JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 34, 611-629 (2018)
DOI: 10.6688/JISE.201805_34(3).0003

Building Mobile Apps by Ordinary Users: A Service-Brick-Based Approach*

SHANG-PIN MA+, PENG-ZHONG CHEN, YANG-SHENG MA
AND JHENG-SHIUN JIANG
Department of Computer Science and Engineering
National Taiwan Ocean University
Keelung, 202 Taiwan
+*E-mail: albert@ntou.edu.tw*

To build an effective, efficient, and easy-to-use mobile service composition and delivery approach for ordinary users, in this research, we propose a framework, called CARSB (Composite App with RESTful service and Service Bricks), to create mobile Apps based on Service Bricks and RESTful services. Three main features are offered by the CARSB framework, including (1) Service Brick, a rectangular UI component used for the display of specific information, is introduced as the basic building block; (2) a mobile service composition framework, which can integrate Service Bricks with local resources or RESTful services in the cloud, is devised; and (3) a web-based software tool, called CARSB Portal, is provided to allow ordinary users to build their customized composite mobile applications, either Web-based or Android-based, according to their requirements. Besides, in this research, quantitative experiments were conducted to verify the proposed CARSB approach. Experiment results demonstrate that the proposed CARSB approach is able to achieve a considerable decrease in user operation time and network transmission load.

Keywords: mobile mashup, service brick, RESTful service composition, mobile service composition, mobile App delivery

1. INTRODUCTION

Mobile applications (*i.e.*, mobile Apps or Apps) are becoming an important software delivery model for the integration of front-end user interfaces (UIs) with back-end services in the cloud [1]. A wide range of mobile Apps are available, including those dealing with news, entertainment, travel, multimedia, and social networking. For example, the number of various Android Apps are close to three millions.[1] However, the retrieval and composition of information from multiple Apps, services, or local resources can be time-consuming, costly, and inconvenient. Mashup [2] technology is used to create websites that combine information and services from multiple sources on the web. Despite recent advances in mobile mashups [3, 4], which apply the mashup techniques to the mobile environment, very few configurable mashup mechanisms have been developed to enable the combination of information and services from front-end as well as back-end resources for the use on mobile Apps. Meanwhile, most existing service composition methods [5, 6] fail to take into account how to deliver services to the user interfaces in

Received January 24, 2017; accepted April 26, 2017.
Communicated by Ren-Hung Hwang.
* This research was sponsored by Ministry of Science and Technology in Taiwan under grants MOST 103-2221-E-019-039 and MOST 105-2221-E-019-054-MY3.
[1] http://www.appbrain.com/stats/number-of-android-apps.

mobile Apps.

In this paper, we introduce the concept of the Service Brick [7] as well as a Service Brick composition framework, called CARSB (Composite App with RESTful Services and Bricks), to enable the integration of front-end UI components with back-end services and information for building customized mobile Apps by ordinary users. The definition of "ordinary user" is a user without programming skills or a novice at programming. In CARSB, a Service Brick (Brick or SB) enables the development of web-enabled or Android-Fragment-based user interfaces, linking server-side or client-side resources, such as external RESTful services or local modules in the smartphone. The fact that external RESTful service could be atomic or composite makes it possible for users to combine multiple Service Bricks through the integration of services and/or information within a customized mobile App. The method proposed in this study for the composition of RESTful services is built atop JOpera [8]. JOpera is a widely-used service composition tool providing a visual language with which to define control flow and data flow for service processes as well as an execution engine. CARSB furnishes a set of utility services and a service mediator to facilitate communication between the mobile App and the JOpera server. Besides, to provide complete functionality of the CARSB approach for ordinary users, we developed a Web-based software tool, referred to as CARSB Portal, to allow users to publish their atomic or composite Service Bricks as well as search and download those developed by other users. CARSB Portal also provides other auxiliary functionality, such as Brick recommendation and Brick preview, to enhance the usability. Quantitative experiments were also conducted to verify the proposed CARSB approach from the viewpoint of reduction for user operation time and network transmission load.

The remainder of this paper is organized as follows: Section 2 presents a review of related research. Section 3 outlines the details of the proposed CARSB concept, including the notion of Service Brick, Event Engine, and Resource Provider, as well as the design of CARSB Portal. Empirical evaluations of the proposed system are presented in Section 4, and the work is concluded in the final section.

2. RELATED WORK

In this section, we review a number of previous studies related to mobile service composition, which can be divided into two research tracks, RESTful service composition and web mashups or mobile mashups.

Rosenberg *et al.* [6] provided an extensible, XML-based language called Bite, in conjunction with an integrated programming model for the composition of RESTful services with interactive flow. Alarcon *et al.* [9] designed hypermedia-centric REST service description, ReLL (Resource Linking Language) in conjunction with Petri Nets for the modelling of service compositions. Pautasso [10, 11] introduced a means by which to extend BPEL (Business Process Execution Language) to invoke RESTful WSs (Web Services) and publish BPEL processes as RESTful Web Services. The above methods can integrate multiple RESTful services from various viewpoints, however, these works do not take into account the issue of service delivery to the user interface in mobile Apps.

Several online services that combine UI components have already been developed by IT companies, including iGoogle, a customizable, Ajax-based startpage comprising multiple portlets providing access to news, weather, sports, and other information. PageFlakes is a website service similar to iGoogle, which features tabs containing user-selected modules called Flakes. NetVibes is a personalized dashboard for the publishing of websites comprising widgets obtained from a list of widgets developed by third-party developers. WidgetBox enables businesses to create novel widgets to be embedded in the personal blogs or web pages of users. Unlike Portlets, Flakes, and widgets, the Service Bricks proposed in this study are able to cooperate with each other for the creation of more complex applications.

Yu *et al.* [12] introduced a framework for building an application UI by combining front-end presentation components. The proposed notion of presentation component encapsulates UI and logic to manage user interactions and facilitates the event-based integration of multiple presentation components to build an overall user interface through a declarative composition language, XPIL (Extensible Presentation Integration Language). Nestler *et al.* [13] presented ServFace Builder, an authoring tool designed to enable individuals without programming skills to design and create service-based interactive applications using a graphical interface. This makes it possible to combine two service components simply by clicking their inputs and outputs, thereby simplifying the creation of new service components with front-ends. Process-Data-Widget (PAW) [14], our previous research result, is a REST-based service mashup framework which functions as a composition model for the construction of mashup applications. Developers are able to design service processes, compose service data, and configure widgets for presentation on a UI simply by constructing a mashup document (MD). The PAW mashup engine parses the MD and generates a corresponding mashup application and associated RESTful services. The above methods can merely be used to create conventional mashup applications, but not those for mobile environment.

Francese *et al.* [15] proposed an approach, called MicroApp Generator (MAG), for sharing and reusing user-generated mobile applications. The MicroApp is able to be designed visually by selecting the MicroApp services in the MicroApp Store. Notably, each service exposes a description of its user interface that enables to generate the user interface. The main difference between MAG and CARSB is that MAG generates specific, graph-based MicroApps whereas CARSB generates mobile web apps or installable Android Apps. Lee *et al.* [16] proposed ICBMS Smart Mediator which provides a unified OpenAPI to interlink various platforms and data sources and to help developers to do mashup without studying APIs for each platform. ICBMS also provides a full-fledged web-based interface to help developers to create new mashup services that combines multiple backend OpenAPIs. ICBMS does not take into account the integration of front-end UI components. Zhai *et al.* [17] presented an ecosystem, including EasyApp and LSCE, to let end users develop mobile applications by creating cross-platform GUIs and service process that are invoked by the GUIs. The EasyApp was developed based on the OSGI (Open Service Gateway Initiative) framework to let users design user interfaces by selecting provided widgets. The core concepts of EasyApp and CARSB is different – EasyApp is a "design" tool to let users design the App using fine-grained UI elements whereas CARSB is a "composition" tool to let users do mashup by combining coarse-grained UI components (*i.e.* Service Bricks).

To elaborate the contributions of the proposed approach, comparison with above related works for five different dimensions: communication among UI components, support of Web service composition, enabling the usage in mobile devices, creation of mashups by ordinary users, and generation of installable Apps, were provided. The analysis and comparison is shown in Table 1. Comparing with other representative efforts, CARSB fully supports all significant requirements for mobile service composition. In a nutshell, CARSB is capable of building mobile Apps, which comprises linkable Service Bricks in the mobile front-end and composite RESTful services in the cloud, in an easy and configurable way.

Table 1. Comparison with other mobile service composition methods.

	Communication among UI components	Support of Web service composition	Enabling the usage in mobile devices	Creation of mashups by ordinary users	Generation of installable Apps
Rosenberg's work (Bite)	No	Yes, process-oriented	No	No	No
Alarcon's work	No	Yes, process-oriented	No	No	No
Pautasso's work	No	Yes, process-oriented	No	No	No
Online Services	No	No	No	Yes	No
Yu's work	Yes	Not mentioned	No	No, merely for developers	No
Nestler's work (ServFace)	Yes	Yes, UI-driven approach	No	Yes	No
Process-Data-Widget (PAW)	Yes	Yes	No	No, merely for developers	No
Francese's work (MicroApp)	Yes	Yes	Yes	Yes	Not mentioned
Lee's work (ICBMS)	No	Yes	Yes	No, merely for developers	Not mentioned
Zhai's work (EasyApp)	Yes, for fine-grained UI elements	Yes	Yes	Yes	Not mentioned
Our approach: CARSB	Yes, for coarse-grained Service Bricks	Yes, both UI-driven and process-oriented ways	Yes	Yes	Yes

3. CARSB: COMPOSITE APP WITH RESTFUL SERVICE AND SERVICE BRICKS

In this section, we outline the proposed approach in detail. We first analyze the requirements and then introduce the core concepts and interactions among them. We then present the methods used to establish the proposed approach.

The objective of this research is to facilitate the composition of multiple UIs and information sources by and for smartphone users. Meanwhile, a composite App was expected to be more efficient on the user operation time and the usage of network bandwidth than the use of separate Apps or websites. An examination of existing mobile Apps and an in-depth survey of the literatures helped to identify the following key requirements:

1. Integration of front-end UI components with local as well as remote resources within composite mobile Apps.
2. Front-end components can communicate with each other.
3. Decreased needs of programming skills in the creation of mobile composite Apps.
4. Reduced reliance on network communication and shortened user operation time.

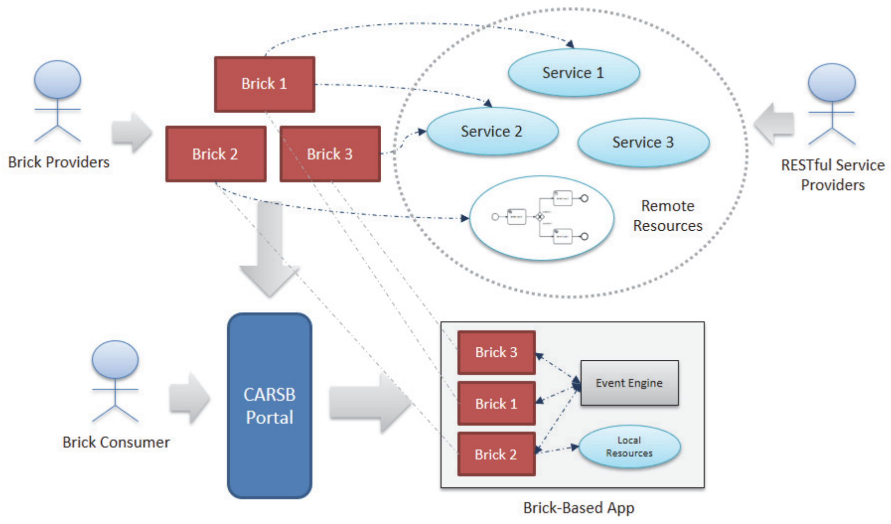


Fig. 1. Operational concepts.

Fig. 1 illustrates the operational concepts of the proposed approach. First, RESTful Service Providers publish RESTful services in the cloud. In CARSB, services are also called external resources and divided into two classifications, atomic services and composite ones; *i.e.*, processes services hosted in the JOpera server. Second, the Brick Providers develop Service Bricks that link resource providers, including external resources and client-side local resources like an address book or GPS module, and publish their Bricks to the CARSB Portal. A Service Brick is a rectangular UI component used for the display of specific information in an App. Third, via the CARSB Portal, the Brick Consumer (*i.e.*, end user) designs Brick-based App by composing multiple Service Bricks for the display of integrated services or information. In the App, Service Bricks communicate with each other through the Event Engine.

The CARSB Portal is a web application that allows users to publish and compose Service Bricks and access those published by others. Resources stored in the Service Brick Repository include HTML, CSS, JavaScript, and JSON-based description documents, including those used for atomic and composite Brick descriptions, and composite RESTful services. Please refer to [7, 18] for information on JSON-based description documents. The CARSB Portal can be used to access these documents via the Brick Manager. In addition to the operational concepts, Fig. 15 illustrates the conceptual model illustrating the core notions on which the proposed CARSB framework was devised. In the following sub-sections, we describe the above concepts in greater detail within the context of the conceptual model.

3.1 Service Brick (SB)

The principal function of Service Bricks is to make available for users a display of the services and/or information they desire. Limitations imposed by the screen size on smartphones require that the layout of Bricks be as simple as possible. Furthermore, the network bandwidth of 3G network or WIFI is generally slower than that of wired networks; therefore, the architecture of the Bricks requires a minimum number of requests and must be optimized for network transmission load. The general structure of a Service Brick was shown in Fig. 2. A Service Brick can accept input events (IE), generate output events (OE), and link to Resource Providers.

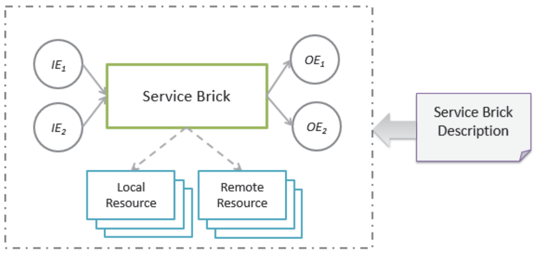


Fig. 2. The structure of a service brick.

We devised Service Bricks from the perspective of technology (*i.e.*, Web-based and Android-Fragment-based) as well as from the perspective of composability (*i.e.*, simple and composite). Details of these types of Bricks are outlined in the following.

A. Simple and Composite Service Bricks

A simple Service Brick is a rectangular UI component used in an App. It is an elementary unit in a composite Service Brick responsible for displaying data received from resource providers. Based on their behaviors, simple Service Bricks are classified as Isolated Service Bricks and Linkable Service Bricks.

- Isolated Service Bricks: This type of Service Brick does not interact with other Bricks by sending or receiving events. One example of an isolated Brick is the time Brick, which only displays the current time and does not transfer or receive any data from other Bricks.
- Linkable Service Bricks: Linkable Bricks interact with other Bricks by exchanging event-related data with other Bricks. One example of linkable Service Bricks is the song Brick, which can find the albums and songs of singers specified by the user and the lyric Brick, which can display the lyrics of the song designated by the user. These

two Bricks can cooperate in the provision of composite functions. Specifically, when the user clicks the song button, the song Brick sends an event containing the song name to the Event Engine. Then the Event Engine forwards the event to the lyric Service Brick to initiate the display of lyrics.

Composite Service Bricks (CSBs) are composed of multiple Service Bricks (both Isolated and Linkable Service Bricks) for the integration of services and the graphical user interfaces. Individual Bricks are generally integrated into a composite Service Brick through the transmission of events, whereas Isolated Bricks are simply embedded within a composite Service Brick.

B. Web-Based and Android-Fragment-Based Service Bricks

Web-based Service Bricks can be constructed using existing Web technologies, such as HTML, CSS, and JavaScript. Android-Fragment-based Service Bricks can be constructed using Android Fragment APIs and web technologies. Both kinds of Service Bricks use the web design patterns proposed by Ribeiro and Carvalhais [19] to render content by vertical layout.

In the Android system, a Fragment represents a modular section of an activity with its own lifecycle. CARSB utilizes Fragment as the container to host Service Bricks and provides two sub-types of Android-Fragment-based Service Bricks:

(1) Web-enabled Service Bricks are created using Web technology prior to implementtation in Android WebView. All resources, such as the source files for HTML, CSS, and Javascript, are cached on the client side, *i.e.* on mobile devices. Thus, unlike Web-based Service Bricks, the proposed mechanism does not require the loading of resources from the server, which enhances overall efficiency. The communication between two Web-enabled Service Bricks is relatively indirect – when a Brick produces an output event, CARSB transforms the event from a JavaScript object to an Android Java object, and sends it from the source Fragment to the target Fragment via Event Engine. In the target Fragment, CARSB conversely transforms the event from an Android Java object to a JavaScript object and triggers the following actions.
(2) Native Service Bricks are created using pure Android native objects used to obtain additional functionalities that conventional web pages are unable to perform. Native Service Bricks make it possible to use sensor data locally and communicate with remote services to obtain value-added functionality. For example, native Service Bricks could use the current geospatial location to obtain weather-related data, or use heart rate sensors or pedometers for the monitoring of exercise performance. The communication between two Web-enabled Service Bricks is direct – CARSB forwards the generated output event to the Service Bricks that can accept the event to trigger the following actions.

3.2 Event Engine (EE)

An Event Engine is responsible for handling events from Service Bricks. Events are stored in an Event Pool before being handled sequentially by the Event Engine or other scheduling strategy using the mechanisms of Publish/Subscribe Handler or Push Handler. The Event Engine is executed on the client-side as a JavaScript Object or an Android object. This makes it possible to avoid unnecessary network traffic by having all event communications occur on the client side.

Collaboration among Service Bricks is achieved through the submission and receipt of events. In other words, when one Brick sends an event, another Brick should receive it, as long as it can be handled by that Brick. Each Brick is allowed to stipulate which output events may be generated during runtime, as well as the input events triggering further actions. Bricks that require an input event are treated as the subscriber of the event, whereas Bricks that generate output events act as the event publisher. The Event Engine plays the role of mediator between output and input events, which means that it is responsible for finding subscribers capable of handling the event emerging from the subscriber. Essentially, the process of matching the publisher and the subscriber is based on the types of input/output events (See Fig. 3). Two kinds of event matching are provided: exact matching and subsumption-based matching. Exact matching means that the types of the submitted output event from one Brick and the acceptable input event for another Brick are the same; subsumption-based matching indicates the type of the submited output event is a subtype for the type of acceptable input event. The type hierarchy is generated by reasoning the global type ontology managed in CARSB.

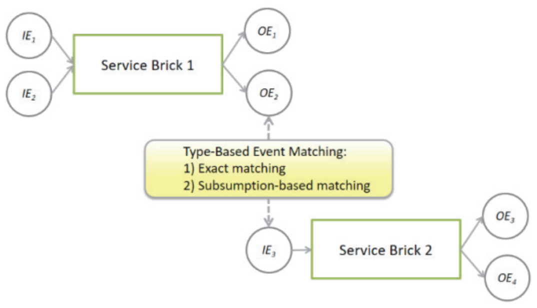


Fig. 3. The concept of event engine.

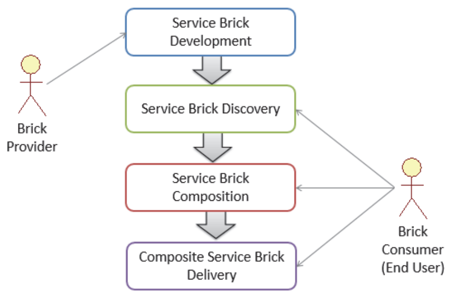

Brick Provider
Service Brick Development
Service Brick Discovery
Service Brick Composition
Composite Service Brick Delivery
Brick Consumer (End User)

Fig. 4. Stages of the development and operation for Service Bricks.

Besides, the user can also add enforced connections to enable the linking of input and output events in cases where the two events do not match. Similarly, the user stipulates which connections are forbidden. Please refer to [7] for further information on the notion of enforced and forbidden connections. The Event Engine also furnishes the Push Handler in the continual updating of Brick content. For example, the Email Brick invokes the back-end Resource Provider to check the mail regularly and ensure that the state of the mailbox remains up-to-date.

3.3 Resource Provider (RP)

The main responsibility of the Resource Provider (RP) is the provision of data requested by the user for the display of application data associated with Service Bricks. Thus, the description of the Brick should contain information related to the associated RPs via the Service Manager. It should be noted that the relationship between Resource Provider and Service Brick remains unchanged. For any Service Brick, the user is able to switch to a different Resource Provider. The Resource Provider usually provides data through a RESTful service interface in the JSON data format. In this research, we divided Resource Provider into 2 categories: Local Resources and External Resources.

- Local Resources are obtained from client-side smartphones, including user contextual data, such as GPS (Global Positioning System) or an address book, and other phone-specific functional modules, such as the phone call module.
- External Resources include all data provided by an external service provider. For instance, news data in a News Brick is supplied by an external provider such as Google or Microsoft.

To deal with external resource providers, we developed a mechanism to bridge front-end Service Bricks with back-end atomic or composite RESTful services in a stateful manner; *i.e.*, service data can be stored and processed during any interchange between Bricks and services. We adopted the widely-used JOpera service process engine as an underlying platform. To ensure the flexibility of service composition, the first objective of CARSB is to facilitate state-enabled service interchange between front-end Bricks and back-end services. The dynamic substitution of component services [20] is another objective for the composition of dynamic services. This involves state analysis as well as the integration of two utility services (a component service substitution module and a session management module), into the service flow. It should be noted that two utility services are expected to be embedded into the service flow.

As a result, the CARSB framework mainly provides three functionalities associated with the linkages among composite RESTful services:

- One-shot service flow: Composite services are able to invoke multiple RESTful services in a specified sequence in order to generate an integrated result following the execution of the service. The results are then displayed in a specific Service Brick.
- Stateful service flow: CARSB also supports conversational composite services by allowing multiple Service Bricks within a CSB to connect to the same composite service via the session management mechanism. The session management mechanism gives each service process instance a unique id to allow multiple Service Bricks in the same App to participate in the same process instance. In this case, each Brick could display different content retrieved in different phases of the service flow.
- Service substitution: Service substitution allows users to substitute composite services with other binding component services. Nonetheless, the substituted service must be interface-compliant with the original component service. For example, a user could simply use Bing Videos as a replacement for the YouTube service embedded in the video Service Brick.

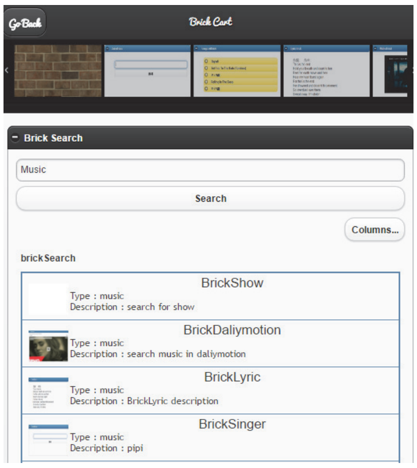
3.4 CARSB Portal

As mentioned, we developed a Web-based tool, referred to as CARSB Portal to implement all desired features in the CARSB approach. The four stages of the development and operation for Service Bricks were realized in the CARSB Portal (See Fig. 4):

1. Service Brick Development

The Brick provider develops atomic Service Bricks based on the associated APIs provided in this research. These APIs include event handling and resource access as well as service invocation. The functionality of Service Bricks is developed using Web technology (for Web-based or Web-enabled Bricks) or Android technology (for Android-Fragment-based Bricks).

2. Service Brick Discovery
   Users are able to search for and obtain Bricks using shopping-cart-like workspace (See Fig. 5). When the user imports a Service Brick (the target Brick) into his/her workspace, CARSB Portal recommends other Service Bricks that could be integrated with the imported Brick. These recommendations may include preceding Bricks, with output events matching the target Brick's input event, or following Bricks with input events matching the output event of the target Brick. The event matching mechanism is the type-based matching mentioned in Section 3.2. CARSB Portal may also recommend similar Service Bricks possessing input and output events that are the same as those associated with the target Brick. The recommendation functionality (See Fig. 6) in CARSB Portal can effectively enhance the usability of Service Brick discovery.



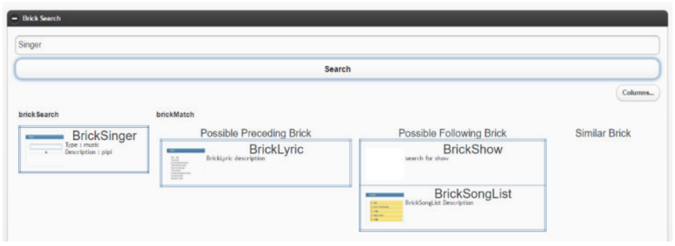
BrickShow
Type : music
Description : search for show

BrickDaliymotion
Type : music
Description : search music in dailymotion

BrickLyric
Type : music
Description : BrickLyric description

BrickSinger
Type : music
Description : pipi

Fig. 5. CARSB Portal: Service brick discovery.



BrickSinger
Type : music
Description : pipi

BrickLyric
BrickLyric description

BrickShow
search for show

BrickSongList
BrickSongList Description

Fig. 6. CARSB portal: Service brick recommendation.

3. Service Brick Composition
   One important objective in the development of CARSB Portal was to allow ordinary users (without programming skills) to design attractive composite Bricks with ease.

Using the provided visual design tool in CARSB Portal, the Brick consumer (*i.e.*, the aforementioned ordinary users) is able to compose Service Bricks In this phase, the user is free to select the positon of each Service Brick and set preferences pertaining to forbidden or enforced connections between events in a WYSIWYG (What You See Is What You Get) manner (See Fig. 7). For example, the user may select four Service Bricks, Singer Service Brick, Song Service Brick, Lyric Service Brick and YouTube Service Brick, in the stage of service brick discovery. In this stage, the user may decide the positions of these four Service Bricks under the vertical layout and add the enforced connections between the Song Brick and YouTube Brick to allow the YouTube Brick display videos according to the selected song name in the Song Brick. In a nutshell, these four Bricks are composed as a composite one based on the user configuration.

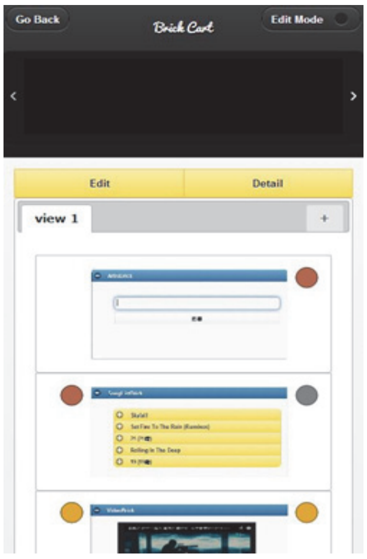


Fig. 7. CARSB portal: Service brick composition.

The user can then preview the composite Service Brick he has just designed before publishing it. The preview functionality can allow the user to trigger Service Bricks step-by-step based on the prepared test data in the Service Brick Descriptions. The preview algorithm is shown in Fig. 8. The Brick Set (BS) is the set that collects all unmarked Bricks. The Brick Graphs (BG), where the Bricks are vertices and the connections between Bricks are edges, are built based on the matching input/output events among Bricks. The Preview Queue (PQ) is the queue that stores found paths of Service Bricks. The basic idea for the preview mechanism in this research is finding the longest path between any two vertices in the BGs and preview the Bricks in the selected path iteratively. Note that multiple BGs may be constructed because it is hard to connect all Bricks in a graph according to the matching of input/output events.

4. Delivery of Composite Service Bricks

CARSB Portal provides two ways in which published Composite Service Bricks can be implemented: Portal model (See Fig. 9 (a), a music App including four web-based Service Bricks) and App mode (See Fig. 9 (b), an exercise App including three native Service Bricks). Portal mode enables the use of Web-based Composite Service Bricks using the CARSB Portal directly. App mode is applicable to all types of Composite Service Brick (particularly Android-Fragment-based ones), using a packaged Android App. In other words, the App of a Composite Service Brick can be used without the need for a hyperlink to the CARSB Portal website. Both modes enable users the browsing of integrated user interface and services using composite Service Bricks, even when using the small screens found on mobile smartphones.

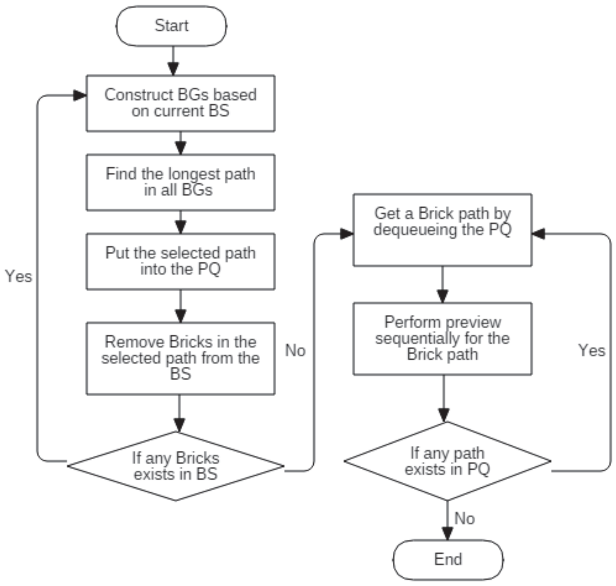
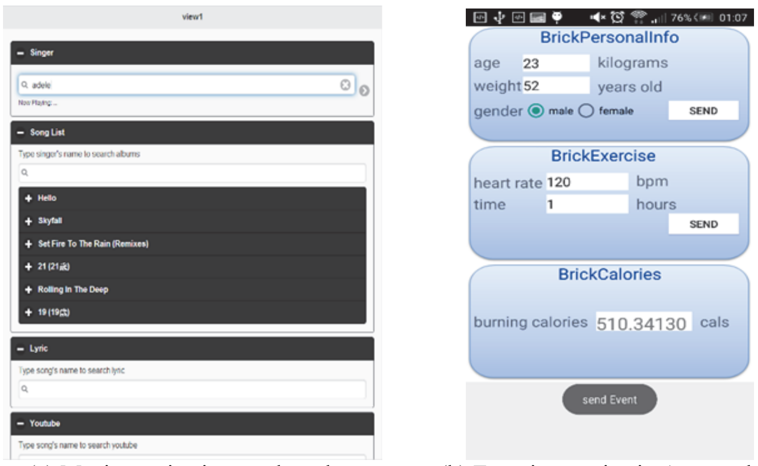

Start
Construct BGs based on current BS
Find the longest path in all BGs
Put the selected path into the PQ
Remove Bricks in the selected path from the BS
If any Bricks exists in BS
Yes
No
Get a Brick path by dequeueing the PQ
Perform preview sequentially for the Brick path
If any path exists in PQ
Yes
No
End

Fig. 8. Brick preview algorithm.



(a) Music service in portal mode. (b) Exercise service in App mode.

Fig. 9. Service brick delivery.

4. EXPERIMENTAL EVALUATIONS

As mentioned in Section 3, the identified requirements for the proposed CARSB Portal are (1) Integration of front-end UI components with local as well as remote resources; (2) Front-end components can communicate with each other; (3) Decreased needs of programming skills in the creation of Apps; (4) Reduced reliance on network communication and shortened user operation time. From the descriptions in Sections 3.1~3.4, it is obvious that the CARSB approach can satisfy requirements #1~#3. To verify the realization of requirement #4, in this section, we outline the methods used to evaluate the efficacy of the proposed Service Brick system, with regard to reducing operating time and decreasing network transmission load.

4.1 Tests for the Proposed RESTful Service Composition (RSC) Mechanism

Since CARSB without RESTful service composition could still work well, we need to verify the efficacy of the proposed RSC mechanism first. Thus, the goal of the first experiment is to check if the proposed RSC mechanism could reduce service response time or decrease network transmission load. We prepared two groups of Service Bricks – the first group, referred to as Separate Service Invocation, includes twenty Service Bricks that invoke one to twenty prepared RESTful services sequentially. And the second group, referred to as Composite Service Invocation, also includes twenty Service Bricks that invoke different composite services which were composed of one to twenty prepared RESTful services using the sequential process pattern. All forty Service Bricks were wrapped as mobile Apps. We recorded the response time, the delay time in seconds between the moment the input data is submitted and the moment the Brick receives and displays the output data, while the network transmission load was monitored by another App, My Data Manager.[2] We conducted the following four steps:

1. Start the My Data Manager App.
2. Start the App to be tested.
3. Start the stopwatch.
4. Record the response time and consumed network transmission load using the My Data Manager App.

Figs. 10 and 11 show the experiment results. The response times of two test groups are almost the same whereas the network transmission load of the first test group is obviously larger than the second test group. It indicates that proposed service composition mechanism is able to reduce consumed network transmission load, especially for the condition of composing a large number of services, but is not effective to shorten the response time since the service invocations spend almost the same time either in the client side (the first test group) or in the server side (the second test group).

4.2 Tests for the Proposed Service Brick Composition Mechanism

In the second experiment, the goal is to verify the efficacy of the proposed font-end Brick composition mechanism to check if it could reduce user-operating time or de-

[2] My Data Manager: https://play.google.com/store/apps/details?id=com.mobidia.android.mdm

crease network transmission load. In these experiments, a total of 8 mobile Apps were prepared for the evaluation with 4 Web-based Service Bricks, which included 3 atomic Bricks as well as one composite Brick (comprising 3 atomic Bricks), and 4 Fragment-based Bricks, which included 3 atomic Bricks and one composite Brick (comprising 3 atomic Bricks). The atomic Service Bricks were as follows: (1) a Service Brick for showing the song list for a given singer; (2) a Service Brick for displaying the lyrics for a given song; and (3) a Service Brick for embedding YouTube video clips. All Service Bricks (both atomic and composite) were wrapped as Android mobile Apps.

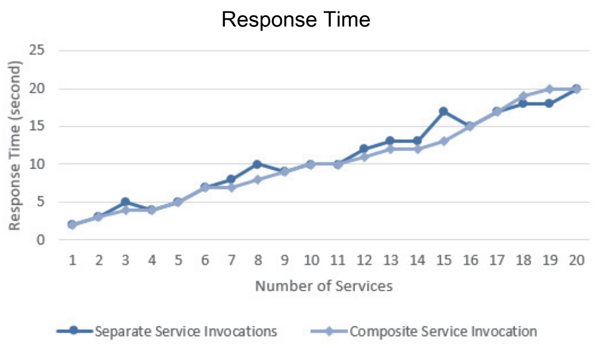


Fig. 10. Response time for separate service invocations and invocation of composite service.

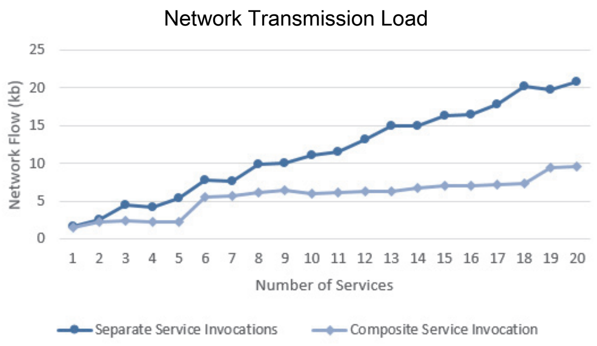


Fig. 11. Network transmission load for separate service invocations and composite service invocation.

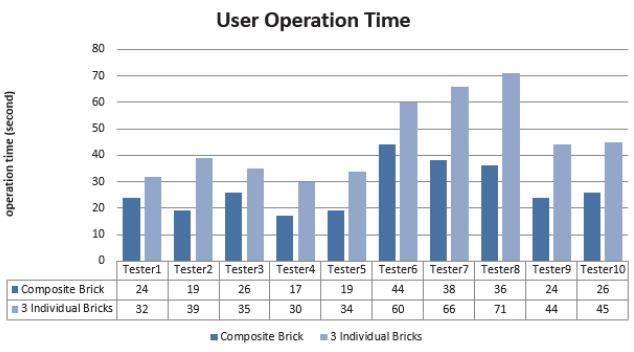
Ten individuals, including five graduate students majored in computer science, two office workers (light smartphone users), and three elder people, were invited to participate as testers to use the prepared Apps on their own Android smartphones. The specifications of the smartphones used by the testers are listed as follows:

	Phone Model	Android Version	Network
Tester1	hTC new One m7 LTE	Android 5.0.2	3G
Tester2	hTC new One m7	Android 5.0.2	3G
Tester3	Sony Xperia V	Android 4.4	3G
Tester4	hTC One E8	Android 5.0.2	4G
Tester5	hTC One M8	Android 5.0.2	4G

Tester6	Asus Zenfone2	Android 5.0	3G
Tester7	hTC Desire 820	Android 4.4.1	4G
Tester8	hTC Desire 826	Android 5.0.2	3G
Tester9	hTC One M9+	Android 5.0.2	4G
Tester10	hTC One M9	Android 5.0.2	4G

The testers were asked to record the user operation time while the network transmission load was also monitored by My Data Manager. Testers conducted the following five steps:

1. Start the My Data Manager App.
2. Start the App to be tested.
3. Start the stopwatch.
4. Browse the content provided by the Apps:
    - For the songlist Brick, testers were requested to input the term "Adele" and click the first album "skyfall".
    - For the lyric Brick, the testers were requested to click the song "skyfall" and glance at the lyrics of "skyfall".
    - For the YouTube Brick, the testers were requested to watch the video until the first lyrics appeared.
5. Record the operation time and consumed network transmission load using the My Data Manager App.



	Tester1	Tester2	Tester3	Tester4	Tester5	Tester6	Tester7	Tester8	Tester9	Tester10
■ Composite Brick	24	19	26	17	19	44	38	36	24	26
■ 3 Individual Bricks	32	39	35	30	34	60	66	71	44	45

Fig. 12. Comparison of operation times.

Figs. 12, 13, and 14 present the experiment results. Note that user interfaces of corresponding Web-based and Fragment-based Bricks were nearly the same, and the time required for manual operations were roughly equivalent. Thus, only the experiment results for Fragment-based Bricks are presented in Fig. 12. These results illustrate that CARSB is able to reduce the user operation time by 25%-51% by composing separate Apps within one. This would be particularly beneficial for light smartphone users or elder people, by allowing them to access all of the related functions without having to deal with multiple Apps simultaneously.

Figs. 13 and 14 present the total network transmission load for three atomic Service Bricks as well as the network transmission load for composite Service Bricks with Web-based and Fragment-based architectures. Our results demonstrate the following: (1)

by comparing atomic Service Brick (Fig. 13) and composite Service Brick (Fig. 14), the proposed Brick composition mechanism reduced network transmission load by approximately 50% when using Web-based Service Bricks. No obvious effects on network transmission load were observed when using Fragment-based Bricks. Due to the fact that all resources were already available on the local mobile device; and (2) by comparing Web-based Service Brick and Fragment-based Service Brick, the proposed Fragment-based Service Bricks enabled an enormous decrease in network transmission load of approximately 75% when using individual atomic Bricks and 50% when using composite Bricks. The strength of Web-based Service Brick is cross platforms whereas the advantage of Fragment-based Service Brick is superior performance.

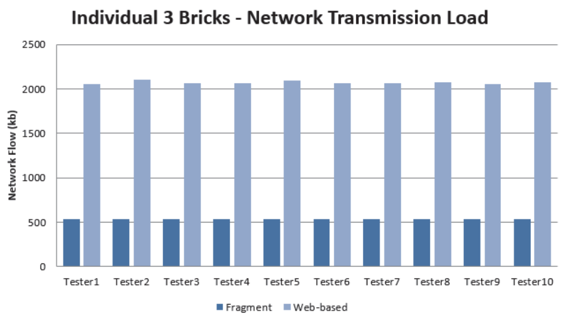


Fig. 13. Network transmission load associated with three atomic bricks.

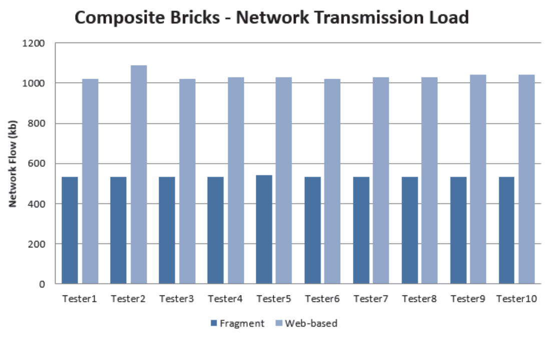


Fig. 14. Network transmission load associated with composite bricks.

5. CONCLUSIONS

In this paper, we introduce the concept of Service Bricks as well as the CARSB framework with Web-based software tool to facilitate the composition and delivery of Service Bricks and RESTful services. The proposed approach enables users to design composite Service Bricks comprising individual Service Bricks. Each Service Brick can send events to and receive events from an Event Engine, which can match and forward events to appropriate Bricks. In order to obtain desired services or information, Service Bricks can also be used for the invocation of resource providers, including local modules, atomic RESTful services, and stateful RESTful service flow. Experiment results demonstrate that the proposed CARSB approach is able to achieve a considerable decrease in user operation time and network transmission load.

The proposed system includes the following contributions:

- A loosely-coupled, component-based software framework that allows developers to build client-side Service Bricks, server-side RESTful services, and fully-fledged customized mobile applications.
- A Web-based software tool that allows ordinary users to compose customized mobile applications visually by integrating front-end UI components with backend RESTful services.
- Generation of mobile user interfaces ideally suited to the relatively small size of mobile phone screens through the application of UI layout patterns.
- Reduced network communication overhead by incorporating local event communication, caching local resources, and linking composite services in the cloud.

The next stage of CARSB will involve the design of data communication mechanism among multiple pages to enable the building of multiple-page Apps.

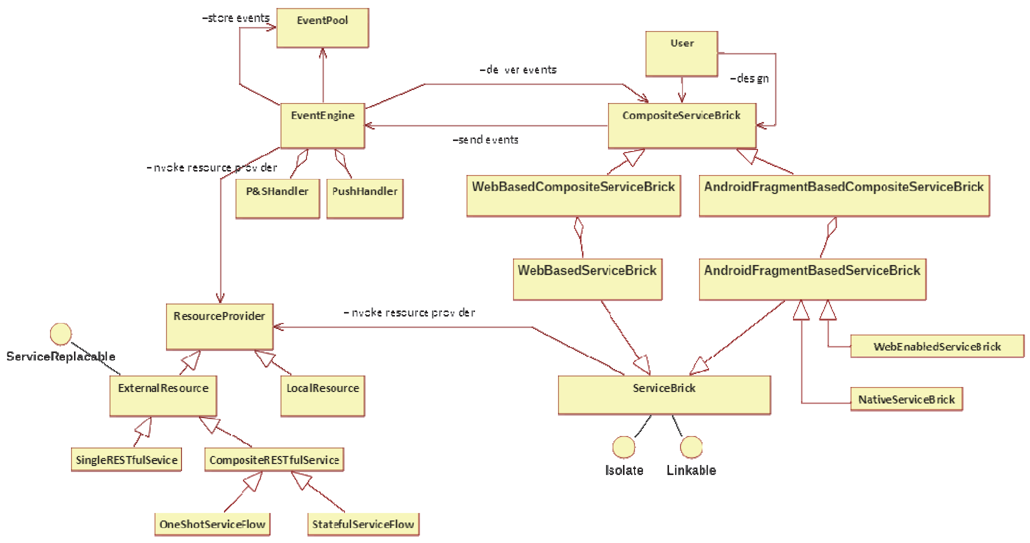


Fig. 15. Conceptual model of the CARSB approach.

REFERENCES

1. S. P. Ma, W.-T. Lee, P.-C. Chen, and C.-C. Li, "Framework for enhancing mobile availability of RESTful services – A connectivity-aware and risk-driven approach," *Mobile Networks and Applications*, Vol. 21, 2016, pp. 337-351.
2. A. Bozzon, M. Brambilla, F. M. Facca, and G. T. Carughu, "A conceptual modeling approach to business service mashup development," in *Proceedings of IEEE International Conference on Web Services*, 2009. p. 751-758.

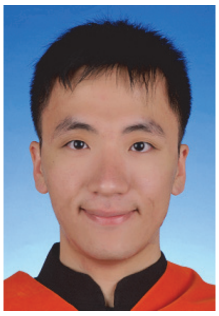
3. E. M. Maximilien, “Mobile mashups: Thoughts, directions, and challenges,” in *Proceedings of IEEE International Conference on Semantic Computing*, 2008, pp. 597-600.
4. S. Kaltofen, M. Milrad, and A. Kurti, “A cross-platform software system to create and deploy mobile mashups,” in *Proceedings of the 10th International Conference on Web Engineering*, 2010, pp. 518-521.
5. C. Pautasso, “RESTful web service composition with BPEL for REST,” *Data & Knowledge Engineering*, Vol. 68, 2009, pp. 851-866.
6. F. Rosenberg, F. Curbera, M. J. Duftler, and R. Khalaf, “Composing RESTful services and collaborative workflows: A lightweight approach,” *IEEE Internet Computing*, Vol. 12, 2008, pp. 24-31.
7. S.-P. Ma, J.-S. Jiang, and W.-T. Lee, “Service brick composition framework for smartphones,” in *Proceedings of the 20th IEEE Asia-Pacific Software Engineering Conference*, 2013, pp. 459-466.
8. JOpera.org, “JOpera: Process support for web services,” http://www.jopera.org.
9. R. Alarcon, E. Wilde, and J. Bellido, “Hypermedia-driven RESTful service composition,” in *Proceedings of International Workshop on Service-Oriented Computing/ServiceWave*, 2011, pp. 111-120.
10. C. Pautasso, BPEL for REST, in *Business Process Management*, M. Dumas, M. Reichert, and M.-C. Shan, ed., Springer Berlin Heidelberg, 2008, pp. 278-293.
11. C. Pautasso, “RESTful web service composition with BPEL for REST,” *Data Knowledge Engineering*, Vol. 68, 2009, pp. 851-866.
12. J. Yu, B. Benatallah, R. Saint-Paul, F. Casati, F. Daniel, and M. Matera, “A framework for rapid integration of presentation components,” in *Proceedings of the 16th ACM International Conference on World Wide Web*, 2007, pp. 923-932.
13. T. Nestler, L. Dannecker, and A. Pursche, “User-centric composition of service front-ends at the presentation layer,” in *Proceedings of International Workshop on Service-Oriented Computing/ServiceWave*, 2010, pp. 520-529.
14. S.-P. Ma, C.-Y. Huang, Y.-Y. Fanjiang, and J.-Y. Kuo, “Configurable RESTful service mashup: A process-data-widget approach,” *Applied Mathematics & Information Sciences*, Vol. 9, 2015, pp. 637-644.
15. R. Francese, M. Risi, and G. Tortora, “Management, sharing and reuse of service-based mobile applications,” in *Proceedings of the 2nd ACM International Conference on Mobile Software Engineering and Systems*, 2015, pp. 105-108.
16. D. Lee, S. Jeong, T. Jeong, J.-H. Yoo, and J. W.-K. Hong, “ICBMS SM: A smart Mediator for mashup service development.” in *Proceedings of the 18th Asia-Pacific Network Operations and Management Symposium*, 2016, pp. 1-6.
17. Z. Zhai, B. Cheng, Z. Wang, X. Liu, M. Liu, and J. Chen, “Design and implementation: the end user development ecosystem for cross-platform mobile applications, in *Proceedings of the 25th International Conference Companion on World Wide Web*, 2016, pp. 143-144.
18. S. P. Ma, Y.-S. Ma, and W.-T. Lee, “State-driven and brick-based mobile mashup,” in *Proceedings of IEEE International Conference on Mobile Services*, 2015, pp. 190-196.
19. J. Ribeiro and M. Carvalhais, “Web design patterns for mobile devices,” in *Proceedings of the 19th Conference on Pattern Languages of Programs*, 2012, pp. 1-48.

20. S.-P. Ma, Y.-Y. Fanjiang, and J.-Y. Kuo, “Dynamic service composition using core service identification,” *Journal of Information Science and Engineering*, Vol. 30, 2014, pp. 957-972.

Shang-Pin Ma (馬尚彬) received his Ph.D. degree in Computer Science and Information Engineering from National Central University, Taiwan, in 2007. Dr. Ma is currently an Associate Professor in the Department of Computer Science and Engineering at National Taiwan Ocean University. His research interests include service-oriented computing, software engineering, mobile computing, and semantic web.

Peng-Zhong Chen (陳鵬中) received his Bachelor degree (2015) from the Department of Computer Science and Engineering, National Taiwan Ocean University, Taiwan. Mr. Chen is currently a graduate student in National Taiwan Ocean University. His research interests include linked open data, software engineering, and service-oriented computing.

Yang-Sheng Ma (馬楊陞) received his Bachelor (2013) and Master’s (2015) degrees from the Department of Computer Science and Engineering, National Taiwan Ocean University, Taiwan. His research interests include mobile computing, component-based software engineering, and service-oriented computing.



Jheng-Shiun Jiang (江政勳) received his Bachelor (2011) and Master’s (2013) degrees from the Department of Computer Science and Engineering, National Taiwan Ocean University, Taiwan. His research interests include web information system, software engineering, and service-oriented computing.