SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

October 2020, Vol. 63 202101:1–202101:20
https://doi.org/10.1007/s11432-019-2646-2

MashReDroid: enabling end-user creation of Android mashups based on record and replay

Jiahuan ZHENG[1,2], Liwei SHEN[1,2*], Xin PENG[1,2],
Hongchi ZENG[1,2] & Wenyun ZHAO[1,2]

[1]*School of Computer Science, Fudan University, Shanghai* 200433*, China;*
[2]*Shanghai Key Laboratory of Data Science, Fudan University, Shanghai* 200433*, China*

Received 3 June 2019/Accepted 9 August 2019/Published online 16 September 2020

Abstract To allow end users to combine different apps for accomplishing various goals, it is desired that they can create mashups of mobile apps in an on-demand fashion. The end user creation of mobile mashups, however, is complicated by the fact that many apps do not expose interfaces for mashup and the lack of user friendly interfaces for end user programming. In this paper, we propose MashReDroid, an end user programming approach for the creation of Android mashups that incorporates the behaviors of backend apps into the execution of a host app. MashReDroid automatically transforms Android apps into mashup enabled apps. It then allows end users to create mashups by recording the interactions between host apps and backend apps and run mashups by replaying the interactions. Our evaluation shows that MashReDroid supports a variety of real scenarios and users can easily create and use mashups with a very low overhead.

Keywords end user programming, human computer interaction, Android, mashup, app transformation

Citation Zheng J H, Shen L W, Peng X, et al. MashReDroid: enabling end-user creation of Android mashups based on record and replay. Sci China Inf Sci, 2020, 63(10): 202101, https://doi.org/10.1007/s11432-019-2646-2

1 Introduction

Over the past decade, we have witnessed the burst of growth of smartphones (e.g., iPhone and Android phones) and mobile apps. Nowadays, there have been millions of apps available in Apple's app store and Android app stores (e.g., Google Play). These apps support a wide range of needs such as online shopping, social networking, gaming, and traveling. And people have been used to using mobile apps to meet their needs.

In reality, users often need to combine several apps for accomplishing a goal [1]. For example, when a user finds a restaurant on a food recommendation app like Yelp, he/she would like to search for the location on Google Map and subsequently call a taxi by Uber [2]. Currently, some apps support the integration with other apps based on predefined interfaces. For example, some payment apps (e.g., ApplePay, AndroidPay, and AliPay) and map apps (e.g., Google Maps and Baidu Map) allow other apps to integrate them by using their SDKs (software development kits). This kind of integration, however, is limited and does not support end users to combine different apps on demand.

To better meet user needs, it is desired that users can create mashups that combine existing content and services to create new applications by end user programming. Existing researches on software mashups mostly focus on web-based content and services [3–7], saying web mashups [8]. Besides, mobile mashups are native mobile apps which integrate data coming from remote or local services [9]. In this field, some

* Corresponding author (email: shenliwei@fudan.edu.cn)

mashup technologies support the creation of mashup apps [2, 9, 10], but only support web-based apps or rely on predefined components, interfaces, and templates.

The challenges with end-user creation of mobile mashups lie in the following two aspects. One is the fact that many apps do not expose interfaces for mashup. Some apps provide programming interfaces via SDKs, but these interfaces are intended for professional developers and only cover a limited part of functionalities of the apps. The other is how to provide a friendly interface for end users to specify their mashups based on existing apps. Existing mashup technologies usually provide a modeling tool like editor for users to create mashup apps. This kind of editor is not friendly for end users and cannot run on mobile devices.

In this paper, we propose MashReDroid1), an approach for end-user creation of Android mashups based on record and replay. We define Android mashups as a special form of mobile mashups which is limited to the integration of Android apps by Android specific techniques. MashReDroid supports the creation of Android mashups that incorporates the behaviors of a backend app into the execution of a host app. The host app triggers the execution of the backend app, passes values to it, and obtains return values from it. MashReDroid does not require Android apps to provide predefined interfaces for mashup. Nor does it require end users to use special purpose editors to define desired mashup apps. MashReDroid automatically transforms an Android app (in source code or packaged in an APK file) into a mashup enabled app. It then allows end users to create and run mashups by recording and replaying the interactions between host apps and backend apps.

Two user studies are designed to evaluate the applicability and usability of MashReDroid. The results suggest that MashReDroid can support a variety of mashup scenarios and both the technical and nontechnical users can easily create and use mashups with MashReDroid. We also conducted an experimental study to evaluate the runtime efficiency of MashReDroid. The results show that the mashups created with MashReDroid can effciently run with a very low overhead.

The contributions of this paper are as follows: (1) we defined the concept of Android mashups based on the interactions between host apps and backend apps; (2) we proposed and implemented a system technique for recording and replaying the interactions between Android apps; (3) we designed a user interaction mode that allows end users to create Android mashups on their mobile devices; (4) we proposed and implemented a transformation technique that can automatically transform Android apps into mashup enabled ones.

The rest of the paper is structured as follows. Section 2 introduces the technical background related to MashReDroid which supports the creation and execution of Android mashups. Section 3 defines the conceptual model behind MashReDroid. Section 4 presents MashReDroid covering its execution mechanism, recording process and app transformation technique. Section 5 evaluates the applicability, usability, and efficiency of MashReDroid. Section 6 discusses the characteristics and limitations of MashReDroid. Section 7 reviews some related work. Section 8 concludes the paper and outlines future work.

2 Background

An Android app is usually written in Java and packaged in an APK file for installation. User interfaces of an Android app are defined by its activity classes (i.e., descendant classes of *Activity*). The lifecycle of an activity class is defined by a set of callback methods, for example *onCreate*, *onWindowFocusChanged*, *onResume*, *onPause*, and *onDestroy*. These methods will be invoked by the Android system during different phases of the lifecycle of an activity object. For example, the *onCreate* method of an activity object is called when it is created; the *onWindowFocusChanged* method of an activity object is called when it becomes visible to the user. Each activity object is associated with a *PhoneWindow* object, which represents a visual window. Android delivers touch screen events to an activity object by invoking its *dispatchTouchEvent* method. The activity object then passes these events to the *PhoneWindow* object.

1) MashReDroid demo video: https://www.youtube.com/watch?v=zOVJejoSEeY.

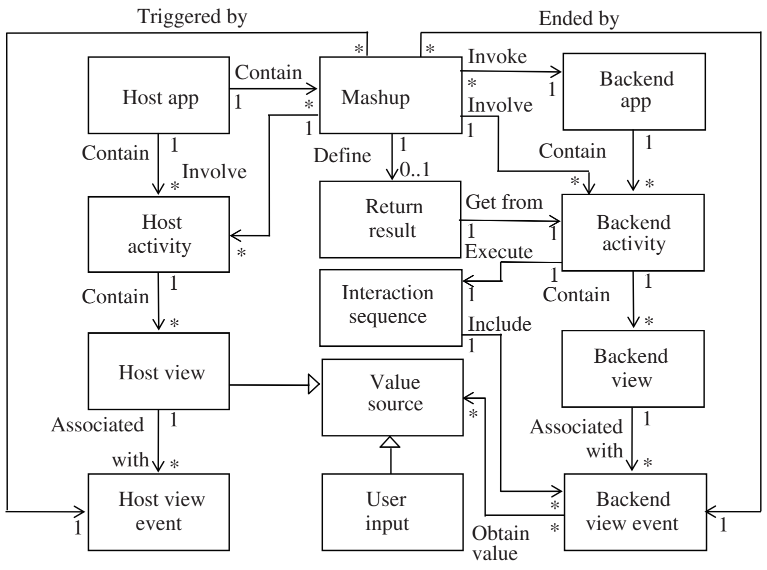


Figure 1 Conceptual model of mashup apps.

A *PhoneWindow* object is associated with a *DecorView* object, which contains all the children views of the window.

Android provides an *Intent* class to support the interactions between different apps. It can be used to send a broadcast message that can be received by any app. Android supports both explicit intents and implicit intents. Explicit intents specify the component to start by class name, while implicit intents declare a general action to perform and allow components from other apps to handle it. For an implicit intent, the Android system finds appropriate components to start by comparing the content of the intent to the intent filters declared by other apps on the device. An intent filter is an expression in an app's manifest file that specifies the type of intents that a component of the app would like to receive. The broadcast receivers (i.e., descendant classes of *BroadcastReceiver*) of an app can receive intents that are broadcasted by other apps, even when the app is not running.

To launch an installed app on a mobile device or open a specific location within the app, we can use the so-called mobile deep link, which is similar to hyperlink in the web. However, mobile deep links need to be developed statically by developers to facilitate navigation to a target page given by its link, thus only a small number of locations within an app are directly accessible via deep links [11]. Another way to repeatedly navigate to arbitrary locations of an Android app is to use the record and replay techniques [12, 13]. These techniques can record an app's execution by capturing inputs and events and automatically replay the execution.

3 Conceptualization

The conceptual model shows the set of concepts as well as their relationships which are referenced in the MashReDroid approach. Figure 1 shows the conceptual model. In Figure 1, a mashup involves a host app and a backend app. The host app runs in the front end of user devices, triggers the backend app to perform additional functionalities and obtains data from it. The backend app runs in the back end of user devices, gets inputs from and returns results to the host app. An app as a host app can contain multiple mashups, each of which implements an interaction scenario with a backend app. Note that a host app of a mashup can be involved in another mashup as a backend app.

An app (host or backend app) contains multiple activities that are involved in a mashup. Note that there may be multiple instances of an activity class that are involved in a mashup. A user needs to follow a sequence of activities and interactions to reach a specific activity. For example, to reach an activity for checking and submitting orders in an online shopping app, a user needs to first log in and then search for and choose desired products. During the process the user needs to perform a series of actions on the activities such as typing text, selecting checkbox, clicking button.

Each activity contains multiple views and each view is associated with multiple events. A mashup is triggered by a view event of the host app, that is, the backend app is triggered to run when the event occurs. After being triggered, the backend app is launched and automatically executed. The mashup is ended by a view event of the backend app, that is, the execution of the backend app is ended when the event occurs.

After being launched with the initial activity, the backend app of a mashup needs to automatically go through a series of activities to accomplish the desired functionality. For each activity, an interaction sequence, which includes a series of view events of the activity, needs to be executed by sequentially replaying the events (e.g., clicking a button). During the process, some of the view events need to obtain values from corresponding value sources, which can be user inputs captured in mashup recording or views of the host app. Note that a view event of the backend app can obtain values from multiple value sources by combining them together.

When the execution of the backend app ends, it returns a result to the host app. This result can be taken from a screen area of an activity of the backend app, which is usually the resulting activity of the event that ends the execution. If the user does not need any results from the backend app, it can return a success message that is defined by the user in mashup recording.

Figure 2 shows an example of Android mashup with New Egg (an online shopping app) as the host app and SimpleNote (a notepad app) as the backend app. The mashup implements the functionality of recording a submitted online shopping order in a notepad. The click event of the “SECURE CHECK-OUT” button of the order submission window of New Egg (Figure 2(a)) triggers the mashup. After being triggered, SimpleNote first makes the user sign in (Figure 2(b)), next lists all the notes (Figure 2(c)), then creates a new note with the order information (Figure 2(d)), and finally shows the updated note list (Figure 2(e)). The mashup records an interaction sequence that includes a series of events for each of these activities (each corresponding to a window), for example the inputs of the username and password and the click of the “Sign In” button on the note login window. An execution of SimpleNote in the mashup is accomplished by replaying these interaction sequences. During the execution, SimpleNote obtains inputs from the views of New Egg and user inputs. For example, the note login window obtains the values of the username and password from the user inputs captured during mashup recording; the note creation window obtains the values of the note title and note content from the product name, price, total of the order window. The mashup is ended by clicking the saving button (the arrow) at the left top of the note creation window. After that, the execution of SimpleNote ends with the note list window with updated note list and returns the area of the newly added note (Figure 2(e)) as the result. Finally, the result is returned to New Egg and shown in a floating panel that is dynamically generated on the current screen, such as the address window (Figure 2(f)). If the user likes, he/she can also set the return result to a success message (e.g., “Order saved in notepad!”) in mashup recording.

4 The approach

The approach named MashReDroid is explained in detail starting from the overview.

4.1 Overview

An overview of MashReDroid is shown in Figure 3, which includes three parts, i.e., app transformation, mashup recording, and mashup execution.

App transformation transforms an original Android app into a mashup enabled app, which can then be used as a host or backend app in a mashup. The transformation augments an app with the capabilities of mashup execution and recording by introducing additional components and weaving them with the app’s behaviors. Note that app transformation is not conducted for a specific mashup, but a general treatment that enables an app to be integrated with other apps in mashups.

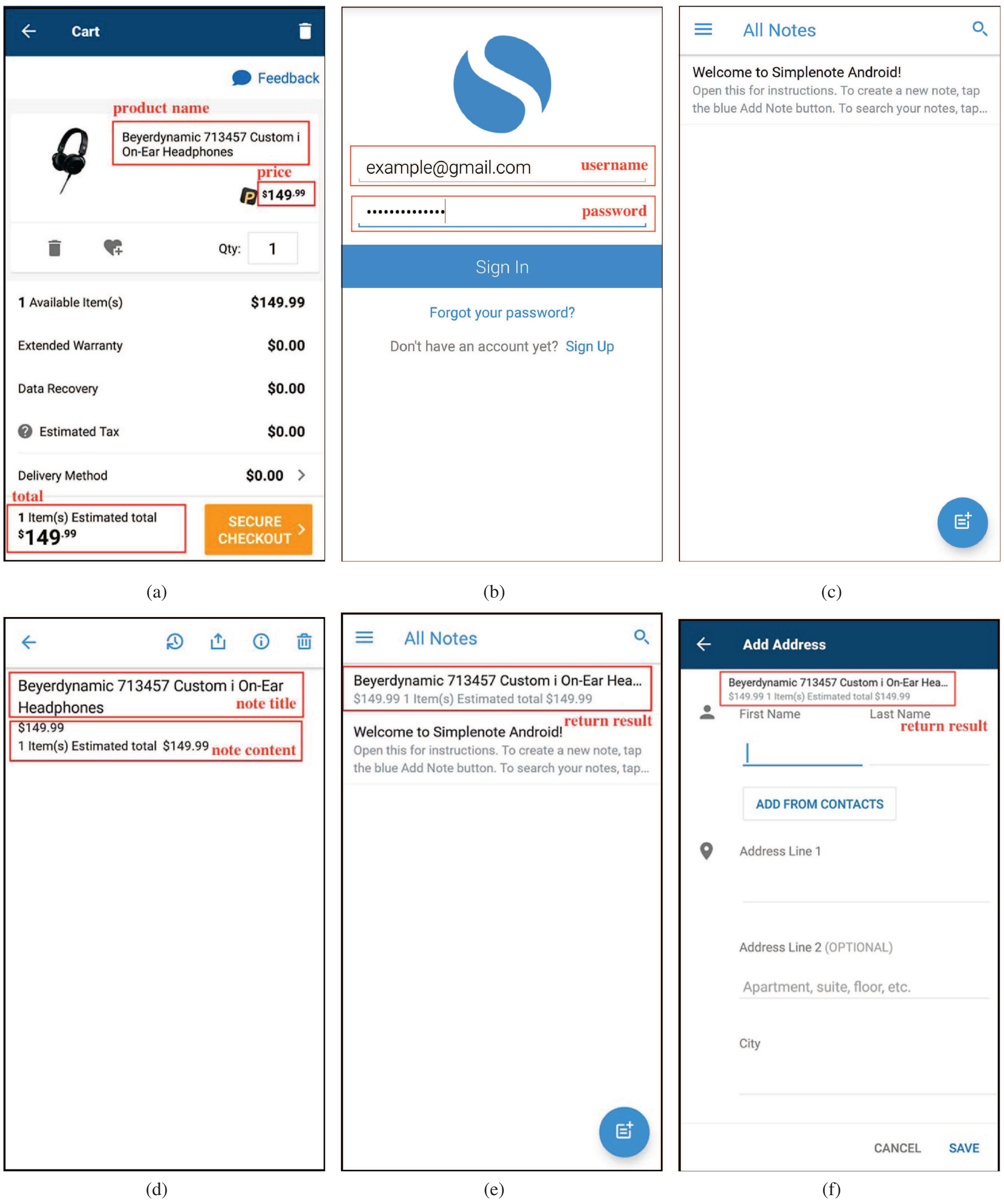

Cart
Feedback
product name
Beyerdynamic 713457 Custom i On-Ear Headphones
price
$149.99
Qty: 1
1 Available Item(s) $149.99
Extended Warranty $0.00
Data Recovery $0.00
Estimated Tax $0.00
Delivery Method $0.00
total
1 Item(s) Estimated total $149.99
SECURE CHECKOUT
example@gmail.com
username
password
Sign In
Forgot your password?
Don't have an account yet? Sign Up
All Notes
Welcome to Simplenote Android!
Open this for instructions. To create a new note, tap the blue Add Note button. To search your notes, tap...
Beyerdynamic 713457 Custom i On-Ear Headphones
note title
$149.99
1 Item(s) Estimated total $149.99
note content
All Notes
Beyerdynamic 713457 Custom i On-Ear Hea...
$149.99 1 Item(s) Estimated total $149.99
return result
Welcome to Simplenote Android!
Open this for instructions. To create a new note, tap the blue Add Note button. To search your notes, tap...
Add Address
Beyerdynamic 713457 Custom i On-Ear Hea...
$149.99 1 Item(s) Estimated total $149.99
First Name
Last Name
return result
ADD FROM CONTACTS
Address Line 1
Address Line 2 (OPTIONAL)
Apartment, suite, floor, etc.
City
CANCEL
SAVE
(a)
(b)
(c)
(d)
(e)
(f)

Figure 2 (Color online) An example of mashup. (a) Order submission; (b) note login; (c) note list; (d) note creation; (e) note list (updated); (f) address input.

Mashup recording is conducted by end users and can be regarded as a kind of end user programming of mashup apps. To create a mashup, a user initiates a mashup recording when using an app, which becomes the host app of the mashup. Next the user chooses another app as the backend app. Then the user marks the views that will act as value sources and sets the trigger event in the host app. After that, the backend app is launched and the user operates it to reach a target activity. During the process, the user inputs values for some views and sets the value sources from the host app for some other views in the backend app. The user operation events on each activity are recorded as an interaction sequence together with the value sources (user inputs or host views) of related events. Finally, the user chooses an area in the target activity of the backend app or specifies a success message as the return result. The

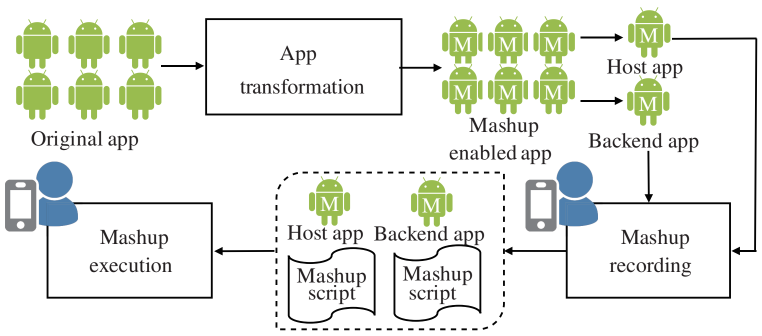


Figure 3 (Color online) Overview of MashReDroid.

trigger event, interaction sequences, value sources, return result are all recorded in a generated mashup script. Once being created, a mashup can be automatically executed many times.

A mashup is executed within its host app. When an event that occurs during an app execution matches the triggering event of a mashup, it triggers the backend app to be executed. The backend app is automatically executed by replaying recorded interaction sequences and obtaining inputs according to the mashup script. During the execution of the backend app, the host app keeps running. When the execution of the backend app ends, the returned result is shown in a floating panel on the current window of the host app.

MashReDroid implements cross-app communication by intent broadcast. It defines five types of mashup intents: MashupInquiry is sent from a mashup enabled app to all the other apps on the same device to inquire candidate backend apps; MashupInquiryResponse is the response of MashupInquiry; MashupExecution is sent from the host app to the backend app to execute a mashup; MashupRecording is sent from the host app to the backend app to record a mashup; ResultReturning is sent from the backend app to the host app to return the result of a mashup.

Mashup recording is designed based on the mashup execution mechanism and app transformation is implemented as required by mashup execution and recording. Therefore, we will first introduce mashup execution and then mashup recording and app transformation in the subsequent sections.

4.2 Mashup execution

For an app that contains mashups (i.e., acts as a host app), the user can set the permission of mashup execution on the mashup setting dialog so that a mashup contained in the app can only be triggered to execute if the mashup is enabled. For an app that is invoked by mashups (i.e., acts as a backend app), it can either be executed normally if manually launched by the user or executed by replaying recorded interaction sequences if automatically launched by a mashup. Note that an app can act as a host app in a mashup, while act as a backend app in another mashup.

Figure 4 shows the process of mashup execution. For a mashup, its host app starts a host controller for each of the mashups that are contained in the app and enabled to execute. The host controller continuously collects input values and monitors the trigger event according to the mashup script. The input values are collected from those views that are defined as value sources in the mashup script.

Once the trigger event of the mashup occurs, the host controller triggers the backend app to execute. To this end, the host controller broadcasts a mashup intent of the type MashupExecution, which includes the name of the backend app as the target app and all the input values collected for this mashup. The mashup receiver of the backend app receives the mashup intent and analyzes its content. As the intent is of the type MashupExecution, the mashup receiver launches the initial activity of the backend app by invoking the *startActivity* method provided by Android with an intent of the category of “android.intent.category.LAUNCHER”. After that, the backend controller controls the execution of the backend app.

Starting from the initial activity, the backend controller controls the execution of each involved activity. When an activity becomes visible, i.e., when its lifecycle method *onWindowFocusChanged* is invoked, the

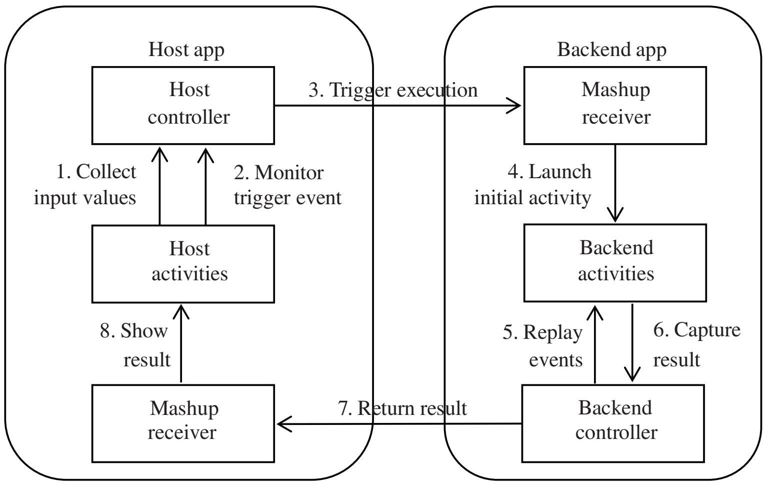


Figure 4 Process of mashup execution.

backend controller retrieves the interaction sequence associated with the current activity from the mashup script. The backend controller then sequentially replays the events in the interaction sequence by invoking the corresponding event listener methods.

This repeats the sequence of operations that the user has conducted on the same activity in mashup recording. For an event that requires input value (e.g., the input event of *EditText*), the backend controller invokes the value setting method of the corresponding view object (e.g., the *setText* method of *EditText*) to provide the value. The value is obtained from corresponding value sources according to the mashup script. If a value source is user input, the value is obtained from the user input recorded in the mashup script. If a value source is host view, the value is obtained from the broadcast mashup intent from the host app.

When the ending event of the mashup is replayed, the backend controller waits for the resulting activity to capture the return result from the area designated in the mashup script. After that, the backend controller broadcasts a mashup intent of the type ResultReturning, which includes the name of the host app as the target app and return result. The mashup receiver of the host app receives the mashup intent and analyzes its content. As the intent is of the type ResultReturning, the mashup receiver generates a floating panel showing the return result on the current window of the host app. In our current implementation, the return result is a screen shot captured from the designated area, which is sent to the host app by providing its file path on the device.

4.3 Mashup recording

Similar to mashup execution, mashup recording also involves a host app and a backend app. A difference is that in mashup execution the backend app is automatically executed at the back end by event replaying, while in mashup recording the backend app is manually executed by the user at the front end to capture interaction sequences. Another difference is that in mashup execution the user uses the host app normally, while in mashup recording the user needs to mark value sources and set trigger event.

During the recording process, the user can open the mashup setting dialog at any time by long pressing the screen of the host app or the backend app. The setting dialog shows a series of options for mashup recording, each of which indicates a specific status of mashup recording. For example, for a host app the setting dialog shows options for mashup creation, value source marking, trigger event setting.

A user starts to record a mashup by choosing the option of creating a new mashup in the host app. The host app then starts a host recorder to control the recording process. To get the list of candidate backend apps on the current device, the host recorder broadcasts a mashup intent of the type MashupInquiry. All the mashup enabled apps on the current device, which have a mashup receiver, receive the mashup intent and return their app names as response by broadcasting a mashup intent of the type MashupIn-

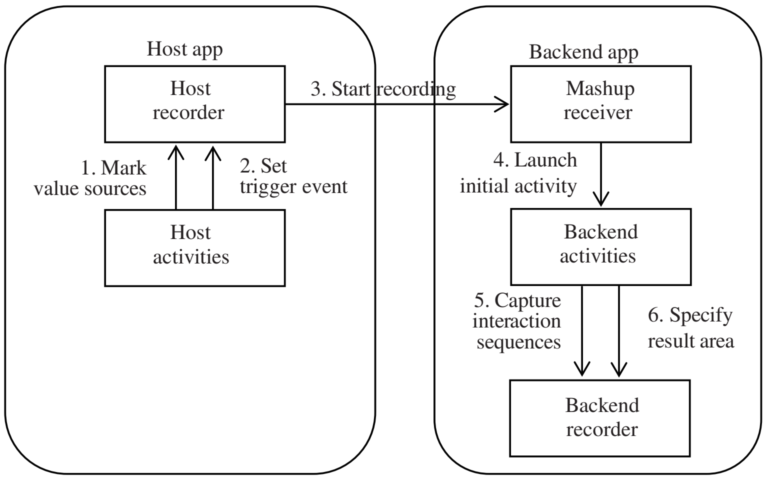


Figure 5 Process of mashup recording.

quiryResponse. Based on the responses from other mashup enabled apps, the host recorder shows a list of candidate backend apps and the user chooses one as the backend app of the current mashup.

Figure 5 shows the process of mashup recording with a chosen backend app, which includes two parts, i.e., the host part and the backend part. In the host part, the user marks host value sources and sets trigger event by touching the screen. During the process, the user can operate the host app as usual, for example, navigating from one activity to another. The default recording status of the host app is HostRecording. To mark a value source on the current screen, the user first sets the recording status to MarkValueSource by choosing the corresponding option on the setting dialog. The user then selects a view (e.g., an *EditText*) by touching it on the screen. To recognize the selected view, the host recorder intercepts the touch event in the *dispatchTouchEvent* method of the current activity and fetches the view that the event is targeted at. After that, the host recorder shows a red box surrounding the selected view and pops up a dialog for confirmation. If the user confirms the selected view as a value source, he/she is asked to give a readable name for it. The name and activity name of the selected view will be recorded in the mashup script together with a unique identifier which consists of the view's resource ID and its position in the *DecorView* object, i.e., the position in the activity layout structure. The unique identifier is used to locate a view in script execution phase according to the resource ID and the position. There is a situation that the views are not assigned with resource IDs. In this case, we create customized resource IDs for these views and the locating of the views mainly depends on their positions. After marking all the value sources in the host app, the user sets the trigger event in the host app. Similarly, the user first sets the recording status to SetTriggerEvent and then selects a view by touching it on the screen. The host recorder recognizes the selected view, shows a red box surrounding it, and pops up a dialog for confirmation. The dialog shows all the candidate events (e.g., the click event of a button) associated with the selected view and the user chooses one as the trigger event.

Once the trigger event is confirmed, the host app part of mashup recording ends. The host recorder broadcasts a mashup intent of the type MashupRecording, which includes the name of the backend app as the target app and all the host views that are marked as value sources. Similar to mashup execution, the mashup receiver of the backend app receives the mashup intent and launches the initial activity of the backend app. Then the backend recorder starts the backend part of mashup recording.

In the backend part, the user demonstrates the execution process of the backend app and specifies the value sources of related views and the return result. During the process, the user operates the backend app in a desired way and the backend recorder captures all the involved activities and the interaction sequence executed in each activity. The default recording status of the backend app is CaptureInteractionSequence. In this status, the backend recorder captures the interaction sequence of the user in each activity.

To this end, the backend recorder intercepts and recognizes all the events of each involved activity in corresponding listener methods. The captured events of an activity are recorded sequentially in the interaction sequence of the activity. For example, to capture the text change of *EditText* views, the backend recorder binds the *TextWatcher* listeners to them. When the text change occurs, the backend recorder captures the event. The text content of the view and its identifier are then recorded in the interaction sequence of the activity. For an event that requires user input in mashup execution such as the value change of *EditText*, *Spinner*, or *CheckBox*, its value source is set to the user input given in mashup recording by default.

The user can also set the value source of a captured event to a host view. To this end, the user first sets the recording status to SetValueSource and then selects a view by touching it on the screen. After that, the backend recorder pops up a dialog to list all the host views that have been marked as value sources and the user chooses a host view as the value source of the value change event of the selected backend view.

When the backend app reaches the final window, i.e., the window where a user wants to end the execution of the backend app, the user specifies the return result of the app. To this end, the user first sets the recording status to SetReturnResult and then selects a rectangular area on the screen. The position and size of the selected area are recorded in the mashup script together with its activity name. If the user does not require any return result from the backend app, he/she can specify a success message as the return result. The last interaction event captured before the return result setting is regarded as the ending event of the mashup. The setting of return result ends the mashup recording process.

The output of mashup recording is a mashup script stored on the current device. The mashup script is a serialized object based on the conceptual model shown in Figure 1.

4.4 App transformation

App transformation introduces mashup components into existing apps and can be implemented for Android apps with source code or packaged in APK files.

4.4.1 *Mashup components*

An overview of app transformation in MashReDroid is shown in Table 1. The transformation introduces components for mashup execution and recording, i.e., mashup receiver, touch manager, setting dialog, two execution controllers, and two mashup recorders, into Android apps. Each of these components has its implementation classes and may be weaved with the behaviors (e.g., lifecycle methods and listeners of activity classes) of Android apps. Moreover, the transformation adds a permission of "android.permission.REORDER_TASKS" to the Android manifest file of each app to allow a host app to be brought to the foreground when a backend app finishes its execution.

(1) Mashup receiver. Mashup receiver is a broadcast receiver (i.e., a descendent class of *BroadcastReceiver*) that implements cross-app communication. In transformation, an implementation class of mashup receiver is added into an app and a declaration of the receiver is added into the Android manifest as shown in Figure 6(a). The declaration specifies the implementation class of the receiver. In addition, it specifies an intent filter which is responsible for receiving intents for mashup.

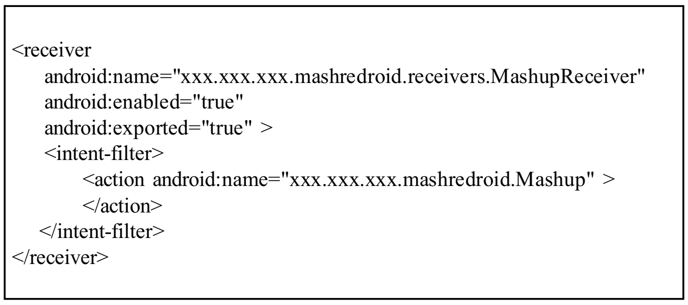
(2) Touch manager. Touch manager intercepts touch events on the screen (e.g., long pressing for opening the setting dialog, choosing a view as value source) and dispatches them to the corresponding handling components (e.g., setting dialog, host recorder). MashReDroid implements a *TouchManager* class for dispatching touch events and a gesture listener class (i.e., a descendent of interface *OnGestureListener*) called *MashupGestureListener* for responding long pressing events to open the mashup setting dialog. The behaviors of touch manager are weaved into the *onCreate*, *dispatchTouchEvent*, and *onDestroy* methods of each activity class.

The code added into the *onCreate* method creates a gesture detector (i.e., an instance of class *GestureDetectorCompat*) and associates it to the current activity and an instance of *MashupGestureListener*.

The code added into the *dispatchTouchEvent* method intercepts each touch event and passes it to the touch manager (see Figure 6(b)). The touch manager further dispatches the touch event to other components according to the type of the event and the status of the app. For example, a touch event

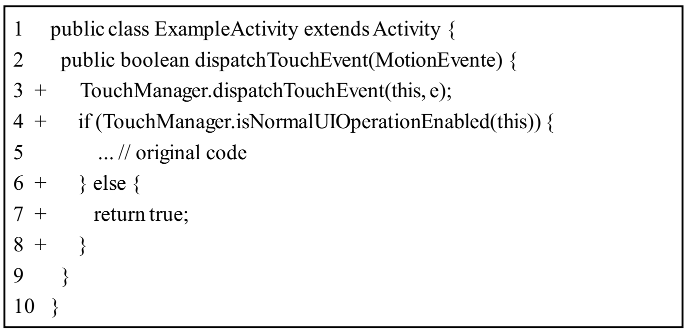
Table 1 Overview of app transformation in MashReDroid

Component		Description of added code
Mashup receiver	Implementation class	A mashup receiver class
	Manifest file	A declaration of the receiver with an intent filter for mashup to the AndroidManifest.xml file
Touch manager	Implementation class	A touch manager class; a gesture listener class
	onCreate	Create a gesture detector and associate it with the current activity and a gesture listener
	dispatchTouch Event	Intercept a touch event and pass it to the touch manager
	onDestroy	Remove the gesture detector of the activity
Setting dialog	Implementation class	A setting dialog class
Execution controllers	Implementation class	A host controller class; a backend controller class
	onCreate	Initialize host controllers; initialize backend controller; set the *currentActivity* reference to the current activity
	onWindowFocus Changed	Bind listeners to related views for capturing value sources and trigger event in host execution; replay the interaction sequence associated with the current activity in backend execution; capture and return the result to the host app in backend execution
	Listeners	Notify host controllers about value changes in event handling methods of developer defined listeners
Mashup recorders	Implementation class	A host recorder class; a backend recorder class
	onCreate	Initialize host recorder; initialize backend recorder; set the *currentActivity* reference to the current activity
	onWindowFocus Changed	Bind listeners to related views for capturing user inputs in backend recording
	Listeners	Notify backend recorder about value changes in event handling methods of developer defined listeners



```
<receiver
    android:name="xxx.xxx.xxx.mashredroid.receivers.MashupReceiver"
    android:enabled="true"
    android:exported="true" >
    <intent-filter>
        <action android:name="xxx.xxx.xxx.mashredroid.Mashup" >
        </action>
    </intent-filter>
</receiver>
```

(a)



```
  public class ExampleActivity extends Activity {
    public boolean dispatchTouchEvent(MotionEvente) {
+     TouchManager.dispatchTouchEvent(this, e);
+     if (TouchManager.isNormalUIOperationEnabled(this)) {
        ... // original code
+     } else {
+       return true;
+     }
    }
  }
```

(b)

Figure 6 Implementation codes of mashup. (a) Mashup receiver declaration; (b) touch event interception in *dispatchTouchEvent*.

will be dispatched to the gesture detector if the status is HostRecording (host app) or CaptureInteractionSequence (backend app), which further determines whether to open the setting dialog; a touch event will be dispatched to the host or backend recorder in mashup recording if the status is MarkValueSource, SetTriggerEvent, SetValueSource, or SetReturnResult. After that, the code uses the touch manager to determine whether normal UI operations are enabled (i.e., the user is manipulating the app). If they are enabled, the original code of the *dispatchTouchEvent* method is executed. Otherwise, it means that the event is a user recording operation (e.g., marking a view as value source), so the original code is bypassed.

The code added into the *onDestroy* method removes the gesture detector of the activity.

(3) Setting dialog. Setting dialog allows the user to set the status of mashup execution and recording and can be opened by long pressing the screen. MashReDroid implements a *MashupSettingDialog* class, which implements the functionalities of setting the permission of mashup execution, starting a mashup recording, inquiring and choosing candidate backend apps, setting recording status (e.g., value source marking, trigger event setting).

(4) Execution controllers. MashReDroid introduces two controllers for mashup execution, i.e., host controller and backend controller. It implements a *HostController* class for controlling the execution process in a host app and a *BackendController* class for controlling the execution process in a backend app. A host controller obtains the host side execution configuration of a mashup, including the trigger event and the views that are marked as value sources, from the mashup script when it is initialized. A backend controller obtains the backend side execution configuration of a mashup, including the views that need inputs from value sources, the interaction sequence of each involved activity, the ending event, and the result returning area, from the mashup script when it is initialized. And the behaviors of these two controllers are weaved into the *onCreate* and *onWindowFocusChanged* methods and related view listeners.

The code added into the *onCreate* method initializes the host (backend) controllers and sets the current activity. It first checks whether the host (backend) controllers of the app have been initialized if the app is being involved in mashup execution as a host (backend) app. If not, it initializes a host controller for each of the mashups that are contained in the app and enabled to execute if the app executes as a host app, or initializes a backend controller if the app executes as a backend app. After that, the host (backend) controllers set their *currentActivity* references to the current activity.

The code added into the *onWindowFocusChanged* method implements different functionalities for host controller and backend controller. For a host controller, the code binds listeners to related views for capturing value sources and trigger event. To this end, it traverses all the views that are marked as value sources or associated with trigger events. For each related view, the code dynamically binds a listener to it to capture value changes if it is a value source or monitor the occurrence of trigger events if it is associated with trigger events. Note that if a view already has a developer defined listener, the code for notifying host controllers will be statically added into the corresponding event handling methods. For a backend controller, the code replays the interaction sequence associated with the current activity, and captures and returns the result to the host app. To this end, it retrieves the interaction sequence associated with the current activity and replays each event in the sequence by invoking the corresponding event listener methods. If an event involves input values, the code invokes the corresponding setter method (e.g., the *setText* method of *EditText*) of the target view to set the value based on input values captured in the host app or user inputs recorded in the mashup script. After the ending event of the mashup is replayed and the final window is ready, the code obtains the result by taking a snapshot of the designated area on the screen or reading the return message specified by the user and then returns the result by sending a broadcast intent.

(5) Mashup recorders. MashReDroid introduces two recorders for mashup recording, i.e., host recorder and backend recorder. It implements a *HostRecorder* class for controlling the recording process in a host app and a *BackendRecorder* class for controlling the recording process in a backend app. The behaviors of these two recorders are weaved into the *onCreate* and *onWindowFocusChanged* methods and related view listeners.

The code added into the *onCreate* method initializes the host (backend) recorders and sets the current activity. It first checks whether the host (backend) recorder of the app has been initialized if the app is being involved in mashup recording as a host (backend) app, and if not initializes a new one. After that, the host (backend) recorder sets its *currentActivity* reference to the current activity.

The code added into the *onWindowFocusChanged* method binds listeners to related views for capturing user inputs. To enable the backend recorder to capture user inputs in recording, it binds a value change listener to each of the views that require input values (e.g., *EditText*). Note that if a view already has a developer defined listener, the code for notifying backend recorder will be statically added into the corresponding event handling methods of the listener.

4.4.2 *Implementation*

MashReDroid has been implemented to support the transformation of Android apps in source code or packaged in APK files. For an app with source code, MashReDroid directly manipulates the source code and manifest file to introduce the MashReDroid runtime library and transform the implementation code.

For an app packaged in an APK file, MashReDroid uses dex2jar[2] to analyze and rewrite the Dalvik byte code in the APK file to introduce the MashReDroid runtime library and weave mashup related behaviors into the app. MashReDroid runtime library includes the implementation classes of all the mashup components. It is first implemented in Java and then converted into Java byte code using java and Dalvik byte code using dx. All the revised classes, the MashReDroid runtime library, and all the other classes in the original app are packaged into a mashup enabled app (also an APK file) together with resources and the revised manifest file. Similar techniques have been used in Zheng et al.'s work [14], which provides more details about the manipulation of APK files.

5 Evaluation

To evaluate the applicability, usability, and efficiency of MashReDroid, we conduct two user studies and an experimental study to answer the following research questions.

- RQ1 (applicability). Do the requirements for the mashups of Android apps commonly exist? To what extent can these requirements be supported by MashReDroid?
- RQ2 (usability). Can the users easily create and use Android mashups with MashReDroid? How much time and effort do they need to create a mashup?
- RQ3 (efficiency). How efficient are the mashups created with MashReDroid at runtime? How much overhead does MashReDroid bring to an app in terms of memory and time?

We used a set of Android phones in our experiments and user studies, including a Redmi Note 4, a Redmi Pro, and a 360 N4S.

5.1 RQ1: applicability

To answer RQ1, we collect a set of Android apps and mashup requirements of these apps and test the feasibility of mashup recording and execution for each of these requirements. We recruit five students (one Ph.D. student and four master students) from our university as the users. These users frequently use Android phones and apps in their lives.

We collect 13 closed source apps from the Android Market and 6 open source apps from GitHub. A complete list of the collected apps can be found in the MashReDroid research page[3]. The closed source apps are selected from those that are commonly used by the users. As many mobile apps provide location based services, those apps that do not operate their services locally are excluded. The selected closed source apps cover a range of life services such as online shopping, calling taxi, booking tickets. The open source apps are selected from those that provide complete functionalities and have no serious problems

2) dex2jar: https://github.com/pxb1988/dex2jar.
3) MashReDroid research page: https://mashredroid.github.io/MashReDroid/src/index.html.

Table 2 Results of applicability evaluation

Ord	Host app	Backend app	Scenario	Supported
1	YiHaoDian	New Egg	When searching for a product in YiHaoDian, show relevant search results in New Egg for comparison.	√
2	New Egg	SimpleNote	When submitting an order in New Egg, record the order information in SimpleNote.	√
3	YiHaoDian	SimpleNote	When submitting an order in YiHaoDian, record the order information in SimpleNote.	√
4	New Egg	YiHaoDian	When adding the shipping address in New Egg, synchronize the addition in YiHaoDian.	×
5	Baidu Travel	Uber	When finding a local attraction in Baidu Travel, call a taxi to that place in Uber.	√
6	YiHaoDian	MiNote	When logging in YiHaoDian, record the username and password in MiNote.	√
7	Yidao	Uber	When looking for a taxi in Yidao, show the results in Uber for comparison.	√
8	Baidu Travel	Weather Live	When finding an attraction in Baidu Travel, obtain the weather of that place from Weather Live.	√
9	Yidao	Baidu Travel	When looking for a taxi in Yidao, obtain the attractions around the destination from Baidu Travel.	√
10	Baidu Nuomi	Uber	When checking a film shown in a cinema, call a taxi to the cinema in Uber.	×
11	HeadNews	WizNote	When reading an article in HeadNews, save the content in WizNote.	×

such as crash. The selected open source apps mostly provide practical functionalities such as taking notes and weather forecast.

The users are asked to propose as many mashup scenarios as they could based on the 19 collected apps. After that we use MashReDroid to transform these apps into mashup enabled apps and test the mashup recording and execution for each of these scenarios to determine whether it is supported by MashReDroid.

The mashup scenarios collected from the users and the results of applicability evaluation are shown in Table 2. The last column shows whether the scenario is supported by MashReDroid. From the table, it can be seen that the mashup scenarios can be categorized into the following four types.

(1) Information comparison. Users compare similar products or services from different apps (Scenarios 1, 7);

(2) Information recording. Users record the information produced in one app with another app (Scenarios 2, 3, 4, 6, 11);

(3) Additional information. Users combine additional information from an app into the context of another app (Scenarios 8, 9);

(4) Service composition. Users compose a service provided by an app into the context of another app (Scenarios 5, 10).

Among all the 11 scenarios, 8 are supported by MashReDroid. The failure of Scenario 4 is owing to the fact that the options in the drop-down boxes for the address in the two apps do not match. To solve this problem, we need to support more accurate mapping between heterogeneous data formats and semantics. The failures of Scenarios 10 and 11 are caused by the use of *WebView* (a view that displays web pages) in the apps. Currently, MashReDroid cannot capture the UI elements and events in a *WebView* page.

From the above analysis, it can be seen that the requirements for the mashups of Android apps commonly exist in mobile users' everyday lives and most of the requirements can be categorized into several common types. Most of the mashup requirements identified in our study can be supported by MashReDroid. The failure cases caused by *WebView* can be resolved in the future by extending the record and replay mechanism of MashReDroid.

5.2 RQ2: usability

To answer RQ2, we conduct a user study to evaluate the usability of MashReDroid. To compare the usability of MashReDroid for technical and nontechnical users, we recruit two groups of students from our

Table 3 Results of usability evaluation

Group	Participant	Task 1		Task 2		Task 3		Task 4		Feedback		
		#	T. (s)	#	T. (s)	#	T. (s)	#	T. (s)	(S1)	(S2)	(S3)
Group A	P1	1	42	1	58	2	123	2	104	4	4	5
	P2	3	145	1	173	1	44	1	75	4	3	5
	P3	2	103	2	132	2	91	1	65	5	4	5
	P4	2	109	2	161	1	41	1	76	5	4	5
	P5	2	162	1	146	1	61	1	140	2	3	5
	P6	2	87	1	117	1	58	1	71	5	4	4
	P7	1	29	4	211	2	93	1	48	4	5	3
	P8	4	143	1	81	1	30	1	59	4	4	3
	Average	2.1	102	1.6	135	1.4	68	1.1	80	4.1	3.9	4.4
Group B	P9	1	58	1	115	1	86	2	168	5	5	4
	P10	1	65	1	105	1	61	1	74	5	5	4
	P11	1	61	2	167	1	67	1	64	3	4	5
	P12	3	146	1	111	1	95	2	102	5	5	5
	P13	1	55	1	77	1	57	1	84	4	2	5
	P14	1	55	1	92	1	77	1	118	4	5	4
	P15	1	51	2	223	2	116	1	124	4	5	5
	P16	1	58	2	214	1	51	1	82	5	5	5
	Average	1.3	68.6	1.4	138	1.1	76.2	1.3	102	4.4	4.5	4.6

university to participate in the study as users, each with 8 members. Group A is the nontechnical group with 2 master students and 6 undergraduate students, and all of them have no programming experience. Among them 7 students major in pharmacy and 1 student majors in economics. Group B is the technical group with 3 Ph.D. students and 5 master students. All of them major in computer science and have programming experience.

The participants are trained to use mashup enabled apps to record mashups. Two of the authors give a tutorial of 10 min with an example and answer the questions raised by the participants. After that, the participants are asked to try mashup recording with the same example.

After the training session, the participants are asked to finish four mashup creation tasks by themselves. The four tasks Task 1, Task 2, Task 3, and Task 4 correspond to Scenarios 1, 3, 8, and 5 in Table 2, respectively. Each of them represents one of the four types of scenarios identified in Subsection 5.1. For each task, the participants are given an Android phone with the host app and backend app installed on it. They run the host app and create a mashup based on the given scenario. They can retry the process numerous times until the task is done. During and after each mashup creation, they run the created mashup to check whether it work as desired.

After the participants finish the tasks, they are asked to complete a questionnaire to provide feedback on the usability of MashReDroid by rating the following three statements on a scale of 1 (strongly disagree) to 5 (strongly agree): (S1) the mashups created with MashReDroid are helpful in their lives; (S2) mashup recording is easy to learn and use; (S3) the mashup recording and execution process is smooth. The questionnaire also includes open questions about the advantages and possible improvement of MashReDroid. Moreover, we conduct a group interview to learn more about their feedback on MashReDroid.

The results of the user study are presented in Table 3, which shows the participant, the times tried (#) and the time used (T.) in each task, and the rating of each statement. It can be seen that most of the participants try once and finish a mashup recording in 1 to 4 min. Some participants retry the recording of a mashup several times. This is usually caused by incorrectly selecting or missing value passing between host views and backend views owing to participants being unfamiliar with the apps or devices (some of the participants are iPhone users). Group A (nontechnical users) significantly tries more times and uses more time on Task 1, while achieves comparable performance with Group B (technical users) on the other tasks. It may indicate that nontechnical users have a steeper learning curve than technical users, but can get familiar with the recording process soon.

From the feedback of the participants on the statements, it can be seen that most of the participants

Table 4 Results of efficiency evaluation

Mashup	Memory (MB)		Response time (ms)		
	Mem.(O)	Mem.(M)	RT$_h(O)$	RT$_h(M)$	RT$_m$
M1	219.4	224.4	521.1	640.1	2144.8
M2	223.6	208.0	1000.4	1196.7	842.4
M3	212.3	207.9	–	–	1639.8
M4	201.9	204.8	–	–	1409.0

have a good impression of the usability of MashReDroid in mashup creation and usage. Group B (technical users) have a little higher rating than Group A (nontechnical users). We learn more about the advantages and possible improvement of MashReDroid from the open questions and group interview. Most of the participants agree that MashReDroid provides a new way for them to combine the services and functionalities of different apps. They say they are happy that they can create something that belongs to their own on their mobile phones. Some of them mention that they have ever encountered situations similar to the given tasks and think that creating mashups with MashReDroid is a good idea to meet their requirements. Almost all of them think that it is not hard to learn mashup recording with MashReDroid, even though some of them rarely use Android phones before. A participant without programming experience say that he can easily understand the basic concepts and operations of MashReDroid because its interaction mode (e.g., marking value sources on the screen) is intuitive and the prompts (e.g., red boxes for highlighting chosen views and floating panels for prompting the next step) provided along the recording process on the screen are quite helpful. Some participants mention an inconvenience in using the mashup created for Scenario 5 (Baidu Travel+Uber): sometimes they need to manually choose a precise address in Uber before calling a taxi because the address obtained from Baidu Travel cannot be exactly matched. This inconvenience is caused by the mismatch of information representation in different apps. Some participants also suggest some improvement of MashReDroid, e.g., supporting the value passing of multimedia information, providing mashup templates for similar situations, reducing response time in mashup recording, supporting undo in mashup recording, suggesting the views to be matched.

From the above analysis, it can be seen that most of the users can easily learn MashReDroid and use it to create and use Android mashups. Even those without programming experience can well understand the basic concepts and operations of MashReDroid and create mashups in a short time.

5.3 RQ3: efficiency

To answer RQ3, we test the runtime efficiency of MashReDroid by measuring its runtime overhead in terms of memory and time. We use the mashups created for the four tasks in Subsection 5.2 as the subjects and a 360 N4S (Qualcomm snapdragon 625 CPU, Mali-T880 GPU, 4 GB RAM) with Android 6.0.1 as the mobile device.

The memory overhead of a mashup is measured by the increased memory consumption of the host app, while the time overhead is measured by the increased response time of the trigger event in the host app. To measure the memory overhead, we use an open source performance testing tool called Emmagee[4], which can monitor the CPU, memory, network traffic, and battery consumptions of Android apps. To measure the time overhead, we instrument code into the host app (both the original version and the mashup enabled version) to compute the following two kinds of response time of the trigger event: RT$_h$, the response time for the host app, i.e., the time between the occurrence of the trigger event and the appearance of the next activity object; RT$_m$, the response time for the mashup, i.e., the time between the occurrence of the trigger event and the receiving of return result of the mashup. For each mashup, we run the two versions of the host apps (i.e., the original version and the mashup enabled version containing the mashup) 10 times and compute the average memory consumption and response time of the two versions for comparison.

The results of the efficiency evaluation are presented in Table 4, which shows the mashup, the memory consumption of the original version (Men.(O)) and the mashup version (Men.(M)), the RT$_h$ in the original

4) Emmagee: https://github.com/NetEase/Emmagee.

version and the mashup version, and the RT$_m$ in the mashup version. The four mashups M1, M2, M3, M4 corresponding to the mashups created for the four tasks Task 1, Task 2, Task 3, Task 4 in Subsection 5.2, respectively. There is no RT$_h$ for M3 and M4, as the next activity responding the trigger event is still the current activity of the host app. It can be seen that the increased memory consumption is negligible and the increased response time is small (within 23%). And the RT$_m$ is small (842.4–2144.8 ms), which is sometimes less than the RT$_h$ in the mashup version, which means that the return result can be received before the next activity is visible.

From the above analysis, it can be seen that the overhead of MashReDroid in mashup execution is very low, and the response time of the mashup execution is small.

5.4 Threats to validity

The major threat is that our user studies and experiments only involve a limited number of apps and users. Our approach and evaluation might not generalize to more complex mashup scenarios and a broader variety of users.

In the applicability evaluation, the mashup requirements are collected from five students and based on the 19 available apps. We think the collected apps and the identified four types of mashup scenarios are representative and more interesting scenarios can be identified if a broader variety of apps are considered. The record and replay mechanism of MashReDroid currently does not support some UI elements (e.g., *WebView* elements) or events (e.g., sensor events). Therefore, it is not clear whether the results of applicability evaluation may differ if more apps and scenarios are considered.

In the usability evaluation, the tasks only involve four mashup scenarios. The participants may need more time and effort to create mashups for more complex scenarios. Another threat lies in that the participants are all undergraduate or master students. Although we intentionally include some participants without programming experience, it is not clear if less educated users can easily use mashup recording to create mashups. The training session in our study is short (10 min). We believe more users can use mashup recording well if they spend more time learning it.

In the efficiency evaluation, we only test four mashups with one Android device. The resource consumption of MashReDroid's execution mechanism may differ in different apps or on different devices. Moreover, the resource consumption measured by the tool may be inaccurate.

6 Discussion

A fundamental challenge of end user programming is how to make end users understand and grasp the involved programming concepts. Although the conceptual model behind MashReDroid involves a number of concepts (see Figure 1), the programming concepts that end users need to understand are only trigger event and value passing between host views and backend views. The other concepts are hidden from the user and naturally embedded in the process of mashup recording, which follows the normal usage process of mobile apps by the user. The concepts of trigger event and value passing are directly related to user visible elements on the screen (e.g., a submit button or an address label), thus can be easily understood by end users. However, this limits the capability of MashReDroid in expressing more complex mashups. For example, the internal variables of a host app cannot be used as value sources. But this avoids breaking the confidentiality of the host app by only passing user visible content of an app to another one.

We acknowledge that there still exist several limitations of MashReDroid. First, there is a lack of branch and loop constructs in the value passing between the host app and the backend app. For example, MashReDroid does not support the mashups that need to execute different paths of the backend app based on values from the host app. Second, there may exist the mismatch of format and UI element of the same content in different apps. For example, the address shown as text in the host app may need to be passed to multiple spinners showing the province and city in the backend app. Third, the replaying of the mashup script may fail especially when the interface structure of the backend app differs from that recorded in the script. It also influences the reusability of user created mashups among different user devices. For example, when a view is removed from an activity of the backed app or its resource ID or position is changed, MashReDroid fails to locate the view in the execution process. The inconsistency is usually caused by the evolution of the app or the fragmentation issues when applying in difference devices. Fourth, the instrumentation of packaged APKs may fail in particular for those with the integrity checking ability. MashReDroid is currently unable to cross this protection mechanism, which may affect the use of this approach on some popular apps.

Introducing constructs for branch, loop, and more complex value passing can make MashReDroid support more complex mashup scenarios, but may affect the usability of MashReDroid.

7 Related work

Mashups are traditionally regarded as a way for end users to repurpose and combine existing web contents and services [3, 7, 15]. Therefore, most of the existing researches on mashup focus on the composition of web-based content, services, or APIs. Yahoo! Pipes[5)] provides a Web-based environment with a graphical user interface for building data mashups that aggregate web feeds, web pages, and other services[6)]. Marmite [3] is a Firefox plugin for end-user programming on web services. It provides a list of operators for users to extract, process and redirect data and a linked dataflow metaphor for users to understand the current state of the data. Hartmann et al. [16] developed a tool for creating web mashups based on a sampling approach that leverages pre-existing web sites as example sets and supports fluid composition and modification of examples. Lin et al. [17] developed a spreadsheet-like environment for creating mashups. It uses direct-manipulation and programming-by-demonstration techniques to automatically populate tables with information collected from various web sites. DashMash [5] is a platform for end users to mashup lightweight services into enterprise systems based on a publish-subscribe model and event-driven execution. MashupEditor [6] is an environment for non-professional users to create web mashups. By parsing web pages through a proxy, it allows users to select elements and connect them via copy-paste metaphor. Liu et al. [7] proposed iMashup, a composition framework to support mashup development and deployment for Web-delivered services. The framework contains a unified mashup component model with semantic tags facilitating developers to connect components by assigning data flows. Built upon iMashup, Ma et al. [18] introduced an approach to aid Web mashup development by suggesting helpful recommendations in an iterative way. The approach is implemented in a browser plugin called iMashupAdvisor which incorporates data-driven recommendation algorithms. The mentioned mashup methods as well as tools support developers to create applications composed of web content and web-delivered services. MashReDroid, relatively, focuses on the mashup for Android apps which is another promising domain for information integration. Moreover, tools for web mashup mainly rely on a standalone environment or editor for programming mashups, which cannot be easily used on mobile devices.

There are also some researches on the creation of mashup apps. iOS Workflow[7)] provides a personal automation tool on iOS platform. It allows users to create workflows of apps by dragging and dropping any combination of app actions which are predefined by developers based on a standard specification. SatinII app development environment [10] allows end users to visually compose social apps that are then compiled into web-based apps that can run on mobile devices. Cappiello et al. [9] developed a model-driven end user development framework for the design and automatic generation of mobile mashups. GALLAG Strip [19] is a visual programming tool for end users to create mashups of sensors and devices. It is based on a linear, if-then rule based model and supports logic for temporal relationships among different kinds of home automation sensors and devices. Ma et al. [2] proposed a data-driven and content-based mobile apps composition approach by leveraging an In-App Search mechanism, which can discover relevant services for the data and content in apps. Wang et al. [20] proposed a client-based MicroServices automatic collaboration framework to collaborate different apps by decomposing them into interfaces and then composing the interfaces. Zhou and Lee [21] introduced a system to provide location-aware semantic

5) Yahoo! Pipes. http://pipes.yahoo.com, 2007.
6) Yahoo! Pipes was shut down on 30 September 2015.
7) iOS Workflow. https://workflow.is, 2014.

mashups of GoogleMap and DBPedia. Context-aware mobile mashups are further proposed to integrate data and service in context-aware mobile applications [22, 23]. This kind of application can access to heterogeneous resources taking the user's current situation into consideration. These techniques aim to build mobile apps by the composition of predefined components and interfaces from the apps or from the other web-based services. Differently, MashReDroid allows end users to compose any functionality of an app by recording its execution process.

Deep links and record/replay of mobile apps have been widely studied for various purposes. Azim et al. [11] proposed a deep linking mechanism, which can transparently track data- and UI-event-dependencies of app pages and encode the information in links to the pages. Ma et al. [24,25] proposed an approach for automatically generating deep links of Android apps by building a navigation graph based on static and dynamic analysis. The approach generates the deep link enabled APKs without additional coding and deployment efforts. However, these deep linking mechanisms generally do not consider the interactions required in mobile mashup, for example cross-app communication and value passing, user specified execution path and user inputs, which is taken by MashReDroid into account. In addition, these mechanisms focus on exposing APIs inside one app to be accessed by other apps. Relatively, MashReDroid considers the incorporation of the backend apps and the host app, thus transforming the both apps in the solution. RERAN [12] is a record-and-replay tool by capturing and replaying low-level GUI-events directly, including touchscreen gestures (e.g., tap, swipe, and pinch) and data from sensor input devices. VALERA [13] is a stream-oriented record-and-replay tool that can record and replay sensor and network inputs, event schedules, and inter-app communication via intents. Different from these approaches and tools, MashReDroid implements a lightweight record-and-replay technique for Android mashup, which considers additional mechanisms for cross-app communication and value passing.

Programming by demonstration (PBD) [26, 27] is a technique to automate activities for end uses without programming knowledge such as web-based tasks [28] and photo manipulation [29]. Recently PBD has been applied in the automation for tasks in Android apps. Li et al. [30] proposed SUGILITE which allows end users to create generalized automation for tasks in mobile apps. It requires the end users to simply demonstrate the process of performing the task according to their conventions in regular UI. They also apply PBD techniques to complement the programming for IoT devices [31] and task-oriented chatbots [32]. Besides, Shen et al. [33] built Eco-Skills for AI Assistant by user's demonstration in one native mobile app. Bellal et al. [34] utilized PBD to build an approach that allows users to add and modify the interaction modalities of their already installed mobile applications. These studies mainly focuses on the recording of the demonstration for a single app, without involving the demonstration for multiple apps among which there exists data coupling. Compared with them, MashReDroid also follows the idea of the demonstration but emphasizes the cross-app communication and value passing.

Existing researches on program transformation or refactoring of mobile apps are targeted at different goals such as performance optimization and remote collaboration. Zhang et al. [35] developed an Android refactoring tool that can augment an Android app with on-demand computation offloading at runtime to improve the performance and save energy. Lin et al. [36, 37] developed refactoring tools that can transform incorrect or misused Android asynchronous programming constructs into correct ones for better performance. Zheng et al. [14] proposed a technique that can transform an Android app into a collaboration augmented one supporting interactive remote collaboration. The app transformation in our study is targeted at a different specific goal, i.e., enabling mashup recording and execution.

8 Conclusion

We present MashReDroid, an end user programming approach for the creation of Android mashups that combines a host app and a backend app. The host app triggers the execution of the backend app, passes values to it, and obtains return values from it. MashReDroid supports the creation and execution of mashups by recording and replaying the interactions between host apps and backend apps. The applicability, usability, and efficiency of MashReDroid have been evaluated using two user studies and an experimental study. In the future, we plan to investigate the approach further to deal with the limitations of MashReDroid. In addition, we will support more UI element and event types and further improve the user interface of mashup recording.

Acknowledgements This work was supported by National Key R&D Program of China (Grant No. 2018YFB1004800).

References

1 Xu Q, Erman J, Gerber A, et al. Identifying diverse usage behaviors of smartphone apps. In: Proceedings of the 11th ACM SIGCOMM Internet Measurement Conference, Berlin, 2011. 329–344

2 Ma Y, Liu X Z, Yu M H, et al. Mashdroid: an approach to mobile-oriented dynamic services discovery and composition by in-app search. In: Proceedings of the 2015 IEEE International Conference on Web Services, New York, 2015. 725–730

3 Wong J, Hong J I. Making mashups with marmite: towards end-user programming for the web. In: Proceedings of the 2007 Conference on Human Factors in Computing Systems, San Jose, 2007. 1435–1444

4 Stolee K T, Elbaum S G. Refactoring pipe-like mashups for end-user programmers. In: Proceedings of the 33rd International Conference on Software Engineering, Waikiki, 2011. 81–90

5 Cappiello C, Matera M, Picozzi M, et al. Dashmash: a mashup environment for end user development. In: Proceedings of the 11th International Conference on Web Engineering, Paphos, 2011. 152–166

6 Ghiani G, Paternó F, Spano L D, et al. An environment for end-user development of web mashups. Int J Human–Comput Studies, 2016, 87: 38–64

7 Liu X Z, Huang G, Zhao Q, et al. iMashup: a mashup-based framework for service composition. Sci China Inf Sci, 2014, 57: 012101

8 Daniel F, Matera M, Weiss M. Next in mashup development: user-created apps on the web. IT Prof, 2011, 13: 22–29

9 Cappiello C, Matera M, Picozzi M. End-user development of mobile mashups. In: Proceedings of the 2nd International Conference on Design, User Experience, and Usability, Las Vegas, 2013. 641–650

10 Rana J, Morshed S, Synnes K. End-user creation of social apps by utilizing web-based social components and visual app composition. In: Proceedings of the 22nd International Conference on World Wide Web, New York, 2013. 1205–1214

11 Azim T, Riva O, Nath S. ulink: enabling user-defined deep linking to app content. In: Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services, Singapore, 2016. 305–318

12 Gomez L, Neamtiu I, Azim T, et al. Reran: timing- and touch-sensitive record and replay for android. In: Proceedings of the 2013 International Conference on Software Engineering, Piscataway, 2013. 72–81

13 Hu Y J, Azim T, Neamtiu I. Versatile yet lightweight record-and-replay for Android. In: Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications, Pittsburgh, 2015. 349–366

14 Zheng J H, Peng X, Yang J C, et al. Colladroid: automatic augmentation of android application with lightweight interactive collaboration. In: Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing, New York, 2017. 2462–2474

15 Erenkrantz J R, Gorlick M, Suryanarayana G, et al. From representations to computations: the evolution of web architectures. In: Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, New York, 2007. 255–264

16 Hartmann B, Wu L, Collins K, et al. Programming by a sample: rapidly creating web applications with d.mix. In: Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology, New York, 2007. 241–250

17 Lin J, Wong J, Nichols J, et al. End-user programming of mashups with vegemite. In: Proceedings of the 14th International Conference on Intelligent User Interfaces, New York, 2009. 97–106

18 Ma Y, Lu X, Liu X Z, et al. Data-driven synthesis of multiple recommendation patterns to create situational Web mashups. Sci China Inf Sci, 2013, 56: 082109

19 Lee J, Garduño L, Walker E, et al. A tangible programming tool for creation of context-aware applications. In: Proceedings of the 13th ACM International Joint Conference on Pervasive and Ubiquitous Computing, Zurich, 2013. 391–400

20 Wang R, Chen S Z, Feng Z Y, et al. A client microservices automatic collaboration framework based on fine-grained app. In: Proceedings of 2018 IEEE International Conference on Services Computing (SCC), 2018. 25–32

21 Zhou D H, Lee Y J. Design and implementation of location-aware semantic mobile mashups. In: Proceedings of the International Conference on Intelligent Science and Technology, 2018. 72–76

22 Cassani V, Gianelli S, Matera M, et al. On the role of context in the design of mobile mashups. In: Proceedings of International Rapid Mashup Challenge, 2016. 108–128

23 Daniel F, Matera M, Quintarelli E, et al. Context-aware access to heterogeneous resources through on-the-fly mashups. In: Proceedings of International Conference on Advanced Information Systems Engineering, 2018. 119–134

24 Ma Y, Liu X Z, Du R G, et al. Droidlink: automated generation of deep links for android apps. 2016. ArXiv: 1605.06928

25 Ma Y, Hu Z N, Liu Y X, et al. Aladdin: automating release of deep-link apis on android. In: Proceedings of the World Wide Web Conference, 2018. 1469–1478

26 Cypher A, Halbert D C. Watch What I do: Programming by Demonstration. Cambridge: MIT Press, 1993

27 Lieberman H. Your Wish is my Command: Programming by Example. San Francisco: Morgan Kaufmann, 2001
28 Leshed G, Haber E M, Matthews T, et al. Coscripter: automating & sharing how-to knowledge in the enterprise. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 2008. 1719–1728
29 Grabler F, Agrawala M, Li W, et al. Generating photo manipulation tutorials by demonstration. ACM Trans Graph, 2009, 28: 66
30 Li T J J, Azaria A, Myers B A. Sugilite: creating multimodal smartphone automation by demonstration. In: Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, 2017. 6038–6049
31 Li T J J, Li Y C, Chen F L, et al. Programming iot devices by demonstration using mobile apps. In: Proceedings of International Symposium on End User Development, 2017. 3–17
32 Li T J J, Riva O. Kite: building conversational bots from mobile apps. In: Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services, 2018. 96–109
33 Shen Y L, Nama S, Jin H X. Teach once and use everywhere–building ai assistant eco-skills via user instruction and demonstration. In: Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services, 2019. 606–607
34 Bellal Z, Benslimane S M, Elouali N. Using programming by demonstration for multimodality in mobile-human interactions. In: Proceedings of the 29th Conference on l'Interaction Homme-Machine, 2017. 243–251
35 Zhang Y, Huang G, Liu X Z, et al. Refactoring android java code for on-demand computation offloading. In: Proceedings of the 27th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, Tucson, 2012. 233–248
36 Lin Y, Radoi C, Dig D. Retrofitting concurrency for android applications through refactoring. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, Hong Kong, 2014. 341–352
37 Lin Y, Okur S, Dig D. Study and refactoring of android asynchronous programming (t). In: Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), Washington, 2015. 224–235