Journal of Web Engineering, Vol. 12, No. 1&2 (2013) 093–130
© Rinton Press

A DESCRIPTION-BASED COMPOSITION METHOD FOR MOBILE AND TETHERED MASHUP APPLICATIONS

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

Received November 25, 2011
Revised October 27, 2012

This paper presents a description-based composition method for rapid development of mashup applications for mobile devices. We designed and evaluated a generator system which allows an automatic generation of the declarative descriptions to mobile mashups. The generator system is based on a mobile mashup composition language called Mobile Application Interface Description Language (MAIDL). The language focused the reuse of mobile applications, Web services and Web applications as mashup components and allows composers to lay out the connection of component data flow of the mashup application. In technical aspect, our generator provides an automated mechanism that can reduce the mashup execution time. In usability aspect, the evaluation shows that our composition method could assist novice composers in interpreting and planning mobile mashup applications. We found no significant difference in composition time and correctness between novice and expert composers. From the evaluation result, we are able to indicate the expressivity, the major patterns, and common composition mistakes in our mobile mashup composition method. The further requirements lead to a new composition approach for single and multiple devices mashups via the use of tethered mashup applications.

Keywords: Mobile application, description-based mashup, end user development

Communicated by: G.-J. Houben & H. Kitagawa

1 Introduction

Mobile technologies have become a highly important area in the current trends of telecommunication technology due to their rapid growth [1]. The advent of high-speed mobile communications and smart phone platforms also gives rise to a divergence between the consumption behaviour of the Web and the use of mobile Internet technology. Device-dependent functionalities such as location sensors, built-in cameras and always-on Internet connections have become essential components when developing mobile applications.

Typically, mobile applications do not complete the user's task in a single application execution. This makes some end-user requirements arise when further customizations are needed to be performed over the processed information. The majority of commercial applications available through online distributions (e.g., App Store [2], Android Market [3]) are built for a single purpose, and users often need to further combine them with other applications and/or Web-enabled resources for their specific queries. For example, suppose a user creates a mashup application that interprets place name from his/her GPS coordinates, and uses the name as a search keyword for local information queries (e.g., weather forecast, transportation route, etc.). To create such an application, it is impractical for non-programmers to: 1) lay out the conceptual plan of the mashup, 2) access and query the relevant Web-enabled resource and mobile device's functionalities, and 3) configure the query output display.

To address the end-user mashup requirements and the development problems, we proposed a method for mashup of device-dependent functionalities and Web-enabled resources using an XML description-based language called *Mobile Application Interface Description Language* (MAIDL). With the aim to study an end-user level mashup composition, we designed the method to improve correctness, and lower the amount of time spent on configuration process. Therefore, in the generation process, the composers are only required to: 1) decide the output context, 2) lay out the mashup components and their messaging pattern, and 3) assign the components' control parameters that will be executed for the generation of the final output.

Our essential result is an automatic generation from the declarative descriptions to mobile mashup applications. In the evaluation process for the method's usability and expressivity, we tested a generator system called *Mobile MashUp Generator System* (MUGS) that is based on the proposed method. We observed the impact of skill differences by conducting an evaluation with novice and expert composer groups. In the technical aspect, we also assessed the execution performance of the generated applications that applied our *Mashup Process Scheduling* mechanism in various mashup combinations and network latency settings. Our proposed composition method also covers the generation of mashups in the context of tethered applications (*Tethered Web Service* and *Tethered Web Application*). It allows remote mashup executions via HTTP connections for platform-independent cooperative mashups in multi-device usage scenarios. The integration of *Tethered Web Service* in HTML and JavaScript also extends the expressivity of the fundamental MAIDL to cover mashups with interactive display elements.

The organisation of this paper is as follows. We review the related research, state our motivation for designing the composition method, and clarify the contributions of our research in the next section. The overview of the composition method, MAIDL mashup components and generator system are explained in Section 3. The mashup generation process, execution-related mechanisms and mashup runtime are explained in Section 4. We give the details of evaluation covering usability, expressivity, execution performance and comparison of our method to other works in Section 5. We discuss current problems and possible improvements in Section 6, and finally give our conclusion in Section 7.

2 Background

2.1 Related Work

In our view, current composition methods and models often target the reuse of Web-enabled resources. Functionalities unique to mobile devices are not focused. Thus, end-users lack methods to create or customize mobile-specific mashup applications. Therefore, to define our research problems and motivations, we review shortcomings in these related fields:

- *Web Page Adaptation and Mobile Web Applications*
- *Web Information Extraction*
- *Mashup Composition Language and Multi-phone Application Frameworks*
- *End User Mashup Development*

The founding generation of mobile mashup researchers focused on Web page adaptations and transformations [4, 5, 6, 7, 8]. The proposed solutions are limited to the reuse of existing Web pages for the device fragmentation problems found in mobile Web platforms. Current trends in mashups are focusing the integration of multiple client-server communications (i.e. Web services) for mashup Web applications [9, 10]. These device-independent Web applications are generated and deployed rapidly on Web servers [11]. Using the user-agent identification, the servers are able to detect the type of a device and deliver a different display when the application is accessed [12]. Naturally, when a device-independent mashup Web application is generated and run on mobile devices, device-dependent functionalities may be absent. Despite the developments of mashup builders and models for data integrations, the current mashup approaches do not define an appropriate integration model for mobile mashup applications. The conventional approaches [13, 14, 15] reuse Web-enabled resources to form new Web applications but fail to support mobile devices' functionalities.

As information available on the Web has been extensively used, Web data extraction and integration has become one of the major fields to support end-user mashup developments. The applications of Web data extraction are found in many research areas of service compositions [9], news content extractions [16] and partial information extractions [17]. However, it is supposed that end-users are not familiar with the application of Web services. To gather information from various sources, they tend to browse and search through Web sites they are normally exposed to. Thus, the model and development process to extract and reuse information Web sites and Web applications at at the end-user level is not concretely defined.

In the composition of mashup applications, there are many description languages designed to support the domain-specific tasks. These approaches lack model, language, and optimal programming paradigms for mobile mashup composition at the end-user level. The classic BPEL [18] can be used to compose the integration of Web services, logic and data operations. EMML [19] enables the capability to perform Web clipping and other programmatic operations such as conditionals and loop statements. The major shortcomings of BPEL and EMML are their exposure to end-users in Web and mobile-specific area. The languages tend to focus more on system integrators who possess a deep comprehension of BPMN and Web service standards. The integration method found in MashArt [20] and CRUISe [14] can support the service-oriented mashup composition in both presentation layer and logical operations. However, mashup applications with device-dependent functionalities can only be created in low-level implementation using mobile native programming languages. In addition, many multi-phone Web-based application frameworks [21, 22, 23] employ Web programming languages. Web programmers can implement device-dependent functionalities since these frameworks promote simpler syntaxes for more rapid development compared to mobile native languages. These frameworks somewhat facilitate programmers, but do not provide non-programmers a practical solution for creating mobile mashup applications.

The mashup development environment for end-users can be divided into many aspects according to the expressivity and the targeted composers. The important elements, that are absent from these studies consist of: the reuse of Web applications, the coverage of device-dependent functionalities, and the empirical evaluation of the composition environment with real end-users. The method of continuous feedback found in DashMash [24] enables composers to immediately compose and correct the mashup composition. This method focused data extraction and visualizations during the composition process. A user-centric experiment has been conducted to show that novice and expert users perceive the ease of use of the tool in the same way. The evaluation of ServFace [25] is able to define concrete composition patterns and problems. This mashup platform performed service-based compositions at the presentation layer focusing the use of the WYSIWYG paradigm and reductions of UI level. Mehandijev et al. [26] use visual representations of service components and a template-based process to assist the end-user mashup composition. The reduced level of user control in this research shows the significant result in low time consumptions and few mistake actions. Paternò et al. proposed the MARIAE authoring environment [27] to enable Web service-based composition using a task model of the interactive application. Various mobile devices can also access the composed Web application. However, the method to create desktop-based mashup applications cannot be applied directly to mobile mashup compositions. The clarification of technical limitations and restrictions, that are found in a mobile-specific mashup composition system, is required.

2.2 Motivation

In the overall perspective, the aforementioned disciplines are not specifically designed to enable the end-user level of mobile mashup composition. We define the current problems and state our research motivations as follows.

- *Mobile Mashup Composition Language and Model*: Many studies in mashups have targeted data, application logic and user interfaces [28] for desktop-based Web applications. We define MAIDL as a description language in which mobile device-dependent functionalities (mobile applications) and Web-enabled resources (Web applications and services) can be integrated. The language is based on an imperative programming paradigm that composers are required to specify the description of how components receive inputs and send outputs in order to construct the final output when the executions are done. To lower the skill requirements, we applied a control-propagated data flow model and a shared tree execution sequence in the design of the language. The generated mashups use *Mashup Process Scheduling* mechanism to reduce the complexity of the execution sequence and shorten the mashup execution time.
- *End-User Web Integration*: Mobile mashup composers, in our hypothesis, are end-users who are able to browse and search through various Web sites for the information they need. However, the current methods to integrate Web information and mobile applications are not well-defined. Moreover, the user operation to reuse Web services (SOAP [29] or RESTful [30]) is not comparable to the operation to reuse normal Web sites. Therefore, to expose and study Web integration feature among end-users, we developed an integration tool, and a runtime engine for mobile device based on a tag-highlight annotation method [31] and used them in our evaluation. Building on our previous study [32], we further evaluate the method in our study for usage and problem patterns.
- *Evaluation of Mobile Specific Development Environment*: The end-user selection of integration components and the composition pattern in mobile mashups might be different from the Web-specific mashup compositions. To observe and effectively support the end-user behaviour, we evaluate and report the results of end-user studies. The evaluation involves the composers' preferences in their selection of components, common composition patterns, time consumption and composition qualities. We also compared

the results taken from novice composers who do not possess any programming skills with expert users who are familiar with programming languages.

- *Mashup Extensibility*: General mobile applications are usually designed for a single purpose and are not available for further integrations with other applications on the same device or with other devices. To provide more extensibility, we enable the generation of mashups in two output contexts: *Mobile Mashup Application* and *Tethered Mashup Application*. We term the first context as a mobile application composed from an integration of components and is used locally on the device. We term the second context as a Web service or a Web application composed from an integration of components that can be used locally (e.g., using the device browser to connect to its local host) or be accessed remotely from tethered (i.e. ad hoc) devices via HTTP connections. A mashup created as a Tethered Web Service (TeWS) can be consumed by the device itself or by remote client devices. The contrast point to WSDL [33] when composers use MAIDL for describing the functionality of TeWS is that: MAIDL uses both mobile-specific and Web-enabled mashup components, and all procedures to construct the final output result have to be defined. The mashup developed as a Tethered Web Application (TeWA) provides the complete remote application that enables platform-independent functionality exchange between devices. Moreover, we can derive a mashup application in a reactive paradigm from the application of TeWS to compose which can overcome the limitation found in the fundamental data flow composition (as discussed in Section 5.4).

In this research, the final mashup application package files are built for the Android platform [34] for single-device mashup. Tethered mashup applications (TeWS and TeWA) are built and deployed on the i-Jetty mobile Web server module [35] of the device that functioned as a host server. The targeted clients, in multiple-device usage scenarios, are devices that are capable of HTTP communications.

3 MAIDL Description-based Mashup Composition Method

3.1 Mashup Composition Method Overview

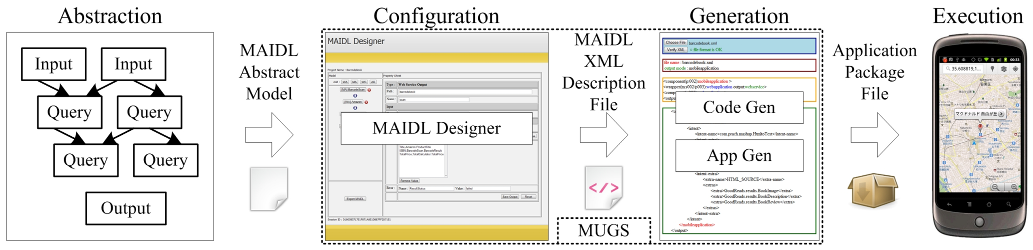


Fig. 1. Overview of MAIDL mashup composition method.

In the process to compose a mashup application using MAIDL description-based mashup composition method, composers are required to follow the steps as demonstrated in Fig. 1:

(i) In the abstraction phase, composers abstract the outline of their mashup. The abstraction method we used is that composers conceptualize the final output at first. They later lay out the relevant components and specify the components' data links that are used in the execution for the final mashup result.

(ii) In the configuration phase, composers create a composition model by selecting, laying out, and configuring all necessary components they have outlined in the abstraction phase. Three important configurations are: 1) the communication links that describe how components send/receive data, 2) the detailed control parameters of mashup components, which vary by category, and 3) the description of the final output component after the execution is done. We record the information of mashup components which is written by following the notation of *MAIDL XML Description File*. To lower the domain of error, composers use *MAIDL Designer* tool to assist their configuration task.

(iii) The model layout and configurations that are translated into a description file is sent to *Code Gen* module. In the generation phase, the module generates source code files which are then compiled by *App Gen* module for the final *Application Package File*.

(iv) In the execution phase, composers deploy the *Application Package File* on the actual device and start the execution. The generated application uses *Mashup Process Scheduling* mechanism to optimize the execution sequence and to reduce the execution time.

In Section 3.2 we give the category-specific component configurations and detail other shared features. In Section 3.3, we clarify the control/data flow model and its messaging pattern in our composition method. The systematic implementation of *Mobile Mashup Generator System* and a mashup example are described in Section 3.4.

3.2 Mashup Components

MAIDL mashup components can be divided into main 4 categories. According to their category, each component employs its unique runtime behavior in foreground or background processes. The runtime is also limited by its behavior that holds the use of the device's display and its inter-application protocol. The components also divided into three handling stages. Components in the *Input Handling Stage* can provide data to one or many components. The *Query Handling Stage* contains components that receive, process, and send data. The *Output Handling Stage* contains one component that represents the final output delivery or the display of the mashup execution result. The handling stages are explained with the data flow model in Section 3.3. We clarify the components' runtime behavior, their applicable coverage, their handling stage, and their shared configurations as follows.

3.2.1 Mobile application component

MAIDL's MA component employs Android's Intent [36] and Service[37] as its inter-application protocol.[a] Intent is a message passing protocol which uses asynchronous messages to enable communication between applications. Intent callers are able to pass their data as parameters to perform an execution at a target application. The target application may display data, or perform an execution and return the result to the caller. We observed and provide a summary of Intent-supported mobile applications in Table 1. The applications are selected according to their compatibility with our handling stage model. Therefore, they must be able to function as an input, output or query component. Applications without the information of Intent or Service specifications are not applicable. Some Intent-supported applications, that do not

[a]iOS platform use Apple URL Scheme [38] for inter-application communication.

Table 1. Summary of Intent-supported mobile applications

Source	Name	Stage	Description
Pre-Installed	Contact list	Input	Pick data from the contact list of the phone [39].
	Browser	Output	Browse the Web page of a given URL [40].
	E-mail	Output	Send a text to the receiver's e-mail address [39].
	Map	Output	Open a map application with the given Geo URI specified [40].
	Market	Output	Perform search in Android Market for the given keyword [41].
	Music player	Output	Launch a music player with the given playback file URI [42].
	Phone dialer	Output	Call to the given phone number [40].
	Search	Output	Perform Web search of the given keyword [40].
	SMS	Output	Send a SMS to the receiver's phone number [43].
	Street View	Output	Open Street View application to the given location [40].
Google-Release	Barcode Scanner	Input	Scan a barcode [64].
	Speech Input	Input	Translate user's speech into a text [44].
	Text to Speech	Output	Machine-read the given text [45].
	Radar	Output	Display a radar like view of the given position [46].
	Youtube	Output	Play a YouTube video from the given URL [47].
Android Market and Third Party Sites	Calculator	Input	Calculate and send the final result to other application [48].
	Calendar	Input	Pick a date from the phone's calendar application [49].
	Motion Gesture	Input	Detect a touch screen gesture [50].
	Motion to Intent	Input	Detect a shake or a jump action [51].
	Location Notifier	Input	Detect an enter/exit action of the registered location range [52].
	Pick Color	Input	Pick a color using a color picker dialog [53].
	Tag to Intent	Input	Detect the registered NFC tag when it is tapped [54].
	Translate Intent	Query	Translate the given text to a text of another language [55].
	Chartdroid	Output	Produce a chart representation from the given data [56].
	Evernote	Output	Send a text to Evernote application [57].
	Photoshop	Output	Open Photoshop photo editor of the given file URI [58].
	Twidroyd	Output	Send a tweet via Twidroyd application [59].
Custom Library	GPSLocator[b]	Input	Detect the current position's GPS coordinate.
	Input Box	Input	Display an input dialog.
	Database[b]	Query	Query or store the data to the phone's database.
	List Display	Output	Display a list of th given data.

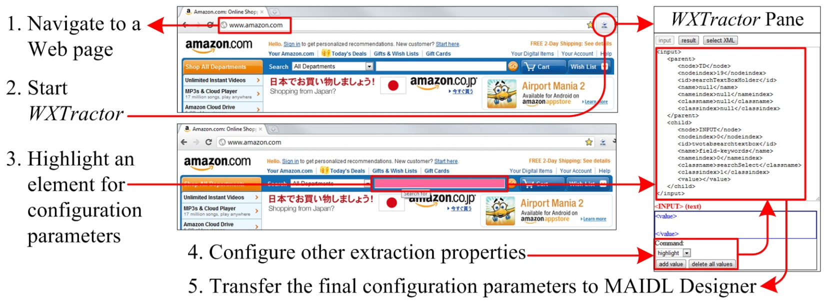
receive any data when it executed, are also not applicable. Some mobile device features, such as accelerometers and real-time GPS, do not support the Intent protocol.Therefore, we also provide some custom component libraries that are unavailable from external resources.

From our observation, most of the available mobile applications are executed as foreground processes and require user interactions to finish the execution before the process release the use of the device's display. This runtime behavior is suited for a single-device usage scenario. For multiple-device usage scenario, applications are executed at the Web server module which runs in background process. It is possible that the multiple-device mashup may be executed from remote clients and yield an interruption issue to the host device. Thus, the design process of multiple-device mashups must consider the issue carefully.

3.2.2 Web application component

A WA component is used when composers require an integration and an extraction of data from Web navigations (e.g., Web search, Web page access). WA components employ JavaScript injections to DOM [60] document objects. We implemented a Web browser's extension called *WXTractor* to work as a Web extraction tool in MAIDL. To generate the extraction parameters for WA components, composers are require to follow the steps shown in Fig. 2.

[b]Upon the configuration GPSLocator and Database can run in either foreground or background processes


1. Navigate to a Web page
2. Start *WXTractor*
3. Highlight an element for configuration parameters
4. Configure other extraction properties
5. Transfer the final configuration parameters to MAIDL Designer
WXTractor Pane
input
result
select XML
<INPUT> (text)
Command:
highlight
add value
delete all values

```
<input>
   <parent>
      <node>TD</node>
      <nodeindex>19</nodeindex>
      <id>searchTextBoxHolder</id>
      <name>null</name>
      <nameindex>null</nameindex>
      <classname>null</classname>
      <classindex>null</classindex>
   </parent>
   <child>
      <node>INPUT</node>
      <nodeindex>0</nodeindex>
      <id>twotabsearchtextbox</id>
      <name>field-keywords</name>
      <nameindex>0</nameindex>
      <classname>searchSelect</classname>
      <classindex>1</classindex>
      <value></value>
   </child>
</input>
```

```
<value>

</value>
```

Fig. 2. Steps to extract control tags for WA components.

(i) Navigate the external Web browser to the desired URL.
(ii) Start the *WXTractor* tool. The current page becomes highlightable and the *WXTractor* pane is displayed in separated window. Composers then highlight the desired element (HTML tag) of the Web page to extract the configuration parameters.
(iii) Configure 1) properties to extract specific attributes from the element (e.g., text content, image link, or the raw tag code), and 2) the filtration parameter to remove part of the data. The configurable properties are limited to the type of highlighted HTML tag. The simulation of the output is displayed at the bottom of the *WXTractor* pane.
(iv) Finally, transfer the final configuration parameters to *MAIDL Designer*. To use data in publisher's results, composers can substitute the variable names at the *Property Sheet*.

When the *WXTractor* tool is started, JavaScript code is injected into the Web page and allows mouse-over, and mouse-click interaction for element selections. When composers select an element of a Web page, the *WXTractor* pane will show the element's attributes (e.g., node, id, class name). Composers further specify the part of the elements to be extracted, such as the link URL or the text content of an anchor tag. The *Code Gen* module uses these attributes to generate JavaScript code for the mashup execution on the mobile device. Some Web applications require navigation along multiple pages for result extraction. In this case, composers are required to generate multiple sets of input extraction parameters. The WA component contains only one result extraction parameter set which represents the resulting page of the navigation. In the runtime environment, WA component can run in both foreground and background processes. For dynamic extraction requiring JavaScript injection in a browser screen, the component runs in a foreground process and holds the use of the device's display. For static data extraction, the component runs in a background process with the use of HTML parser-based extractions instead of JavaScript injections.

Fig. 3 describes the extraction algorithm used in WA components. In the implementation of the runtime environment of WA components, we designed the extraction algorithm based on a node order and the *getElementsByTagName* method. The method is applied to the selected tag and its parent tag. This method is used because of the absence of *getElements-ByClassName* method in some versions of WebKit-based mobile browsers and WebView API [61]. Additionally, since the page rendered in desktop and mobile Web browsers are different, we designed the algorithm to match neighbouring nodes in the node index range of ± 10. The algorithm also matches id, name and class name attributes when they are available.

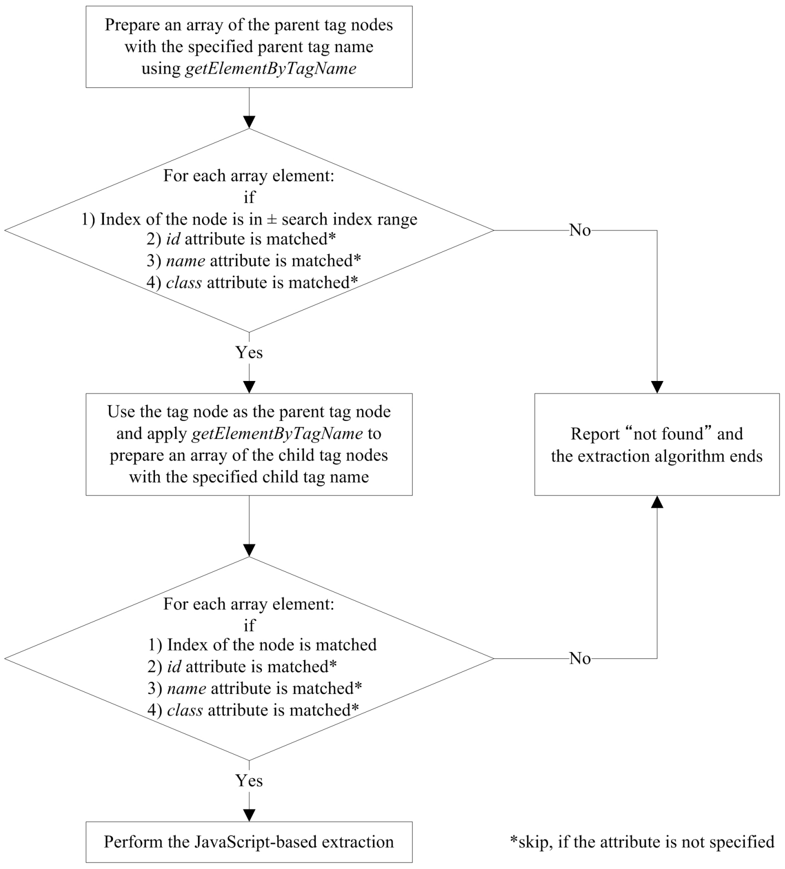

Prepare an array of the parent tag nodes with the specified parent tag name using *getElementByTagName*
For each array element: if 1) Index of the node is in ± search index range 2) *id* attribute is matched* 3) *name* attribute is matched* 4) *class* attribute is matched*
No
Yes
Use the tag node as the parent tag node and apply *getElementByTagName* to prepare an array of the child tag nodes with the specified child tag name
Report "not found" and the extraction algorithm ends
For each array element: if 1) Index of the node is matched 2) *id* attribute is matched* 3) *name* attribute is matched* 4) *class* attribute is matched*
No
Yes
Perform the JavaScript-based extraction
*skip, if the attribute is not specified

Fig. 3. Extraction algorithm used in Web application component.

The extraction algorithm first uses the tag name of the selected element's parent tag as the starting point. All tags with the same name are listed in an array using DOM's *getElementByTagName* method. Listed tags are then matched with other attributes (id, name, and/or class attribute of the tag) that are defined in the configuration parameters. If a match is found, the algorithm continues to the second phase by listing all child tags of the matched parent tag. It matches the child tag name with other attributes for the final element. The JavaScript code is executed to extract information of the final element if the match is found, and returns the final result from the component back to the mashup application.

WA components can be used in any handling stage as they can provide an input at the beginning of the mashup application. In normal cases, WA component functions as a query component that receives data from a component, performs a Web extraction, and sends the result back to the mashup application. It also can be used as the output component to display the final result page after the extraction is finished. The limitation is that a WA component can only be applied a Web application that has its DOM representation and can be executed with JavaScript injection. Therefore, Flash-only Web pages are not compatible.

3.2.3 Web service component

WS components in MAIDL use XPath [62] and JSONPath [63] to query data from REST Web services via network connections in a background process. We chose to use only RESTful services because of their simplicity in terms of resource identification (i.e. URI). SOAP services, on the other hand, would require composers to have specific knowledge in order to send a request. Composers specify a URI and a query expression to access a part of the whole data. Composers may integrate Web services they have discovered or use the templates we have provided. In general, Web service components work in query handling stage of a mashup such as a transformation from location data to the position's information.

3.2.4 Arithmetic component

Arithmetic component functions as a mathematical and logical operator that applies to one or more inputs (operands) received from a publisher component. The operation includes addition, subtraction, division, multiplication, summation, comparison, and GPS distance calculation. (given two pairs of GPS coordinates). The program code for this type of component is generated as Java threads and runs in a background process. We limited the AR component to perform basic calculations. For a conditional statement, AR component can use a ternary operator to assign the value to the variable when the condition is met. Therefore, composers cannot use the AR component as a conditional statement of a control flow (e.g. publish data to different components according to the configured condition). The description of AR components also does not cover loop statements (e.g., for and while).

3.2.5 Shared configuration parameters in MAIDL

There are two common configurations that are shared among all components: data links and filters. Firstly, all components in input and query handling stages have to define their publisher and/or subscriber inter-component data links. The mashup generator uses this information to declare the variable names at the code level. The *Mashup Process Scheduling* mechanism also decides the mashup execution sequence by optimizing the shared tree representation of the components and their links. Secondly, composers can configure data filtration parameters at the result variable configuration of mashup components. The data filters can be configured to remove, replace or add a set of characters to a text or number string. The filtration parameter is designed to solve data format conflicts in inter-component communication and visual representation.

3.3 Data Flow Model

A MAIDL mashup application is designed on the basis of a control-propagated data flow model (we use *flow-based model* to represent this terminology). In the abstraction phase, composers can abstract their mashup by freely laying out the components and specifying the components' data flow in a mashup application. The control flow and the execution sequence of the components are propagated according to the configured data flow. Fig. 4(a) demonstrates the abstract data model in a shared tree representation. The model represents mashup components as nested nodes and their inter-component data links in top-down level order. We divided components in a mashup application into three handling stages. The *Input Handling Stage* contains root nodes or a group of *Input Components* that can provide data to one or many components. The *Query Handling Stage* contains components that are nested

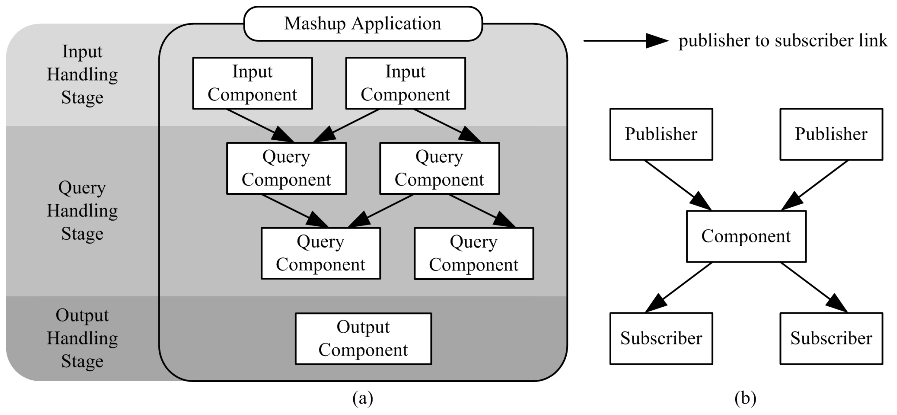


Fig. 4. Abstract data flow model. (a) shared tree model of a mashup application and (b) direction of data flow of a component with its publishers and subscribers.

as child nodes. It represents a group of Query Components that receive, process, and send data. The *Output Handling Stage* contains one component that represents the final *Output Component* or display of the mashup execution result. Table 2 summarizes the characteristics and gives component examples according to its handing stage in a mashup application.

Table 2. Characteristic and component examples divided by handling stage in mashup applications

Stage	Characteristic	Component example by category
Input	Receive an input from user interactions or from data sources.	Mobile application: Start a camera, GPS, voice input, etc. Web service: Make a request with a fixed query string. Web application: Start user's or automated Web navigations.
Query	Execute a query to related data sources, perform a calculation.	Mobile application: Run and finish an application for results. Web service: Prepare a query string and make a request. Web application: Extract data from the result page. Arithmetic component: Make a calculation of the data.
Output	Display output data, populate an output message.	Mobile application: Start an application (map, browser, etc.). Web service: Populate and return a JSON or an XML message. Web application: Populate and return an HTML document.

The data links in the model, shown in Fig. 4(b), represent the configured direction of data flow from one component to another. We call the component at the head of a link as a *publisher*, and the component at the tail of a link as a *subscriber*. According to the publisher-subscriber model, one component can be configured to send data to any other components, and to receive data from the components it subscribed to. Therefore, data flows of mashup results from publishers are visible only to their subscribers. The *Output Component*, however, is not directed to or from any components and is executed last. In the final sequence after other executions are finished along the configured data flow. Result data propagated from all components will be visible for use at this component.

The control flow and the execution sequence in MAIDL are based on the device single display basis: the use of display is paused until the execution is done. In unoptimised execution sequence, the components are executed one-by-one by its breath-first order. The tree model is divided into levels in which the topmost level will be executed first. Due to the top-down data flow, loop executions can not be defined in MAIDL. Moreover, MAIDL limits the definition to perform a retrieval of any data or an interruption in the middle of the execution of any component (e.g., event trigger, interrupt routine call, etc.). Some components such as Web services and arithmetic components run in background processes and do not hold the use of the device's display. To reduce execution time, the *Mashup Process Scheduling* mechanism merges and starts execution threads of foreground processes and background processes simultaneously in the same hierarchical level.

3.4 Mobile Mashup Generator System

In this section we describe how composers use our implemented mashup generator system called *MUGS* that is based on MAIDL and its composition model. The tool contains three separated parts: the *MAIDL Designer* tool, the *Code Gen* module, and the *App Gen* module. This section explains the *MAIDL Designer* tool and an example to demonstrate how our method is used in the composition of mobile mashup application.

3.4.1 MAIDL Designer tool

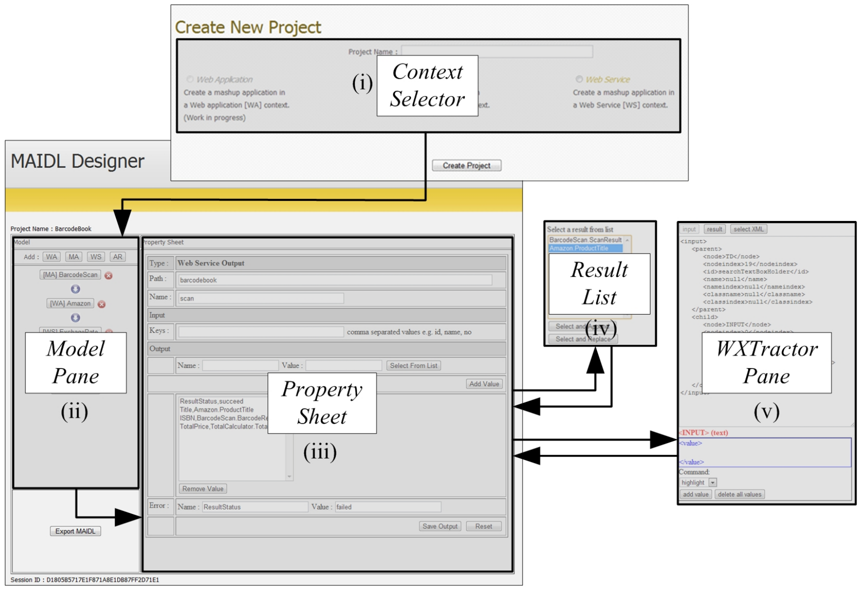


Fig. 5. MAIDL Designer tool and its related working components.

In the configuration phase, composers use the *MAIDL Designer* tool to compose their mobile mashups. It is a Web-based tool that enables composers to correctly choose the output context, lay out mashup components, and finally create MAIDL description files for the next generation phase. As demonstrated in Fig. 5, the tool consists of five working components which are used by these steps:

(i) First, composers select the mashup output context using *Context Selector* which appears at the beginning of the composition. The output context can be configured as a mobile application, a TeWS, or a TeWA.

(ii) Next, composers select and lay out all mashup components using the *Model Pane*. A component model, similar to the abstract model described in Section 3.3, is displayed in this pane. Composers may use component templates to produce the component configuration or use a blank template to manually add and configure a component.

(iii) Composers use the *Property Sheet* to show configuration details of the component and to complete the configuration of all the required parameters. The components require two configuration sets: 1) the publisher-subscriber links that describe how components send/receive data, and 2) the category-specific control parameters of each mashup component (as described in Section 3.2). The *Model Pane* will update its link representation between components after the configurations are completed.

(iv) During the configuration at *Property Sheet*, composers can view and select result variable names of the publisher from *Result List*, when the components require references to the data from the components they are subscribed to. To limit the confusion of variable visibility, these two rules that are applied from the abstract model are used: 1) result variable names from publishers are visible only to its subscribers, and 2) all result variables from all components are visible to the output component.

(v) We provide a Web extraction tool called *WXTractor* for composers to use when they require the integration of Web application inside their mashup application. During the composition of a WA component, composers can use the Web browser to navigate to a Web application and start the *WXTractor* tool. Composers then select the element of the actual Web page and simulate/configure the data filtration of the selected information. Finally, composers acquire the extraction parameters and apply them to the Web application component they are currently working on. The details of the *WXTractor* tool and its extraction algorithm are described in Section 3.2.2.

After the configuration finished, the composers export the working project as a *MAIDL XML Description File* for the generation process.

3.4.2 MAIDL Mashup Example

In this section, we give an example to explain how a mobile mashup application is created using our composition method. The example *Barcode Book Price*, as demonstrated in Fig. 6, is a mobile mashup application functioned as a barcode reader that displays a translated book price extracted from Amazon's Web application as its mashup result. It contains all four types of components available in MAIDL. This mashup consists these following components:

- *BarcodeReader*: This MA component is an input component that only provides data to its subscriber. It employs a barcode reader application [64] and is used to scan a barcode of a product for a barcode number.
- *Amazon*: This WA component extracts data from Amazon's Web application [65]. As a subscriber to *BarcodeReader*, it receives and uses the barcode number as a search keyword. After the keyword is submitted and the result page is loaded, the component extracts an image, a title, and a price of the product for the next execution sequence.
- *ExchangeRate*: This WS component connects to ExchangeRate's Web service [66] using a price from *Amazon* WA component as its Web service query parameter. The price in US dollars is then translated into Japanese yen.

- *TotalPrice*: This AR component calculates the summation of the translated price from the *ExchangeRate* WS component and a fixed product shipment cost.

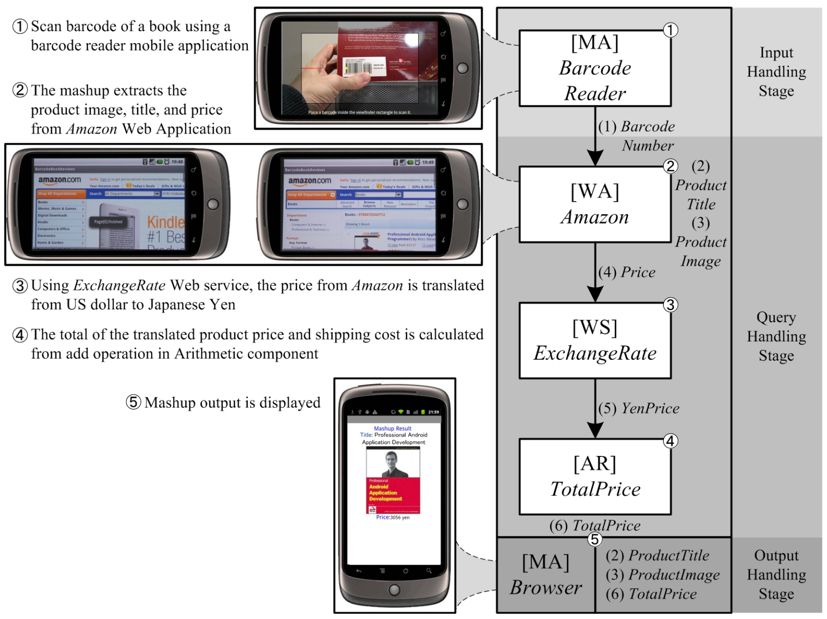

① Scan barcode of a book using a barcode reader mobile application
② The mashup extracts the product image, title, and price from *Amazon* Web Application
③ Using *ExchangeRate* Web service, the price from *Amazon* is translated from US dollar to Japanese Yen
④ The total of the translated product price and shipping cost is calculated from add operation in Arithmetic component
⑤ Mashup output is displayed
[MA] *Barcode Reader*
(1) *Barcode Number*
[WA] *Amazon*
(2) *Product Title*
(3) *Product Image*
(4) *Price*
[WS] *ExchangeRate*
(5) *YenPrice*
[AR] *TotalPrice*
(6) *TotalPrice*
[MA] *Browser*
(2) *ProductTitle*
(3) *ProductImage*
(6) *TotalPrice*
Input Handling Stage
Query Handling Stage
Output Handling Stage
Mashup Result
Title: Professional Android Application Development
Price:3056 yen

Fig. 6. MAIDL mobile mashup example *Barcode Book Price*

The execution sequence follows the order ① to ④ in the input and query handling stage. At the output handling stage in ⑤, as displayed at the bottom of the model in Fig. 6, we reuse the mobile device's Web browser to display the mashup result. The output component is able to gather any result variable from any component after the execution is finished. The result consists of the product image and title from the *Amazon* WA component, and the total price from the *TotalPrice* AR component. In the evaluation Section 5, we use these following steps in the tutorial task to provide the overall comprehension of MAIDL composition method.

(i) First, composers starts the *MAIDL Designer* tool and choose a mobile mashup application at *Context Selector*.
(ii) Composers respectively add a MA component, WA component, a WS component, and an AR component to the *Mode Pane* to match the abstract model provided in Fig. 6.
(iii) Composers bring up the *Property Sheet* by selecting the first MA component in the *Model Pane* and configure the MA component with an Android Intent name to connect to a barcode reader application. Two additional parameters have to be configured: 1) use barcode scan mode and 2) define the result parameter name as *BarcodeNumber*.
(iv) Composers select the next WA component and configure its publisher to the *BarcodeReader* MA component. For the data extraction parameter, composers navigate to the external Web browser and use the *WXTractor* tool to generate configuration parameters from Amazon.com. The extraction parameters are divided into *input* and *result* sets. The *input* parameter set can be generated at the index page of Amazon.com

that contains a search box. To use the publisher's barcode number as a search keyword, composers bring up the *Result List* pane to select and substitute the available publisher's variable name to the *input* parameter set. The *result* parameter set can be generated at the final result page where the browser shows the search result. Composers simulate the result by inputing the barcode at the index page and submit the data for the result page. Composers then retrieve the *result* parameters from this page and transfer them to the *Property Sheet*. In our example, composers are required to extract three *result* extraction parameter sets, and specify three components' result variable names of *ProductTitle*, *ProductImage*, and *ProductPrice*. To match the data format of the next WS component, composers are required to configure the filtration parameter of the *ProductPrice* to remove the leading dollar sign of the extracted data. We describe the details of the input/result parameter extraction steps in Section 3.2.2.

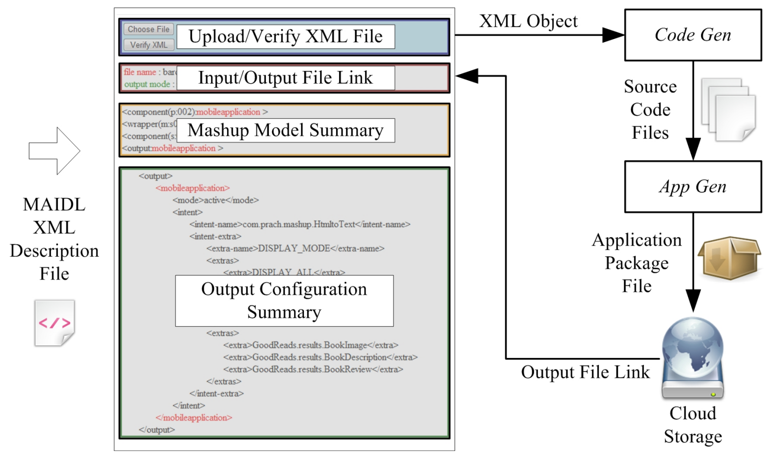
(v) Composers select the next WS component and configure its publisher to *Amazon* WA component. Composers configure the component to connect to the *ExchangeRate* Web service by using the extracted price from *Amazon* WA component, and the currency pair (i.e. USD and JPY) as its query parameters. As similar to the previous step, composers use the *Result List* pane to select and substitute the publisher's variable name to the query parameters. Finally, composers configure the result variable name as *YenPrice*.

(vi) Composers configure the next AR component as a subscriber to *ExchangeRate* WS component and calculate the total price using its publisher's *YenPrice* variable name and a fixed shipping cost. Finally, composers configure the result variable as *TotalPrice*.

(vii) At the final step, composers configure the output component that uses the system's Web browser to display data received from all components. The required parameter is the Intent name to connect to the device's browser and the HTML code to be displayed. Composers use *Result List* to select and substitute the variable name in the code.

4 Mashup Generation and Runtime

4.1 Mashup Generation



```
<component(p:002):mobileapplication >
<wrapper(m:s0
<component(s:
<output:mobileapplication >
```

```
<output>
    <mobileapplication>
        <mode>active</mode>
        <intent>
            <intent-name>com.prach.mashup.HtmltoText</intent-name>
            <intent-extra>
                <extra-name>DISPLAY_MODE</extra-name>
                <extras>
                    <extra>DISPLAY_ALL</extra>
                <extras>
                    <extra>GoodReads.results.BookImage</extra>
                    <extra>GoodReads.results.BookDescription</extra>
                    <extra>GoodReads.results.BookReview</extra>
                </extras>
            </intent-extra>
        </intent>
    </mobileapplication>
</output>
```

Fig. 7. Workflows of *Code Gen* and *App Gen* Module in the generation phase.

Mashup output contexts in MAIDL are divided into two groups according to their runtime behavior. We called the application created as a mobile application as *Active Context* which aimed for single user interaction and have its output component run as foreground (FG) processes. *Passive Context*, which includes TeWS and TeWA, are mashups created for multiple device use via network connections. According to the deployment to the Web server module, this context's output components runs as background (BG) processes. Each output contexts has a different inter-application protocol and interactions with users or client devices. In Section 4.2, we explain the architecture, how data is exchanged between components, and how the *Mashup Process Scheduling* mechanism is used. The overall runtime behavior of two output contexts is explained later in Section 4.3.

In the generation phase, after composers complete the configuration process and have their MAIDL XML description file exported. As demonstrated in Fig. 7, the code generation process begins after a MAIDL XML description file is uploaded to the *Code Gen* module. It verifies the XML syntax and sends the XML object to generate source code files. The generated code is then compiled by *App Gen* module. The source code and the output application package file are generated differently according to the configured output context as described in the followings.

- *Mobile Mashup Application (Active Context)*: *Code Gen* module first generates all the working files. The main source code file is programmed to send and receive information with other components via scheduled multiple threads. All GUI elements including the final output display are borrowed from the integrated components that are executed in sequence. The *App Gen* module compiles the prepared files for the final Android package file (APK [67]). The module then returns the final package file.
- *Tethered Mashup Application (Passive Context)*: In this context, *Code Gen* uses i-Jetty file and folder templates, similar to conventional Java Servlets. The difference from the mobile mashup application context is that the Servlet code is run in a background process. The background process cannot directly communicate with foreground process to start an execution of mobile applications which hold the use of the device display. We describe the solution to this restriction as the conventional inter-application protocol is altered via the use of *Switcher Mobile Application*. It is able to indirectly receive and send data between background and foreground processes in the tethered mashup application context. *App Gen* module then compiles the prepared files for the final Web application archive file (i-Jetty compatible WAR [68]). The mashup application in this context is deployed to the i-Jetty application on the device. In an advanced deployment of multiple TeWSs and TeWAs in the same path, composers can change the content of deployment descriptor file (web.xml) to enable access to the same context path.

It should be also noted that, during the execution process, all prerequisites applications required for each mashup component are installed on the device running the created mashup.

4.2 Switcher Mobile Application and Mashup Process Scheduling Mechanism

In *Passive Context*, the conventional inter-application protocol does not permit background processes to call and execute foreground processes. In other words, tethered mashup applications cannot call and retrieve data from mobile applications. Therefore, we altered the protocol to enable the mashup components in foreground process to communicate with ones

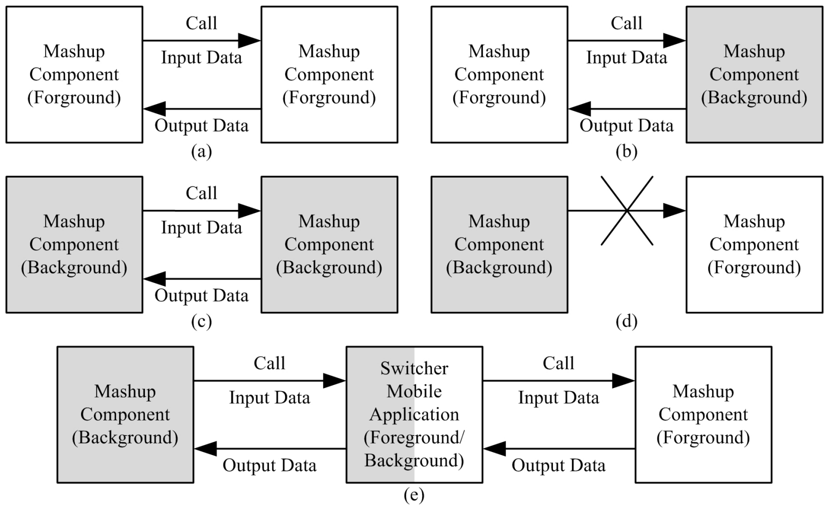

Mashup Component (Forground)
Call
Input Data
Output Data
Mashup Component (Forground)
(a)
Mashup Component (Forground)
Call
Input Data
Output Data
Mashup Component (Background)
(b)
Mashup Component (Background)
Call
Input Data
Output Data
Mashup Component (Background)
(c)
Mashup Component (Background)
Mashup Component (Forground)
(d)
Mashup Component (Background)
Call
Input Data
Output Data
Switcher Mobile Application (Foreground/ Background)
Call
Input Data
Output Data
Mashup Component (Forground)
(e)

Fig. 8. Conventional inter-application protocols are shown in (a) to (c) where (d) is not permitted. A SMA-based inter-application protocol (shown in(e)) enables an indirect communication between a call from a background process to a foreground process

Table 3. Details of the mashup mechanism in each output context group

Group	Context	Inter-component messaging protocol	Details
Active (FG)	MA	MA: Intent (FG) or Service (BG) WS: Service (BG) WA: Intent (FG) or Service (BG) AR: Native code	The mashup output context transformation mechanism translates the messaging protocol from Intent (FG) to Intent (SMA) when these components are configured and executed.
Passive (BG)	TeWS and TeWA	MA: Intent (SMA) or Service (BG) WS: Service (BG) WA: Intent (SMA) or Service (BG) AR: Native code	MA: Mobile applications are executed from a remote connection via TeWS or TeWA. WA: JavaScript-based extractions are executed from a remote connection via TeWS or TeWA

in background process via an application called *Switcher Mobile Application* (SMA). Fig. 8 demonstrates the difference between the conventional messaging protocol and the SMA-based protocol. For AR components, data filtrations, and publisher/subscriber configurations: the working code is generated as native program code and does not employ the SMA-based protocol. Table 3 shows the detail of inter-application protocol used in each output context. In *Active Context*, the mashup is generated as a mobile application. The inter-application protocol is based on the Intent and Service protocols of the Android platform. We found no limitation in the implementation of the runtime environment in this context.

After the output context is decided, mashups will be generated using MAIDL applied the *Mashup Process Scheduling* mechanism to optimize the execution sequence and to reduce the execution time. The optimization algorithm contains two phases. As demonstrated in Fig. 9, the first one is based on the reduction of shared tree levels that follow these rules:

(i) If there is no background component, the execution sequence follows the breadth-first order (① to ⑤) as demonstrated in Fig. 9(a).

(ii) If a foreground component is followed by a background component, the child component

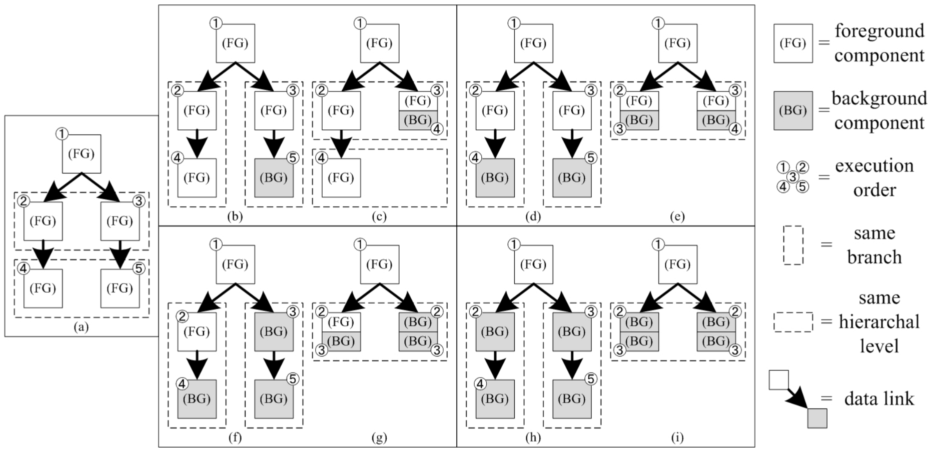


Fig. 9. Tree Level Reduction in the first phase of Mashup Process Scheduling Mechanism.

is executed immediately after the parent component is executed. Therefore, after the execution sequence Fig. 9(b) is optimized, the background component ④ is executed at the same time as the foreground component ④, as shown in Fig. 9(c).

(iii) In Fig. 9(d), if first foreground components in two branches is followed by a background component, the followed background component ③ of the first branch is executed at the same time as the foreground component ③ of the second branch, as shown in Fig. 9(e).

(iv) In Fig. 9(f) and (h), if one branch contains a foreground or a background component and is followed by a background component while the other contains two background components, the first component ② in the first branch will be executed at the same time as the first background process ② in the other branch. The next background component in both branches are executed at the same time after the first two components are executed, as shown in Fig. 9(g) and (i).

In the second phase, as shown in Fig. 10, the optimization algorithm then looks up the component in the same hierarchical level that has the same publisher or subscriber and tries to synchronize the execution time of a foreground component with other background components in the same hierarchical level. We summarized the rule as follows:

(i) If there is no background component in the same hierarchical level, the execution sequence follows the breadth-first order as demonstrated in Fig. 10(a) and (f).

(ii) If there is one foreground component while others are background components in the same hierarchical level, the foreground component is executed at the same time as other background components, as demonstrated in Fig. 10(b), (c), (g) and (h).

(iii) If there is more than one foreground component in the same hierarchical level, other background components are executed at the same time as the first foreground component, as demonstrated in Fig. 10(d), (e), (i) and (j).

On a technical level, our runtime environment uses Java Thread synchronization to catch a signal message that is sent by the component to inform the system when its execution has finished. We also measure and report the performance of the mechanism in Section 5.1.

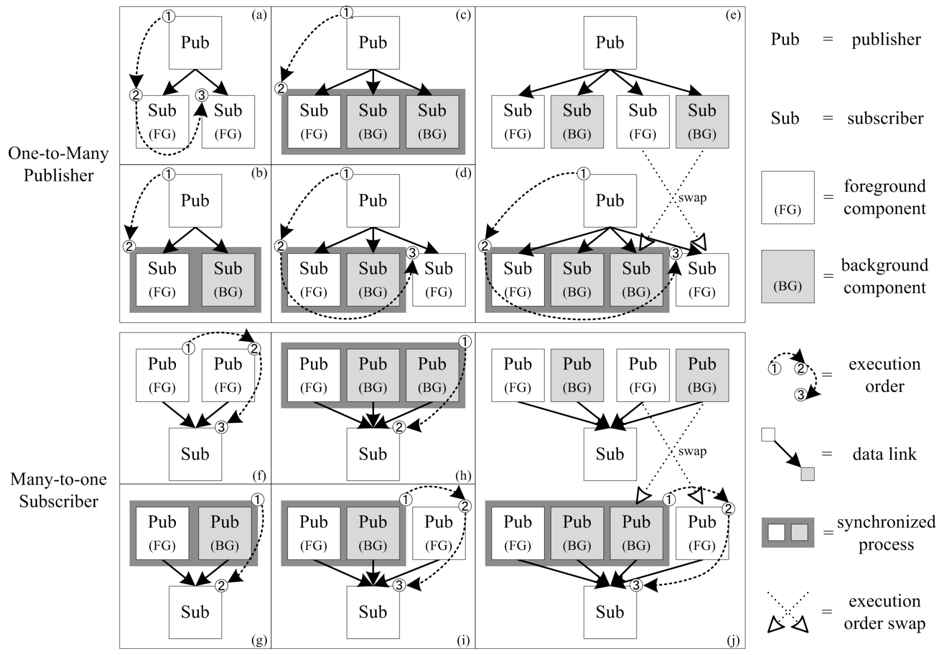


Fig. 10. Process Synchronization in the second phase of Mashup Process Scheduling Mechanism.

4.3 Mashup Runtime

4.3.1 Runtime in active context

The mashup in *Active Context* is configured as a mobile application when composers intended to design the mashup application for single-device use. During the mashup execution process, all interactions with the components are to be done by the user. According to what we demonstrated in an example in Section 3.4.2, components in this context work separately as mobile applications that hold the use of the device's display and release it once the execution of the component is done. Other components that work in background processes will not be displayed. Mashups in this context apply conventional Intent and Service messaging protocols and do not employ the SMA-based one. After the execution of components in input and query handling stages are finished, the output component gathers all information and uses it as its execution parameters with the configured Intent name. For example, the mashup can display the location's detail on a map application, show the search result in the form of an HTML document using the device's browser, or use the result as e-mail content to send to recipients.

4.3.2 Runtime in passive context

The mashup in passive context is configured as a TeWS or a TeWA when composers intend to broadcast information or send the mashup results via HTTP communications. They allow the exposure of functionalities from the host device (e.g., location-based and other information retrieved from device's sensors) to be used as a data source for mashup executions on the client devices. In addition, the HTTP communications via TeWS or TeWA mashups enable platform-independent functionality exchanges which solves the device platform constraints.

For TeWS, this output context is aimed for further consumptions by other client-server scripts for functionality exchanges (e.g., using XMLHttpRequest and JavaScript, or parsing JSON messages in native program code). During the execution process, the client-side scripts can send an input parameter via GET or POST methods to the TeWS and retrieve the mashup output in the form of a Web service message. Since the execution is requested via a remote connection, all components should be configured to run in background processes (e.g, Web service query, GPS location retrieval from built-in sensor). In contrast to the active context, the TeWS output context does not require a mobile application to work at the output stage. Instead, it is necessary to configure how results of the execution are populated as a Web service output message in JSON format or plain text.

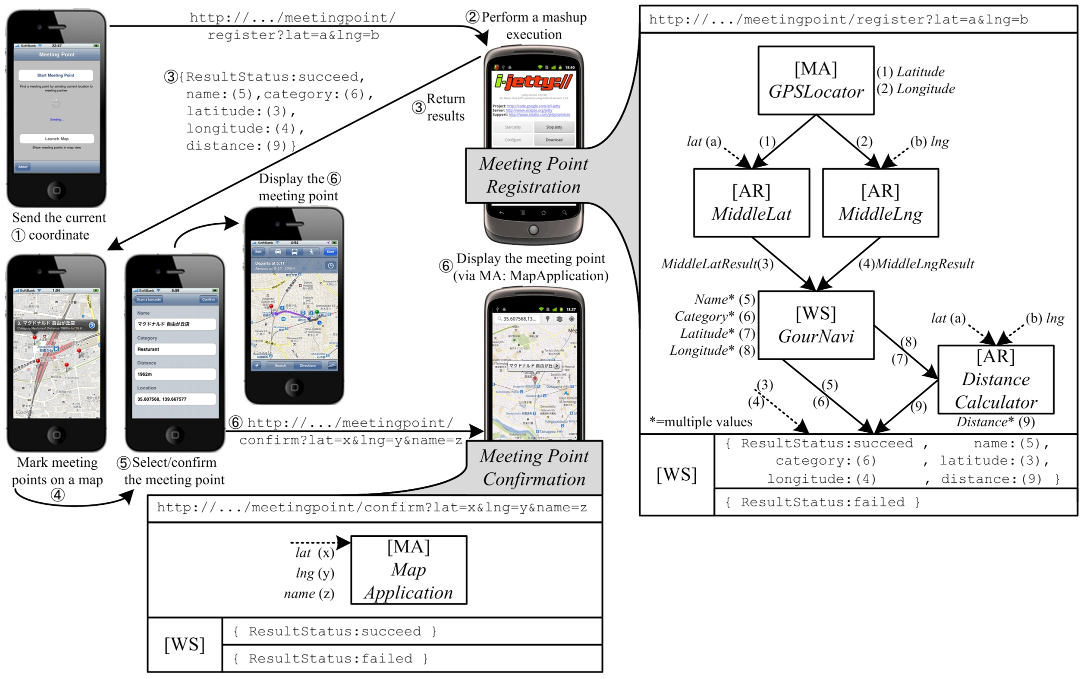

http://.../meetingpoint/register?lat=a&lng=b
② Perform a mashup execution
③ {ResultStatus:succeed, name:(5),category:(6), latitude:(3), longitude:(4), distance:(9)}
③ Return results
Send the current ① coordinate
Display the ⑥ meeting point
Meeting Point Registration
⑥ Display the meeting point (via MA: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

Fig. 11. Meeting Point Cooperative Mashup.

Fig. 11 demonstrates the multi-device cooperative mashup that involves the consumption of TeWS which requires interaction between two devices. The *Meeting Point* mashup demonstrates a restaurant search scenario using two mobile devices' location-based information. From number notations in Fig. 11, *Meeting Point* is operated by the following steps:

(i) The client device starts the client side application[c] which sends the device's current location coordinates to the host device.
(ii) The host device uses the received location information and uses its own current location as inputs for the mashup execution of *Meeting Point Registration*.
(iii) The result of the mashup execution, containing restaurants' information, is returned to the client device. The client application populates the pinned map GUI.

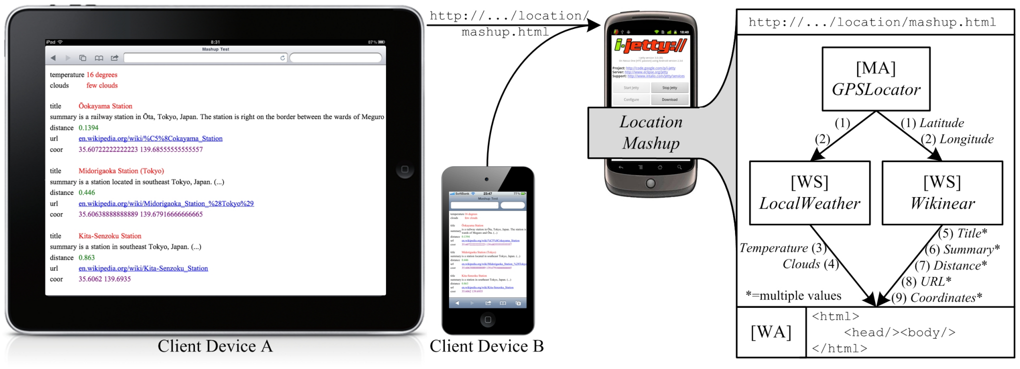
[c]The client application is a manually created iOS application which consumes the host's Web service messages.

(iv) The user navigates the map application to see information of each place.
(v) The user decides the point of interest and broadcasts it back to the host device.
(vi) The host device receives the broadcasted information. Both the client and host devices simultaneously switch to their map application which display a navigation to the place.

In this mashup, the application is separated into two mashups. *Meeting Point Registration* mashup performs the following actions: gathers the devices' location-based information, calculates the middle coordinates, submits the calculated coordinates to perform a query to a Web service [69] for nearby restaurant information, and sends the relevant results back to the client device. *Meeting Point Confirmation* is a mashup created in TeWS output context which allows the foreground process (map application) on the host device to be called via SMA-based protocol. The mashup receives the selected POI information, that the client device intended to broadcast back, and starts the device's map application to display the point.

The important role of MAIDL in this cooperative mashup is to enable the cross-platform device connections for cooperative purposes via a TeWS output. The output message can be consumed by client devices or can be used to create a rich interactive GUI using the platform's APIs instead of HTML-based GUIs. In a more complex mashup, more information from devices can be gathered for information-assisted decisions. The significant constraint of this type of mashup is that it requires a unique identification (e.g., mobile IP address) that allows the deployed mashups to be accessed directly from other devices. A proxy architecture might be required for non-IP communication between multiple devices.

For TeWA, this output context is aimed for a Web-based interactive mashup using HTML and JavaScript. In contrast to TeWS, a TeWA can be created as a terminal application without further consumption from client devices. An output message of TeWA is be populated as a hardcoded HTML document with results of the execution aligned in body or scripting part of the document. Fig. 12 demonstrates a mashup application designed to allow requests from client devices to access the location-based information and to perform mashup execution on the host device. The prepared application *Location Mashup* is created in TeWA output context with its execution procedures as the followings:


```
temperature 16 degrees
clouds      few clouds

title       Ōokayama Station
summary is a railway station in Ōta, Tokyo, Japan. The station is right on the border between the wards of Meguro
distance    0.1394
url         en.wikipedia.org/wiki/%C5%8Cokayama_Station
coor        35.60722222222223 139.68555555555557

title       Midorigaoka Station (Tokyo)
summary is a station located in southeast Tokyo, Japan. (...)
distance    0.446
url         en.wikipedia.org/wiki/Midorigaoka_Station_%28Tokyo%29
coor        35.60638888888889 139.67916666666665

title       Kita-Senzoku Station
summary is a station in southeast Tokyo, Japan. (...)
distance    0.863
url         en.wikipedia.org/wiki/Kita-Senzoku_Station
coor        35.6062 139.6935
```

Fig. 12. Location Functionality Exchange Mashup.

(i) The client device A and B use their default Web browser to access the Web application *Location Mashup* on the host device.
(ii) The host device uses its location-based information to execute the mashup that queries Web services for weather [70] information and Wikipedia articles of nearby places [71].
(iii) The host device populates the HTML documents using the results from the connected Web services. The document is then transferred and displayed on the client's browser.

In this functionality exchange mashup, the application *Location Mashup* is able to fulfill the device's (e.g., iPod Touch and iPad) absence of GPS sensors. It is practical when the client device can perform an ad-hoc connection to the host device located nearby and borrow some functionalities. This is advantageous as the variety and pace of mobile hardware updates sometimes leave devices without certain functionalities. For instance, Wifi-based tablets often lack a GPS sensor and a high-definition camera. Platform-dependent APIs also made those sensors inaccessible without native program code. In our composition method, the mashups created in a TeWA can work as a functionality portal for the client devices via HTTP. It is considerable to compose a remote application such as a remote Web camera probing system, or an application that securely exchanges personal information with other devices.

Mashups in passive context can integrate and execute foreground components using a SMA-based protocol. The *Mashup Context Difference Conflict* was found when these two exceptional combinations are integrated: 1) WA components that employ JavaScript-based extraction, and 2) MA components that run in foreground processes. In these cases, when the mashup is executed from a remote client device, the device may see no difference in the use of these components. However, it yields an interruption by holding use of the host device's display. For example, a client device connects to the host device and executes a barcode scanner application. The host device's interaction is interrupted to finish the requested task.

5 Evaluation

The evaluation of our research is divided into 4 subsections. We report the performance evaluation of the *Mashup Process Scheduling* mechanism, when applied to various mashup execution settings, in Section 5.1. We report and explain the empirical evaluation result concerning the composer usability and expressivity of MAIDL in Section 5.2. The comparison of our research to other mashup-related methods is discussed in Section 5.3. Finally, we summarize the expressive boundary and explain the extensibility of MAIDL in Section 5.4.

5.1 Performance Evaluation of Mashup Process Scheduling Mechanism

To evaluate the *Mashup Process Scheduling* mechanism, we selected and tested applicable composition samples given by composers in the freestyle task. The applicable examples are mashups consisting of parallel WS, AR and background MA components which can be represented as two shared tree branches model. To reduce the execution time at the input handling stage, we simplified the mashup examples to receive constant inputs. The simplified models are used as inputs for the *Code Gen/App Gen* modules with two configurations as shown in Fig. 13: (1) with and (2) without the scheduling mechanism applied. The mashup examples were modified to record timestamps when each component finished its execution.

The mashup examples were divided by their integrated components. Table 4 shows the average execution time difference of each examples. The execution time is measured from the

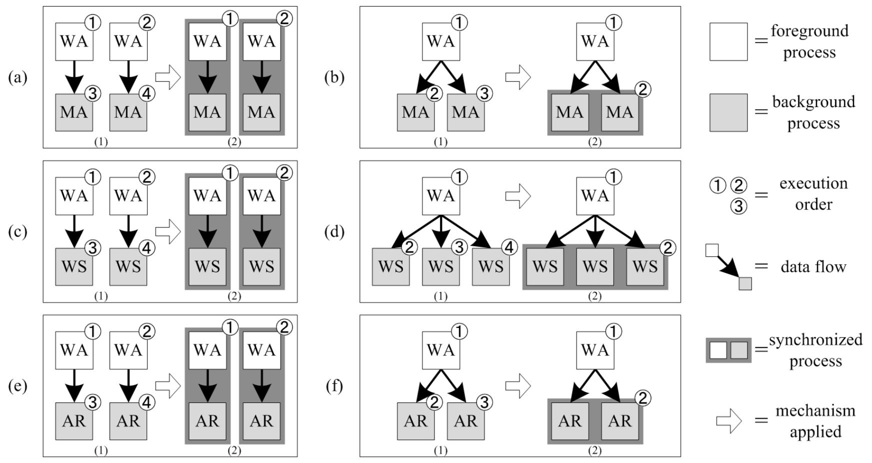


Fig. 13. The Model of Mashup Example used in Performance Evaluation. (a),(c) and (d) applied the first phase of the *Mashup Process Scheduling Mechanism*. (b), (d) and (f) applied the second phase of the mechanism. In (a) and (b), we use Amazon WA component, and database MA component. In (c) and (d), we use Amazon WA component and ExchangeRate WS component. In (e) and (f) we use Wikipedia WA component, and AR component that calculate GPS distance.

Table 4. Average Execution Time in Performance Evaluation (sec).

	$3G^m$	$WLAN_1{}^n$	$WLAN_2{}^o$
T_{a_2}-T_{a_1}	3.7437 (16.97%)	1.0474 (11.88%)	0.3001 (2.85%)
T_{b_2}-T_{b_1}	0.1425 (38.60%)	0.1134 (24.48%)	0.0176 (5.73%)
T_{c_2}-T_{c_1}	14.9168 (41.57%)	3.2902 (22.89%)	1.8671 (13.92%)
T_{d_2}-T_{d_1}	0.2211 (34.85%)	0.1677 (29.83%)	0.1596 (10.46%)
T_{e_2}-T_{e_1}	0.0231 (18.30%)	0.0066 (5.23%)	0.0052 (2.75%)
T_{f_2}-T_{f_1}	4.4953 (11.28%)	3.0980 (10.09%)	0.3283 (3.27%)
Average(%)	26.93%	17.40%	6.50%

moment the execution starts until all the components received their results. We observed the average from ten execution times of the mashup application deployed on a Nexus One mobile device [72] in three network connection environment settings: one $3G^m$ and two wireless LAN connections ($WLAN_1{}^n$ and $WLAN_2{}^o$). From time difference results in Table 4, the scheduling mechanism shows significant improvement in execution time when it is applied to mashups using two WA-WS connected components (Fig. 13(c) and (d)) that were executed in a 3G connection environment. For the same connection speed, we found about a 40% execution time reduction when MA subscribers in the same hierarchical level are scheduled to be executed at the same time (Fig. 13(b)). There is less improvement when the mechanism is applied with AR components since they do not employ any inter-application protocol. The overall percentage of time differences is decreased when the mechanism is applied to mashup executions in faster connectivity environments. In summary, the *Mashup Process Scheduling* mechanism is able to accelerate the mashup execution speed well when it is applied to the integration of Web components in slower network connectivity environments.

[m] 7.2 Mbps download/384kbps upload
[n] 54 Mbps download/upload (IEEE 802.11g) connected to 100 Mbps router
[o] 300 Mbps download/upload (IEEE 802.11n) connected to 1 Gbps router

5.2 Composer Usability and Expressivity of MAIDL

We observe four elements in our evaluation with novice and expert composers: *Questionnaire Result*, *Composition Pattern*, *Composition Quality*, and *Composition Time*. The main objective of this evaluation was to observe how composers react to two main questions. The first was about composers' expectations (i.e., "What did end-users expect before and after using MAIDL?"). The second was about the method's expressivity (i.e., "How well could MAIDL express end-users' compositions?"). Composers were asked to use *MUGS* to complete two composition tasks, tutorial and freestyle task. In the tutorial task, composers used the *MAIDL Designer* and followed tutorial steps to build a simplified version of the mashup example shown in Section 3.4.2. In this modified task, the final output component is simple text messages rather than a whole HTML document. We explained the objective of the task and demonstrated the output mashup before composers continued to the freestyle task. In the freestyle task, composers were asked to create a mashup and apply the configuration they have learned in the prior task. Composers were able to search for and discover Web services, Web applications or mobile APIs for their mashup before the composition began. We collected and observed the common composition patterns from the composers' mashups. This task also aimed to observe the expressivity of the description language towards the aforementioned questions of expressivity, composers' preferences regarding the composition method.

At a certain time frame of the tasks, composers were asked to fill out 5-point Likert scale evaluation questionnaires. The questionnaires were divided into 4 sets[d] as the following:

(i) S-set observes composers' mobile phone usage, knowledge about mobile and Web technologies, programming skills, and the concept of mashup compositions.
(ii) C-set observes composers' expectations to our description language in the level of which components and in what aspect composers would like to be assisted (and are assisted) in creating a mashup application.
(iii) R-set observes the expressivity of the description language in the level of how well composers can understand and apply the method to compose their own mashup.
(iv) P-set observes composers' preference regarding each component they have used.

The pre-questionnaire set consists of S-set and C_1-set. After the first task finished, composers took R_1-set before the freestyle task begins. To observe the outcome of the experiment, the C-set and R-set questionnaires were repeated after the freestyle task finished (this time recorded as C_2 and R_2, see footnote). P-set was additionally taken after the aforementioned composition tasks and questionnaires are finished. We give the evaluation result by comparing answers from pre- and post-questionnaire between two composer groups in Section 5.2.1. The common mashup composition patterns and study cases are provided in Section 5.2.2. Timestamp information, which was recorded during the tasks, is interpreted and discussed in the same section. We report the observation result of composition mistakes in Section 5.2.3.

5.2.1 Questionnaire result

The experiment was conducted with 40 composers which are divided into two groups. The novice group consists of 20 composers who are familiar with Web and mobile technologies. The expert group consists of 20 composers who are familiar with Web, mobile technologies

[d] C-set contains C_1 and C_2. R-set contains R_1 and R_2. The pair in each questionnaire set are identical.

Table 5. *S*-set Questionnaire Result

Question Item	Novice Composers	Expert Composers
HTML backgrounds	Composers know some HTML tags such as links (`<a>`), and images (`<img>`)	Composers understands HTML tags' attributes and their relationship to DOM and JavaScript
Web service backgrounds	None of them understand (0%)	The majority knows the concept, how to use and apply (75%)
Daily mobile phone usage hours	2-3 hours	1-2 hours
Top 3 mobile phone function usage	1)Internet, 2)Phone,and 3)E-mail	1)E-mail, 2)Internet, and 3)Phone
Mashup knowledge	None of them knows (0%)	Most of the subjects know mashups (95%)
Mashup component examples given	GPS, camera, voice recognizer, and Web pages	GPS, camera, and Web services
Composers' confidence in creating a mashup from the given examples	None of them were confident (0%)	More than half were confident (65%)
Programming skills	Some know HTML, or Excel	Most of them possess intermediate level of Java, C, or PHP

and are well experienced in one or more programming languages. Novice composers are expected to compose mashups with basic functions and a simple pattern. Expert composers are expected to able to use most of the features provided to compose complex mashups. They might also suggest corrections and propose further development to our system. From results of *S* question set, we summarise the characteristics of both composer groups in Table 5.

Results from question items in *S*-set show that novices know basic HTML and some are able to use Microsoft Excel expressions. Experts have high skills in Web programming languages such as HTML, JavaScript DOM or PHP. For mashup compositions, experts possess more skills to use Web services, create mashups and most of them possess intermediate level of one or more programming languages. Novices, however, possess low to no mashup composition-related skills. Interestingly, when asked to give a mobile mashup example, both composer groups similarly chose the same set of mashup components for their example composition.

The results from the next three question sets are compared based on: three items with highest/lowest ratings in both pre-/post-questionnaires, statistical differences between both groups, and statistical differences between pre-/post-questionnaires. *C*-set questionnaire consists of 12 question items concerning the composer's expectations of our mashup method. In Table 6, we compare the results from C_1-set (which was taken before the tutorial task) and C_2-set (which was taken after the freestyle task). The results from *C*-set indicated that:

- Before the tasks were taken, the expert and novice group shared high expectations about the generation of source code/application files, and automatic process scheduling mechanism in our method. In particular, the novice group expected that MAIDL could help them integrate Web services into their mashups. The expert group expected that MAIDL could assist them in creating mobile mashups. Both groups shared low expectations that MAIDL may not assist them in the planing procedure of mashup data flows, applying mathematical functions, and applying data filtrations.
- From Fig. 14(a), the overall ANOVA results indicated that after the tasks finished: the expectation rating is increased in the novice group, but was lowered in the expert group. It can be interpreted that novices do not expect or have more requirements. Experts,

Table 6. C-set Questionnaire Result

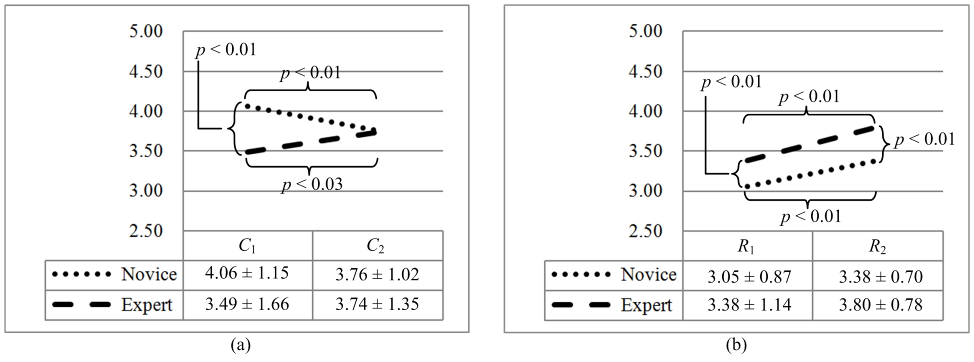
Question Item	Novice[e] ($\bar{x} \pm \sigma$)	Expert[e] ($\bar{x} \pm \sigma$)	T[f] N_{C_1}-E_{C_1}
1) Select and reuse parts of Web pages/Web applications	4.10±1.12 3.60±1.05	3.20±1.40 3.80±1.24	0.031
2) Connect to Web services and query the result messages	4.25±0.85 3.80±1.11	3.30±1.63 3.80±1.24	0.028
3) Select integrating mobile phone features and connect them to mashups	4.05±0.83 3.45±1.28	3.80±1.20 4.00±1.26	
4) Plan data flows of the mashup application	3.55±1.36 3.25±1.12	3.05±1.28 3.50±1.00	
5) Apply mathematical function and logical operation with the data	3.95±1.05 3.50±1.10	2.65±1.18 3.00±1.34	0.001
6) Filter the data (e.g., cut some unimportant texts/characters)	3.95±1.05 3.60±1.19	3.10±1.55 3.15±1.35	
7) Automatically synchronize running processes for better performance	4.20±0.95 4.10±0.72	4.00±0.79 3.85±0.75	
8) Automatically generate source code files	4.20±1.15 4.10±0.79	3.55±1.15 3.95±0.89	
9) Compile source code and generate an application package file	4.30±1.08 4.05±0.83	3.85±1.31 3.85±1.14	
10) Create the final mashup as a mobile application (to use on a device)	4.05±1.23 4.10±0.85	3.90±1.17 4.15±0.99	
11) Create the final mashup as a Web service or Web application (for collaborative applications/functionality exchanges to other devices)	4.10±1.12 3.90±0.85	3.75±1.07 4.20±1.11	
12) Reuse existent information or functionality without creating new ones (parts of Web pages, Web service messages, mobile applications)	4.05±1.05 3.65±0.88	3.75±1.12 3.60±1.14	

however, expected that MAIDL should perform better in the items explained below.

- After the tasks were performed, both groups differed in their expectations. Novice composers developed the following requirements: more automatic optimizations, an understandable source code file generation, and fewer steps to create and deploy the final mobile mashups. Expert composers would like MAIDL to provide more mobile-specific components and fewer steps to construct the mashup output. Both composer groups do not expect or require further assistance in the planning procedure of mashup data flows or the application of mathematical functions in MAIDL. Novice composers show low expectations about the integration of mobile-specific components while expert composers believe that MAIDL includes enough data filtrations to apply to their mashup.
- From T-test results of the C_1-set, we found overall statistical differences of both groups' expectations. When compared to the expert group, the novice group has higher expectations about the reuses of Web services/Web applications and the layout planning procedure of data flows. There was no statistical difference found in C_2-set.

R-set questionnaire consists of 14 question items concerning the composer's expressivity using our method. In Table 7, we compare the results of R_1-set, which was taken after the tutorial task, and R_2-set, which was taken after the freestyle task. The results indicated that:

[e]Data is shown in average($\bar{x}$)±standard deviation(σ) format. The upper value is C_1, the lower is C_2 result.
[f]We omitted the T-value that is larger than 0.05, therefore, the data in this column shows only the T-value calculated from novice(N)/expert(E) result pairs of the pre-questionnaire. There was no significant difference between any pre/post result pairs or novice/expert result pairs of the post-questionnaire.



	C_1	C_2
•••••• Novice	4.06 ± 1.15	3.76 ± 1.02
▬ ▬ Expert	3.49 ± 1.66	3.74 ± 1.35

	R_1	R_2
•••••• Novice	3.05 ± 0.87	3.38 ± 0.70
▬ ▬ Expert	3.38 ± 1.14	3.80 ± 0.78

Fig. 14. The Analysis of Variance Result of (a) *C*-set Questionnaire and (b) *R*-set Questionnaire.

- In R_1-set, both groups indicated their high expressivity to plan a workflow of a mashup and to apply data filtration parameters. In particular, novices believed that MAIDL could help them integrate Web services to their mashup. Experts indicated that the *WXTractor* tool could help reduce the amount of work and steps to get extraction parameters from Web applications. Both groups shared low ratings about the use of MAIDL without reference documents, their comprehension of publisher/subscriber patterns and their comprehension of the AR component's configuration.
- In Fig. 14(b), the overall ANOVA results indicated that both groups' expressivity ratings were increased but the novice group gave lower ratings than the expert group. We interpreted that both groups believed that the use of MAIDL can help them express their compositions, albeit at different levels. Experts can use almost all the features while novices may have difficulties in the inter-component and data-related features.
- After the tasks were taken, both groups shared the same opinion that they could plan a workflow of mashups and reuse Web applications. The novice group could understand the configurations of AR components while the expert group shown that they could use *Result List* to import the inter-component variables. Both groups still do not agree to use MAIDL without reference documents. The novice group found difficulties in the use of *Result List* and data filtrations. The expert group found difficulties in the configuration of MA components to use some specific features. They also indicate confusion in the process to get the extraction parameter sets for page navigations of WA components.
- T-test results of R_1 set indicate that the expert group gave higher expressivity ratings than the novice group in data-related items consisting the use of *WXTractor* tool and data filtrations. T-test results of the R_2 set further indicate that the expert group could integrate Web services/applications in MAIDL and import variables using *Result List*. We found the statistical difference between pre/post data sets in the novice group concerning: planning a mashup work flow, understanding publishers/subscribers, configuring Web applications in their mashup, and configuring the output component. Both groups shared the same statistical difference found in understanding publishers/subscribers. The results can be interpret that MAIDL provides a moderate expressivity level for novices in planning their mashups. Experts have a higher level of expressivity that they can apply more features to their mashups.

Table 7. *R*-set Questionnaire Result

Question Item	Novice[g] ($\bar{x} \pm \sigma$)	Expert[g] ($\bar{x} \pm \sigma$)	T[h] N_{R_1}-N_{R_2}/ E_{R_1}-E_{R_2}	T[h] N_{R_1}-E_{R_1}/ N_{R_2}-E_{R_2}	T[h] All_{R_1}-All_{R_2}
1) Plan a workflow of a mashup using MAIDL.	3.50±0.69	3.65±0.88	0.03/ ϕ		0.01
	3.95±0.60	4.10±0.72			
2) Create a mashup from MAIDL without reference documents.	2.60±0.88	2.55±1.10			
	2.50±0.95	2.75±1.25			
3) Apply the configuration of MA component to your mashups.	3.30±0.73	3.50±0.83			
	3.35±0.88	3.60±1.10			
4) Understand the publisher and the subscriber messaging pattern.	2.80±0.70	2.95±1.19	<0.01/<0.01		<0.01
	3.60±0.50	3.90±0.64			
5) Apply the configuration of WA component to your mashups.	3.15±0.93	3.65±0.88	0.04/ ϕ	ϕ / 0.01	0.01
	3.65±0.49	4.15±0.67			
6) Use *WXTractor* to extract parameters for your mashups.	3.15±1.14	3.95±1.23		0.04/ ϕ	
	3.40±0.60	3.75±0.97			
7) Apply the input/result extraction parameter sets of WA components.	3.15±1.04	3.25±0.97			
	3.25±0.72	3.55±0.89			
8) Apply the JavaScript-based navigation/extraction parameters.	2.80±1.01	3.50±1.05	0.04/ ϕ	0.04/ ϕ	0.04
	3.40±0.75	3.75±0.72			
9) Apply the *Result List* to import parameters from publishers.	3.25±0.85	3.25±0.91	ϕ /<0.01	ϕ /<0.01	
	2.95±1.00	4.10±0.64			
10) Apply filtration parameters to the data.	3.35±1.04	4.15±0.99		0.02/ 0.04	
	3.10±0.85	3.65±0.75			
11) Apply the configuration of WS component to your mashup.	3.35±0.88	3.50±1.15		ϕ / 0.02	
	3.40±0.88	4.00±0.65			
12) Apply the configuration of AR component to your mashup.	3.00±1.08	3.05±0.94	ϕ /< 0.01		< 0.01
	3.50±0.89	4.00±0.86			
13) Understand the relation of operations/results in AR component.	2.50±0.89	2.75±1.02	<0.01/<0.01		< 0.01
	3.80±0.77	3.90±0.79			
14) Understand how to configure the output component.	2.85±0.67	3.65±0.81	0.01/ ϕ		< 0.01
	3.55±0.83	4.00±0.79			

In the final questionnaire, *P*-set, we observed the composers' ratings of each particular feature in our mashup method. We compare the results between the two groups and found no statistical difference in any items except that the novice composers gave lower ratings for advanced execution features (such as loops and data type) than the expert composers. Table 8 shows the average ratings for 20 question items answered by two composer groups. We discussed the five question items with lowest and highest rating found in both groups.

- Both groups gave high ratings about the ability of *MAIDL Designer* to assist their compositions and the comprehensible method to reuse Web applications. Composers prefer that the mashup give them immediate feedback when it is being created. They indicated that integration components should be simplified to templates. The expert group agreed that our method and its ability to perform the integration of Web services was applicable.
- Both groups gave low preferences towards further applications of MAIDL. MAIDL fails to covers a sufficient integration component domain, including queries of multiple items from one data source. They disagreed that the reuse of Web services were more preferable than Web applications. The novice group narrowly disagreed with the statement that *WXTractor* is comprehensible.

[g]Data is shown in average($\bar{x}$)±standard deviation(σ) format. The upper part is R_1, the lower is R_2 result.
[h]The symbol ϕ represents the T-value that is larger than 0.05. ϕ/ϕ representations are omitted from the table.

Table 8. *P*-set Questionnaire Result

Question Item	Novice[h] ($\bar{x} \pm \sigma$)	Expert[i] ($\bar{x} \pm \sigma$)	T[j] N-E
1) *MAIDL Designer* can assist compositions of mashup applications.	4.10±0.72	4.50±0.51	
2) *Model Pane* and *Property Sheet* were applicable.	3.85±0.67	4.00±0.65	
3) The configuration of WS and WA output was comprehensible.	3.60±0.60	3.95±0.60	
4) The configuration of MA output/component was comprehensible.	3.65±0.75	3.65±0.81	
5) The configuration of WS component was comprehensible.	3.75±0.64	4.05±0.60	
6) The configuration of AR Component was comprehensible.	3.55±0.60	3.90±0.64	
7) The configuration of WA component was comprehensible.	3.75±0.55	4.05±0.69	
8) The *WXTractor* tool was comprehensible.	3.30±0.98	3.75±1.02	
9) The configuration of data filters was conprehensible.	3.50±0.69	3.85±0.93	
10) The *Result List* feature was comprehensible.	3.75±0.91	3.70±0.98	
11) Prepared component libraries covered all your requirements.	3.20±0.62	2.90±0.91	
12) `type`, `datatype` and `execution` parameters were comprehensible.	3.00±0.79	3.60±0.82	0.024
13) It was more preferable to have component templates.	3.80±0.77	4.05±0.89	
14) It was more preferable to create your own MA components.	3.50±0.95	3.30±1.08	
15) The method to reuse WAs are more preferable than WSs.	3.20±1.06	2.80±0.70	
16) Composers are able to crate a mashup using *MAIDL Designer.*	3.65±0.67	4.05±0.60	
17) It was more preferable to be able to immediately test the mashups.	3.95±0.76	4.15±0.67	
18) There were some limitations in *MAIDL Designer.*	3.70±0.66	3.85±0.93	
19) Any mashups are effectively composable using *MAIDL Designer.*	3.10±0.97	3.00±1.03	
20) Mashup composition subjective ratings.	76%	77%	

- Composers have given additional comments about our method. They requested more user-friendly GUI elements such as filter simulations, dialog composers and mobile GUI editors. Interactive elements such as real-time mobile device simulators, data flow simulators and video tutorials might deliver a better comprehension during the compositions.

5.2.2 Composition patterns and composition time

We observed the patterns in the mashups composed by the subjects in the freestyle task. Table 9 shows the common composition patterns found. We divided the patterns by their combinations of components in different handling stages (*I*=Input, *Q*=Query, and *O*=Output). The result gives an overview perspective that mobile mashups are commonly composed in *IQO* or *IQQO* patterns. These patterns are generally a linear flow that receives an input, uses the input data to perform a query with one or two executions and generates an output of the mashup. We found that 80% of novice composers use these patterns. Expert composers tend to have variety in composition patterns and they are able to compose compose complex mashups with more than six components integrated.

In order of most frequent use, Table 10 shows the top four types of components that are used in compositions, divided by their handling stage. The frequently used inputs are geolocation, text, camera and speech. Query components employ WA/WS components as the mashup data sources. The main difference found is that novice composers tend to use more WA components while expert composers use more WS components as their query component. The most common output components were list displays and WS or WA outputs. AR components are less used by novices. The most common mashup in this study is a GPS component

[i] Data is shown in average($\bar{x}$)±standard deviation(σ) format.
[j] The data in this column shows only the T-value calculated from novice/expert result pairs of the *P*-set. We omitted the T-value that is larger than 0.05 from the table.

Table 9. Composition Pattern

Composition Pattern[k]	Total	Novice	Expert
IQO	18	11	7
IQQO	8	5	3
IO	4	1	3
IQQQO	3	1	2
2IQO	1	1	0
2IO	1	1	0
I2QO	1	0	1
2IQO	1	0	1
I2Q2QO	1	0	1
I3Q2QO	1	0	1
IQQ2Q2QO	1	0	1

Table 10. Frequently Used Components divided by Handling Stage

Stage	Type	Number found
Input	MA:GPSLocator	16
	MA:Input Box	7
	MA:Barcode Scanner	5
	MA:Speech Input	4
Query	WA Component	26
	WS Component	24
	AR Component	4
	MA:Database	1
Output	MA:List Display	14
	WS or WA Output	10
	MA:Map	9
	MA:Browser	5

connected with one or two Web services (restaurant, hotel, address, nearby places, train stations) that displays the query output as a map application. Composers also use the same pattern to compose a price checker, a word translator/voiceover, and simple Web search. In more complex patterns, there was a mashup of a barcode scanner to check prices from multiple sources, a multiple language translator, and a mashup with calculations of GPS distance.

During the freestyle task, we measured the time composers use *MAIDL Designer* tool to configure their mashup. The amount of time does not include the abstraction phase where composers plan their mashup or confirm the availability of data sources by browsing Web pages. The summary of average total composition time and average per-component composition time during the task are shown in Table 11. We found no statistical difference in overall composition time between two groups (p=0.39). Composers used on average 6-7 minutes to configure a single component. We perform a closer observation with the compositions that shared the exact same pattern and component combination. We found no statistical difference in composition time of MA-WS-MA pattern (p=0.50). Composition time is decreased in the case of simple mashups. A longer composition time is required in the case where the libraries employ explicit configurations. However, we observed that composers tend to get familiar when they repeat the use of a component in the mashup that performs a query to

[k] Pattern can be represented in regular expression $[2-9]*I([2-9]*Q)*O$ where a number in front of I or Q represent the number of components that runs in parallel (we use nine components maximum as an example). Repeated characters (such as QQ or *2Q3Q*) represent sequential executions in hierarchical orders

Table 11. Average Composition Time in Freestyle Task (minutes)

	Novice[l]	Expert[l]
Overall	23.00±9.64	22.00±9.72
MA-WS-MA	22.80±5.68	22.60±5.24
Per-component	6.82±3.47	6.56±3.26

different resources with the same pattern (e.g., international Amazon Web pages, multiple language Google translation). We conclude that MAIDL provides instructional scaffolding, so composition time is lowered and the skill gap between novices and experts is narrowed. Combining these results with the questionnaire results, it can be considered that composers are guided when a top-down model (i.e. flow-based model) was used. Therefore, they could clearly identify the start points of their compositions and progress further in steps. Further development of our mashup system could exploit the patterns found in our evaluation.

5.2.3 Composition quality

Table 12. Problematic Mistake Pattern found in Novice and Expert Composer Groups

	Novice	Expert
MA	1) Use wrong Intent name. 2) Miss some important parameters.	1) Unable to use some specific feature of a mobile application (e.g., map pin captions).
WA	1) Select wrong elements of the Web page. 2) Generate and use wrong extraction parameter sets (input/result sets). 3) Select the extraction parameters.	1) Configure multiple input extraction parameter sets for multi-page navigations.
WS	1) Map publisher's variables as query parameters. 2) Query of single and multiple data from the Web service output (JSON and XML).	1) Query of multiple data from the Web service output (JSON and XML).
AR	1) Number of operands are not matched with the selected function. 2) Input and output mapping in ternary operations was not correct.	
Others	Data filters: data format is not compatible with the next component's requirement.	Multiple value data: cannot use the right index of a list/array

The composition quality (i.e. correctness percentage) is measured by the correct configurations divided by all configurations. We found 78% correctness in the mashup created by novice composers while expert composers potentially have higher correctness at 87%. There was no statistical difference between correctness percentage in two groups. In Table 12, we report details of mistakes. The two groups had different patterns of mistakes. Critical mistakes found among novice composers surrounded the use of correct component parameters. The WA components were used mostly amongst novices but they sometimes selected the wrong extraction elements and configured the parameter sets for the wrong Web page. The other interesting aspect was that mashups composed by novices used only one input and one output. In contrast, expert composers frequently made mistakes when trying to manipulate data from multiple sources. Both groups made mistakes when trying to integrate Web services into their mashups. In Section 6.1, we provide detailed suggestions for further improvement from composers' opinions and lessons learned from our evaluation.

[l] Data is shown in average($\bar{x}$)±standard deviation(σ) format.

5.3 Comparison to Other Mashup-Related Methods

Table 13. Comparison to Other Mashup Approaches

	MAIDL	MARIA [73]	YahooPipes [13]	AppInventor [74]	PhoneGap [21]
Data Flow Formalism	Flow-based	Event-based	Flow-based	Programming	Programming
Composition Method	Pub/Sub components	GUI/Services migrations	Data pipe connections	Program blocks	Source code
Input[p]	MA,WS,WA	WS, WA	WS_{pre}	WS_{pre},MA_{pre}	X (Coding)
Output	MA,WS,WA	WA	WA	MA	MA
Deployment	Mobile device/server	Proxy server	Proxy server	Mobile device	Mobile device
Simulation	Separated	Separated	Built-in	Built-in	Built-in
Composition Level[p]					
GUI	△ (Borrowed)	◯	◯	◯	◯
Logic	◯	◯	◯	◯	◯
Data	◯	◯	◯	◯	◯
Web Integration Tool[p]	*WXTractor*	Mashup Editor[31]	X	X	X
Extraction Algorithm[p]	2-Level	1-Level	X	X	X
Data Filter[p]	◯	X	◯	△ (Coding)	△ (Coding)
Programming Skill Requirement	X Not required	△ Partially required	X Not required	X Not required	◯ Required
Target Users	Novice	Intermediate	Novice	Novice	Expert
Reusable[p]	◯	◯	◯	X	◯ (Code)
Process Scheduling Mechanism[p]	◯	X	X	X	X

Many mashup methods and development systems have been introduced to fulfil end users' requirements to compose their mashups. The method in this research can deliver fast-paced development and lower the skill gap between novice and expert groups. Table 13 shows a comparison of our approach with other approaches that enable end-user development of mobile mashup applications and are otherwise similar to ours. We also added PhoneGap [21] for the comparison between mashup methods and a programming framework.

When compared to other methods, our approach can deliver the reuses and creations of mobile applications, Web services and Web applications. In general, other device-independent approaches require a proxy server architecture and they are only capable of the integration of Web services and/or Web applications. They also tend to limit the use of components as pre-defined sets Web services while composers can freely apply more sets in our approach. In contrast to our flow-based approach, an XML language MARIA can apply the integration of WS and WA components with customizable GUI and event trigger mechanism. The major drawback of our method is that the custom GUI cannot be newly created, but instead borrowed from the integrated components. The other disadvantage is that simulations of the mashups are separated from the generation process. Simulations have to be conducted after everything is prepared, so composers cannot see immediate feedback and errors. Moreover, the mechanism to shorten mashup execution time is not found in other approaches.

In an end-user aspect, we provide the *WXTractor* tool for composers to integrate Web information to their mobile mashups. The extraction algorithm in our approach is specifically

[p]Subscription $_{\text{pre}}$ denotes pre-defined components. ◯=available, △=partially available, and X=not available

designed for mobile devices which might parse the Web pages differently from what is found in the proxy server approaches. Due to the capability of creating mashups as TeWS and TeWA, the mashups are extensible and consumable by other clients in different device platforms. In other approaches the mashup is created as an application and cannot be reused. Finally, our flow-based method requires less programming skill compared to the others (or even none).

5.4 Expressivity and Extensibility of MAIDL

5.4.1 Expressive Boundary

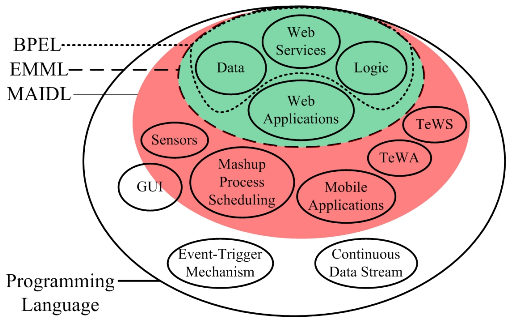


Fig. 15. Expressive boundary of MAIDL compared to other Web mashup methods.

Fig. 15 demonstrates the expressivity coverage of MAIDL and conventional Web mashup methods compared to programming languages. MAIDL covers the composition of mobile applications and device's sensors as an extension to a conventional Web mashup. The compositions of tethered mashup applications including the process scheduling mechanism are presented in MAIDL. The limitation of MAIDL is that it does not cover the use of continuous data streams or an event-trigger mechanism in composition of mashups with high interactivity. For instance, a mashup of a map displaying person's running route online cannot be composed using MAIDL due to the incompatibility of continuous data stream usage in flow-based composition. The programming language provides highest expressivity while it provides low composition usability when compared to other mashup methods. MAIDL also has a limitation that GUI elements cannot be newly created but are borrowed from integrated components instead. Composers sometimes need to customize the interactive elements and dialog controls of the mashup. We limited this customization because when exposing these customizable features to composers, the learning curve may be raised too high for beginners.

5.4.2 Extensibility of MAIDL

MAIDL has its extensibility to cover the composition of GUI, event-trigger mechanism and continuous data stream. In an alteration to the fundamental MAIDL, we considered an event-based composition method. The main difference from flow-based composition's abstract model, as described in Section 3.3, is that the Input-Query-Output handling stages and propagated control flow are absent. This is because all the components periodically update their data which leads to event-trigger operation to the other components. Therefore, the components are virtually integrated inside the output display as demonstrated in Fig. 16(b). There are several pre-composition steps which composers are required to carry out:

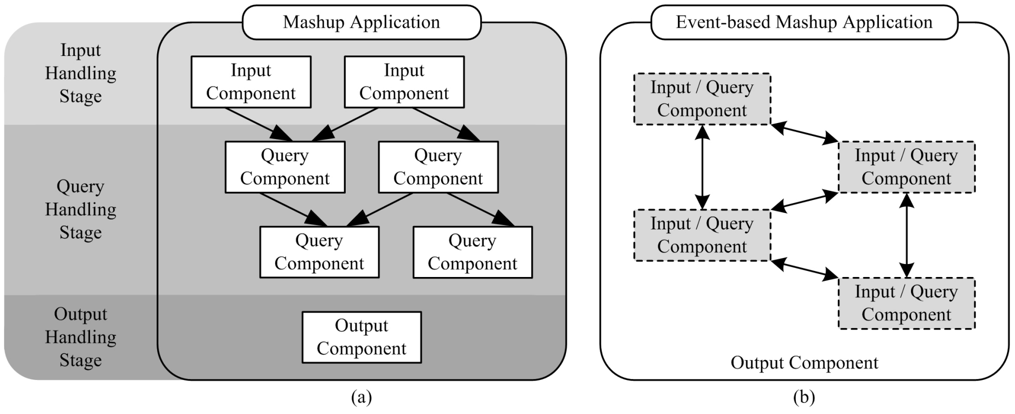


Fig. 16. The comparison between flow-based and event-based mashups. (a) In flow-based mashups, the execution order follows the top-down shared three models of data and control flow. The output is generated and displayed in the last order. (b) In event-based mashups, the contained components communicate with each other in back-ground processes by handling the input/output events. The output display is updated in real-time executions of the integrated components.

(i) MA and WS components are prepared as TeWSs using MAIDL.
(ii) The TeWS converted components have to be installed on the device before the mashup execution begins. These TeWSs are accessed via the device's loop back IP address (e.g., `http://127.0.0.1/gps`) and the configured path of each TeWS.

Composers are then able to integrate the TeWS inside an HTML document using conventional AJAX technique. This enables the rapid development of Web programming language and the extension to cover the access to mobile device functionalities. However, in an event-based composition method, the process scheduling and the propagated control flow have to be implemented manually by composers. Therefore the method requires the knowledge of HTML and JavaScript which are usually not understood by novice composers. In addition, to avoid *Mashup Context Difference Conflict*, uses of active components are not recommended for event-based mashup. However, composers are able to use these components by first converting them to TeWSs that run as background processes.

6 Discussion

We discuss problems and limitations found in our research and next we purpose solutions for further improvement of the composition method as the followings:

6.1 Composer Evaluation

The use of MAIDL for mobile mashups might be optimal for simple composition patterns, as we found no significant difference in composition time between two composer groups as stated in Section 5.2.2. In the composer evaluation, the questionnaire results told us that the composers in two groups could perceive and use different features available in our method. Further investigation from comments about integrations of Web services in MAIDL found differences between the two composer groups in the term of structured data model comprehension. Most of the novice composers gave up the integration of Web services and use Web applications instead. They stated that they are not familiar with Web services and they could not see the concrete output when the query operation is performed. Expert composers tend to use external tools which can help them correctly visualise XML and JSON data formats. In interviews about how to improve the mashup composition, composers from both groups indicated that a predefined set of Web services, filter simulation and some example description would help them realise the importance of this integration component. Moreover, an interactive data model and real-time visualised work flow might gave better understanding of the description language and give better composition quality.

6.2 Mashup Components and Implementation Issues

Mashup applications in general are combinations of working components. Therefore, they rely on the availability of the integrable components. It can be considered that Web-enabled data resources and downloadable mobile applications are created for reuse purposes. However, it is possible that some legal issues can occur when those components are reused or republished. Therefore, mashups with those components should be created for personal use only. Most mobile applications, on the other hand, do not support the messaging protocol for integration with other components. Moreover, complex algorithms (such as an image pattern recognition) are obviously not composable using mashup composition methods. To overcome these availability issues, such components can be created as external Web services which allow devices to send data for executions. The development of *Scripting Component*, which allows the integration of some coding to the components, is also considerable.

Data reliability may also be a major concern in a mashup development. A well-defined extraction algorithm might not be able to point to precise information pieces from Web-enable resources due to their rapid updates. The extraction method should be robust and sustainable to changes made over time [75]. The other important issue in mobile mashup is mobile platform constraints. Some closed platforms (e.g., Apple's iOS and NTT docomo's i-appli[76]) do not allow inter-application communications. Mashups in these platforms have to be integrated in low-level programming code. Therefore, the contributions of creators in the form of packaged applications cannot be reused due to this platform constraint. The development of tethered mobile applications that are able to communicate via HTTP connections might pave the way for the possibility of more development of integrated approaches in mashups.

In regard to security, the application of WA components in MAIDL to invoke cross-domain connections and navigations of secured Web sites are possible. Moreover, the use of MA components in mashup compositions also raised some security issues such as the publications of personal information. Therefore, a security assessment should be conducted for each component in a mashup and reported to the user before the mashup is installed and used.

6.3 Mashup Context Difference Conflict

A mashup created in passive context is suggested to use only background components. It is quite non-typical when a host device is interrupted and requested to do an interactive task (such as taking a photo) during the middle of other usages. In MAIDL, the usage of foreground components in passive context is allowed via the use of SMA-based protocol. However, these features might have to be controlled. When a host device is requested to perform an interrupting task, a permission request dialog is recommened for smoother operations.

7 Conclusion

This research proposed a description-based composition method using *Mobile Application Interface Description Language*. This method allows a control-propagated data flow compositions for reuses and creation of mobile applications, Web services and Web applications. In technical aspect, the generated mashups took shorter execution time in a 3G network connection setting when the mashup execution applied the *Mashup Process Scheduling* mechanism. In usability aspect, the composer evaluation results indicated that the flow-based composition method could moderately assist novices while experts could effectively adapt their own libraries to their mashup. The result also showed no composition time difference between two composer groups. However, the composition patterns found in the novice group have less complexity than in the expert group. We are able to indicate the composition patterns, commonly used components, and composition mistakes of both groups from our experiment.

For tethered mashup applications, our method could deliver the composition of a tethered Web service and a tethered Web application. This non-typical context can solve the platform-dependent constraint of inter-application communications and allows the composition of co-operative and functionality exchange mashups. Regarding expressivity, it is considerable that the synchronous and flow-based manner of our approach might suppress the complexity of the composition model. The future work for our research includes the composition of interactive and scripting elements. It is also preferable to promote the event-based approach for mashups of continuous data streams.

References

1. International Telecommunication Union (2010), *Key Global Telecom Indicators for the World Telecommunication Service Sector*. `http://www.itu.int/ITU-D/ict/statistics/at_glance/KeyTelecom.html`.
2. Apple iOS App Store `http://www.apple.com/iphone/from-the-app-store/`.
3. Android Market `https://market.android.com/`.
4. M. Gaedke, M. Beigl, H. Gellersen and C. Segor (1998), *Web Content Delivery to Heterogeneous Mobile Platforms*, LNCS, Vol.1552, , pp.1429-1437, Springer-Verlag (Berlin, Heidelberg).
5. Y. Chen, X. Xie, W. Ma, H. Zhang (2005), *Adapting Web Pages for Small-Screen Devices*, IEEE Internet Computing, Vol. 9, pp. 50-56, IEEE Computer Society (Los Alamitos, CA, USA).
6. B. MacKay, C. Watters and J. Duffy (2004), *Web Page Transformation When Switching Devices*, LNCS, Vol. 3160, pp.228-239, Springer-Verlag (Berlin, Heidelberg).
7. N. Bila, T. Ronda, I. Mohomed, K. N. Truong and E. de Lara (2007), *PageTailor: reusable end-user customization for the mobile web*, In *Proceedings of the 5th international conference on Mobile systems, applications and services*, pp. 16-29, ACM (New York, NY, USA).
8. Y. Kao and T. Kao and C. Tsai and S. Yuan (2009), *A personal Web page tailoring toolkit for mobile devices*, Computer Standards & Interfaces, Vol. 31, 2, pp.437-453, Elsevier Science Publishers B. V. (Amsterdam, The Netherlands, The Netherlands).
9. X. Liu, Y. Hui, W. Sun and H. Liang (2007), *Towards Service Composition Based on Mashup*, IEEE Congress on Services, pp. 332-339, IEEE Computer Society (Los Alamitos, CA, USA).
10. D. Benslimane, S. Dustdar and A. Sheth (2008), *Services Mashups: The New Generation of Web Applications*, IEEE Internet Computing, Vol. 12, pp.13-15, IEEE Computer Society (Los Alamitos, CA, USA).
11. S. Pietschmann, J. Waltsgott and K. Meißner (2010), *A Thin-Server Runtime Platform for Composite Web Applications*, In *Proceedings of the 2010 Fifth International Conference on Internet and*

Web Applications and Services, pp. 390-395, IEEE Computer Society (Washington, DC, USA).

12. T. Laakko and T. Hiltunen (2005), *Adapting Web Content to Mobile User Agents*, IEEE Internet Computing, Vol. 9, pp.46-53, IEEE Computer Society (Los Alamitos, CA, USA).
13. Yahoo Pipes. `http://pipes.yahoo.com/pipes/`.
14. S. Pietschmann, M. Voigt, A. Rümpel and K. Meißner (2009), *CRUISe: Composition of Rich User Interface Services*, LNCS, Vol. 5648, pp.473-476, Springer-Verlag (Berlin, Heidelberg).
15. J. Guo, H. Han and T. Tokuda (2010), *Towards Flexible Mashup of Web Applications Based on Information Extraction and Transfer*, LNCS, Vol. 6488, pp.602-615, Springer-Verlag (Berlin, Heidelberg).
16. H. Han and T. Tokuda (2009), *An Automatic Web News Article Contents Extraction System Based on RSS Feeds*, J. of Web Engineering, Vol. 8, No. 3, pp. 268-284, Rinton Press.
17. J. Guo, H. Han and T. Tokuda (2010), *A New Partial Information Extraction Method for Personal Mashup Construction*, In *Proceeding of the 2010 conference on Information Modelling and Knowledge Bases XXI*, pp. 155-168, IOS Press (Amsterdam, The Netherlands, The Netherlands).
18. Web Services Business Process Execution Language Version 2.0. `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html`.
19. Open Mashup Alliance Enterprise Markup Language Documentation. `http://www.openmashup.org/omadocs/v1.0/index.html`.
20. S. Pietschmann, T. Nestler and F. Daniel (2010), *Application Composition at the Presentation Layer: Alternatives and Open Issues*, In *Proceedings of the 12th International Conference on Information Integration and Web-based Applications & Services*, pp. 461-468, ACM (New York, NY, USA).
21. PhoneGap. `http://www.phonegap.com/`.
22. Rhodes. `http://rhomobile.com/`.
23. Appceletor Titanium. `http://www.appcelerator.com/`.
24. C. Cappiello, F. Daniel, M. Matera, M. Picozzi and M. Weiss (2011), *Enabling End User Development through Mashups: Requirements, Abstractions and Innovation Toolkits*, In *Proceedings of the Third international conference on End-user development*, pp. 9-24, Springer-Verlag (Berlin, Heidelberg).
25. T. Nestler, A. Namoun and A. Schill (2011), *End-user Development of Service-based Interactive Web Applications at the Presentation Layer*, In *Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems*, pp. 197-206, ACM (New York, NY, USA).
26. N. Mehandjiev, F. Lecue, U. Wajid and A. Namoun (2010), *Assisted Service Composition for End Users*, In *Proceedings of the 2010 Eighth IEEE European Conference on Web Services*, pp.131-138, IEEE Computer Society (Washington, DC, USA).
27. F. Paternò, C. Santoro and L.D. Spano (2011), *Engineering the authoring of usable service front ends*, J. Syst. Softw., Vol. 84, No. 10, pp. 1806-1822,Elsevier Science Inc. (New York, NY, USA).
28. J. Yu, B. Benatallah, F. Casati and F. Daniel (2008), *Understanding Mashup Development*, IEEE Internet Computing, Vol. 12, pp. 44-52, IEEE Computer Society (Los Alamitos, CA, USA).
29. SOAP Specifications `http://www.w3.org/TR/soap/`
30. RESTful Web Services `https://www.ibm.com/developerworks/webservices/library/ws-restful/`
31. G. Ghiani, F. Paternò, Fabio and L. Spano (2011), *Creating Mashups by Direct Manipulation of Existing Web Applications*, LNCS, Vol. 6654, pp.42-52, Springer-Verlag (Berlin, Heidelberg).
32. P. Chaisatien, K. Prutsachainimmit and T. Tokuda (2011), *Mobile Mashup Generator System for Cooperative Applications*, LNCS, Vol. 6757, pp. 182-197, Springer-Verlag (Berlin, Heidelberg).
33. Web Services Description Language `http://www.w3.org/TR/wsdl`
34. Android Developers `http://developer.android.com/sdk/index.html`
35. I-Jetty: Web server for the Android mobile platform `http://code.google.com/p/i-jetty/`
36. Android Intents and Intent Filters, `http://developer.android.com/guide/topics/fundamentals/services.html`
37. Android Services `http://developer.android.com/guide/topics/fundamentals/services.html`

38. Apple URL Scheme Reference http://developer.apple.com/library/ios/#featuredarticles/iPhoneURLScheme_Reference/Introduction/Introduction.html
39. Intent http://developer.android.com/reference/android/content/Intent.html
40. Intents List: Invoking Google Applications on Android Devices http://developer.android.com/guide/appendix/g-app-intents.html
41. Publishing on Android Market, Android Developers http://developer.android.com/guide/publishing/publishing.html
42. MediaStore http://developer.android.com/reference/android/provider/MediaStore.html
43. Send SMS via Intent http://www.androidsnippets.com/send-sms-via-intent
44. Speech Input http://developer.android.com/resources/articles/speech-input.html
45. Using Text-to-Speech http://developer.android.com/resources/articles/tts.html
46. Paranomio, Radar http://android-developers.blogspot.com/2008/09/panoramio.html
47. Resources for Mobile Development with YouTube http://code.google.com/apis/youtube/articles/youtube_mobileresources.html
48. RpnCalc, A full-function scientific calculator for Android phone http://www.efalk.org/RpnCalc/
49. Calendar Picker https://market.android.com/details?id=org.openintents.calendarpicker
50. Motion Gesture https://market.android.com/details?id=RabiSoft.MotionGestureAd
51. Motion to Intent https://market.android.com/details?id=RabiSoft.MotionToIntentAd
52. Location Notifier https://market.android.com/details?id=RabiSoft.LocationNotifier
53. Color Picker https://market.android.com/details?id=org.openintents.colorpicker
54. Tag to Intent https://market.android.com/details?id=RabiSoft.TagToIntent
55. Translate Intent http://android.kupriyanov.com/apps/translate-intent
56. chartdroid http://code.google.com/p/chartdroid/
57. Developing on Android, Evernote http://www.evernote.com/about/developer/android.php
58. Adobe Photoshop Express for Android - For Developers http://mobile.photoshop.com/android/developers.html
59. Twidroyd, Plugins http://twidroyd.com/plugins/
60. Document Object Model (DOM) http://www.w3.org/DOM/
61. WebView http://developer.android.com/reference/android/webkit/WebView.html
62. XML Path Language (XPath) http://www.w3.org/TR/xpath/
63. JSONPath - XPath for JSON http://goessner.net/articles/JsonPath/
64. ScanningViaIntent http://code.google.com/p/zxing/wiki/ScanningViaIntent
65. Amazon.com http://www.amazon.com/
66. Exchange Rate API http://www.exchangerate-api.com/
67. Android Developers: Building and Running http://developer.android.com/guide/developing/building/index.html
68. i-Jetty: Creating Downloadable Web Applications http://code.google.com/p/i-jetty/wiki/DownloadableWebapps
69. Gourmet Navigator API. http://api.gnavi.co.jp/api/manual.htm
70. Weather Stations with most recent Weather Observation. http://www.geonames.org/export/JSON-webservices.html#weatherJSON
71. Find nearby Wikipedia Entries. http://www.geonames.org/export/wikipedia-webservice.html#findNearbyWikipedia
72. Nexus One Phone Gallery http://www.google.com/phone/detail/nexus-one
73. F. Paternò, C. Santoro and L.D. Spano (2009), *Maria: A universal, declarative, multiple abstraction level language for service-oriented applications in ubiquitous environments*, ACM Trans. Comput.-Hum. Interact., Vol. 16, 4, pp. 19:1-19:30, ACM (New York, NY, USA).
74. Google App Inventor for Android. http://appinventor.googlelabs.com/
75. S. Lingam and S. Elbaum (2007), *Supporting End-Users in the Creation of Dependable Web Clips*, In *Proceedings of the 16th international conference on World Wide Web*, pp.953-962, ACM (New York, NY, USA).
76. Docomo i-appli http://www.nttdocomo.co.jp/english/service/imode/make/content/iappli/