Mobile Mashup Generator System for Cooperative Applications of Different Mobile Devices

Prach Chaisatien, Korawit Prutsachainimmit, and Takehiro Tokuda

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

Abstract. This paper presents a development and an evaluation of a mobile mashup generator system to compose mobile mashup applications and Tethered Web services on a mobile device (TeWS). With less programming efforts, our system and description language framework enables a rapid development, a reusability of working components and a delivery of new cooperative mobile mashup applications. Working components in the mashup execution are derived from a combination of existent mobile applications, JavaScript automated Web page data extractions, and RESTful Web service consumptions. The state of art generator system is evaluated with novice and expert composer groups, to validate the usability of the system and the expressibility of our Mobile Application Interface Description Language (MAIDL). Complex mashup examples are provided to demonstrate new cooperative applications of generated Web services, which enable platform-independent functionality exchange across devices via tethered HTTP communications.

Keywords: Mobile mashup application, description language, tethered Web service, mobile Web server.

1 Introduction

A software development for mobile devices is becoming an essential task in the field of information technology. The devices' unique capabilities such as on-the-go Internet access, GPS and camera-based applications are becoming the key functional components in developing modern mobile applications. The major drawback when creating a multi-platform mobile Web application is that it tends to make less use of the device's useful features. Mobile mashup development using the device's native programming language requires more explicit knowledge and does not allow the mashups to be developed as fast as the Web-based ones are.

As an alternative, the second iteration of approaches is related to the code-to-code (C2C) [1], [2] and model-to-code (M2C) [3] approaches. The C2C approach, however, is not designed for developers (or mashup composers) without programming skills. This approach emphasizes on a conversion of Web language to the device's native programming language and allows sensors be accessed with less code having to be written. On the other hand, the M2C approach tends to promote Web mashup without the integration of mobile devices' unique features.

Our study addresses problems found in these platform centric and model-code centric approaches by providing a fast-paced development using an XML-based description language called Mobile Application Interface Description Language (MAIDL) and its mashup composition tool. With less programming efforts, composers can freely integrate parts of Web information (annotated Web navigations, and queries from Web services) and mobile devices' unique features (existent mobile applications accessing device's sensors). We proposed automatic code generation algorithms for mobile mashup application, which are *Mashup Output Context Transformation* and *Mashup Process Scheduling* algorithm. The output application can be designated for a single device, as a normal mobile application, and for multiple devices, as a Tethered Web service on the mobile device (TeWS). Thus, platform-independent communication between devices for functionality exchange and cooperative application can be simply created.

2 Related Work

Our previous study [4] proposed a composition model for mashup of Web applications, Web services and mobile applications. This approach shows the reusability of Web information and mobile devices' capabilities to outline a mashup application. As a continuation, this research aims at the composition tool to deliver the *End Users Mashup Development on Mobile Devices*. We also extend our approach's expressibility to compose cooperative mashups for multiple devices via a TeWS. The contrary point of our approach to the conventional mobile Web mashup [5] is that other approaches extensively create or reuse Web information as a part of user interface without integrating devices' sensors and existent application. Mashups also cannot be created as a TeWS.

In academic research, mashup approaches are often referred to as a M2C approach in which a composer's created composition model is later translated to programming code. These approaches allow the integration of Web application from many Web application components. In applying these models to mobile mashups, the runtime of mashups on actual devices is not as powerful as the one used in PCs or Web servers. In order to reuse the same composition pattern on a mobile device, the generation and execution procedures of the mashup application have to be adjusted for a limited computation environment [6].

In contrary to the M2C approach, the recent C2C mobile development approaches are designed so that developers use Web languages to build multi-platform mobile applications. Despite the C2C advantage in its Web language compatibility, its frameworks are not designed for non-programmers. This research is based on the M2C approach, which consists of 2 parts, M2M (model abstraction) and M2C (automatic software generation).

Code, which is generated from MAIDL, is in a procedural paradigm since the control part mainly consists of procedures that are passing results and synchronizing processes in the mashup runtime environment. For this reason, we proposed automatic code generation algorithms, which assist composers in creating mobile mashup applications. Moreover, the final output is not limited to mobile application as traditional methods are [7] [8]. A TeWS can be generated and later consumed by other clients. Later in complex mashup examples, we show how the composed TeWS is applied to platform-independent communications between devices. In addition to the main composition tool, the Web extraction assistant tool (WXTractor) in this research is created based on methods of Web APIs and Web automation [9] and partial information extraction [10]. The tool automatically generates mashup execution parameters for MAIDL from an annotated tag of an HTML document. To expose these features to novice composers, our tool also visually provides result samples in the model abstraction process.

3 An Overview of Our Research Approach

3.1 Objective and Motivation

1. *Explore a mobile mashup model.* The conventional disciplines discussed in section 2 show that a mashup model for the mobile mashups is not concretely defined. We aim to find an optimal mashup model which leads to a better solution in creating mashups for mobile devices.
2. *Deliver reusability.* Our mashup components include existent mobile applications and Web information. Therefore, developing mashups with low-level API, such as creating an image recognition component with a new algorithm, is beyond our research scope.
3. *Enable fast prototyping.* Mashups can be created from a Web-based software generation tool. Composers are allowed to generate source code, compile, and test it immediately after the composition model is correctly prepared. Methods called *Mashup Output Context Transformation* and *Mashup Process Scheduling Algorithm* would assist composers by automatically managing foreground and background runtime behaviors of the mashup components.
4. *Target novice and expert composers.* Most of the abstraction models are designed for programmers. Through our composition tool, we aimed to let novice composers be able to create mashups for mobile devices. More advanced and customizable features are added for expert composers.
5. *Demonstrate a TeWS.* A mashup in our approach can be created as a mobile application to run on a device or as a TeWS. Functionality exchanges and interactive collaborations between devices can be derived from our approach, and these are unique contributions which do not appear in other approaches.

In order to run most flexible configuration on mobile devices (such as third party mobile applications and embedded server modules), we use the Android open source platform [11] as our mashup runtime environment.

3.2 MAIDL and its Abstract Model Composition

The general concept of MAIDL (shown in Fig. 1) is to provide data flows between mashup components for its execution and output. The components consist of:

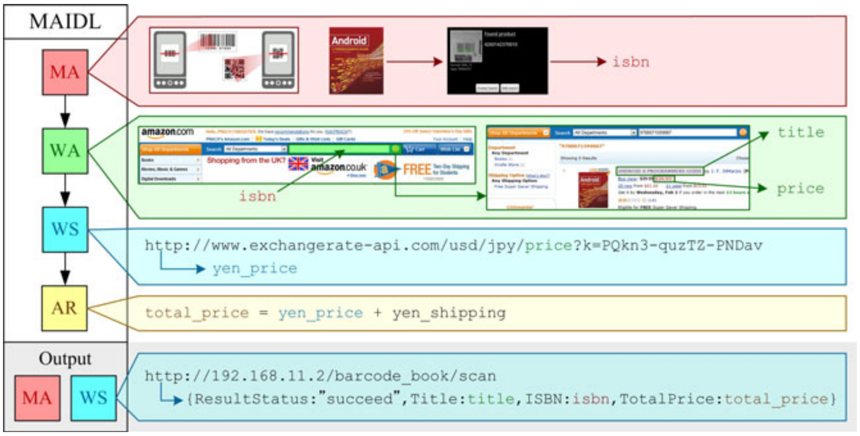
MAIDL
MA
WA
WS
AR
Output
MA
WS
isbn
title
price
isbn
http://www.exchangerate-api.com/usd/jpy/price?k=PQkn3-quzTZ-PNDav
yen_price
total_price = yen_price + yen_shipping
http://192.168.11.2/barcode_book/scan
{ResultStatus:"succeed",Title:title,ISBN:isbn,TotalPrice:total_price}

Fig. 1. Overview of MAIDL and its abstract model composition

1. Web Application Component (WA). A part of a Web page or a query through HTML forms can be reused through a WA component. Composers are provided with a tool called WXTractor to annotate HTML tags and specify execution commands. JavaScript code will be generated according to the specification and executed in the runtime environment on the mobile device.
2. Web Service Component (WS). Connections to REST Web services are applicable to our composition. Composers specify a URL and a query expression (such as XPath or JSON dot notation) to access a part of the whole data.
3. Mobile Application Component (MA). A part of mashup execution can be derived from a mobile application. Our method allows an application which implemented Intent and Service messaging protocol [12] to be integrated.
4. Arithmetic Component (AR). A mathematical operation between results from one or more components can be performed through AR. The operation includes addition, subtraction, division, multiplication, summation, comparison, and GPS distance calculation from 2 pairs of GPS coordinates.

The composition process begins where composers select the output context (mobile application or TeWS). Components can be added in any order but not starting with the AR component. Composers then configure each component's parameter according to their data flows and logical specifications. Results from the publisher components attached in the upper hierarchical order are listed and are selectable in the nested subscriber components. Finally, composers configure the output component and export the abstracted model to a MAIDL script file.

3.3 Mashup Mechanism, Output Context and Process Scheduling

The mechanism in each output context is different Therefore, it is crucial that composers specify the context first. The mashup composition tool is

Table 1. Mashup mechanism in a mobile application and a TeWS output context

Output Context	Runtime Process	Inter-Component Messaging Protocol	Working process in detail
MA	Fore-ground (FG)	WA:Intent(FG) or Service(BG)	WA: Custom browser and JavaScript navigation
		WS: Service(BG)	WS: HTTP connection and message retrieval/query
		MA: Intent(FG) or Service(BG)	MA: Intent or Service call

TeWS	Back-ground (FG)	WA:Intent(FG/SMA) or Service(BG)
		WS:Service(BG)
		MA: Intent(FG/SMA) or Service(BG)
			Same as MA output context

context-sensitive and will allow integrations of only compatible components. Table 1 shows the mashup mechanism in each output context.

Mashup Mechanism in a MA Context. Components in this context work separately as mobile applications. WA components employ a custom Web browser and JavaScript navigation sequences generated from MAIDL. In the case where form submissions (e.g. fill a form with keywords and click on a submit button) or user interactions (e.g. select a link from search results) are required, the runtime will work in a foreground process. When a part of a static Web page is extracted without user interaction, the WA components work in a background process. WS components always run in a background process. The process includes HTTP connections, message retrievals and queries of the data. The runtime of MA components depends on its messaging protocol. If the Intent protocol is used, they work in a foreground process. When the Service protocol is used, they work in a background process. After all components finished their tasks, all parameters will be passed to the final output mobile application according to the configuration.

Mashup Mechanism in a TeWS Context. A mashup runtime in a TeWS output context does not allow MA components to be called directly, for the reason that the process in WS component itself runs as a background process. Therefore we use a *Switcher Mobile Application* (SMA) to indirectly call each MA component in the same way the mechanism works in the MA output context. Fig. 2 shows the messaging processes in a normal direct call and an indirect call via SMA. The final TeWS, is generated from results passed from each component. The TeWS can be accessed using the device's IP address and the path specified in the project's MAIDL script. For performance reasons, we applied the REST architecture to the TeWS output, to deliver JSON messages.

Mashup Process Scheduling. Processes in a linear FIFO order are scheduled according to the hierarchical order of a publisher-subscriber pattern and their runtime behavior. If no parallel execution occurred, a process will sequentially wait for prior processes to finish its task. In a parallel execution, processes under a

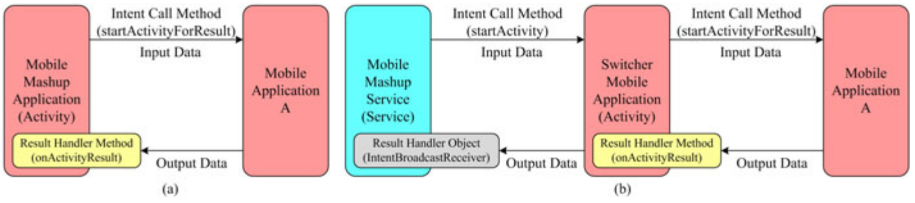
Intent Call Method
(startActivityForResult)
Input Data
Mobile Mashup Application (Activity)
Result Handler Method (onActivityResult)
Output Data
Mobile Application A
(a)
Mobile Mashup Service (Service)
Result Handler Object (IntentBroadcastReceiver)
Intent Call Method
(startActivity)
Input Data
Output Data
Switcher Mobile Application (Activity)
Result Handler Method (onActivityResult)
Intent Call Method
(startActivityForResult)
Input Data
Output Data
Mobile Application A
(b)

Fig. 2. Messaging processes using a direct call (*a*) and an indirect call (*b*)

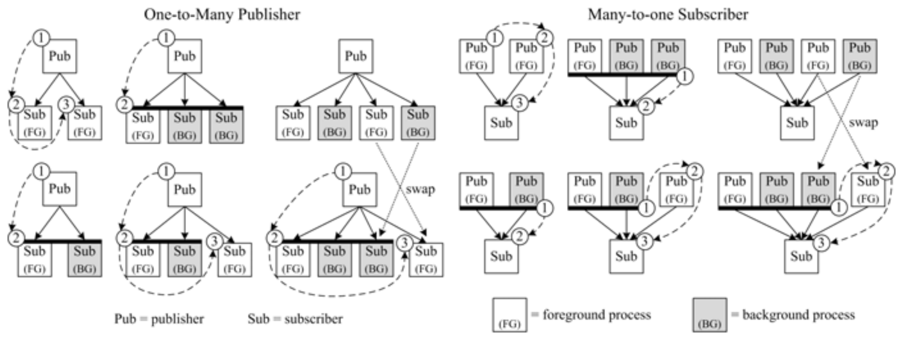

Fig. 3. *Mashup Process Scheduling Algorithm.* This figure shows data flows (*solid lines*), task sequences (*dashed lines*), merged sequences (*black bars*) and swaps (*dotted lines*)

one-to-many publisher are executed from a top-to-bottom order as defined in the MAIDL script file. Many-to-one subscribers or the terminal output component wait for all prior components to finish before its own execution begins.

The process synchronization in our runtime environment employs a Java Thread to wait for messages sent from components when its task is finished. As demonstrated in Fig. 3, the algorithm merges execution sequences of background processes to the latest foreground process in the same hierarchical order. The process order is swapped to execute background processes with the first foreground process in the same hierarchical order. The equations below shows how the total execution time t is reduced to t' after the algorithm is applied. i and j are process indexes after the algorithm is applied, where $n + m \geq i + j$

$$t = \sum_{N}^{n=0} t(F_n) + \sum_{M}^{m=0} t(B_m) \ . \tag{1}$$

$$t' = \begin{cases} \sum_{N}^{n=0} t(F_n) \text{ if all } t(F_n) \geq t(B_m) \ . \\ \sum_{I}^{i=0} t(F_i) + \sum_{J}^{j=0} t(B_j) \text{ if some } t(F_m) < t(B_m) \ . \\ \sum_{M}^{m=0} t(B_m) \text{ if all } t(F_m) \leq t(B_m) \ . \end{cases} \tag{2}$$

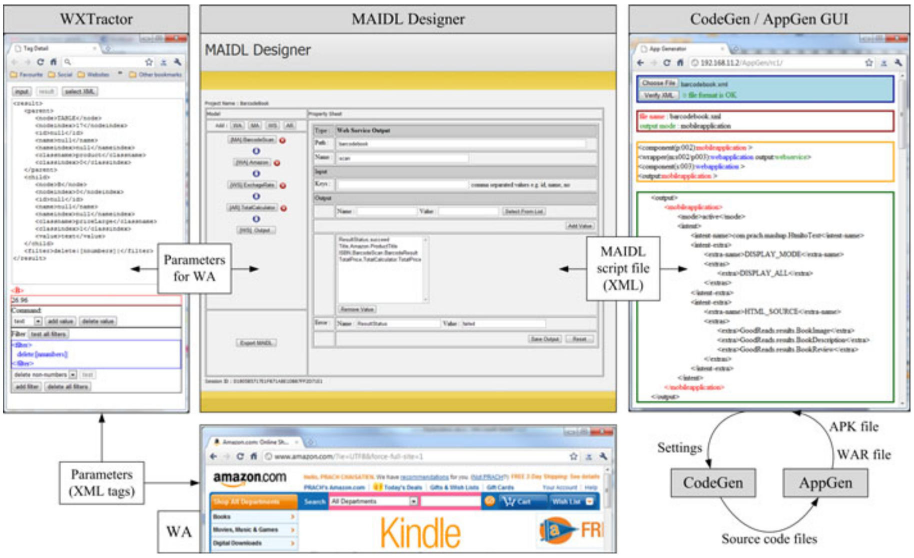

Fig. 4. Screenshots and a workflow in each mashup composition tool

3.4 Mashup Composition Tool

1. *MAIDL Designer* is a Web-based XML document builder, which provides a grammar check and context-sensitive settings. Composers select the output context and then add mashup components. The output of this tool is an XML file (a MAIDL script file), which works with the Code Generator.
2. *Web Extractor* (WXtractor) is a Web browser extension, which generates XML tags used in WA components in MAIDL script files. It will be available for use when composer adds a WA component into his/her mashup. WXtractor provides a visual preview of the annotating part of a Web page (e.g. text content, HTML source code, link URL and image representation if applicable). Composers are also able to specify navigation sequences (e.g. text input, highlight, click) and data filtrations (e.g. delete commas or dollar signs from the text content) using this assistant tool.
3. *Code Generator* (CodeGen) receives a MAIDL script file generated from the MAIDL Designer or manually created by composers. It generates Java source code according to the output context. MA output context's code is generated as files compatible with Android's SDK. The TeWS output context's code is generated as i-jetty [13] files.
4. *Application Generator* (AppGen) is a Web-based compiler, which applies a compilation command to source code generated from CodeGen. It returns a URL link of an mobile application package file (Android package - APK file) or a Web service package file (i-jetty compatible Web archive - WAR file).

4 Evaluation

The evaluation is divided into 3 sections. First, *MAIDL and Mashup Tool* section shows the expressibility test result of MAIDL in composing mashups. Use cases of cooperative applications are given in *Complex Mashup* section. Finally we discuss the mashups' overall *Security Performance* in the third section.

4.1 MAIDL and Mashup Tool

In the evaluation of our tool and description language, we tested the *MAIDL Designer* with 2 subject groups, 5 novice and 6 expert composers. A pre-questionnaire was given to observe the composers' background. After the tool session finished, we also asked composers to fill in a post-questionnaire about the tool's features, their creations, MAIDL's expressibility and their expectations.

The tool session was divided into a tutorial and a freestyle task. The first task was given to let composers use all mashup components (WA, WS, MA and AR) and basic features. The mashup is the one shown in Fig. 1, composers integrated a barcode reader a MA component to read a product barcode, search for the product title and price from *Amazon.com* WA component [14], translate the price currency from dollars to yen using *ExchangeRate* WS component [15], add shipping cost using the AR component ,and finally display the product title and total price via a TeWS. Later, composers were given the freestyle composition task where they could freely plan and create a mashup using our *MAIDL Designer*. We compared the composer's expectations and MAIDL's expressibility by using 5-point Likert scale questions. The data was analyzed using T-test and ANOVA, divided by novice/expert and pre/post task. Finally, the composers asked for comments concerning the mashup runtime and usability.

Composer Groups. Novice composers were able to use the Internet and mobile applications. Our expectation was that they would be able to use our tool to compose mashups with basic functions. We also expected that a similar basic usage pattern amongst novice composers would be found. Expert composers were involved in the use of Web information and are able to program in one or more languages. We expected that they are able to use most of the features provided, suggest corrections and propose more functionality to our system.

Pre-questionnaire Result. The majority of novice composers understand HTML and its simple tags (forms and links). They are familiar with mobile applications but none of them understand the concept of Web services. Novice composers do not know what a mashups are. When asked to give an example of a mobile mashup, they were able to propose ones using GPS and camera functionality as a terminal input. About their expectations for MAIDL (question set C_1), high rating scores were found in: connecting the mashup to a Web service ($\bar{x} = 4.00, \sigma = 1.67$), data filtration ($\bar{x} = 4.00, \sigma = 1.10$), automatic code generation ($\bar{x} = 4.00, \sigma = 1.34$) and creating software package files ($\bar{x} = 4.00, \sigma = 1.73$).

Most of the expert composers understand HTML at the level of tags' attributes. They are familiar with mobile applications and all of them understand the concepts of Web services and mashups. Mashup examples given by these composers consisted of a wider variety of existent mashup components. Their expectations rated in question set C_1 were low on average since they are highly skilled in programming. High ratings score were found in: connecting to the mashup to mobile applications ($\bar{x} = 3.17, \sigma = 1.60$) and automatic process synchronization ($\bar{x} = 3.67, \sigma = 0.52$). We found low ratings for: the use of mathematical functions ($\bar{x} = 2.00, \sigma = 0.89$) and data filtration ($\bar{x} = 2.00, \sigma = 1.55$).

Tutorial Composition Task. Novice composers finished the task in an average of 34 minutes ($\sigma = 2$). About the expressibility of MAIDL (question set R_1), we found high ratings in these items: planning a workflow using MAIDL ($\bar{x} = 4.00, \sigma = 0.71$) and data filtration ($\bar{x} = 4.00, \sigma = 0.55$). Novice composers are likely to follow the tutorial steps with less details studied.

Expert composers took longer to finish the task with the average of 55 minutes ($\sigma = 27$). In question set R_1, we found high ratings in: planning a workflow using MAIDL ($\bar{x} = 4.00, \sigma = 0.71$), WA components ($\bar{x} = 3.67, \sigma = 0.52$), WXTractor ($\bar{x} = 3.83, \sigma = 1.60$) and data filtration ($\bar{x} = 4.17, \sigma = 0.75$). From further observations, we found that they took more attention to the reusability of Web applications after WXTractor and the data filter functions were used. Expert composers also tend to spend more time study all features in detail.

Freestyle Composition Task. Novice composers were able to plan the dataflow, choose the right components, and lay out their abstract model using MAIDL. However, advanced features (such as background processing, loop execution) were not used. Therefore, their mashups consisted of only components listed in the manual and time taken in this session was 51 minutes on average ($\sigma = 17$).

The majority of mashups created by expert composers contained 1 to 2 MA components. The complexity of the abstraction models between 2 composer groups was approximately 3 components per a mashup. Differ from the novice composers, expert composers tended to use external Web services not available in the manual as a middle component linking 2 MA components. The average time the group used was 60 minutes ($\sigma = 34$). The reason they took longer might be that they acquired external libraries with custom settings. It takes time to understand how the libraries can be properly configured in MAIDL.

Post-questionnaire Result. A post-questionnaire was taken after the freestyle task finished. Question sets were divided into 3 parts, set C_2 and R_2 which are similar to C_1 and R_1, and a question set P concerning the composers preference towards MAIDL. In set C_2, novice composers gave high rating to the reusability of Web applications ($\bar{x} = 4.40, \sigma = 0.55$), simple Web service connections ($\bar{x} = 5.00, \sigma = 0.00$), source code generation, compilation and the creation of software package file ($\bar{x} = 4.80, \sigma = 0.45$). Expert composers gave high ratings to the ability to configure the mashup as a TeWS ($\bar{x} = 4.00, \sigma = 1.55$) and simple reuses of mobile applications as a component ($\bar{x} = 3.83, \sigma = 1.60$). Novice composers' ratings in set R_2 was not significantly high. In contrast, we found high ratings in expert composers' answers emphasizing: planning a workflow using MAIDL, the use of AR components, WA components, and WXTractor ($\bar{x} = 4.50, \sigma = 0.55$).

In set P, we found high novice composers' preference ratings towards: the ability of MAIDL in mashup compositions ($\bar{x} = 4.40, \sigma = 0.55$), the model layout ($\bar{x} = 4.00, \sigma = 0.71$), WXTractor ($\bar{x} = 4.00, \sigma = 1.00$), data filtration ($\bar{x} = 4.60, \sigma = 0.55$) and the publisher's parameter list ($\bar{x} = 4.20, \sigma = 0.84$). Novice composers tended to ignore the use of features involving data type and loop execution ($\bar{x} = 3.20, \sigma = 0.84$). They also did not quite agree that the component library covered all components they needed ($\bar{x} = 3.40, \sigma = 0.55$). They agreed that component's settings should be prepared as templates ($\bar{x} = 4.00, \sigma = 0.71$). The answers from expert composers show high rating in 2 items: the ability of MAIDL in mashup compositions, and the model layout ($\bar{x} = 4.50, \sigma = 0.55$).

Comparison of pre- and post-questionnaire. In novice composer group, a significant difference was not found in the majority of questions in set C and R. However, novice composers might found that the data filtrations are hard to configure as its ratings decreased after all tasks had been finished ($p = 0.03$). Expert composers might be able to reduce their efforts to write the components' code manually as a significant increase in rating are found in these items: the use of WA components ($p = 0.02$), and the use of AR components ($p = 0.01$).

Runtime and Usability. The runtime performance of the applications created by novice composers was normal when running MA components and WS components without complex queries. One application, which connected 2 Web applications, was slow and sometimes became unresponsive due to high network traffics. Most of the applications created in the expert composers freestyle task contained one WS component as a middle component and had a faster runtime. However, when an expected query result of Twitter API [16] was not found, the mashup process halted and the application needed to be restarted. In MAIDL, there is still no conditional statement for deciding processes or skip errors. Expert composers suggested that status and dialog configurations should be included.

Interpretation and Conclusion. Using MAIDL, novice composers agree that they are facilitated with simple configurations to reuse Web information. After the tutorial task, they were able to abstract the model of their mashup. WXTractor is able to assist their data extraction from Web applications. However, advanced features such as data type and loop executions were not preferred. The code generator and compilation tools could assist their lack of coding knowledge. They tended not to use external libraries, therefore, component libraries and templates should be added. The majority of novice composers also believe that MAIDL enables easy mashup compositions and opens opportunities for non-programmers and new comers to this area. More visualizations should be used and they also suggested that the look and feel of the mashup should be customizable.

Expert composers' expectation towards MAIDL's expressibility was low in the first place and increased after they finished the tasks. Composers in this group tended to use more time in the tasks because they were trying to adapt their skills (e.g. applying Web applications and Web services). The expressibility and customizability of MAIDL met their needs when applying it to external libraries. However, they suggested that some automatic functions (e.g. data adaptations, component templates) should be implemented or manually addable. In addition, expert composers found that the use of WXTractor to annotate tags and apply filters to them was easy. They suggested that it would be helpful for non-programmers if the same method was applied to XML or JSON messages. Just as novice composers did, expert composers suggested that the *MAIDL Designer* have more visual context-aware GUIs and more component templates. A high customizability in MAIDL was good for them to apply external libraries but might not be appropriate for non-programmers. If possible, data adaptations should be done automatically when the data are sent to the component.

Comparison between Novice and Expert Composers. T-test and ANOVA was applied to C_1, C_2 and R_1, R_2 data pairs of novice and expert composers which can be interpreted as follows:

1. Expectations of the expert and novice groups were met after the evaluation.
2. The novice group has higher expectations towards MAIDL than the experts.
3. The expert group rated expressibility of MAIDL in assisting mashup compositions higher than the novice group.
4. After the evaluation was finished, the expert group agreed that MAIDL gave higher rating towards expressibility in assisting mashup compositions.

The result can be interpret that MAIDL might not perform well when mashups are composed by novice composers because of its complexity. Expert composers are able to use MAIDL without confusion and may apply it to external libraries. However, both groups' expectations are met. Composers in both groups rated that the approach delivered 75% subjective rating for creating mashups.

4.2 Complex Mashup

To study cooperative mashup applications, we created 2 complex mashups using our approach. The mashups require interaction between 2 or more mobile devices. In this way, the mashup created in a TeWS output context is deployed on an Android device. On the other hand, iOS devices [17] are manually programmed to consume the deployed TeWS. We evaluate their runtime, connection performance and interaction usability on each device in the actual running settings.

Meeting Point: Cooperative Geolocation Mashup. In this mashup, geolocation of 2 devices are used as a data to find a list of restaurants located near the middle point between each device's GPS coordinates (via the *GourNavi* Web service [18]). Fig. 5 shows 2 mashup models and mashup applications, *Meeting Point Registration* and *Meeting Point Confirmation*, which communicate between devices via a TeWS in separated contexts.

Internal Runtime and Connection Performance. Application on the iOS side was presumably lightweight. Since this is a cooperative mashup for 2 devices with handshaking-like protocol, multiple connections are not considered as a performance factor. The overall performance of this mashup depends on the performance of *GourNavi* Web service.

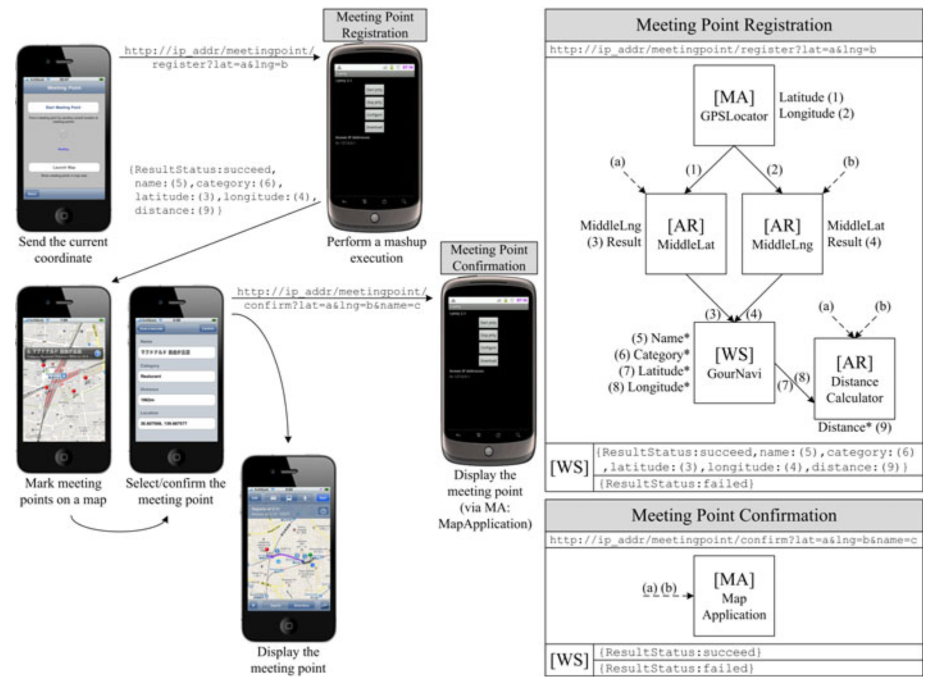
Meeting Point Registration
http://ip_addr/meetingpoint/register?lat=a&lng=b
Send the current coordinate
Perform a mashup execution
{ResultStatus:succeed, name:(5),category:(6), latitude:(3),longitude:(4), distance:(9)}
Meeting Point Confirmation
http://ip_addr/meetingpoint/confirm?lat=a&lng=b&name=c
Mark meeting points on a map
Select/confirm the meeting point
Display the meeting point (via MA: MapApplication)
Display the meeting point
Meeting Point Registration
http://ip_addr/meetingpoint/register?lat=a&lng=b
[MA] GPSLocator
Latitude (1) Longitude (2)
(a) (1) (2) (b)
MiddleLng (3) Result
[AR] MiddleLat
[AR] MiddleLng
MiddleLat Result (4)
(3) (4)
(a) (b)
(5) Name* (6) Category* (7) Latitude* (8) Longitude*
[WS] GourNavi
(7) (8)
[AR] Distance Calculator
Distance* (9)
[WS]
{ResultStatus:succeed,name:(5),category:(6),latitude:(3),longitude:(4),distance:(9)}
{ResultStatus:failed}
Meeting Point Confirmation
http://ip_addr/meetingpoint/confirm?lat=a&lng=b&name=c
(a),(b)
[MA] Map Application
[WS]
{ResultStatus:succeed}
{ResultStatus:failed}

Fig. 5. Mashup models and screenshots of *Meeting Point*

Usability and Interactions. If we assumed that 2 devices are connected using global IP addresses and are placed outdoors, the interactions between 2 devices might be interrupted by signal loss. Both sides must have timeout configuration and reconnection arrangement in the case of failure execution.

4.3 Book Shopping: Camera and Data Server Cooperative Mashup

This complex mashup application is designed for a shopping scenario in a book store for 2 or more users. One user holds an Android phone functioned as a server, the other users are holding iOS devices and are moving around the store searching for books. Mashup applications in a TeWS output context consist of *Book Shopping Add* and *Book Summary*. iOS device clients were installed with a manually written program to read a book's barcode and send the translated data to the *Book Shopping Add* TeWS deployed on the Android phone. The TeWS on the phone will search for the product title and price from *Amazon.com* WA component, translate the currency of the price from dollars to yen using *ExchangeRate* WS component, and finally add the title and price in Japanese yen into the phone database. Users with an iOS device may request for the added book title, book price and price summary using the *Book Summary* TeWS.

Internal Runtime and Connection Performance. In this setting, 2 mashups are installed on the server and handle multiple connections. In *Book Shopping Add*,

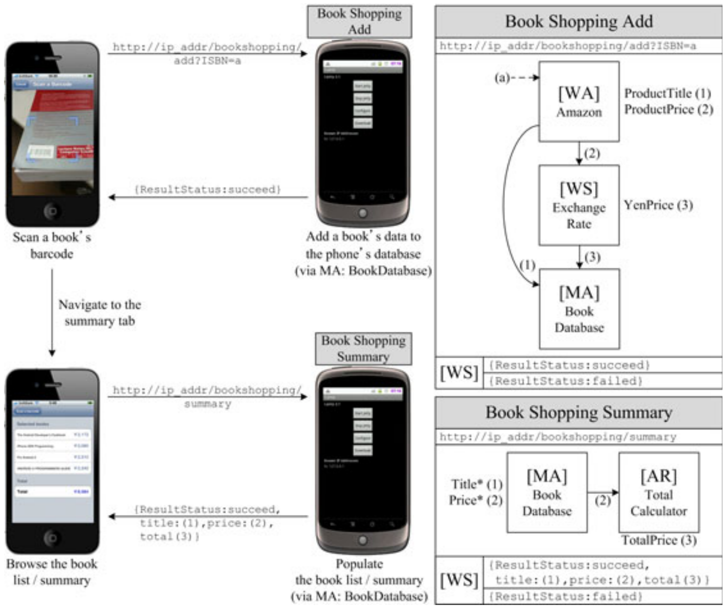
Book Shopping Add
http://ip_addr/bookshopping/add?ISBN=a
{ResultStatus:succeed}
Scan a book's barcode
Add a book's data to the phone's database (via MA: BookDatabase)
Navigate to the summary tab
Book Shopping Summary
http://ip_addr/bookshopping/summary
{ResultStatus:succeed, title:(1),price:(2), total(3)}
Browse the book list / summary
Populate the book list / summary (via MA: BookDatabase)
(a)
[WA] Amazon
ProductTitle (1)
ProductPrice (2)
[WS] Exchange Rate
YenPrice (3)
[MA] Book Database
[WS] {ResultStatus:succeed}
{ResultStatus:failed}
Title* (1)
Price* (2)
[MA] Book Database
[AR] Total Calculator
TotalPrice (3)
[WS] {ResultStatus:succeed, title:(1),price:(2),total(3)}
{ResultStatus:failed}

Fig. 6. Mashup models and screenshots of *Book Shopping*

the overall performance depends on the *Amazon.com* WA component and the *BookDatabase* MA component. We implemented 2 versions of *BookDatabase*, foreground and background ones. In *Book Shopping Summary*, when the MA component is accessed as a foreground process, server connections are queued by MAIDL process synchronization feature. The mashup runs more smoothly when the composited MA components are accessed as a background process. However, the WA component in *Book Shopping Add* runs in foreground process, yields lower speed and the server queues all incoming connections. Therefore, this type of mashup should not include foreground WA and MA components.

Usability and Interactions. We have tested the actual mashup using wireless network connection and found that the range between the server and client devices are important. This mashup might also be applied to other shopping scenarios which are best suited for Ad-hoc connections.

4.4 Security Performance

Since the application of WA components in MAIDL to invoke cross-domain connections and navigations of secured Web sites are possible. We believe that users should be warned before the mashup is installed on the device. For MA components, faulty MA components, which enter infinite loops, are automatically terminated by Android's system. However, the endless loop of sending data in a circle can be implemented. The use of MA component in mashup applications also altered some security issues. Android's OS allows softwares to be removed through the Intent messaging protocol. To solve these problems involving insecure WA and MA components, a MAIDL script might also work as a manifest file to look for information of the integrated component for security reasons.

The other important issue is when a mashup called a MA component which accessed personal information. In general, a manifest file of a mobile application is used to indicate what function to be used and what information to be accessed. Therefore, security measurement of each component in a mashup should be conducted and informed to the users before the mashup is installed and used.

5 Discussion

The results of the evaluation indicate that our method provides a good solution to mobile mashup compositions. The composers emphasized that the use of visual previews of data extractions and filtrations are preferred. However, the system do not allow mashups be simulated as a whole. Some MA components required an actual runtime environment. Moreover, we believe that mobile application providers or online application stores should contain the application's description concerning the messaging protocol for mashups.

In general, a mobile mashup is created for one task. Compared to mashups of Web applications, we found that mobile mashups have less complexity. They tend to be created as an integration between the phone's sensors and Web information. We found fewer mobile mashups that perform comparisons of multiple lookups to a variety of Web sites. More factors have to be considered in creating mashup in TeWS output context. To deliver smooth interactions between devices, the behavior of running process, network latency and usage scenario has to be observed. Since MAIDL script files contain information about each component and its runtime behavior, an alternative application of MAIDL for performance measurement can be considered.

MAIDL script files also contain a concrete description of the output messages sent via a TeWS. Applications on the client side might be generated or adapt themselves according to the description. A good example for the combination of a TeWS and a desktop Web application is to exchange multiple data from a mobile phone to automatically fill in personal information in an HTML form. The Web application first observes the applicable TeWS on the device and connects to it.

For the power consumption of the mashup, energy footprint of each component have to be observed, including the amount of data sent via a TeWS.

6 Conclusion and Future Work

In this research, we proposed a fast-paced mashup development using MAIDL. The mashup created by our approach can be designated for a single device, as a normal mobile application, or for multiple devices, as a TeWS. We proposed the method of *Mashup Output Context Transformation* and *Mashup Process Scheduling Algorithm.* The composition enables integration of annotated parts of Web pages, connections to Web services and the use of existent mobile applications.

In the evaluation with novice and expert composer groups, we found that the approach's simplicity in reuse of Web applications, Web services and mobile applications as mashup components serve the composers well. For the approach's expressibility, novice composers tend to use normal features and pre-defined templates, while expert composers require customizability when applying external libraries. Both composers groups gave higher preference rating after they used our tool. This implies that our method is optimal for low skilled composers.

In the complex mashup section, we demonstrated how a mashup works in a TeWS output context to deliver functionality exchange and cooperative application between devices. We were able to characterize the components that are appropriate to be integrated in a complex mashup in TeWS context. Our future work is to enable mobile mashup in the context of a Web application on a mobile device. To support higher interactivity to run on desktop computers, the process control and composition method might be different from the two contexts we have observed. The combination of multiple TeWS might be considered.

References

1. Appcelerator Titanium Mobile, `http://www.appcelerator.com/products/titanium-mobile-application-development/`
2. PhoneGap, `http://www.phonegap.com/`
3. Pietchmann, S., Tietz, V., Reimann, J., Liebing, C., Pohle, M.: Meißner, K.: A Metamodel for Context-Aware Component-Based Mashup Applications. In: Proceeding of the 12th International Conference on Information Integration and Web-Based Applications & Services. ACM, New York (2010)
4. Chaisatien, P., Tokuda, T.: A Description-based Approach to Mashup of Web Applications, Web Services and Mobile Phone Applications. In: Information Modelling and Knowledge Bases XXII, Frontiers in Artificial Intelligence and Applications, vol. 225, pp. 174–193. IOS Press, Amsterdam (2011)
5. Paternò, F., Santoro, C., Spano, L.D.: Maria: A universal, declarative, multiple abstraction level language for service-oriented applications in ubiquitous environments. In: Computer-Human Interaction, vol. 16 (2009)
6. Aijaz, F., Ali, S.Z., Chaudhary, M.A., Walke, B.: The Resource-Oriented Mobile Web Server for Long-Lived Services. In: 6th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (2010)
7. Google App Inventor for Android, `http://appinventor.googlelabs.com/`
8. 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)
9. Maleshkova, M., Pedrinaci, C., Domingue, J.: Semantic Annotation of Web APIs with SWEET. In: Proceedings of the 6th Workshop on Scripting and Development for the Semantic Web (2010)
10. 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)
11. Android Developers, `http://developer.android.com/index.html`

12. Android Intents, http://developer.android.com/guide/topics/intents/
13. i-jetty, http://code.google.com/p/i-jetty/
14. Amazon.com, http://www.amazon.com/
15. Exchange Rate API, http://www.exchangerate-api.com/
16. Twitter Search API, http://dev.twitter.com/doc/get/search
17. iOS Technology Overview, http://developer.apple.com/technologies/ios/
18. Gourmet Navigator API, http://api.gnavi.co.jp/api/manual.htm