A Mashup Construction Approach for Cooperation of Mobile Devices

Korawit Prutsachainimmit, Prach Chaisatien, and Takehiro Tokuda

Department of Computer Science, Tokyo Institute of Technology
Meguro, Tokyo 152-8552, Japan
{korawit,prach,tokuda}@tt.cs.titech.ac.jp

Abstract. The purpose of this paper is to present a description based mashup approach for integration of mobile applications, Web services, and Web applications in order to realize cooperation of mobile devices. We define a description language called C-MAIDL for describing logic of mashup. We use a mashup generator for generating mashup applications from the description. We aim to allow composition of existing mobile applications, extracted information from Web pages and RESTful Web services. We use a mashup execution environment to automate cooperation among devices. Finally, we demonstrate that our approach allows users to create mobile mashup applications dealing with cooperation of devices easily and efficiently.

Keywords: Mobile Mashup, Cooperation Mashup, Description Language.

1 Introduction

Mobile mashup is a new tool for mobile application development. It is a combination of Web resources and mobile Internet for enriching mobile services and enhancing user experiences [1]. Mobile mashup takes advantages of mobile devices' capabilities. Data from mobile sensors such as camera and GPS can be integrated with existing Web resources. Mobile applications such as map-based applications or barcode scanner applications can be an important component in mashup. With these advantages, mashup approaches were proposed to allow users to compose mashup applications for mobile environment. However, existing approaches share a common characteristic where they are targeted on mashup for single device. Existing approaches still lack attention to enable mashup for multiple mobile devices.

Recently, trend of mobile application usage is constantly changing from individual use to collaborative use. Collaborative applications such as groupware or social applications are adapted to the mobile platform. Similarly, mashup development for cooperation of mobile devices is now taken into account. With the collaboration of the devices, information from the devices can be shared and integrated with other mashup components to produce new variety of mashup output.

Mashup with cooperation of mobile devices has clear benefits for mobile computing. Multiple mobile devices can participate in a mashup to exchange information and share mashup result. A simple example is the location-based mashup. A mashup application may request location data from multiple mobile devices and use it to compute the middle coordinate among devices. Then, the middle coordinate can be given to mashup APIs to find the best nearest restaurant or choosing the lowest car rental service around the location. Finally, the selected place can be shared to all participating devices. Hence, mashup with cooperation of mobile devices can be considered as an essential research topic in mobile mashup.

In our previous work [2], we have presented a mobile mashup approach for end-users by using a description language and a mashup generator. We also have presented Tethered Web service (TeWS) to support cooperative mobile application. However, the previous work still has limitations, especially on the cooperation of devices. Thus, this research aims to improve efficiency and extend functionalities of our previous work. In this paper, we present a description based mashup approach dealing with cooperation of mobile devices. Our approach is designed for flow-based mashup where each mashup component is sequentially executed. We aim to allow the integration of mobile applications, Web applications, and Web services with cooperation information from multiple mobile devices. We propose a description language for describing mashup and use a mashup generator to leverage mashup composition effort. We develop a mashup execution environment to automate cooperation tasks among devices.

2 Related Work

Different mobile mashup solutions have been proposed to assist end-users in composing mashup for mobile environment. Mashup editors such as Yahoo Pipes [3] and Intel MashMaker [4] are capable to provide mobile mashup via mobile Web browsers. However, with these tools, we cannot integrate data from devices' sensors into mashup. TELAR mashup platform [5] presents a way to combine mobile devices' features such as GPS with existing Web resources. Kaltofen et al. presents an end-users' mobile mashup for cross-platform deployment [6]. The proposed solutions share a common characteristic where they focus on mashup development for single device. They also have limited capabilities to develop mashup for cooperation of multiple devices.

In our previous study [2], we have proposed a mobile mashup generator system for cooperative applications of different mobile devices. Our work aims to deliver the mashup development for end-users by using a description language and a mashup generator. We have applied a mobile Web server and TeWS to allow cooperation of mobile devices. However, our previous work still has limitations. To compose a cooperation mashup, manual programming effort is still required. The participating devices have to maintain a connection with other devices during the mashup process. In addition, mobile applications on client devices cannot be integrated into mashup.

In existing mashup approaches, cooperation of mobile devices is not explicitly proposed. Therefore, this research aims to find an efficient mashup approach which enables cooperation of mobile devices. Our goals are to reduce the limitations and extend the functionalities of our proposed approach to create a more powerful mashup construction system.

3 Mashup Approach

The general concept of our approach is using a description language for mashup construction. We define an XML-based description language called C-MAIDL (Cooperation - Mobile Application Interface Description Language). C-MAIDL allows mashup composers to specify mashup components and details of their integration. Our approach is designed for flow-based integration where mashup components will be executed in sequence. The mashup components can be Web applications, Web services and mobile applications. To build a mashup application, a mashup composer creates a mashup description file by using C-MAIDL. Then, the file will be used as an input for our mashup generator to generate the mashup application. Our approach aims to support mashup on single device and cooperation mashup on multiple devices. To enable cooperation of devices, we use a mashup execution environment to exchange data among participating devices.

3.1 C-MAIDL

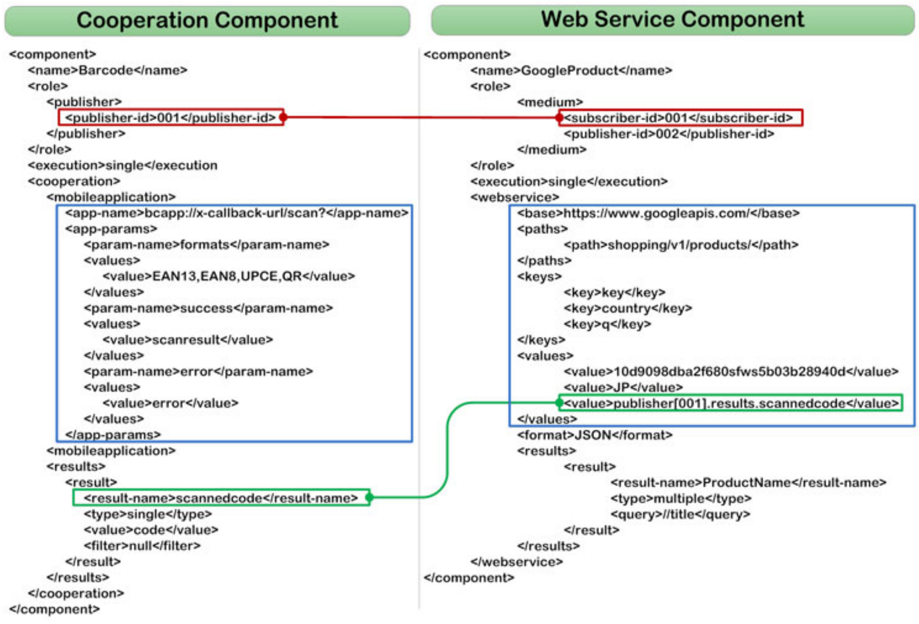
C-MAIDL is an XML-based description language which is designed for describing mashup applications. It provides ways to describe detail and data flow of mashup components which will be used in a mashup application. The components can be arranged as a workflow according to logic of mashup composers. The composers then configure each component's parameters. Results from the components in upper hierarchical order can be used in the lower ordered component. Finally, the composers configure the output component and export the abstracted model to a C-MAIDL description file.

C-MAIDL is an extension of our proposed mashup description language called MAIDL. The general concept of MAIDL is to provide data flows between mashup components for its execution and output. The components consist of Web Application Component (WA), Web Service Component (WS), Mobile Application Component (MA) and Arithmetic Component (AR). By configuring those components, mashup composers can extract parts of Web pages, consuming Web services, invoke existing mobile applications and perform arithmetic operations between outputs of components. However, MAIDL still has limitations about cooperation of multiple devices. Manual programming effort is required to create cooperative applications. Therefore, C-MAIDL is extended from MAIDL to support cooperation tasks by adding new components to the existing language definition. Additional mashup components, Cooperation Component and Output Component, are added to expand functionalities. Thus, C-MAIDL's mashup components consist of:

1. ***Web Application Component (WA).*** Web applications are applicable to our integration. This component is used for extracting a part of a Web page or querying through an HTML form. Mashup composers are provided with a Web extraction

assistant tool [7] to indicate part of required information on a Web page. The description of this component will be generated to JavaScript code and executed in the runtime environment on a mobile device.

2. ***Web Service Component (WS).*** This component is used for consuming a REST Web service by specifying a URL and query expressions (such as XPath or JSON). The target Web service will be invoked to extract a whole or a part of the result.
3. ***Mobile Application Component (MA).*** A mobile application can be used as a mashup component. This component allows the application which implemented Intent and Service [8] messaging protocol to be integrated in mashup.
4. ***Arithmetic Component (AR).*** This component provides pre-defined mathematical operations between results from one or more components. The operation includes addition, subtraction, division, multiplication, summation, comparison, and GPS distance calculation.
5. ***Cooperation Component (CC).*** This component will be used for cooperation of multiple devices. Required information from participating devices can be described in this component. The description of this component will be generated to code for communicating with the mashup execution environment to exchange information with other devices.
6. ***Output Component (OC).*** Output of mashup application can be defined by using this component. Mashup composers can select to show the mashup result as points on the map view or display as a Web page in the Web view.


Cooperation Component

```
<component>
  <name>Barcode</name>
  <role>
    <publisher>
      <publisher-id>001</publisher-id>
    </publisher>
  </role>
  <execution>single</execution
  <cooperation>
    <mobileapplication>
      <app-name>bcapp://x-callback-url/scan?</app-name>
      <app-params>
        <param-name>formats</param-name>
        <values>
          <value>EAN13,EAN8,UPCE,QR</value>
        </values>
        <param-name>success</param-name>
        <values>
          <value>scanresult</value>
        </values>
        <param-name>error</param-name>
        <values>
          <value>error</value>
        </values>
      </app-params>
    <mobileapplication>
    <results>
      <result>
        <result-name>scannedcode</result-name>
        <type>single</type>
        <value>code</value>
        <filter>null</filter>
      </result>
    </results>
  </cooperation>
</component>
```

Web Service Component

```
<component>
    <name>GoogleProduct</name>
    <role>
        <medium>
            <subscriber-id>001</subscriber-id>
            <publisher-id>002</publisher-id>
        </medium>
    </role>
    <execution>single</execution>
    <webservice>
        <base>https://www.googleapis.com/</base>
        <paths>
            <path>shopping/v1/products/</path>
        </paths>
        <keys>
            <key>key</key>
            <key>country</key>
            <key>q</key>
        </keys>
        <values>
            <value>10d9098dba2f680sfws5b03b28940d</value>
            <value>JP</value>
            <value>publisher[001].results.scannedcode</value>
        </values>
        <format>JSON</format>
        <results>
            <result>
                <result-name>ProductName</result-name>
                <type>multiple</type>
                <query>//title</query>
            </result>
        </results>
    </webservice>
</component>
```

Fig. 1. Samples of C-MAIDL description

To illustrate C-MAIDL, the description of Cooperation Component and Web Service Component are shown in Figure 1. The Cooperation Component is configured as a publisher to provide data to other components. A target mobile application and its launching parameters are specified to activate the barcode scanning on the participating devices. Output data from this Cooperation Component is defined, i.e. in this case, the "scannedcode", to be referred by the other components. The Web Service Component is configured as a subscriber and publisher. As a subscriber, this Web Service Component uses the scanned barcodes from the Cooperation Component as an input of a Web Service API. As a publisher, the result from the Web service execution will be available to the other components.

3.2 Mashup Construction Process

The mashup construction process is shown in Figure 2. To compose a mashup application, a mashup composer creates an abstract model of mashup by using C-MAIDL. The composer composes a C-MAIDL description file to transform the abstract model into mashup description. The description file will be used as an input of the mashup generator to generate Java source code. This generated code will be compiled into a mobile application which can be deployed on a target device. The mashup application can be used as an ordinary mobile application.

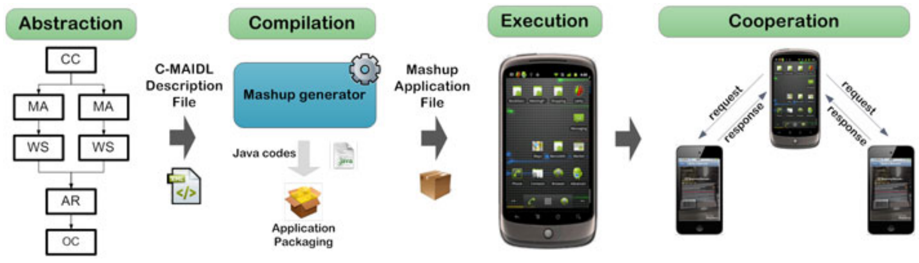


Fig. 2. Mashup Construction Process

According to the mashup generator, this tool takes a C-MAIDL description file as an input to generate a mobile mashup application. First, the generator extracts component's description from the C-MAIDL, and then generates Java source code corresponding to the specification. Next, all the source code will be manually compiled into an Android's package file (apk). Then, the package file will be manually installed to the target device by using Android Debug Bridge (adb). After the generated mashup application is installed and invoked by a mashup user, the flow which is defined in C-MAIDL description will be executed. The connection between participating devices will be established upon needs of cooperation information. The mechanism of requests and responses are automatically handled by our mashup execution environment.

3.3 Mashup Execution Environment

To achieve mashup for cooperation of devices, participating devices need a capability to communicate with each others for exchanging mashup required information. We use a mashup execution environment to automate this task. Our mashup execution environment allows the devices to exchange information by using custom mobile applications called *Cooperation Agent* and *Cooperation Center*.

In our mashup execution environment, we have categorized the participating devices into two types which are *Guest Device (Guest)* and *Host Device (Host)*. The guest device is a mobile device which provides device's information to be used in mashup applications. The host device is a mobile device which executes mashup applications by using information from the guest devices. The Cooperation Agent and Cooperation Center will work together to enable information sharing between guests and a host. The Cooperation Agent will be installed on the guest devices to provide device's information for mashup. The Cooperation Center will be installed on the host device to collect required information from guests. Overview of our mashup execution environment is shown in Figure 3.

With our mashup execution environment, mobile applications on guest devices can be integrated into mashup. When a mashup application needs information from guest devices, it will interact with the Cooperation Center. The Cooperation Center provides programming interfaces for sending request messages to all guests. When a request has arrived to a guest device, the Cooperation Agent will invoke a mobile application which corresponds to the request. The target mobile application can be specified by using the Cooperation Component in the C-MAIDL description file. For example, a request for barcode scanning, a barcode application on guest device will be executed to return the scanned code to the host device. When the host device received all response messages, it starts integrating the received information with other mashup components according to the mashup description.

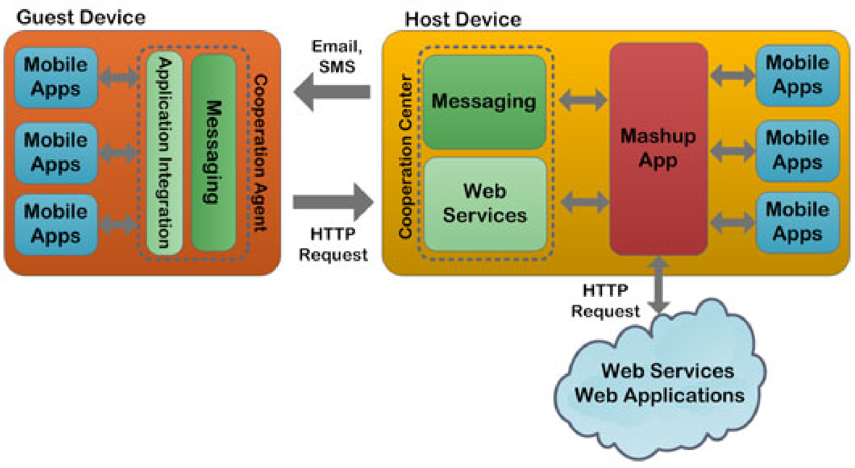


Fig. 3. Overview of Mashup Execution Environment

An important function which enables the cooperation of mobile devices is about sending and receiving information among different devices. An efficient messaging system must be taken into carefully consideration. Our approach presents a messaging system for exchanging cooperation information between mobile devices. This research uses two different mobile platforms to find an effective approach for the messaging system. Google's Android device was implemented as the host device while iOS devices were implemented as guest devices. The Android phone was selected to be the host device because it has a flexible mobile operating system. Using special purpose mobile software is possible, especially the mobile Web server. We apply functionalities of i-Jetty [9] mobile Web server in our messaging system. It is used as a container of Web service APIs to enable communication between different mobile platforms. RESTful Web services and JSON were adapted for better performance [10].

To enable cooperation between an Andriod and iOS devices, we found that there is an important limitation on iOS platform. The iOS platform does not allow a custom mobile application to run as a background process. With this limitation, the capability of communication software which requires listening to incoming request is limited. From this reason, our messaging system applies different techniques for request and response activities to overcome the limitation.

Request Message. For collecting cooperation information from guest devices, the Cooperation Center on the host device creates and sends request messages to all guest devices. The messages will be sent via a standard messaging protocol such as SMS or Email. The Cooperation Agent on guest devices will receive the messages and reply the requested information. However, according to the iOS limitation, the Cooperation Agent has to be an active mobile application to reply the request messages to the host device. To activate the Cooperation Agent we use a technique called URL Scheme Mapping [11]. The custom URL scheme (e.g. cma://) can be registered to the iOS device for invoking a particular mobile application. When a URL with registered scheme was touched by a user, the corresponding application will be brought to active context of the iOS device. In our system, the messages that are sent to guests are included with a registered URL Scheme and additional parameters. Guest device's users can invoke the URL from the received messages. Then, the Cooperation Agent is brought up to extracts query parts of the URL and determines which information is requested. User interfaces of the Cooperation Agent will ask for confirmation before replying the request. An example of request URL is shown in Figure 4-A.

A. *An example of Request for Location (URL Scheme)*

```
cma://host/cooperation/request?mid=cm001&gid=cma@me.com&cmd=gps&lat=[lat]&lng=[lng]
```

B. *An example of Location Response (HTTP)*

```
http://host/cooperation/request?mid=cm001&gid=cma@me.com&cmd=gps&lat=36.1551&lng=15
```

Fig. 4. Examples of Request and Response URL

Response Message. To return a requested data to a host device, the Cooperation Agent will determine the required resources by extracting parameters from the URL in the received message. When the URL was decoded, Cooperation Agent will invoke the target mobile application to acquire the requested information. For example, a request for barcode scanning, the Cooperation Agent will invoke a barcode scanner application and get the result after user has finished scanning. The integration of existing mobile applications is done by using *x-callback-url* specification [12]. The x-callback-url for the iOS platform is aimed to standardize the inter-application communication. However, only several numbers of iOS applications are now supporting the x-callback-url specification. Therefore, we developed testing applications conform to the x-callback-url specification to demonstrate how to enable inter-application integration in the iOS platform. To send the data back to the host device, the Cooperation Agent builds a reply HTTP request by adapting from the original requested URL, and then submits to Web service APIs on the host device. The Web services APIs are implemented with Java Servlet on i-Jetty mobile Web server. An example of response URL is shown in Figure 4-B.

4 Implementation

In order to demonstrate capabilities of our mashup construction approach, we have implemented sample mobile mashup scenarios. In this paper, we present two cooperation mashup sample called *Shopping Assistance* and *Meeting Point*. We have also discussed various aspects of our approach in this section.

To enable host's functionalities, some software is required. The Cooperation Center and i-Jetty mobile Web server must be installed on the host device. For guest devices, Cooperation Agent must be installed to accommodate connectivity among devices. In addition, to demonstrate mobile application integration on guest devices, custom mobile applications (e.g. GPS Locator and Barcode Scanner) have been installed to the guest devices.

4.1 Cooperation Mashup Scenarios

Shopping Assistance: Camera and Data Integration Mashup. This sample scenario simulates a shopping situation in a department store for 3 or more users. Goal of this mashup is to help users to compare prices of products on a local store with online stores, and then create a summary list of selected products. An Android device works as a host device. Two iOS devices coordinate with the host as guests. The guest devices will scan barcodes of selected products and send it to the host device. The host device then executes the mashup by using the collected barcodes to get information of selected products.

Mashup model and screenshots of the mashup application show in Figure 5. In this mashup, a host device sends a request for a barcode to all guest devices. The guest devices read a barcode of selected product and submit it to the host device. The barcode is given to Google's Search API for Shopping [13] to find available online stores and prices. The arithmetic component filters and extracts the lowest price. The price is converted into the designed currency with Exchange Rate API [14]. Selected products from each guest is processed and combined into a list. Finally, the list of products and comparable prices is shared among all devices.

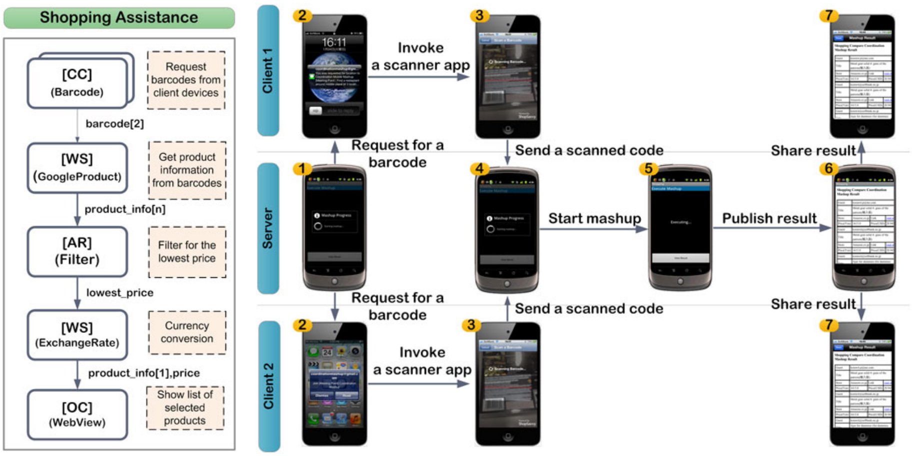


Fig. 5. Mashup Model and Screenshots of Shopping Assistance

Meeting Point: Geolocation Mashup. This mashup scenario aims to find the best ranked restaurant located near the middle point between each device's locations. Geolocation of 3 devices are used as an input to find the middle point. The middle point from arithmetic calculation is used to find the nearest train station via Google Place API Web services [15]. The best nearest restaurant around the selected train station was discovered by Gourmet Navigator API [16]. Finally, detail of the meeting point is shared among all devices by using map views.

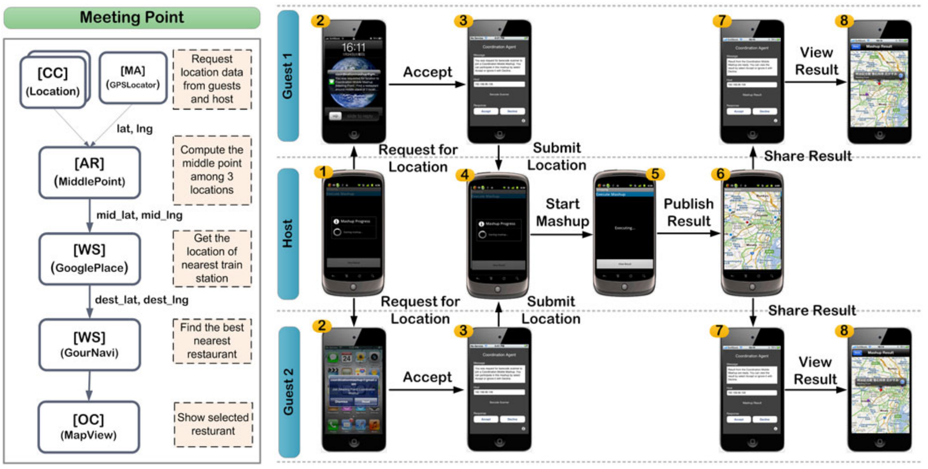


Fig. 6. Mashup Model and Screenshots of Meeting Point

4.2 Discussion

Performance. From the sample scenario, we have noticed that the major performance factors of the cooperation mashup application depend on performance of consuming Web resources and performance of the messaging system. By using multiple Web resources in a mashup application, the host device has to create multiple Internet connections to get the results. This task is resource-consuming. For instance, in the shopping assistance scenario, the major workload of the host device is for querying to Google Products Web services. As for the Cooperation Messaging performance, since our system has applied a standard protocol (e.g. Mail and SMS) for sending and receiving cooperation messages, the additional performance issue is up to the performance of theses protocols. Waiting time for sending and arriving of a message is up to servers and network utilization at that time.

Privacy Protection. For usability, users may use the cooperation mashup applications mostly with other mobile applications. The user interfaces of Cooperation Center will guide users through all the process of mashup. For guest users, our approach also provides a mechanism for privacy protection. The confirmation dialogs of the Cooperation Agent allow users to verify which information will be shared in the mashup. However, there is a trade-off between mashup execution and privacy protection. When we apply the privacy protection which required users to interact in sharing mashup information, the capability of automatic mashup execution will be disabled. User interaction is required through all process of the mashup.

Messaging System. With cooperation messaging, we assumed that participating devices are connected by using global IP addresses. Guests and host require Internet connection to consume Web resources and connect to each others. In some case, problem of losing network connection may interrupt the mashup execution. However, our messaging system leverages the failure of this case by using asynchronous manner. A host and guest devices will wait for the messages similar to waiting for an Email or SMS. User will be notified about incoming messages via the notification features of the mobile operating system. This allows the guest devices to temporarily disconnect from the network after they have shared mashup required information. Later, guest devices require the connection again when mashup result is ready. Anyway, timeout configuration should be considered in case of permanently or long-time disconnected.

Mashup Composition. The implementation of the sample scenario indicates that our approach provides an efficient solution for cooperation mobile mashup. However, our approach is not designed to support event-based mashup where mashup components are executed by events. In event-based mashup, guest devices may publish its information to the host and updating their data when an event is triggered. Host device has to aware for changing of cooperation information to update mashup result. For instance, our approach will request for locations from guests only once, but in some case, the participating devices may move to other locations. Host device needs to trace for the new locations to update the mashup results.

Scope of Integration. According to the integration of mashup components, our system is able to create mashup applications which integrate various types of mashup components. However, we found that some specific type of resources cannot be included in our mashup composition, e.g., Java Applet, Flash Object, authentication required Web services, especially, mobile applications that are not implemented with application integration mechanism. In general, a mobile application is created for a specific purpose. They may not provide the mechanism to collaborate with other applications. Thus, this kind of mobile applications cannot be used in our system.

Mobile Platform. As for the host device, in this research, we implemented host's functionality only on Andriod device. Since our messaging system uses Web services, the target platform must be able to function as a Web server and Web services container. We found that Andriod devices are suited to be the host device because several mobile Web servers are available. However, if there is a new mobile device platform which can be used as the Web service container, it may be applied as a host device for our approach. For guest devices, the participating devices have to install the Cooperation Agent that we have provided for both Android and iOS platform. We can expand coverage of mobile platforms by developing Cooperation Agent software for additional mobile operating system.

5 Conclusion and Future Work

This paper has presented a mashup construction approach that enables composition of cooperation mobile mashup. The mashup created by our approach targeted for multiple mobile devices working together for cooperation. We proposed a description language called C-MAIDL, which enables defining mashup logic and collaboration behavior. The mashup generator is implemented as a fast-paced mashup development tools aiding end-user's mashup composition. We have presented the mashup execution environment that is used to automate cooperation of devices. We have demonstrated our system applicability for cooperation mobile mashup with the sample scenario.

Our future research is targeted towards designing, implementing, and evaluating a novel mashup construction approach for cooperation of mobile devices. We want to enable event-based mashup where mashup components are executed by events. We also aim at easing the mashup composition by using a GUI mashup designer tool to create and deploy the mashup applications. Furthermore, user's evaluation should be conducted.

References

1. Jin, L., Song, M., Song, J.: Mobile Mashup architecture solution, direction and proposal. In: 2010 IEEE 2nd Symposium on Web Society, SWS (2010)
2. Chaisatien, P., Prutsachainimmit, K., Tokuda, T.: Mobile Mashup Generator System for Cooperative Applications of Different Mobile Devices. In: Auer, S., Díaz, O., Papadopoulos, G.A. (eds.) ICWE 2011. LNCS, vol. 6757, pp. 182–197. Springer, Heidelberg (2011)

3. Yahoo Pipes, Inc. (2008), `http://pipes.yahoo.com/`
4. Intel Corp.: Mash maker (2007), `http://mashmaker.intel.com/web/`
5. Brodt, A., Nicklas, D., Sathish, S., Mitschang, B.: Context-Aware Mashups for Mobile Devices. In: Bailey, J., Maier, D., Schewe, K.-D., Thalheim, B., Wang, X.S. (eds.) WISE 2008. LNCS, vol. 5175, pp. 280–291. Springer, Heidelberg (2008)
6. Kaltofen, S., Milrad, M., Kurti, A.: A Cross-Platform Software System to Create and Deploy Mobile Mashups. In: Benatallah, B., Casati, F., Kappel, G., Rossi, G. (eds.) ICWE 2010. LNCS, vol. 6189, pp. 518–521. Springer, Heidelberg (2010)
7. Guo, J., Chaisatien, P., Han, H., Noro, T., Tokuda, T.: Partial Information Extraction Approach to Lightweight Integration on the Web. In: Daniel, F., Facca, F.M. (eds.) ICWE 2010. LNCS, vol. 6385, pp. 372–383. Springer, Heidelberg (2010)
8. Android Developers, `http://developer.android.com/index.html`
9. i-Jetty, `http://code.google.com/p/i-jetty/`
10. Tsai, C.-L., Chen, H.-W., Huang, J.-L., Hu, C.-L.: Transmission reduction between mobile phone applications and RESTful APIs. In: Proceedings of the 2011 ACM Symposium on Applied Computing (SAC 2011) (2011)
11. iOS Dev Center, `https://developer.apple.com/devcenter/ios/index.action`
12. x-callback-url, `http://x-callback-url.com/`
13. Search API for Shopping, `http://code.google.com/apis/shopping/search/`
14. Exchange Rate API, `http://www.exchangerate-api.com/`
15. Google Place API Web Services, `http://code.google.com/apis/maps/places/`
16. Gourmet Navigator API, `http://api.gnavi.co.jp/api/manual.html`