出國報告（出國類別：國際會議）

MS2015 研討會出國報告

服務機關：國立高雄師範大學軟體工程系
姓名職稱：李文廷 助理教授
派赴國家：美國
出國期間：104/6/25~104/7/5
報告日期：104/8/28

摘要

IEEE 行動服務國際研討會(IEEE International Conference on Mobile Service) 是行動服務領域的重要國際研討會，此次出國目的為參與 2015 IEEE 行動服務國際研討會，發表論文並與會議學者討論研究議題和相關技術。今年度的會議於 2015 年 6 月 27 至 7 月 2 日於美國紐約曼哈頓 Millennium Broadway Hotel 舉行，會中有來自多國的與會學者共同參與交流，發表行動服務研究領域之論文。此次大會的議程長達六天，大會邀請了許多服務導向領域的重量級學者進行多場 Keynote Speech 以及 Panel Discussion，從這些演講與座談也獲得了不少的研究上的靈感。本人亦在會議中發表了一篇論文，並與國外的學者進行討論，討論之內容對於研究之進行與推展有相當之助益。

目次

MS2015 研討會出國報告

一、 目的

IEEE 行動服務國際研討會(IEEE International Conference on Mobile Service) 是行動服務領域的重要國際研討會，此次出國目的為參與 2015 IEEE 行動服務國際研討會(2015 IEEE International Conference on Mobile Service, MS2015)發表論文，論文題目為 State-Driven and Brick-Based Mobile Mashup，並與會議學者討論研究議題和相關技術。

二、 過程

IEEE International Conference on Mobile Service 是行動服務領域的重要國際研討會，與其他數個頂尖國際研討會，如 ICWS 2015 (International Conference on Web Services)、SCC 2015 (International Conference on Service Computing)、Cloud 2015、BigData Congress 2015 共同舉行，今年度的會議於 2015 年 6 月 27 至 7 月 2 日於美國紐約曼哈頓 Millennium Broadway Hotel 舉行，會中有來自多國的與會學者共同參與交流，發表 Mobile Service Personalization, Mobile Service Delivery, Mobile Service Framework, Mobile Service Security, Mobile Service Applications 等研究領域之論文。此次大會的議程長達六天，大會邀請了許多服務導向領域的重量級學者，如 NASA 的 Dr. Tsengdar J. Lee, Google 的 VP of Infrastructure at Google: Dr. Eric Brewer、IBM 的 VP of Cognitive Computing: Dr. Guruduth Banavar、Transactions of Service Computing 的主編：Prof. Ling Liu 等進行多場 Keynote Speech 以及 Panel Discussion，這些演講與座談均相當精采，令人獲益良多，從這些演講與座談也獲得了不少的研究上的靈感。大會於 7 月 2 日順利閉幕，結束了此次長達六天的 MS 2015 研討會。

三、 會議議程

2015 IEEE行動服務國際研討會之行動應用品質會議，兩場session分別安排於6月30日上午與下午，議場主題為行動應用品質(Mobile Application Quality,MAQ)，議程如下:

(一) 議程 1 (Session 1): Modeling and Development for Mobile Applications (06/30 Tuesday, 9:25-10:25) 議程主席(Session Chair): Shang-Pin Ma, National Taiwan Ocean University, Taiwan

1. 論文題目: State-Driven and Brick-Based Mobile Mashup (MS2015-3025)

   作者: *Shang-Pin Ma (National Taiwan Ocean University TW) Yang-Sheng Ma (National Taiwan Ocean*

University TW) ***Wen-Tin Lee (National Kaohsiung Normal University TW)***

2. 論文題目:The Study of Cloud-Based Testing Platform for Android (MS2015-3026)
   作者:*Jong Yih Kuo (National Taipei University of Technology TW) Wei Ting Yu (National Taipei University of TechnologyTW)*
3. 論文題目:Improving Resource Utilization of a Cloud-Based Testing Platform for Android Applications (MS2015-3027)
   作者:*Chien-Hung Liu (National Taipei University of Technology TW) Shu-Ling Chen (Southern Taiwan University ofScience and Technology TW) Woei-Kae Chen (National Taipei University of Technology TW)*

(二) 議程 2 (Session 2): Mobile Application Quality Assurance (06/30 Tuesday, 13:00-14:00; 4.04/4.05) 議程主席(Session Chair): Ci-Wei Lan, IBM, Taiwan

1. 論文題目: Code Coverage Measurement for Android Dynamic Analysis Tools (MS2015-3028)
   作者:*Chun-Ying Huang (National Taiwan Ocean University TW) Ching-Hsiang Chiu (National Taiwan Ocean University TW) Chih-Hung Lin (Institute for Information Industry TW) Han-Wei Tseng (National Taiwan Ocean University TW)*
2. 論文題目: Applying Genetic Programming for Time-aware Dynamic QoS Prediction (MS2015-3029)
   作者:*Yang Syu (National Taipei University of Technology TW) Yong-Yi Fanjiang (Fu Jen Catholic University TW Jong-Yih Kuo (National Taipei University of Technology TW) Shang-Pin Ma (National Taiwan Ocean University TW)*
3. 論文題目: A Study of a Life Logging Smartphone App and Its Power Consumption Observation in Location-based Service Scenario (MS2015-3030)
   作者:*Fu-Ming Huang (Academia Sinica TW) Yu Hsiang Huang (Academia Sinica TW) Christopher Szu (Academia SinicaChina) Addison Y.S. Su (National Central University TW) Meng Chang Chen (Academia Sinica TW) Yeali S. Sun (NationalTaiwan University TW)*

四、 報告內容

本人在 2015 行動服務國際研討會中上午的議程中發表了一篇論文，論文題目為 State-Driven and Brick-Based Mobile Mashup，此研究目標為提供一個可建構整合前端 UI 元件與後端服務元件的行動複合應用程式。圖一為本人在研討會報告的現況，報告時間 15 分鐘回答問題 10 分鐘，報告內容為建構整合前端 UI 元件與後端服務元件的行動複合應用程式，與會者也詢問是否可以適用此論文所開發的應用程式，討論熱絡反應良老。研討會當天報告論文內容如附加檔案。

圖一、研討會論文報告

圖二、研討會報告者合影

五、 心得及建議

本人此次相當榮幸獲邀在 MS 2015 中的 special track 擔任議程委員(Program Committee Member)，主題為 Mobile Application Quality (MAQ)，最後 MAQ 特別議程共收錄了六篇論文，分為兩場 session 進行簡報。大會將此兩場 session 分別安排於 6 月 30 日上午與下午，兩場均吸引了許多聽眾前來參與，會議中也就這些論文進行了深度的研

討。本人亦在上午的議程中發表了一篇論文，論文題目為 State-Driven and Brick-Based Mobile Mashup，此研究目標為提供一個可建構整合前端 UI 元件與後端服務元件的行動複合應用程式，在會議中也與國外的學者進行討論有關服務導向架構、網際服務和服務品質等相關最新研究內容與趨勢，討論之內容相當有助於本研究之推展。

六、 附錄

State-Driven and Brick-Based Mobile Mashup

Shang-Pin Ma and Yang-Sheng Ma
Department of Computer Science and Engineering
National Taiwan Ocean University
Keelung, Taiwan
E-mail: albert@ntou.edu.tw, 10257039@ntou.edu.tw

Wen-Tin Lee
Department of Software Engineering
National Kaohsiung Normal University
Kaohsiung, Taiwan
E-mail: wtlee@nknu.edu.tw

***Abstract*—Mobile applications (i.e., mobile apps or apps) are becoming an important software delivery model. Users can employ a wide range of services associated with mobile apps, such as entertainment, news, travel, and social networking. Unfortunately, the retrieval of information from multiple apps, services, or local resources can be time-consuming, costly, and inconvenient. This paper proposes a novel mobile mashup approach, referred to as brick-based, state-driven mobile service composition (BSMSC) to overcome these difficulties. BSMSC comprises two primary mechanisms: (1) Android-fragment-based service bricks; and (2) a state-driven linkage for composite RESTful services, which supports one-shot service flow execution as well as stateful service flow execution. The proposed BSMSC approach makes it possible to assemble fully-fledged, reconfigurable mobile mashup applications.**

Keywords-mobile mashup; service brick; RESTful service composition

I. INTRODUCTION

Mobile applications (i.e., mobile apps or apps) are becoming an important software delivery model. Users can employ a wide range of services associated with mobile apps, such as entertainment, news, travel, and social networking. However, the retrieval of information from multiple apps, services, or local resources can be time-consuming, costly, and inconvenient. Despite recent advances in service mashups, there remains very few configurable mashup mechanisms capable of combining information and services from front-end as well as back-end resources for mobile devices. Most solutions in the area of mobile mashups are based on mobile widget mechanisms, which require the installation of a widget engine or widget runtime [1-3]. Meanwhile, existing service composition methods do not consider the delivery in the mobile clients.

This paper presents a novel approach to mobile mashups, referred to as brick-based, state-driven mobile service composition (BSMSC), which features two primary mechanisms: Android-fragment-based service bricks and a state-driven linkage to composite RESTful services. For the front end of the mobile mashup, we extended our previous work [4], by developing front-end UI components based on Android fragment APIs and web technologies. This enables users to create composite “service bricks” (CSB). We adopted the popular [5] RESTful (Representational State Transfer) services for the back end. Based on the service composition platform JOpera [6], we devised a novel mechanism, referred to as state-driven composite RESTful service linkage, to enable the connection of CSB to back-end RESTful services (atomic or composite). The proposed mechanism supports one-shot service flow execution, in which multiple services can be invoked according to a specified flow sequence followed by the direct return of combined results. The proposed mechanism also supports stateful service flow execution, which enables the execution of conversational services, where the front-end CSB is able to issue requests iteratively to the same service flow in order to obtain service results based on flow states. The proposed BSMSC approach makes it possible to assemble fully-fledged, reconfigurable mobile mashup applications

The remainder of this paper is organized as follows: Section 2 presents a review of research related to mobile mashup and RESTful service composition. Section 3 presents the details of the proposed state-driven and brick-based approach. Illustrative examples are presented in Section 4. The final section presents our conclusions.

II. RELATED WORK

In this section, we review several studies related to mashups and RESTful service composition. Park [7] introduced a platform for the integration of widgets on the web. In that architecture, a web page comprises a number of widgets, which can be categorized as simple or complex. The widgets interact with each other using a mechanism called Publish and Subscribe. Nestler et al. [8] 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. Users can connect two service components simply by clicking their inputs and outputs, to simplify the creation of new service components. Ma et al. [9] proposed a REST-based service mashup framework, referred to as Process-Data-Widget (PAW), which functions as a composition model for the construction of mashup applications. That framework enables developers to design service processes, compose service data, and configure widgets for presentation on a user interface (UI) simply by constructing a mashup document (MD). The PAW mashup engine also parses the MD and generates a corresponding mashup application and associated RESTful services. Most of above efforts can only produce conventional mashup applications, not for the mobile environment. Conversely, our approach can realize mashups for mobile devices by applying the proposed service brick mechanism.

Rosenberg et al. [10] provided an extensible, XML-based language, Bite, in conjunction with an integrated programming model for the composition of RESTful services with interactive flow. Alarcon et al. [11] designed a hypermedia-centric REST service description, ReLL (Resource Linking Language), in conjunction with Petri Nets for the modelling and simulation of service compositions. Zhao and Dosh [12] introduced three types of RESTful service: resource set service, individual resource service, and transitional service, which are presented within a situation calculus based on sate transition systems (STS) to automate service composition. Pautasso [13, 14] introduced a means by which to extend BPEL (Business Process Execution Language) to invoke RESTful WSs (Web Services) and publish a BPEL process as a RESTful Web Service. In [15], Pautasso and Wilde proposed an architecture for push-enabling RESTful business processes. They also designed an engine, Push-enable RESTful Process Execution Engine, to enable push notification of tasks and the detection of changes in process state. In [16], Aghaee and Pautasso proposed a Mashup Component Description Language (MCDL), which is a domain-specific language based on JSON (JavaScript Object Notation) to describe heterogeneous mashup components based on various access methods, such as POX (Plain Old XML), REST, or SOAP (Simple Object Access Protocol). In [17], Bellido et al. proposed a set of control-flow patterns (include sequence, iteration, alternative, and parallel) within the context of stateless compositions of RESTful services. All above researches are back-end RESTful composition methods, without considering the client side, especially for the mobile clients. Besides, these works do not fully realize state-based service flow execution, which is an important feature for the mobile composite applications.

JOpera [6] is a widely-used service composition tool based on a range of theoretical methods [13-17]. JOpera provides a visual language with which to define control flow and data flow for service processes as well as an execution engine. JOpera supports numerous adapters with which to invoke various programming languages or services, such as Java and JavaScript as well as SOAP and RESTful services. The method proposed in this study for the composition of RESTful services is built atop JOpera through the provision of a set of utility services and a service mediator to facilitate communication between the mobile app and the JOpera server.

III. BSMSC: State-driven Mobile Service Composition

In this section, we outline the proposed approach in detail, including the system architecture, the newly-devised Android-fragment-based composite service brick, a novel means of linking state-driven composite RESTful services, the role of users in the development of brick-based applications, and the JSON-based description document for composite RESTful services.

A. System Architecture

Fig. 1 illustrates the proposed system architecture including the basic schema of the proposed approach. This approach allows users to design a composite service brick (CSB), comprising multiple service bricks used for the display of integrated services or information. A service brick (SB) is a rectangular UI component in an app used for the display of specific information. We extended our previous work [4] with the addition of a new type of service brick: Android-fragment-based service bricks, which can be further divided into web-enabled service bricks (using HTML, Javascript, and CSS) as well as native service bricks. The native Android code makes the implementation of these bricks smoother and more efficient. In addition, BSMSC is able to present a composite service brick as a mobile app, which means that users need not refer to a specific URL in order to utilize web-based service bricks via their browser. Furthermore, source files, such as JSON-based description documents, HTML pages, and Javascript source code, can be stored on the client-side to decrease network flow requirements.

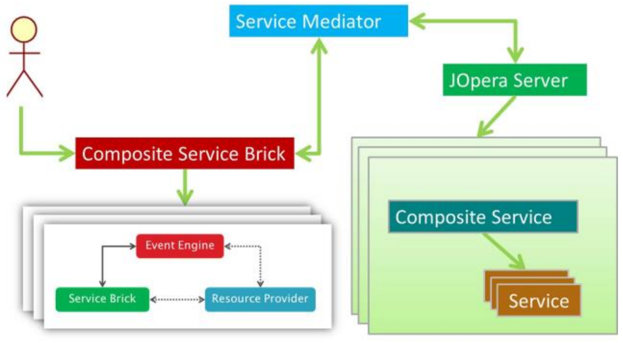


Fig. 1. System Architecture

A service brick can send to and receive events from the event engine, which is responsible for matching and forwarding events to appropriate service bricks. In addition, service bricks can be used to invoke resource providers to obtain services or information. Resource providers include client-side local resources, such as an address book or GPS module, as well as external resources, such as the atomic and composite RESTful services introduced in this paper. Composite RESTful services are hosted in the JOpera server. The service mediator is used by the composite service brick to link JOpera services in order to invoke RESTful service flow.

In the following sub-sections, we explain the above concepts in greater detail.

B. Android-Fragment-Based Composite Service Brick

As mentioned previously, this study developed a new type of service brick, the Android-fragment-based service brick, to improve its efficiency and usability. We also devised two sub-types of Android-fragment-based service bricks:

(1) Web-enabled service bricks are created using HTML, CSS, or Javascript prior to implementation in Android WebView. All resources, such as source files for HTML, CSS, and Javascript, and image files, are cached on

the client side; i.e., mobile devices. Thus, unlike web-based service bricks [4], the proposed mechanism does not require the loading of resources from the server, which enhances overall efficiency.

(2) Native service bricks are created using pure Android native objects 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 functions. For example, native service bricks are able to use the current geospatial location to obtain weather-related data, or use heart rate sensors or pedometers for the monitoring of exercise performance.

This study extended these bricks in the construction of front-end UI components based on Android fragment APIs and web technology.

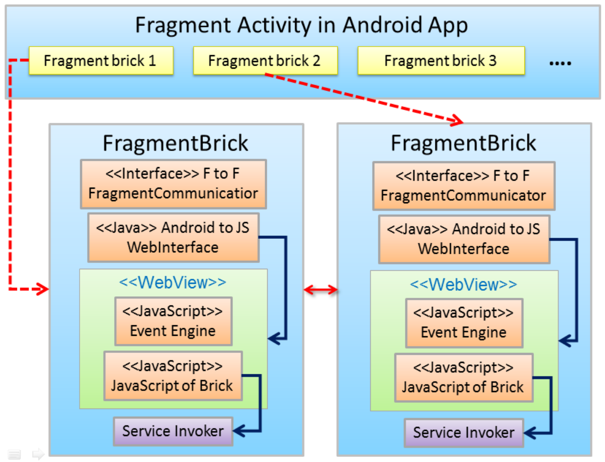

Fragment Activity in Android App
Fragment brick 1
Fragment brick 2
Fragment brick 3
....
FragmentBrick
<<Interface>> F to F FragmentCommunicator
<<Java>> Android to JS WebInterface
<<WebView>>
<<JavaScript>> Event Engine
<<JavaScript>> JavaScript of Brick
Service Invoker
FragmentBrick
<<Interface>> F to F FragmentCommunicator
<<Java>> Android to JS WebInterface
<<WebView>>
<<JavaScript>> Event Engine
<<JavaScript>> JavaScript of Brick
Service Invoker

Fig. 2. Architecture used in Android-fragment-based composite service bricks

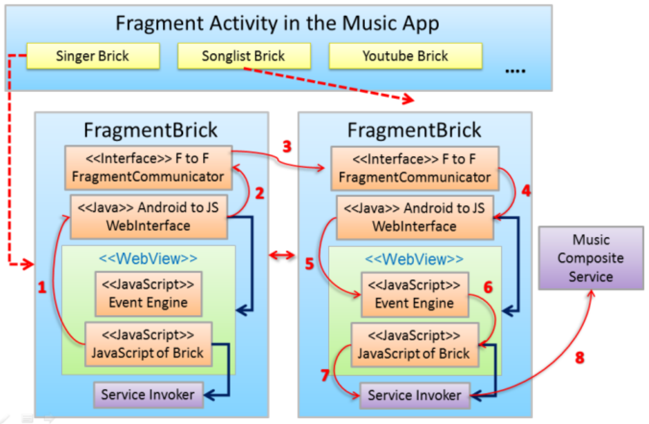

Fragment Activity in the Music App
Singer Brick
Songlist Brick
Youtube Brick
....
FragmentBrick
<<Interface>> F to F FragmentCommunicator
<<Java>> Android to JS WebInterface
<<WebView>>
<<JavaScript>> Event Engine
<<JavaScript>> JavaScript of Brick
Service Invoker
FragmentBrick
<<Interface>> F to F FragmentCommunicator
<<Java>> Android to JS WebInterface
<<WebView>>
<<JavaScript>> Event Engine
<<JavaScript>> JavaScript of Brick
Service Invoker
Music Composite Service
1
2
3
4
5
6
7
8

Fig. 3. Example of Android-fragment-based composite service bricks

Fig. 2 presents the architecture of Android-fragment-based service bricks. An Android fragment represents only one service brick, which can be embedded into WebView or another view in Android Layout. In a fragment-based brick, the "FragmentCommunicator" interface is responsible for sending and receiving data between its parent fragment and other fragments. The "WebInterface" object is responsible for communication between native Android objects and Javascript in the WebView-embedded service brick, thereby making it possible to invoke Javascript functions in WebView. WebView includes the following two main components: an event engine, which receives and transmits events among service bricks, and the Javascript in a brick, which is used to display the desired user interface. As mentioned previously, web-based service bricks can be implemented in the WebView of android-fragment-based bricks to provide the same functionalities as the original web-based service brick.

Fig. 3 presents an example android-fragment-based composite service brick of a music composite application. First, the singer service brick (SB) uses JavaScript to send an event to its WebInterface object (step 1). Next, the WebInterface object forwards the event to the FragmentCommunicator interface (step 2). In step 3 the FragmentCommunicator interface of the singer SB transmits the event to the songlist SB. In steps 4 and 5, the FragmentCommunicator interface of songlist SB submits the event to its event engine via its WebInterface object. Finally, in step 6, the event engine checks the event type and forwards it to the Javascript functions in the songlist SB. Finally (in steps 7 and 8), the songlist SB utilizes the service invoker to connect to the music composite service in order to display the song list that the singer submitted from the singer SB.

C. State-Driven Composite RESTful Service Linkage

To strengthen the capability of the service bricks, we also devised a mechanism for linking to back-end RESTful services, referred to as state-driven composite RESTful service linkage. The proposed mechanism is able to bridge front-end service bricks and back-end single or composite RESTful services in a state-driven manner; i.e., service data can be stored and processed during the conversation between the front-end bricks and the back-end services.

We adopted the widely-used JOpera [6] service process engine as our underlying platform. In addition to performing one-shot service delivery for the composite service, supporting state-enabled service conversation between front-end bricks and back-end services is the first requirement of BSMSC. Second, dynamic substitution of component services in a composition [18] is also required for the composition of dynamic services; therefore, we also integrated this feature in order to allow the assignment of preferred component services in the composite brick development phase. Third, publishing the composite RESTful service as a normal RESTful service with JSON output is required to ease integration. However, a composite service in JOpera must be output in a specific format. Transforming the service output into a validate JSON format is another issue. Thus, we devised a service mediator as an intermediate layer between the bricks and the JOpera, to deal with output wrapping and state analysis and integrate two utility services (a component service substitution module and a session management module), into the service flow. It should be noted that the developer of the composite RESTful should embed the two utility services into the service flow.

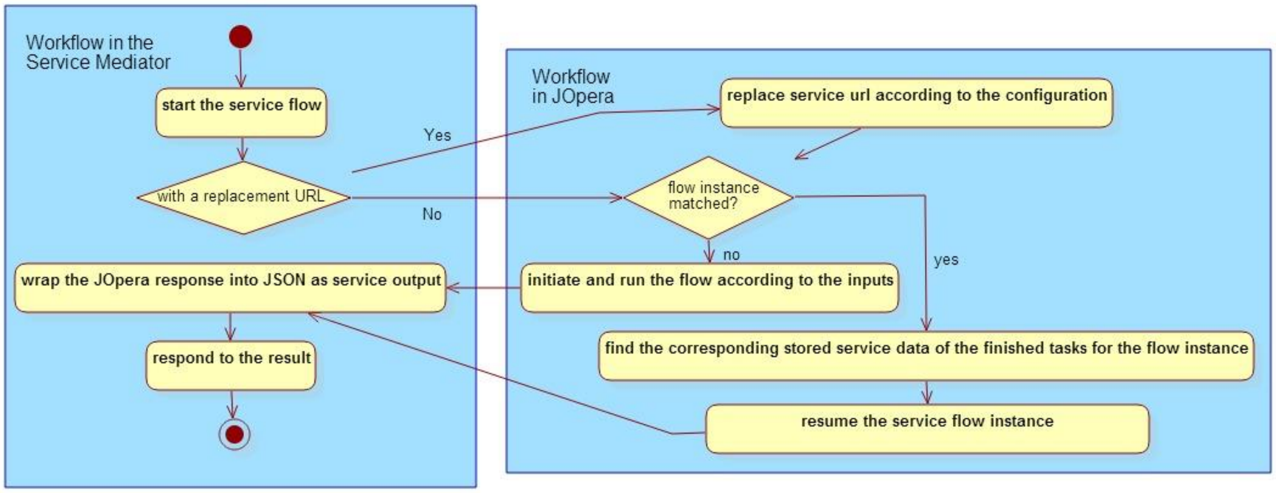

Workflow in the Service Mediator
start the service flow
with a replacement URL
Yes
No
wrap the JOpera response into JSON as service output
respond to the result
Workflow in JOpera
replace service url according to the configuration
flow instance matched?
no
yes
initiate and run the flow according to the inputs
find the corresponding stored service data of the finished tasks for the flow instance
resume the service flow instance

Fig. 4. Workflow of stateful RESTful service composition

Fig. 4 illustrates the system process of the service mediator. First, the service mediator invokes the composite services of JOpera. Then, if the user assigns a new URL to replace a component service in the service brick description, then the service substitution module replaces the component service according to the new configuration. Next, the service mediator checks the flow state to determine whether the service request is the first one in the conversion between the front-end composite brick and service flow. If it is the first service request in the conversation (i.e., no matching instance of service flow is available for the session ID), then the service mediator transfers control to JOpera to implement service flow according to inputs from the service brick. During the execution of service flow, all service input data and output data is maintained by the session management module. In cases where there is a matching instance of service flow, then the session management module retrieves the corresponding stored service data for the client based on the session ID. The stored service data is then used to resume that procedure. Finally, the service mediator wraps the JOpera response into validate JSON format and responds to the client with the results.

Briefly, BSMSC provides four main functionalities associated with the linkage of composite RESTful services:

(1) *One-Shot Service Flow Execution*

Composite services are able to invoke several RESTful services in the specified sequence, and then generate an integrated result following service execution. The results are displayed in a specific service brick.

(2) *Stateful Service Flow Execution*

BSMSC also supports conversational composite services by allowing multiple service bricks in a CSB to connect to the same composite service but display different contents related to the state of the service conversation retrieved in different phases of the service flow.

(3) *Service Substitution*

Service substitution allows users to substitute binding component services for composite services. Note that the substitute service must be interface-compliant with the original component service. For example, the user can simply change the YouTube service embedded in the video service brick to DailyMotion.

(4) *Format Wrapping for Service Output*

The service mediator is able to reformat the service outputs produced by JOpera from the JOpera-specific format into the widely used JSON. The front-end service bricks then use composite RESTful services as atomic RESTful services. The service mediator also solves other minor issues related to the connection of the JOpera server, such as supporting Unicode encoding and overcoming limitations in cross-domain requests for RESTful services.

D. Role of the User in Development of Brick-Based Applications

In this section, we introduce the three roles assumed by users in the proposed approach:

(1) *Composite RESTful Service Developer (CRSD)*

Composite RESTful Service Developers (CRSDs) must be familiar with the use of Eclipse and the JOpera plugin in order to develop composite RESTful services. The CRSD must be able to provide a description document in JSON format for the modelling of composite RESTful services for subsequent processing by BSMSC. Specifically, the CRSD must begin by making a new JOpera project for the development of an OML file, which is the specialized file format in JOpera. Second, the CRSD can then use the JOpera design tool to visually create a service flow, which contains the control flow and data flow required for the composition of multiple RESful services. Finally, the CRSD initiates the JOpera server and tests to determine whether the composite RESTful service has been successfully deployed on the JOpera server.

(2) *Service Brick Developer (SBD)*

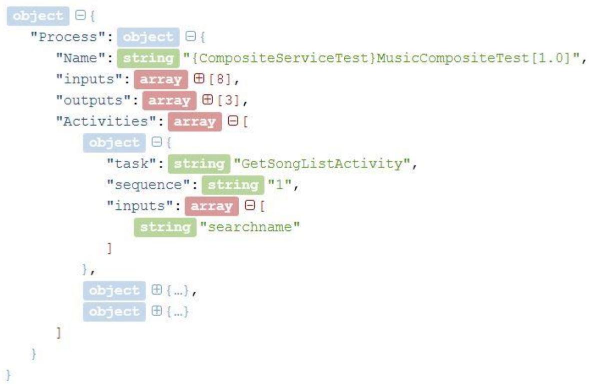
A service brick developer (SBD) oversees the development of service bricks, including all resources, such as the source files of HTML, CSS, and Javascript as well as image/video files. The SBD can also upload the service brick in the form of a zip file via our developed CSB design tool [4], whereupon a preview can be reviewed and the service brick tested.

(3) *Composite service brick developer (CSBD)*

A composite service brick developer (CSBD) functions as a developer as well as an end user of composite service bricks. The CSBD uses the visual design tool to construct a CSB tp connect back-end RESTful services, preview the CSB via the CSB design tool [4], and use the CSB directly.

E. JSON-based descriptions

In this section, we outline the specifications of descriptive documents for the RESTful composite service (as shown in Fig. 5), which are provided by the CRSD. Descriptive documents are in JSON format rather than XML to reduce size and ease the difficulties associated with parsing and processing.



```
object ⊟{
  "Process": object ⊟{
    "Name": string "{CompositeServiceTest}MusicCompositeTest[1.0]",
    "inputs": array ⊞[8],
    "outputs": array ⊞[3],
    "Activities": array ⊟[
      object ⊟{
        "task": string "GetSongListActivity",
        "sequence": string "1",
        "inputs": array ⊟[
          string "searchname"
        ]
      },
      object ⊞{…},
      object ⊞{…}
    ]
  }
}
```

Fig. 5. Description of RESTful Composite Service

Descriptions of RESTful Composite Services include the following:

- Name: name of the RESTful composite service
- Inputs: inputs of the RESTful composite service
- Outputs: outputs of the RESTful composite service
- Activities: component services in the RESTful composite service
  - task: the name of the RESTful component service in the composite service.
  - sequence: the sequence in which component services are executed. For example, if there are three activities in the service flow to be executed sequentially, then the sequence properties of the three services are 1, 2, and 3, respectively.
  - inputs: inputs of the activity.

The specifications of description documents for simple service bricks and composite service bricks are presented in [4]. The specifications of simple service bricks are designated by the SBD and automatically generated by the proposed design tool. Composite service bricks designed by the CSBD are also generated by the CSB design tool.

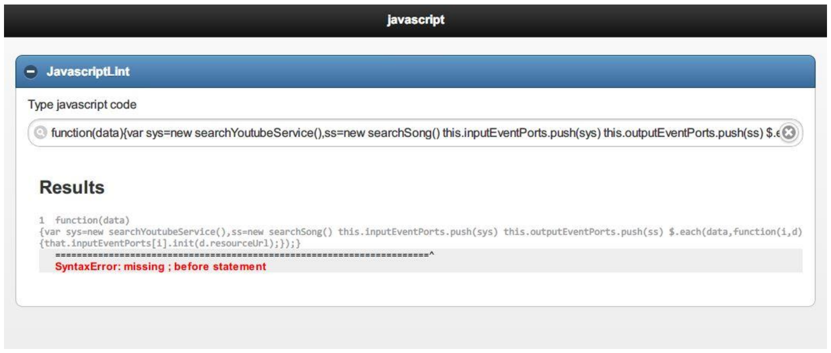
IV. Illustrative Examples

In this section, we evaluate the proposed method using three examples to illustrate differences in the execution flow of composite applications.

A. Javascript minifying and checking for one-shot service flow execution

The first example is a service brick for Javascript minifying and checking, as shown in Fig. 6. The service brick connects a back-end service flow used for the sequential execution of two RESTful services: Javascript minifying and Javascript checking services. This service brick enables the user to input a piece of Javascript code and then obtain minified, validated Javascript code that has been checked for mistakes.

From the view of responsibilities of three user roles, the CRSD first prepares two component RESTful services: Javascript minifying and Javascript checking, and compose them into a composite RESTful service, which can invoke these two services in sequence. The SBD develops an Android-fragment-based service brick to accept the user input, link the backend composite service, and display the service results. The CSBD, i.e. the end user, uses the CSB design tool to choose the above service brick for automatically generating his own brick-based mobile application.



```
function(data){var sys=new searchYoutubeService(),ss=new searchSong() this.inputEventPorts.push(sys) this.outputEventPorts.push(ss) $.e
```

Results

```
function(data)
{var sys=new searchYoutubeService(),ss=new searchSong() this.inputEventPorts.push(sys) this.outputEventPorts.push(ss) $.each(data,function(i,d)
{that.inputEventPorts[i].init(d.resourceUrl);});}
   ====================================================================^
   SyntaxError: missing ; before statement
```

Fig. 6. Demonstration of one-shot service flow

B. *Composite Service Brick for Music using Stateful Service Flow Execution*

The second example is a composite service brick designed to provide music-related services in order to demonstrate the execution of a stateful service flow.

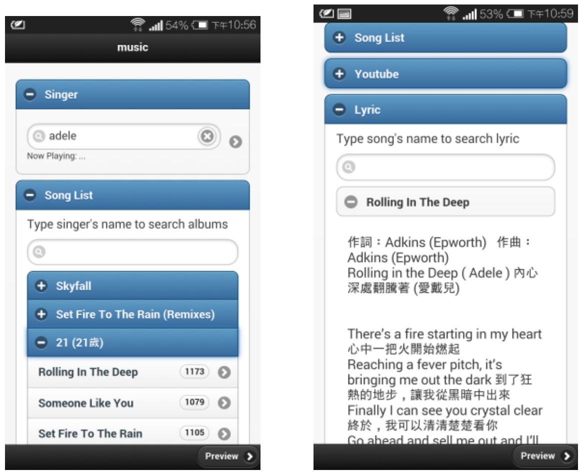
(1) The singer service brick transmits the name of a singer to the service mediator to check the flow state of the service on the JOpera Server, according to the session ID. Initially, no matching flow instance exists, which means that the service brick has not previously communicated with the compsosite service. In this situation, the service mediator calls the first service (get-album-list) in the service flow.

(2) When the service mediator obtains a response from the first service, it throws the result to the event engine at the front end, which matches and forwards events to a service brick which accepts this type of event. In this case, the appropriate service brick is the songlist service brick. The songlist service brick then displays a list of albums by the specified singer.

(3) When the user clicks an album, the songlist service brick sends the album ID to the service mediator. According to the session ID, the service mediator checks the state of that service flow instance. Because the flow instance is matched, the service mediator calls up the get-album-songs service. Note that the get-album-songs service requires the name of the album and its artist as inputs in order to retrieve the songs from the cached session data. The songlist brick used to display the list of songs on an album is presented on the left side of Fig. 7.

(4) When the user clicks the name of a song, the songlist service brick sends the name to the service mediator. As above, the service mediator determines which service brick to call. In this case according to the session ID, the lyric service brick is what is required, as shown on the right side of Fig. 7. The lyric service requires the name of the singer, name of the album, and name of the song as inputs; however, only the name of the song is required at this point because the other data have already been stored in the session management module.

From the view of responsibilities of user roles, the CRSD first prepares five component RESTful services: singer, get-album-list, get-album-songs, lyric, and video (YouTube), and compose them into a session-enabled composite RESTful service, which can invoke these five services based on the state of the service flow. The SBD develops four Android-fragment-based service bricks: singer brick, song list brick, lyric brick, and video brick, to accept the user input, link the backend composite service, and display the service results. The CSBD uses the CSB design tool to choose preferred service bricks for automatically generating his own music mobile application. In other words, the CSBD can use all four service bricks in his app or select less service bricks based on his requirements.



作詞：Adkins (Epworth) 作曲：Adkins (Epworth)
Rolling in the Deep (Adele) 內心深處翻騰著 (愛戴兒)

There's a fire starting in my heart
心中一把火開始燃起
Reaching a fever pitch, it's bringing me out the dark 到了狂熱的地步，讓我從黑暗中出來
Finally I can see you crystal clear
終於，我可以清清楚楚看你
Go ahead and sell me out and I'll

Fig. 7. Demonstration of stateful service conversation

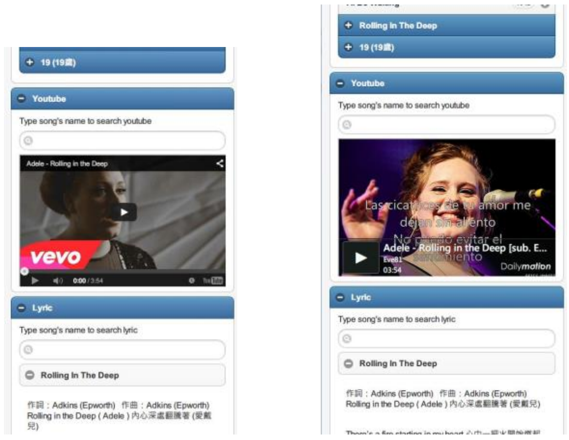

19 (19歲)
Youtube
Type song's name to search youtube
Adele - Rolling in the Deep
vevo
Lyric
Type song's name to search lyric
Rolling In The Deep
作詞：Adkins (Epworth) 作曲：Adkins (Epworth)
Rolling in the Deep (Adele) 內心深處翻騰著 (愛戴兒)
Rolling In The Deep
19 (19歲)
Youtube
Type song's name to search youtube
Adele - Rolling in the Deep [sub. E...
03:54
Dailymotion
Lyric
Type song's name to search lyric
Rolling In The Deep
作詞：Adkins (Epworth) 作曲：Adkins (Epworth)
Rolling in the Deep (Adele) 內心深處翻騰著 (愛戴兒)

Fig. 8. Illustration of service replacement

C. *Service Replacement using the YouTube service*

The third example is a video service brick intended to illustrate service replacement. The original video service brick connects to Youtube to show video clips; however, when the user assigns a replacement URL to the DailyMotion service, the original URL is replaced in the JOpera work flow. The left side of Fig. 8 presents the original YouTube service and the right side of Fig. 8 presents the new DailyMotion service.

V. CONCLUSIONS

This paper presents a novel mobile mashup approach, referred to as Brick-based and State-driven Mobile Service Composition (BSMSC), which features two main mechanisms: (1) Android-fragment-based service bricks; and (2) state-driven linkage to composite RESTful services. Three illustrative examples in two domains are also presented in this paper to demonstrate the feasibility of the proposed approach.

The contributions of the proposed approach include: (1) a loosely-coupled, component-based software framework is provided to allow developers build client-side service bricks, server-side RESTful services, or fully-fledged reconfigurable mobile applications; and (2) a mobile mashup tool is furnished to let end users visually compose their customized mobile applications to ease further use.

Acknowledgment

This research was sponsored by Ministry of Science and Technology in Taiwan under the grant MOST 103-2221-E-019-039.

References

[1] R. Zhou, H. Meng, X. Liu, S. Hu, J. Li, J. Rao, *et al.*, "Design and implementation of mobile widget composition framework and tool for end-user," in *2012 8th International Conference on Computing Technology and Information Management (ICCM)*, 2012, pp. 767-770.

[2] S. Kaltofen, M. Milrad, and A. Kurti, "A cross-platform software system to create and deploy mobile mashups," presented at the 10th international conference on Web engineering, Vienna, Austria, 2010.

[3] E. M. Maximilien, "Mobile Mashups: Thoughts, Directions, and Challenges," presented at the 2008 IEEE International Conference on Semantic Computing, 2008.

[4] S.-P. Ma, J.-S. Jiang, and W.-T. Lee, "Service Brick Composition Framework for Smartphones," in *20th Asia-Pacific Software Engineering Conference (APSEC 2013)*, 2013, pp. 459-466.

[5] R. T. Fielding, D. Software, and R. N. Taylor, "Principled Design of the Modern Web Architecture," *ACM Transactions on Internet Technology,* vol. 2, pp. 115--150, 2002.

[6] JOpera.org. *JOpera: Process Support for Web Services*. Available: http://www.jopera.org

[7] I. B. Park, "Envoy: A Platform for Cooperating Widgets on the Web," 2009.

[8] T. Nestler, L. Dannecker, and A. Pursche, "User-centric composition of service front-ends at the presentation layer," in *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops*, 2010, pp. 520-529.

[9] 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 (AMIS),* vol. 9, pp. 637-644, 2015.

[10] F. Rosenberg, F. Curbera, M. J. Duftler, and R. Khalaf, "Composing RESTful Services and Collaborative Workflows: A Lightweight Approach," *IEEE Internet Computing,* vol. 12, pp. 24-31, 2008.

[11] R. Alarcon, E. Wilde, and J. Bellido, "Hypermedia-driven RESTful service composition," in *Service-Oriented Computing*, ed: Springer, 2011, pp. 111-120.

[12] H. Zhao and P. Doshi, "Towards Automated RESTful Web Service Composition," presented at the Proceedings of the 2009 IEEE International Conference on Web Services, 2009.

[13] C. Pautasso, "BPEL for REST," in *Business Process Management*. vol. 5240, M. Dumas, M. Reichert, and M.-C. Shan, Eds., ed: Springer Berlin Heidelberg, 2008, pp. 278-293.

[14] C. Pautasso, "RESTful Web service composition with BPEL for REST," *Data Knowledge Engineering,* vol. 68, pp. 851-866, 2009.

[15] C. Pautasso and E. Wilde, "Push-Enabling RESTful business processes," presented at the Proceedings of the 9th international conference on Service-Oriented Computing, Paphos, Cyprus, 2011.

[16] S. Aghaee and C. Pautasso, "The mashup component description language," in *Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services*, 2011, pp. 311-316.

[17] J. Bellido, R. Alarc, #243, and C. Pautasso, "Control-Flow Patterns for Decentralized RESTful Service Composition," *ACM Transactions on the Web*, vol. 8, pp. 1-30, 2013.

[18] S.-P. Ma, Y.-Y. Fanjiang, and J.-Y. Kuo, "Dynamic Service Composition Using Core Service Identification," *Journal of Information Science and Engineering (JISE),* vol. 30, pp. 957-972, 2014.