Tanzirul Azim *University of California, Riverside*
Oriana Riva and Suman Nath *Microsoft Research*

Editors: Nic Lane and Xia Zhou

uLink: user-defined deep links in mobile apps

Excerpted from "uLink: Enabling User-Defined Deep Linking to App Content," from ACM MobiSys 2016, *Proceedings of the 14th ACM International Conference on Mobile Systems, Applications, and Services Singapore* with permission. http://dl.acm.org/citation.cfm?id=2906416 © ACM 2016

Web deep links are instrumental to many fundamental user experiences, such as navigating from one web page to another, bookmarking a page, or sharing it with others. Such experiences are not possible with individual pages inside mobile apps, since historically mobile apps did not have links equivalent to web deep links. Mobile deep links, introduced in recent years, still lack many important properties of web deep links. Unlike web links, mobile deep links need significant developer effort to be exposed, cover a small number of predefined pages, and are defined statically to navigate to a page for a given link, but not to dynamically generate a link for a given page. We have built uLink, a novel deep linking mechanism that addresses these problems. uLink is implemented as an application library, which transparently tracks data- and UI-event-dependencies of app pages, and encodes the information in links to the pages; when a link is invoked, the information is utilized to recreate the target page quickly and accurately. uLink also employs techniques, based on static and dynamic analysis of the app, that can provide feedback to users about whether a link may break in the future due to, e.g., modifications of external resources such as a file the link depends on. We have implemented uLink on Android, and tested with 30+ apps. Compared to existing mobile deep links, uLink requires minimal developer effort, achieves significantly higher coverage, and can provide accurate user feedback on a broken link.

uLink GOES BEYOND MOBILE DEEP LINKS

Mobile deep links are URIs that point to specific locations within apps[1] [2] [3]. A mobile deep link can launch an already installed app on a user's mobile device (similar to loading the home page of a website) or it can directly open a specific location within the app (similar to deep linking to an arbitrary web page). For example, the URI fandango://thelego movie_159272/movieoverview directly navigates to the page with the details of the "The Lego Movie" in the Fandango app.

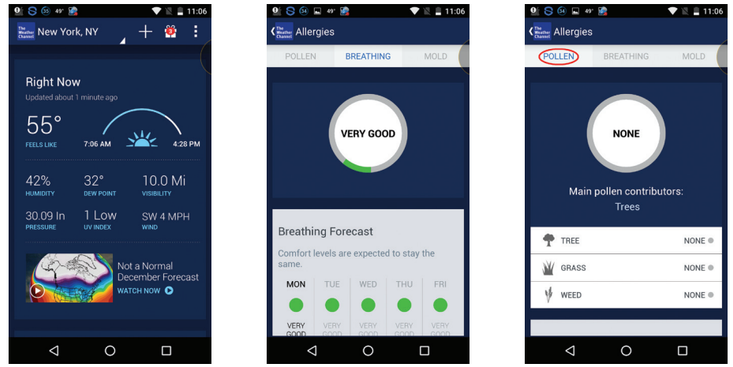
Even though mobile deep linking is an important first step towards accessing any arbitrary location within an app, it lacks many useful properties of web deep linking. First, unlike web deep links, mobile deep links require nontrivial developer effort – several lines of codes per deep link – resulting in a low adoption rate even within the top apps [4]. Second, unlike its web counterpart, mobile deep links have poor coverage – a small number of locations within an app, predefined by the developer, are directly accessible via deep links. Finally, today's deep links are defined statically by developers to facilitate navigation to a target page given its link; the dual process of dynamically determining the link for a given page is not possible even if a deep link to that page exists.

In [5], we introduced *uLink*, a lightweight approach that addresses the above problems. uLink requires minimal developer effort, it supports dynamic link creation, and it achieves significantly higher coverage than existing mobile deep links. Moreover, it is compatible with existing mobile deep links (i.e., the underlying mobile OS handles them in the same way). All this enables many novel user experiences that so far existed only in the web.

A key challenge uLink addresses is improving coverage – creating links to any app location (referred to as app view or view hereafter), including to the ones that depend on previous views or on user interactions. uLink uses two key mechanisms. The first mechanism is *shortcut*. uLink continuously monitors for explicit data dependency between successive runtime views in an app. In Figure 1, view (a) launches view (b), by providing the location "New York, NY" selected by the user in (a). In some cases, e.g., if (a) and (b) are separate Android activities (i.e., pages), uLink can transparently capture the data transferred from (a) to (b) and encode it in the link to (b). This allows uLink to quickly invoke the link to go to (b), without first going to (a). More importantly, it improves coverage to views that depend on data from previous views (location in this example).

Shortcuts do not cover all app views. The view shown in Figure 1(c) is created when the user taps on the "POLLEN" tab, and there is no explicit data transfer between (b) and (c) for uLink to capture – both views are within the same Android activity. To create links to such views, uLink uses a limited form of record and replay. uLink continuously records UI actions in the current view, and encodes them in the link (we call this a *shortcut-and-replay* link). When the link is invoked, uLink first directly navigates to the most recent shortcut-reachable view (e.g., (b) in Figure 1), and then replays the UI actions to navigate to the target view.

Figure 2 shows two examples of links: the first link points to page 598 in a Kindle book, and the second encodes the sequence of actions for requesting a lift in the Lyft app (the result is the dialog for entering payment). After being saved, a link can later be invoked to quickly access the view, by taking shortcuts to views that depend only on data encoded in the link (e.g., book page), and/or by replaying, in the background, the



(a) An independent view

(b) A view dependent on a previous view

(c) A view dependent on a UI action (tap on "POLLEN" tab)

FIGURE 1. Example of views uLink can support. Mobile deep links can supply only (a).

UI events encoded in the link (a clickable button in the second link in Figure 2).

uLink is implemented on Android, as a library that a developer includes in the app with minimal effort. The library continuously monitors various data dependencies and UI events of the current view so that, anytime, it can dynamically create a link with the dependencies encoded in it.

uLink DESIGN

A page in a mobile app may contain many views: the default view is what is shown on the screen when the user navigates to the page (Figure 1 (b)), and the user can navigate to a different view within the same page by UI interactions such as selecting a tab (Figure 1(c)), choosing a date from a date picker control, filling out a search box, or clicking on a search button. A UI interaction can also lead from a view to the default view of a separate page.

There are three broad classes of views a user may want to link to in an app:

1. Stateless view: A view whose state does not depend on states created in previous pages/views (e.g., a view showing weather, as in Figure 1(a)).
2. Stateful view: A view whose state depends on app states created in previous pages (e.g., showing breathing forecast at a location selected in the previous page, as in Figure 1(b)).
3. UI-driven view: A view created by UI events generated on the same page (e.g., Figure 1(c), created by tapping on the "POLLEN" tab in Figure 1(b)).

uLink IS DISTRIBUTED AS A SMALL LIBRARY THAT DEVELOPERS INCLUDE IN THEIR APPS WITH TINY CHANGES

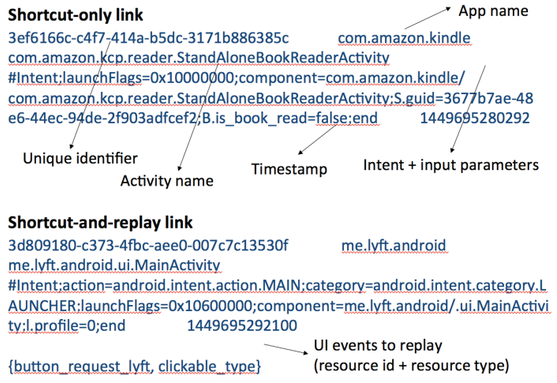


FIGURE 2. Examples of shortcut-only and shortcut-and-replay links.

Existing mobile deep links support stateless views only. They cannot observe the internal state of the app (i.e., they live outside the app), and this is precisely the reason why they cannot cover stateful or UI-driven views that depend on states (e.g., location selected by the user) and UI events (e.g., tapping on a particular UI control) inside the app. In contrast, uLink supports links to all the three types of views, and thus achieves its high coverage goal.

KEY MECHANISMS

uLink uses a novel technique called shortcuts to generate links to stateful views. We observe that a page in an app is usually instantiated through a launcher method responsible for rendering the page in the foreground *(startActivity(intent,options)* in Android and *prepareForSegue:(uiStoryboard Segue) in iOS)*. This method usually expects as input a description of the page to render and possibly other parameters, which are not known to processes external to the app. Our key insight is that uLink can program links to stateful views by demonstration: by observing how views are assembled during user interaction, uLink can learn how to re-construct them. Specifically, uLink continuously intercepts all messages sent to the page launcher method, so as to infer message structures and input parameters necessary to render a view. uLink encodes the message structure and input parameters in a URI generated for the view. To open a saved link, the uLink library simply invokes the page launcher method with properly structured messages assembled using the parameters stored in the URI. We call these *shortcut-only* links. The above idea is simple and can be implemented by overloading the launcher method of the framework page classes.

The above technique of intercepting data passed between pages does not capture UI events within a page, and hence is not sufficient to recreate a UI-driven view. To support such views, uLink adopts a limited form of record and replay. uLink continuously monitors UI events triggered during user interactions, and associated event handlers that are fired. To reduce overhead, uLink monitors UI events only in the current page; when the user moves to a different page, the UI events of the previous page are discarded. To create a link to a UI-driven view, uLink encodes two pieces of information in the link: (1) input parameters to launcher method of the current page (same as *shortcut-only* links), and (2) UI events that lead the user from the page's default view to the current view. When the link is invoked, uLink first launches the page's default view by using its input parameters, and then replays the UI events to navigate to the target view. The UI events are replayed in the background, and so the user sees the same click-and-go *experience as shortcut-only* links. We call such links *shortcut-and-replay* links.

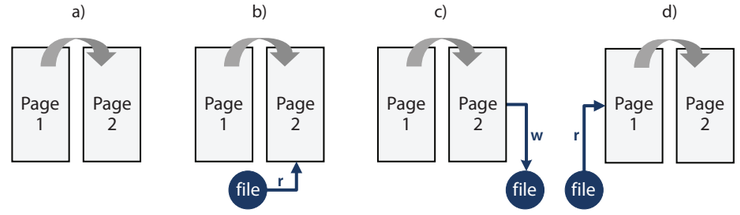


FIGURE 3. uLink can replay correctly a link to Page 2 in case a), c), and d), and in case b) if the file doesn't change after link creation.

Compared to record and replay tools [6] [7] [8], this approach does not require any recording start point, and it is much faster. On the other hand, it is limited by the fact that it captures only UI events (button clicks, checkbox selections, etc.). Capturing I/O and sensor access operations would bring us closer to the ideal world of deterministic replay, but monitoring these events would lead to unsupportable overheads in terms of annotations that developers would have to provide, in terms of OS modifications, or in terms of runtime overhead. By capturing only UI events, uLink hits a sweet spot between existing lightweight but low-coverage deep links and heavyweight but high coverage full-blown record and replay.

LINK VALIDATION

An important challenge uLink must address is identifying captured links that may not open correctly at some later point in time. Broken links are common in the web as well. A link may not open correctly e.g., if the target view opens a file that is deleted after the link is saved, if a user is logged out from the app, or if some UI events cannot be captured or replayed (e.g., Android does not provide APIs for applying long taps on list items).

uLink provides feedback to users (or applications on their behalf) at the time of link creation and of link execution. Let us consider the example of file system resources through the four cases shown in Figure 3. uLink can replay correctly a link to Page 2 in case a), c), and d), and in case b) if the file doesn't change after link creation.

uLink can correctly replay a link to Page 2 in case (a) and (c) because the link either has no external dependencies or the dependencies do not affect the content of Page 2. Also in case (d), uLink can correctly open Page 2 because if the external resource somehow affects the content of Page 2, its value must propagate through the data passed from Page 1 to Page 2. Finally, in case (b), Page 2 reads an external resource: uLink may not be able to correctly open Page 2, or it may be able to open the page, but with potentially different content. This may happen if the content of the external resource is modified after the link is created.

To address situations like case (b), uLink notifies users or companion applications at link creation or execution time that the link may not be replayed correctly, if a specific resource is modified. We call this process link validation. We rely on an offline, automated analysis of the application code to generate an app-specific summary of resource dependencies of relevant event handlers. By design, this approach cannot be as accurate as heavyweight API instrumentation, taint tracking or other approaches requiring OS modifications, but it provides a first, practical approximation of the problem, while not compromising our goals of low overhead and minimal developer effort.

COMPANION SERVICES

We have built two uLink companion services. (1) **Bookmark** (left-hand side of Figure 4) collects links to content, actions, tasks a user wishes to save. Each time a user shakes her phone, a link to the current view is saved into the Bookmark app. Links are opened by clicking on them. (2) Users browse lots of content inside their apps (e.g., hotels to book, restaurants to visit, news article to read),

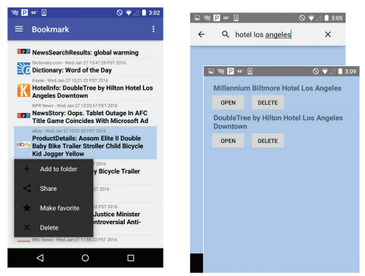


FIGURE 4. Bookmark (left) and Stuff-I've-Seen (right) services we have built using uLink.

and sometimes would like to be able to search through "all the stuff they have seen," and not through all the content those apps (or the web) offer. The **Stuff I've Seen** app (right-hand side of Figure 4) transparently logs content the user sees in her apps, indexes it, and provides a basic search capability.

EVALUATION

The uLink library was integrated successfully in 34 Android apps. Among the top 1000 Android apps, we selected apps based on popularity and compatibility with Android 5.0 from a variety of app categories, with the exclusion of games.

Developer effort

uLink is implemented as an application library. To make an app uLink-enabled, a developer includes the uLink library and extends the uLinkPage class provided by uLink, instead of the original Page class provided by the underlying framework (this is needed to overload the framework's page launcher method). Once the library is added, shortcut-only links are readily enabled. To support shortcut-and-replay links, app developers must add one line of code in each UI event handler of the app.

We counted how many LoC we had to modify to integrate uLink in our 34 test apps. To obtain an estimate for closed source apps, we counted the LoC after decompiling the app to Java source code using the dex2jar (dex2jar) [9] and jd-gui (JD-GUI) [10] tools. On average, shortcut-only required to change only 8.4 LoC in the app code. The smallest effort was 1 LoC

(Lyft), and the largest 32 LoC (Dictionary.com). The developer effort for shortcut-and-replay is higher (196 LoC on average) because it depends on the number of UI event handlers in the app, but the changes are still relatively few (on average 0.07% LoC of the entire codebase needed to be changed). Since these changes are mechanical, they could also be automated.

LINK COVERAGE AND CORRECTNESS

We evaluated whether uLink can provide high coverage of an app views. We picked 6 apps, and manually enumerated all possible views in them. Then, we manually saved links to every such view, and opened them to verify whether the result was correct. Across the 6 apps we found that on average there were 55 views one may save in a link. uLink provided coverage for 71% of them (see Figure 5). In particular, shortcut-only alone provided an average of 19 links per app, and successfully enabled links to almost all pages' default views in the tested apps. The unsupported links were mainly due to failures in replaying UI events caused by binary instrumentation. In fact, for NPR News, the only open source app of the 6 we tested, the coverage was 91%.

We also explored whether uLink can generate links that are reliable over time. We found that links are relatively stable over a short period of time (e.g., 50 days after link creation links still work) and provide the expected content. Links with dependencies on file system, sensors, and databases can break. We conducted a controlled study and found that in 94% of the cases uLink could detect a broken link and provide detailed feedback on the root cause (e.g., a file was deleted). uLink currently monitors only file system dependencies at fine-granularity. With the addition of fine-grained database analysis, we expect the accuracy to be close to 100%.

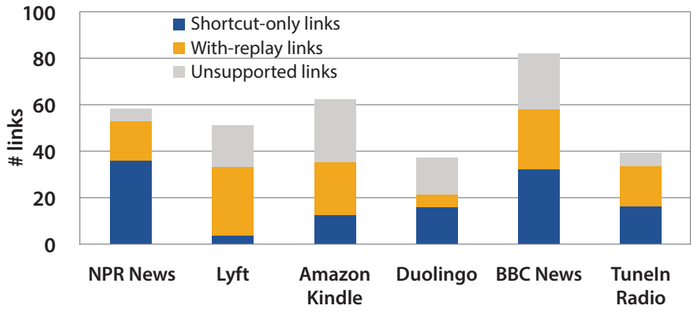


FIGURE 5. Link coverage with 6 apps (NPR News is open source, others are closed source).

CONCLUSION

uLink is a novel approach to enable deep links in mobile apps. uLink is distributed as a small library that developers include in their apps with tiny changes. Compared to mobile deep links, uLink provides higher coverage of an app views with less developer effort. uLink goes beyond the state-of-the-art: it provides links that are stateful and that can be specified by a user on demand, and it achieves these benefits without incurring large resource overheads nor modifying the OS. Although usability is not (yet) a goal of our system, uLink provides the first elements towards that goal: fast experience, no specification of a session start point, and feedback for links that may not work properly. ■

Tanzirul Azim received his Ph.D. in Computer Science from the University of California, Riverside. His dissertation work mainly focused on designing and developing techniques for fault discovery, localization, and recovery in smartphone applications. As well, he obtained his Bachelor of Science from the computer science and engineering department of Bangladesh University of Engineering and Technology.

Oriana Riva is a researcher at Microsoft Research, Redmond. Prior to joining MSR in 2010, she received her PhD from the University of Helsinki, and was a postdoctoral scholar at ETH Zurich. Her research interests revolve around mobile systems, including the programming abstractions, developer tools and cloud infrastructures required to expand their role in the computing world.

Suman Nath received his Ph. D. and Masters from Carnegie Mellon University, and B.Sc. from Bangladesh University of Engineering and Technology. He is a principal researcher at Microsoft Research. His research interests lie in the intersection of mobile, sensing, and cloud systems. His research work has been recognized by best paper awards in ACM MobiSys 2012, SSTD 2011, IEEE ICDE 2008, USENIX NSDI 2006, and IEEE/CreateNet BaseNets 2004. He is an ACM Distinguished Scientist.

REFERENCES

[1] "Enabling Deep Links for App Content," [Online]. Available: https://developer.android.com/training/app-indexing/deep-linking.html.
[2] "App Links," [Online]. Available: https://developers.facebook.com/docs/applinks.
[3] "Support Universal Links," [Online]. Available: goo.gl/9YP40S.
[4] U. Blog, "How Many of the Top 200 Mobile Apps Use Deeplinks?," [Online]. Available: http://marketingland.com/study-22-percent-top-200-apps-using-deep-links-90177
[5] T. Azim, O. Riva and S. Nath, "uLink: Enabling User-Defined Deep Linking to App Content," in *MobiSys*, 2016.
[6] Z. Mao and J. Flinn, "Can deterministic replay be an enabling tool for mobile computing?," in *Proc. of the 12th Workshop on Mobile Computing Systems and Applications (HotMobile '11)*, 2011.
[7] L. Gomez, I. Neamtiu, T. Azim and T. Millstein, "RERAN: Timing-andTouch-sensitiveRecordandReplayforAndroid. In Proc. of the 2013 International Conference on Software Engineering," in *ICSE'13*, 2013.
[8] Y. Hu, T. Azim and I. Neamtiu, "Versatile yet lightweight record-and-replay for android," in *OOPSLA'15*, 2015.
[9] dex2jar. [Online]. Available: https://github.com/pxb1988/dex2jar.
[10] JD-GUI. [Online]. Available: http://jd.benow.ca/.