI prototype preview: If the UI designer wants to view the results of the actual operation after the UI editing, the UI designer can use the preview system in the visual UI authoring tool to check whether the target UI prototype meets the user's UI requirements.

Eventually, the target UI prototype will be finalized by the end of this stage.

6.3 Code Generation

After the target UI prototype is finalized, the UI program will be generated and the binding of associated function code and the UI program will be performed by the code generator in this stage. The major components in this stage are consisted of the following three subsystems:

(1) AP function and UI binder: If the target UI prototype meets the UI requirements, we use the function binding system in the program generator to bind the associated functions to the target UI program generated by the visual UI authoring tool.
(2) Application system code generator: When UI components (in the target UI program) have been bound with the associated functions; it enters into this step to produce the target application system code by program generator.
(3) Code optimizer: If the efficiency is not good during program execution, it enters into this step to adjust application system code until the efficiency is met.

Eventually, the target application system code for the mobile phone device is generated by the end of this stage.

6.4 Simulation

After the above stages are completed, we receive the application program code of the target platform and enter the last simulation stage. In this stage, we have software and hardware simulation two phases:

(1) Software simulation: We combine device module and application system code to do software simulation on laptop through the simulator, which verifies whether the generated application system works normally and meets the expectations.
(2) Hardware emulation: When the software simulation is completed, we could download program to EPROM (Erasable Programmable Read-Only Memory) directly, and execute it on hardware (the target mobile phone device).

7. ASSESSMENT OF GENERATING TARGET UI PROTOTYPE USING TEMPLATE-BASED APPROACH

7.1 Purpose

The purpose of this experiment is to explore the efforts spent with the use and without the use of created UI templates to produce the UI prototype for mobile phone devices. Three application examples contain six, twelve, and eighteen screens of UI prototype were used to make such a comparison. On average, there are about 7 UI components for each screen.

7.2 Participants

Study participants were 39 students enrolled in a graduate software engineering course. The students' majors were computer science (29), electrical engineering (6), in-

formation management (2) and other computer-related departments (2). Those students who participated in the experiment all have rich programming experience (at least three years working experience in software programming related area).

7.3 Design

All the students were asked to be familiar with visual UI authoring tool, and use it to generate target UI prototype for a mobile phone device. Then two weeks later, those students were asked to use the created UI template to do the same task.

7.3.1 Without the use of created UI template to generate target UI prototype

If there is no suitable UI template in the UI templates database can be used directly, we have to use a visual UI authoring tool to generate target UI prototype.

7.3.2 Using the created UI template to generate target UI prototype

If there is suitable created UI template, we can modify it into target UI prototype by a visual UI authoring tool. This modification includes (1) Replacing icon and text from one of the screens; (2) Deleting some screens; (3) Creating one screen to the created UI template.

7.4 Data and Analysis

7.4.1 Case 1: development time of generating a six-screen UI prototype

Fig. 16 presents the time spent to generate a six-screen UI prototype with the use of created UI template and without the use of created UI template. Based on the collected data, we found that the average time spent to generate this UI prototype with the use of created UI template is 9.92 minutes and the average time spent without the use of created UI template is 36.38 minutes. The development time without the use of created UI template is about 3.7 times [(36.38/9.92)] longer than with the use of created UI template. The 27-minute difference represents a time savings of 73%.

7.4.2 Case 2: development time of generating a twelve-screen UI prototype

Fig. 17 presents the time spent to generate a twelve-screen UI prototype with the use of created UI template and without the use of created UI template. Based on the collected data, we found that the average time spent to generate this UI prototype with the use of created UI template is 10.38 minutes and the average time spent without the use of created UI template is 54.10 minutes. The development time without the use of created UI template is about 5.2 times [(54.10/10.38)] longer than with the use of created UI template. The 44-minute difference represents a time savings of 81%.

7.4.3 Case 3: development time of generating an eighteen-screen UI prototype

Fig. 18 presents the time spent to generate an eighteen-screen UI prototype with the use of created UI template and without the use of created UI template. Based on the

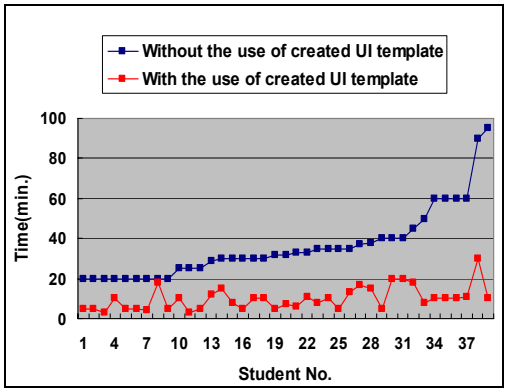


Fig. 16. Time spent in generating a six-screen UI prototype.

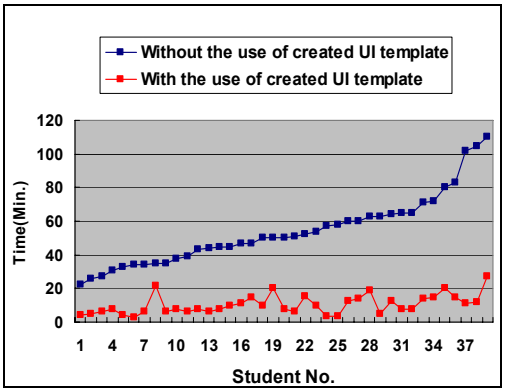


Fig. 17. Time spent in generating a twelve-screen UI prototype.

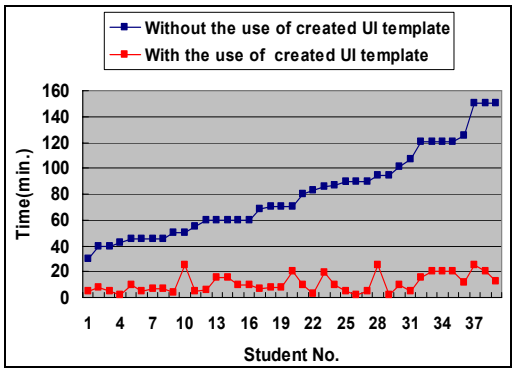


Fig. 18. Time spent in generating an eighteen- screen UI prototype.

collected data, we found that the average time spent to generate this UI prototype with the use of created UI template is 10.85 minutes and the average time spent without the use of created UI template is 80.05 minutes. The development time without the use of created UI template is about 7.4 times [(80.05/10.85)] longer than with the use of created UI template. The 69-minute difference represents a time savings of 86%.

From Table 1, we can note that (refer to column 3) without the use of created UI template to generate UI prototype development time increases tremendously as the number of screens increased. But using the created UI template to generate UI prototype even as the number of screens increased (refer to column 2) the time increment never exceeds 1 minutes. From above, one can see the time ratio increments approximately 1.5 times each time as we add six screens.

From Fig. 19, we can note that the average time spent without the use of created UI template sharply raises. But using the created UI template remains smoothly. From the observation, we can foresee that as the number of screens increased then the time ratio between those two approaches will become wider.

Table 1. Average time spent in generating six, twelve, and eighteen screens UI prototype.

Time spent / Screens	Average time spent with the use of created UI template (Minutes)	Average time spent without the use of created UI template (Minutes)	Time ratio between average time spent without the use of created UI template and average time spent with the use of created UI template
six screens	9.92	36.38	3.7
twelve screens	10.38	54.10	5.2
Eighteen screens	10.85	80.05	7.4

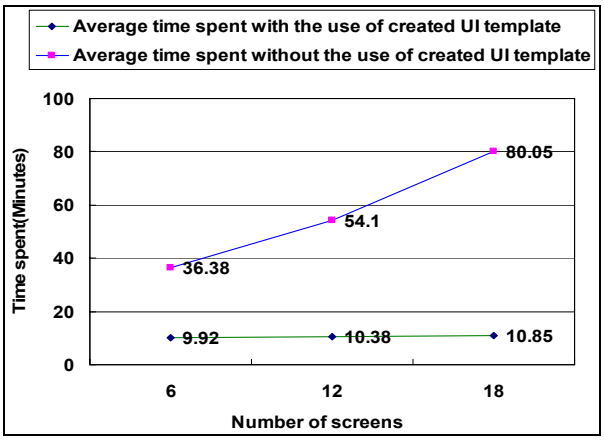


Fig. 19. Average time spent with the use of created UI template and without the use of created UI template to generate target UI prototype.

8. CONCLUSIONS

The well designed UI allows user to operate the product conveniently. So, the design of UI becomes very important which can give customers different impression on the mobile phone devices and influence their willing of purchasing it or not.

UI plays an important role during software development. However, it is also time-consuming for creating a good UI. If UI can be developed in a short time, it can be a great help to reduce development time for application software system. Therefore, many researchers in software engineering area have been seeking better solutions to aid UI designers to builder UI.

By the proposed approach, UI designers use the UI design templates generator to create UI templates and store them for future usage. Furthermore, it could be modified to be a custom-designed UI prototype by the visual UI authoring tool. Based on this innovative approach, the UI designers alone can complete the UI design and implementation without bothering the UI programmers since the UI program will be generated automatically by using the UI design templates generator. With the assistance of the program generator, the UI component is equipped with associated function. We only need to bind the relevant system function with the UI component to generate the target application system code. Finally, we provide a simulator for software simulation.

The proposed generating UI for mobile phone devices using visual based software construction model, template-based approach and generic software framework has following characteristics.

(1) Ideal for UI designer (nonprogrammer) to produce the UI prototype for the target application software system. An UI designer does not need to interact with UI programmers anymore because the UI design templates generator and visual UI authoring tool will generate the target UI program automatically.
(2) Ideal for system developer to plan and manage the team to build the target application system. The proposed approach has separated the UI system with the application function implementation. These two parts are developed by UI designer and programmer separately. A system developer just needs to bind these two parts by using function binding system. In this way, any future changes of the UI will not affect the corresponding application function implementation. Vice versa for the case that the UI is not changed but the associated application functions needed to be changed.
(3) Long iterative process between UI designers and UI programmers can be avoided while using the proposed approach to design and implement the UI prototype of the mobile phone devices.
(4) Rich UI template can be supported by the proposed UI design templates generator to ease a UI designer to reuse to create target UI prototype. Hence, it can reduce time spent while developing application system.
(5) The software system architecture of mobile phone device can be designed by the proposed generic software framework, which reduces the need to re-design UI following OS or hardware device changes.

REFERENCES

1. E. Lee, "User-interface development tools," *IEEE Software*, Vol. 7, 1990, pp. 31-36.
2. B. A. Myers, "User interface software tools," *ACM Transactions on Computer-Human Interaction*, Vol. 2, 1995, pp. 64-103.
3. B. A. Myers, S. E. Hudson, and R. Pausch, "Past, present, and future of user interface software tools," *ACM Transactions on Computer-Human Interaction*, Vol. 7, 2000, pp. 3-28.
4. B. Shneiderman, "Creating creativity: user interfaces for supporting innovation," *ACM Transactions on Computer-Human Interaction*, Vol. 7, 2000, pp. 114-138.
5. C. Frauenberger, T. Stockman, V. Putz, and R. Holdrich, "Mode independent interaction pattern design," in *Proceedings of the 9th International Conference on Information Visualization*, 2005, pp. 24-30.
6. S. Lauesen, *User Interface Design*, Addison Wesley, 2005.
7. R. Jeffries, "Designing interfaces for programmers," *IEEE Software*, Vol. 14, 1997, pp. 89-91.
8. D. J. Chen, M. J. Tsai, J. C. Dai, and D. T. K. Chen, "Visual based software construction: visual requirement authoring tool and visual program generator," in *Proceedings of the International Computer Symposium*, 2004, pp. 171-176.
9. D. J. Chen, W. C. Chen, and K. M. Kavi, "Visual requirement representation," *The*

Journal of Systems and Software, Vol. 61, 2002, pp. 129-143.
10. D. J. Chen and S. K. Huang, "Interface of reusable software components," *Journal of Object-Oriented Programming*, Vol. 5, 1993, pp. 42-53.
11. M. Barr, *Programming Embedded Systems in C and C++*, O'RELLY, 1999.
12. T. Noergaard, *Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers*, Newnes, 2005.
13. S. Babin, *Developing Software for Symbian OS: An introduction to creating smart phone applications in C++*, John Wiley, 2005.
14. R. Harrison, *Symbian OS C++ Mobile phones Vol. 2*, John Wiley, 2004.
15. B. B. Bederson, A. Clamage, M. P. Czerwinski, and G. Robertson, "A fisheye calendar interface for PDAs," *ACM Transactions on Computer-Human Interaction*, Vol. 11, 2004, pp. 90-119.
16. J. A. Landay and T. R. Kaufmann, "User interface issues in mobile computing," in *Proceedings of the 4th Workshop on Workstation Operating Systems*, 1993, pp. 40-47.
17. G. E. Krasner and S. T. Pope, "A cookbook for using the model-view controller user interface paradigm in Smalltalk-80," *Journal of Object-Oriented Programming*, 1998, Vol. 1, 1998, pp. 26-49.
18. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns*, Addison Wesley, 1995.
19. D. Martin and I. Sommerville, "Patterns of cooperative interaction: linking ethnomethodology and design," *ACM Transactions on Computer-Human Interaction*, Vol. 11, 2004, pp. 59-89.
20. K. Hornbæk, B. B. Bederson, and C. Plaisant, "Navigation patterns and usability of zoomable user interfaces with and without an overview," *ACM Transactions on Computer-Human Interaction*, Vol. 9, 2002, pp. 362-389.
21. D. R. Olsen and B. W. Halversen, "Interface usage measurement in a user interface management system," in *Proceedings of the 1st Annual ACM SIGGRAPH Symposium on User Interface software*, 1988, pp. 102-108.
22. V. Novak, C. Sandor, and G. Klinker, "An AR workbench for experimenting with attentive user interfaces," in *Proceedings of the 3rd IEEE and ACM International Symposium on Mixed and Augmented Reality*, 2004, pp. 284-285,
23. X. Kong, L. Liu, and D. Lowe, "Supporting web user interface prototyping through information modeling and system architecting," in *Proceedings of the IEEE International Conference on e-Business Engineering*, 2005, pp. 63-70.

Ming-Jyh Tsai (蔡明志) is a senior lecturer in the Department of Information Management at Fu Jen Catholic University. He is also a Ph.D. candidate at the Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan. His current research interests include software engineering, e-learning, and data mining.

Deng-Jyi Chen (陳登吉) received the B.S. degree in Computer Science from Missouri State University (cape Girardeau), U.S.A., and M.S. and Ph.D. degree in Computer Science from the University of Texas (Arlington), U.S.A. in 1983, 1985, 1988, respectively.

He is now a professor at Computer Science and Information Engineering Department of National Chiao Tung University, Hsinchu, Taiwan. Prior to joining the faculty of National Chiao Tung University, he was with National Cheng Kung University, Tainan, Taiwan. So far, he has been publishing more than 130 related papers in the area of software engineering (software reuse, object-oriented systems, visual requirement representation), multimedia application systems (visual authoring tools), e-learning and e-testing system, performance and reliability modeling and evaluation of distributed systems, computer networks. Some of his research results have been technology transferred to industrial sectors and used in product design. So far, he has been a chief project leader of more than 10 commercial products. Some of these products are widely used around the world. He has been received both research awards and teaching awards from various organizations in Taiwan and serves as a committee member in several academic and industrial organizations.