Supporting View Transition Design of Smartphone Applications Using Web Templates

Kazuki Nishiura[1], Yuta Maezawa[1], Fuyuki Ishikawa[2], and Shinichi Honiden[1,2]

[1] The University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo, Japan
[2] National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, Japan
{k-nishiura,maezawa,f-ishikawa,honiden}@nii.ac.jp

Abstract. Many developers have implemented native smartphone applications (NSAs) that have the same functionalities as those of existing web applications (WAs). They need to redesign web pages as views of NSAs owing to their constraints, such as compact displays. However, it can produce a NSA with low global navigability. We propose a framework that can support developers in designing view transitions of NSAs on the basis of WAs. We focus on web templates to leverage well-designed web page transitions. Our framework 1) extracts a page transition model from a WA to create candidate view transitions of a NSA, 2) provides an interface where developers design these views to solve the constraints, and 3) suggests design modifications to increase global navigability calculated by proposed criteria of navigation costs for users. After examining case studies, we concluded that our framework could support developers to design easy-to-navigate NSAs.

1 Introduction

The rapid spread of smartphones[1] has enabled users to access the web almost anywhere. However, owing to features of smartphones, such as small screens and touch panels, users cannot comfortably browse web applications (WAs) designed for desktop computers. To meet demands of smartphone users, many WA providers have published native smartphone applications (NSAs).

Many efforts have addressed the problem of web browsing on small devices [1]. Some researches took user input to custom pages [2], and others automatically reorganized structures of pages by using heuristic rules and machine learning techniques [3]. Although they focused on individual pages or single tasks, developers should consider global navigability of applications in order to increase usability. Owing to the importance of transition designs in WA development [4], researchers have attempted to improve navigability of web sites [5].

In this paper, we propose a framework that can support developing NSAs on the basis of existing WAs. Developers will be able to not only inherit well-designed page transitions of WAs but also provide users consistent navigation with WAs,

[1] http://www.gartner.com/it/page.jsp?id=1543014

M. Brambilla, T. Tokuda, and R. Tolksdorf (Eds.): ICWE 2012, LNCS 7387, pp. 323–331, 2012.

which will increase usability of NSAs [6]. Our framework extracts a transition model of WAs to provide candidate view transitions of NSAs. To handle dynamic web pages, our model focuses on web templates that are de-facto standards in WA development [7]. Using our framework, developers can design views of a NSA from web templates in order to overcome the display size constraints. They can (i) extract elements from web templates and (ii) divide views to create a child view linked from the original view to reduce the number of elements in each view. Although these operations affect global navigability of the NSA, developers cannot perceive whole transitions in the NSA when designing a single view. Therefore, our framework estimates global navigability by defining a formula for navigation costs of users. Our framework suggests modifications of view transition design to increase the estimated navigability. Our suggestions consist of (i) recovering a link removed by developers, (ii) deleting a link, and (iii) shifting a link from or to child views. Developers can design view transitions interactively by accepting or rejecting these suggestions in accordance with their heuristics.

Our contributions are as follows:

- Proposal of leveraging page transitions of WAs when developers design view transitions of NSAs. We focus on web templates to build a transition model.
- Implementation of our framework. Using our framework, developers can design views of NSAs from web templates and receive suggestions for design modifications to increase global navigability of NSAs.
- Evaluation of our framework by means of case studies that showed our tool could support developers in designing easy-to-navigate NSAs.

2 Web Application-Based Native Smartphone Applications

To leverage page transitions of a WA, developers redesign web pages as views of a NSA. This involves two difficulties. First, contents of web pages are changed at runtime. We focus on web template to handle this dynamic nature as described in Sect. 2.1. Second, developers can reduce global navigability of NSAs (Sect. 2.2).

2.1 Web Templates

When developers design views of NSAs using web pages that have dynamic nature, looking at the page in a particular situation is not enough. To handle slight changes of a page, previous work [2] calculated the similarity among elements by using the Document Object Model (DOM). With this approach, however, developers may fail to notice elements hidden when redesigning the page.

Our approach leverages web template files as bases for views of NSAs. Web templates are predesigned web pages that specify fixed aspects of a presentation. They also contain logical specifications (e.g., *if*, *foreach*) that dynamically control the page structure. A library called a template engine combines web templates with dynamic contents to produce a response. Web templates are widely used in WA development [7], because web page designers can separate a presentation

from the logic behind the page. Our framework uses web templates that contain all possible structures by their very nature, thus developers can handle all displayable elements regardless of any scenarios. In our prototype tool, we consider PHP-based WAs and the *Smarty* template engine library (www.smarty.net).

2.2 Navigability in Native Smartphone Applications

Developers design views of NSAs on the basis of each single web page. They (i) select elements from a page and (ii) divide elements into multiple views if the page contains too many elements. In other words, they (i) remove unselected links and (ii) place links where they want. These operations decide view transitions, and thus affect global navigability of NSAs.

In Fig. 1, we show examples of page transitions in an online bookstore WA. Boxes represent web pages, and arrows represent links. Look at a link from *Index* to *Audio Books* (b). It helps users reach from *Music* to *Audio Books* in two clicks (a, b), from *Index* to *English* in three clicks (b, c, d), and so on. If developers remove the link, users can follow a detour that goes through a *Books* page (e, f). These examples demonstrate two facts: a link contributes to multiple paths and the impact of deleting a link depends on existences of other links. Therefore, developers cannot estimate an importance of a link by simply looking at its source and destination pages. However, they can barely figure out whole transitions while designing a certain single view or consider all the numerous transitions together.

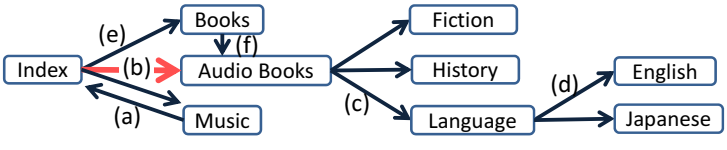

Fig. 1. Simplified page transition graph of an online bookstore web application

3 Proposal Framework

We propose a framework to support developers in designing view transitions of NSAs by using page transitions of existing WAs. Our framework 1) extracts a transition model of a WA to provide candidate view transitions of a NSA, 2) offers an interface where developers redesign web pages of the WA as views of the NSA, 3) calculates global navigability of the NSA, and 4) suggests modifications of transition design to developers in order to increase global navigability.

3.1 Definition of Transition Models

First, our framework generates a Web Application Transition (WAT) model from a WA. To extract the model, we implemented a static path tracer, which pessimistically traces all possible execution paths of a given PHP code and remembers all displayable templates. From given template files, our framework can analyzes transition targets by extracting *a* and *form* tags. The model is then transformed into a Native Smartphone Application Transition (NSAT) model, which reflects the design of developers for the NSA.

Figure 2 shows a meta WAT model given in the Unified Modeling Language (UML). The two main entities in WAs are a *URL* and a *WebPage*. A user sends a request to a URL, and then a WA sends a *response* that contains an appropriate *WebPage* or *redirects* the user to another *URL*. A *WebPage* contains *links* and *forms* by which users send a request to a WA. *WebPages* are *StaticPages* or *DynamicPages* whose contents are fixed or generated at runtime, respectively. Our framework focuses on the WAs using web templates, thus a dynamic page is generated by assigning variables to a web template file.

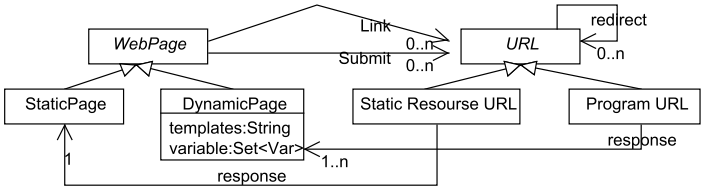

Fig. 2. Meta model of page transitions in general web applications

We model view transitions in NSAs by extending the WAT model (Fig. 3). A *WebPage* is transformed into a *View* of the NSA. In addition, we add *ChildViews* and *internal links*. Note that *ChildView* is a view derived from the source of *internal link*, thus users need not send any additional requests to access it. Additionally, users need not communicate with a server to access *StaticViews*, because NSAs possess layout files on devices. We renamed an association towards *StaticView* as *display*.

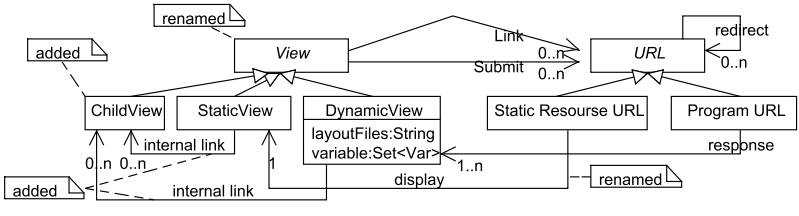

Fig. 3. Meta native smartphone applications transition model. Extension of Fig. 2.

3.2 Estimation of Navigation Costs in Smartphone Applications

In this section, we formulate navigability in NSAs that our framework aims to optimize. Hollink et al. [5] organized previous studies that modeled navigation costs in WAs. They noted that navigation costs for users were determined by the number of pages they visited and the number of choices (i.e., links) on each page. In addition, when they simply assumed these members have a liner relationship to navigation costs, this appeared to fit actual usage logs reasonably.

We extend their formula to model the navigation cost in NSAs generated by our framework. First, we consider *redirections* that consume communication time

between clients and servers. Second, among edges in a NSAT graph[2], *display* and *internal link* edges do not require communications with servers, as described in Sect. 3.1. Therefore, we need to consider only the number of *response* in a path. Let α, β, and γ parameters. We can formulate time consumption as follows:

$$Time(Path) = \alpha \mid Responses \mid + \beta \mid Redirections \mid + \gamma \sum choice(View) \quad (1)$$

Here, *Choice* represents the number of links on a view. Using (1), we define *Global Navigability (GN)* as a weighted sum of the minimum navigation time between two arbitrary views in NSAs. Given *ViewPair* (v_1, v_2), we define *ShortestPath* as a path from v_1 to v_2 that minimizes *Time*. In addition, developers specify a *Value* function that returns the importance of a path. They can define the function based on access logs or their own intentions (e.g., leading users to profitable items).

$$GN = \sum (Value(ViewPair) \times Time(ShortestPath(ViewPair))) \quad (2)$$

3.3 Suggestions for Modifications of View Transitions

Our framework suggests three kinds of operations: recovering, deleting, and shifting (Fig. 4). Among candidate operations, the one that improves *GN* the most is suggested. The computation of *GN* requires all-pairs shortest paths, which takes $O(|Views|^3)$ time by simply using Floyd-Warshall algorithm. Here $|Views|$ denotes the number of views in a NSA. Once we calculate *GN*, we can calculate how each operation would change *GN* in $O(|Views|^2)$ time. We can define a *utility* of a link as the difference in *GN* between whether the link exists. Intuitively, our framework aims to leave links with high utility and remove those with low utility.

The recovering (Fig. 4.a) operation recovers a link removed by developers. While developers design each view, they can hardly estimate a link utility that

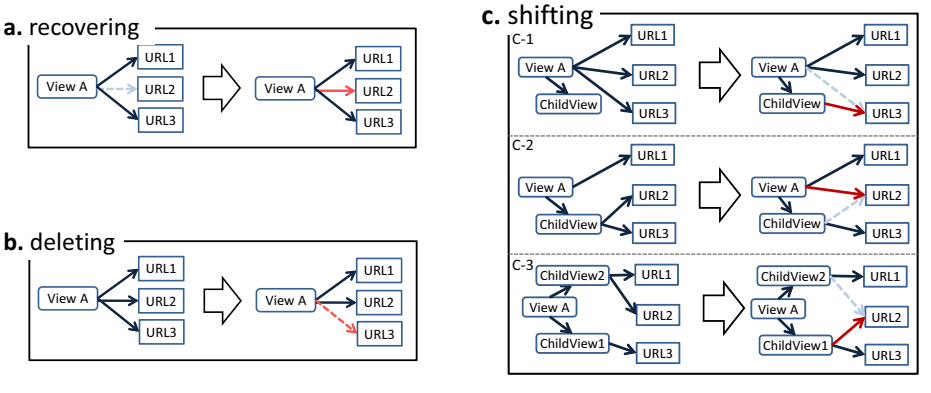

Fig. 4. Transition design of developers (left) and suggestion by our framework (right)

[2] The model described in Sect. 3.1 can be re-interpreted as a graph. In the following, we use model and graph terms (e.g., page v.s. node) interchangeably.

is determined by subsequent paths from the link target. Therefore, they might delete useful links. The deleting (Fig. 4.b) operation, in contrast, deletes unnecessary links. The more links on a view, the harder for users to choose the desired link. Therefore, deleting an unimportant link reduces the cost for users to choose links. The shifting (Fig. 4.c) operation shifts a link from a view to its child view (c-1), from child view to its parent view (c-2), or from a child view to another child view (c-3). Creating a child view can distribute costs of choosing links. Our framework assists in shifting less important links to child views and vice versa.

3.4 Tool Implementation

We implemented a prototype tool of our framework (Fig. 5). To design a view of a NSA, developers can select elements from a web template visualized on the left of our tool. Selected elements are displayed on the emulation screen at the center. If a view contains too many elements, developers can create a child view, which is shown on the right. They are required to specify an appropriate text for the link to the child view. After developers finish designing views, our tool suggests modifications to the design in order to improve global navigability of the NSA (Fig. 6). Our tool generates layout files of NSAs for the Android platform (`developer.android.com`). Elements selected by developers are converted into layout files by removing trivial HTML tags (e.g., decoration tags such as *b*, *i*) and replacing tags to corresponding GUI elements of NSAs. These transcoding rules are based on our heuristic.

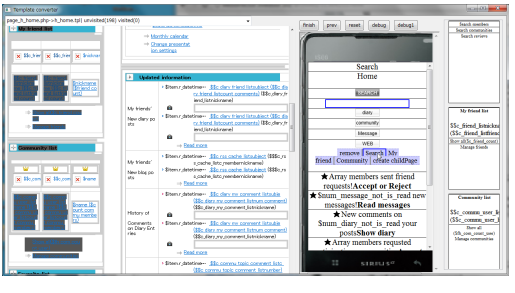

Fig. 5. Designing view (center) ant its child view (right) based on web templates (left)

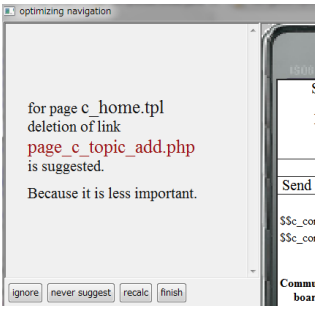

Fig. 6. Suggestion to improve navigability

4 Evaluation and Discussion

We conducted case studies with *OpenPNE2.14.7* (`www.openpne.jp`)[3], which is an open-sourced social networking service (SNS). The page shown in Fig. 7 contains too many links. Each category has a few item links and a *read more* link that leads to an item list page. Developers delete some links that seem less important, as users can reach these pages via *read more* links anyway. However,

[3] We translated pages shown in this section into English.

the destination page of a *post on community board* link contains useful links, such as *send message* and *cancel attending events*. Our framework calculates *GN* and suggests a recovering operation of the link. This case shows that if there are links whose targets have relationships as parent-child or siblings, developers may delete one of them because an alternative path obviously exists. In contrast, developers can hardly estimate tradeoffs between an additional cost incurred by a link deletion and a benefit of decreasing the number of links.

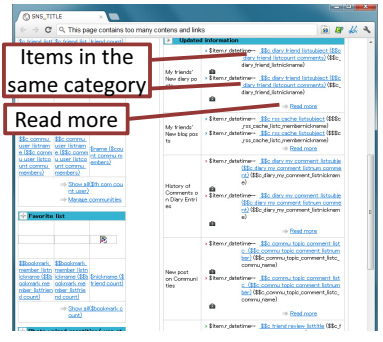

Fig. 7. Many contents and links on a page

Fig. 8. A redundant link on a page

Because this page contains too many elements, developers create child views. They shift some links to leave only useful links in a parent view. Most of the remaining links have a high utility, but a few do not. In addition, the recovering operation recovers a relatively unimportant link, which increases the cost of selecting other important links. Our framework suggests the shifting operation of a *post item review* link that has low utility. A shifting operation can assist the placing of many links among multiple views based on their utility.

Figure 8 shows a community board page. There are links to *topic*, *topic list*, and *post a new topic* pages. Developers decide to use all of them in the NSA. However, users prefer to post a new topic from a *topic list* page after checking existing topics. Our framework suggests a deleting operation of the link to *post a new topic* so that costs of selecting other important links are reduced. Generally, if multiple pages contain links for the same target, developers may use both links. While this choice may increase or decrease navigability, our framework suggests a deleting operation if and only if it works well.

Our framework has some limitations. It cannot deal with links to external WAs nor distinguish different contexts with the same URL and web templates. In addition, our current implementation to extract a WAT model simply traces all execution paths, thus it takes several seconds to trace even a simple code.

5 Related Work

Researchers have attempted to make web pages designed for desktop computers comfortable to browse on narrow screens [1]. Chen et al. [3] utilized both DOM

structures and presentations of pages to split them into small pages. Because their successes were based on heuristics and machine learning techniques, their algorithms do not always work optimally. The *Highlight* [2] tool enables users to make task-based mobile WAs from existing WAs. Users can extract elements from each page related to the task to reduce the page size. The tool reapplies the customizations when users visit a page with a similar structure. However, as described in Sect. 2.1, the reapplication may fail owing to dynamics of web pages.

Our method helps WA developers implement NSAs. Titanium SDK (`www.appcelerator.com/platform/titanium-sdk`) enables NSA development using HTML and JavaScript, which are popular among WA developers. In addition, Prach et al. proposed a mashup framework that could utilize existing web services [8]. However, these approaches did not consider global navigability of NSAs.

Several methods have been proposed to improve web site navigability [5]. Smyth and Cotter [9] reorganized the menu structure to reduce navigation efforts of users modeled by the number of clicks and scrolls. Anderson et al. [10] proposed an algorithm that could suggest shortcut links for mobile web users to reduce the number of communications required. These researchers, however, unrealistically assumed that any two pages could be linked. In fact, modern WAs generate web pages at run time depending on sessions, parameters, and so on.

6 Conclusion and Future Work

We proposed a framework to support developers in designing view transitions of NSAs. We focuses on web templates to extract page transitions in WAs as candidate view transitions in NSAs. Our framework provides an interface to design views of NSAs and suggests design modifications by estimating navigation costs. Using our framework, developers can avoid decreasing global navigability of a NSA while designing each view. By investigating case studies, we conclude that our framework can support designing easy-to-navigate view transitions for NSAs.

Our future work is to implement and publish real world NSAs using our framework. In addition, we aim to extend this work, which leverages page transitions, to handle modern AJAX-based WAs by extracting state transitions on each web page [11]. Moreover, we intend to establish a method for modifying NSAs with keeping consistency in response to frequent updates of WAs.

References

1. Zhang, D., Lai, J.: Can convenience and effectiveness converge in mobile web? a critique of the state-of-the-art adaptation techniques for web navigation on mobile handheld devices. IJHCI 27(12), 1133–1160 (2011)
2. Nichols, J., Hua, Z., Barton, J.: Highlight: a system for creating and deploying mobile web applications. In: UIST, pp. 249–258. ACM (2008)
3. Chen, Y., Xie, X., Ma, W.Y., Zhang, H.J.: Adapting web pages for small-screen devices. IEEE Internet Computing 9(1), 50–56 (2005)
4. Nielsen, J.: User interface directions for the web. CACM 42(1), 65–72 (1999)

5. Hollink, V., Someren, M., Wielinga, B.J.: Navigation behavior models for link structure optimization. UMUAI 17, 339–377 (2007)
6. Gong, J., Tarasewich, P.: Guidelines for handheld mobile device interface design. In: DSI Annual Meeting (2004)
7. Gibson, D., Punera, K., Tomkins, A.: The volume and evolution of web page templates. Special Interest Tracks and Posters of WWW, pp. 830–839. ACM (2005)
8. Chaisatien, P., Prutsachainimmit, K., Tokuda, T.: Mobile Mashup Generator System for Cooperative Applications of Different Mobile Devices. In: Auer, S., Díaz, O., Papadopoulos, G.A. (eds.) ICWE 2011. LNCS, vol. 6757, pp. 182–197. Springer, Heidelberg (2011)
9. Smyth, B., Cotter, P.: The Plight of the Navigator: Solving the Navigation Problem for Wireless Portals. In: De Bra, P., Brusilovsky, P., Conejo, R. (eds.) AH 2002. LNCS, vol. 2347, pp. 328–337. Springer, Heidelberg (2002)
10. Anderson, C.R., Domingos, P., Weld, D.S.: Adaptive web navigation for wireless devices. In: IJCAI, vol. 2, pp. 879–884. Morgan Kaufmann Publishers Inc. (2001)
11. Maezawa, Y., Washizaki, H., Honiden, S.: Extracting interaction-based stateful behavior in rich internet applications. In: CSMR, pp. 423–428. IEEE C.S (2012)