JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 37, 123-138 (2021)
DOI: 10.6688/JISE.202101_37(1).0009

Cache-Enabled and Context-Aware Approach to Building Composite Mobile Apps*

SHANG-PIN MA[1], CHI-CHIA LI[1], SHIN-JIE LEE[2], HSI-MIN CHEN[3] AND WEN-TIN LEE[4]
[1]*Department of Computer Science and Engineering*
National Taiwan Ocean University
Keelung, 202 Taiwan
E-mail: albert@ntou.edu.tw
[2]*Department of Information Engineering and Computer Science*
National Cheng Kung University
Tainan, 701 Taiwan
[3]*Department of Information Engineering and Computer Science*
Feng Chia University
Taichung, 407 Taiwan
[4]*Department of Software Engineering and Management*
National Kaohsiung Normal University
Kaohsiung, 802 Taiwan

Mobile Applications (Mobile Apps or Apps) are an important software delivery model for the composition of front-end user interfaces (UIs) and back-end services in the cloud. However, variations in wireless network conditions can undermine the stability of Mobile Apps. Furthermore, developers face numerous difficulties in customizing Apps based on user preferences. In this paper, we propose the MASA (Mobile Application Slice Architecture), to address the above issues from the viewpoint of reusable software components. MASA includes three main features: (1) a programming model (called MAS) for building cross-platform UI components to facilitate the creation of Mobile Apps; (2) a broadcast mechanism to facilitate the exchange of data among MAS components; and (3) a rule-based and context-aware service prefetch and caching mechanism to ensure uninterrupted and partial offline access to RESTful services. A web-based software tool, MASA Portal, was also developed to assist users in the publication, discovery, composition, and consumption of composite MAS. Quantitative experiment results demonstrate that MASA is able to shorten service response times when using the proposed service prefetch function in various contexts.

Keywords: mobile application, mobile service composition, service cache, service prefetch, mobile application slice architecture

1. INTRODUCTION

Mobile Applications (Mobile Apps or Apps) are an important software delivery model [1] for the composition of front-end user interfaces (UIs) and back-end RESTful services (REST: Representational State Transfer) in the cloud [2]. Millions of Mobile Apps are currently available; however, there are few reusable composable models for the creation of Mobile Apps [3]. Developers face numerous difficulties when seeking to combine existing Apps, and users are unable to customize Apps due to differences among mobile platforms [4]. The challenge in overcoming these difficulties is the development of a com-

Received November 20, 2018; revised March 2, 2019; accepted June 4, 2019.
Communicated by Hung-Yu Kao.
* This research was sponsored by the Ministry of Science and Technology in Taiwan under grants MOST 105-2221-E-019-054-MY3 and 108-2221-E-019-026-MY3.

ponent architecture suitable for the creation of cross-platform Mobile Apps. Meanwhile, the stability of Mobile Apps can also be affected by wireless network conditions. Service caching, a technology of service-oriented computing (SOC) [5], can be used to enhance the availability and usability of services [6]; however, developing the means to apply service caching under varying network conditions poses additional challenges. Several software frameworks [7, 8] have been proposed for the development of Mobile Apps; however, none of these systems provide a mechanism by which to build composable units, or they do not support the invocation and caching required for RESTful services in the cloud.

In this paper, we propose a component architecture, referred to as MASA (Mobile Application Slice Architecture), for the creation of Apps based on MAS (Mobile Application Slice) components. MASA focuses on three important issues: (1) cross-platform compatibility; (2) automatic component integration; and (3) efficient service caching to shorten response time and allow offline use. Accordingly, MASA offers three main features: (1) a cross-platform programming model for building MAS components based on hybrid mobile web techniques. MASA leverages Cordova[1] to produce installable Mobile Apps [9]; (2) a channel-based "Broadcast" mechanism with associated APIs to facilitate the exchange of data among multiple MAS components to integrate MAS components as a composite (Composite MAS, or CMAS shortly); and (3) a relational and context-aware service prefetching and caching mechanism with associated APIs to allow quick online and uninterrupted offline access to RESTful services under unstable connections. Based on the concept of SOA (service-oriented architecture), we also developed a web-based software tool, called MASA Portal, to assist MAS providers and CMAS users in the publication, discovery, and composition of MAS components, as well as the consumption of CMAS applications. Fig. 1 illustrates the conceptual architecture of the proposed MASA system.

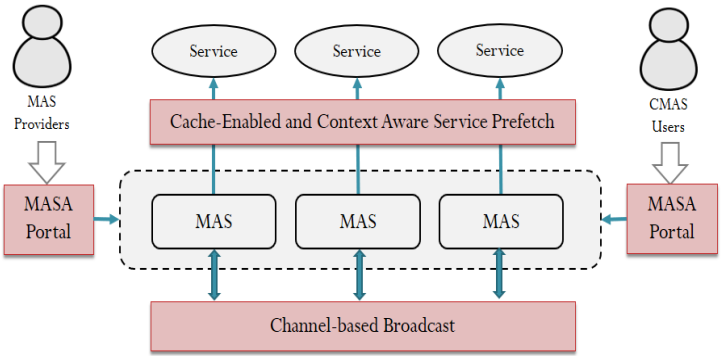


Fig. 1. Overview of the MASA approach.

The remainder of this paper is organized as follows: Section 2 introduces the background of MASA, as well as a review of extant research related to the software architectures used in mobile computing. Section 3 outlines the details of our approach. Section 4 presents the implementation of a MASA prototype system, called MASA Portal. Section 5 presents the experiment results. Conclusions are drawn in the final section.

[1] https://cordova.apache.org/

2. RELATED WORK

2.1 Service Cache and Prefetch

Fernandez *et al.* [10] developed a service caching system for SOAP services. The system uses two types of caching: strong and weak. Strong caching takes the freshest response message from the service provider to ensure that the cache is always fresh. Weak caching allows the client-side to decide how long cached data remains alive by designating an expiry point. In our work, we allow the developers to select the cache lifetime for services associated with a Mobile App. Papageorgiou [11] reported that SOAP services are unsuitable for mobile devices due to their heavy XML payload. They presented a cache model called PCV (Proxied Call with Validity check) to reduce the amount of data transmitted wirelessly between the client-side and the service-side. Liu and Deters [12] developed a dual-caching system for PDAs to enable caching on the client-side as well as the provider-side, both of which have a prefetching component. The client-side predicts which responses may be used, and set up is based on the BPEL (Business Process Execution Language). The server-side decides which cached data should be kept. Liyanaarachchi and Weerawaranl [13] proposed an end-to-end caching scheme based on the work of Fernandez. They devised a set of SOAP headers to realize cache functionality for SOAP services. The above systems and models were designed only for SOAP services, whereas our approach concentrates on popular RESTful services.

For the service caching and prefetch of RESTful services, Spillner *et al.* [14] proposed a service invocation method called RAFT-REST. RAFT-REST uses the cache mechanism to respond to the service request, tests the condition of network to reduce the unnecessary service invocations, and utilizes the request queue to collect all outgoing but failed request and to re-invoke these services. Ollite and Mohamudally [15] devised a 2-tier caching proxy system with one component implemented on the client device (mobile devices) and a server component to be used by the RESTful service provider. The performance evaluation showed a performance (response time) gain of up to 59% could be achieved. The above two methods did not consider the various context of mobile devices.

2.2 Software Architectures in Mobile Computing

Sanaei *et al.* [4] sought to define Mobile Cloud Computing (MCC) and discuss heterogeneity in convergent computing; *i.e.*, mobile computing and cloud computing. They reported that the issue of application fragmentation differs in web, online, and native applications. They suggested that designers use industrial toolkits, such as PhoneGap, Marmalade, Appcelerator, or Titanium to auto-generate cross-platform Mobile Apps to alleviate the problem of fragmentation. Heitkötter *et al.* [9] proposed several criteria for the development of mobile applications: (1) license and costs; (2) supported platforms; (3) access to advanced device-specific features; (4) long-term feasibility; (5) look and feel; (6) application speed; and (7) distribution. They identified four types of application (web, native, PhoneGap and Titanium) to enable comparisons under the proposed criteria. They reported that PhoneGap is able to build Mobile Apps with a look and feel similar to their native counterparts. In our work, we applied Cordova to transform web applications into native applications.

Nestler *et al.* [16] presented an authoring tool called ServFace Builder to help typical users (lacking programming skills) in the design and creation of service-based interactive applications via a graphical interface. Users can link data from two service components by clicking their inputs and outputs. Unlike ServFace, MASA focuses on the development of Mobile Apps, and provides a Broadcast mechanism to perform data exchange in a more flexible manner. Francese *et al.* [7] proposed a novel approach to the sharing and reusing of user-generated Mobile Apps, called MicroApp Generator (MAG). MAG automatically creates Mobile Apps (called MicroApps) on a smartphone. The MicroApps can be designed visually by reusing services available in the MicroApp Store. Our approach is similar to MAG; however, we provide additional features, such as service caching, service prefetching, and broadcast for messages. Bouras *et al.* [8] presented a software framework for the development of Mobile Apps using various state-of-the-art web technologies for cross-platform functionality. Their framework provides three main components: UI Helper, Core Helper and Third-party Services Helper, to help designers in the creation of web applications for mobile devices. They provide several of the features found on MASA; however, the notion of composable units and data exchange mechanisms are missing. CARSB (Composite App with RESTful service and Service Bricks) [17] is our previous research result. CARSB is able to create Web-based or Android-based Mobile Apps based on Service Bricks, rectangular UI components used for the display of specific information, and RESTful services. MASA is devised based on similar basic design concepts of CARSB, but additionally provide the functionality of Broadcast as well as service caching and prefetch.

3. MASA: MOBILE APPLICATION SLICE ARCHITECTURE

In this section, we detail the proposed Mobile Application Slice Architecture (MASA), including the system requirements, system architecture, schemes used for service invocation and caching, the proposed service prefetch method, and the channel-based broadcast mechanism.

3.1 System Requirements

Our objective in this research was to facilitate the composition of multiple UIs and information sources by and for smartphone users. Based on the above research goal, we identified the following key requirements:

1. A mechanism to allow communication between components
2. A component architecture capable of performing a range of actions involving RESTful services under various wireless network conditions
3. The ability to publish, search for, compose, and utilize mobile components
4. A cross-platform component architecture suitable for Android and iOS platforms

3.2 System Architecture

Based on the identified requirements, we designed MASA as shown in Fig. 2. The building blocks of MASA are MAS, which operate as page-style components in Mobile

Apps. MAS includes a view function to assume the role of a user interface, and several MAS APIs, which provide the core functionality of MAS components. MAS APIs include Service Invoker, Cache Manager, Context Manager, and Broadcast. Service Invoker invokes RESTful services in the cloud, and performs service prefetch actions based on designated prefetch rules. Cache Manager stores cached service data, and monitors cache TTL (time to live), to determine whether the cached data has expired. Context Manager maintains the MAS configuration file and provides contextual information retrieved from the user's smartphone. Broadcast furnishes a data exchange mechanism (called Channel), to allow MAS components to communicate with one another.

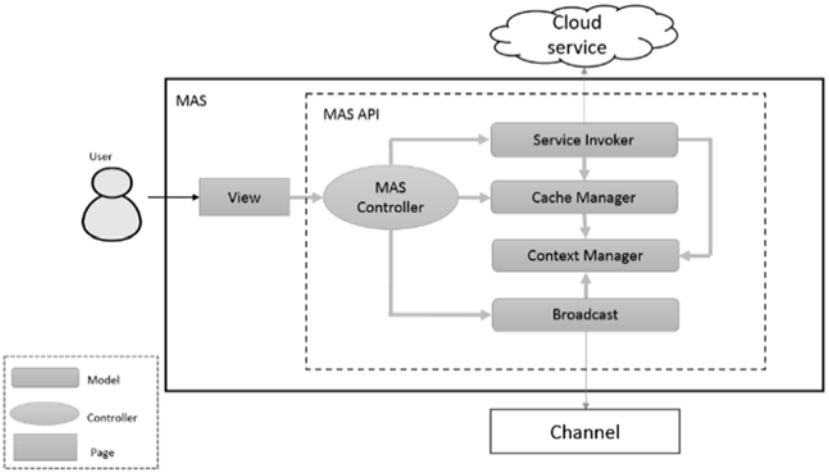


Fig. 2. MASA system architecture.

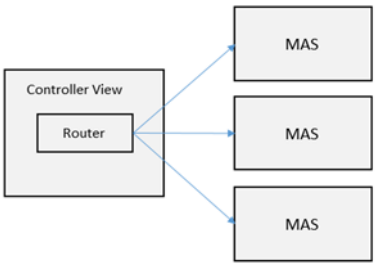


Fig. 3. Controller in MASA.

Using MASA to construct an app involves composing multiple MAS components. This raises the following two issues: (1) How can we compose a user interface with integrated views for multiple MAS components; and (2) How can we enable the movement of data among multiple MAS components. We dealt with the first issue by applying the mechanism found in SPA (single page applications) [18], and used the React router plugin to switch among multiple MAS components, as shown in Fig. 3. The user can trigger the router when swiping the screen or clicking on items representing a MAS on the menu bar. The second issue is dealt with in Section 3.5.

3.3 Cache Manager

Cache Manager is a component of the MAS API, which controls all of the cached data in MASA. We define two types of cached data in Cache Manager: service cache and user action cache. The service cache stores responses received from RESTful services. These are verified and stored when the service is invoked, and then retrieved when the same request is repeated. The user action cache is for temporary data, which is verified, stored, and retrieved by the MAS designers, as in programming scenarios. All of the cached data are given TTL (Time-to-Live) attributes to record how long the cache has been alive and valid [11]. It should be noted that we adopted the web storage mechanism in HTML5 for our service cache mechanism.

As mentioned, the Cache Manager stores service responses within the cache when they are verified as valid. A valid response means that the status code is successful (HTTP code 200) and the response includes expected tokens. The expected tokens are set by the MAS developer and checked using JSONPath [19]. For example, the developer could assert that the response of the Stock service should contain a "stock list" token. Thus, any response that does not contain the token "stock list" is deemed invalid.

When a new service request arrives, the Cache Manager checks whether any of the cached data matches the request, in order to determine whether the cached data could serve as a response. In other words, the Cache Manager fetches cached data with the same URL, service method, and parameters as the service request. It also checks whether the retrieved cached data is alive (not expired), based on its TTL.

Success in the above process depends on properties of linked RESTful services specified in the MAS configuration: (1) service name; (2) service method; (3) service parameters; (4) expected tokens in the service response; and (5) TTL (time to live). As for the user action cache, designers are free to stipulate the data to be cached as well as its TTL. Compared with the service cache, the user action cache can be any variable in the app. It should not be the response of RESTful services. It is usually the output of a time-consuming task, such as mathematical computation procedures on the client side.

3.4 Service Prefetching and Invoker

Prefetch is crucial to the efficiency of service-based systems, and particularly for mobile service systems with limited resources and unstable network connectivity. We integrated prefetch within the Service Invoker module of MASA. The Service Invoker is divided into two parts: (1) Service Manager: responsible for invoking RESTful services, and (2) Prefetch Manager: responsible for handling the prefetch for RESTful services. In other words, Service Invoker can actually invoke RESTful services and prefetch responses of selected services. Furthermore, the Context Manager is able to retrieve current contextual information from the user's smartphone.

As mentioned previously, we propose two methods, identification of prefetchable services and the context-aware service prefetch, to enable predictions and pre-invocations for commonly used services in appropriate occasions.

3.4.1 Identification of prefetchable services (IPS)

The basic idea of the method for the identification of prefetchable services is as fo-

llows: When a base service is invoked by a user-triggered command, relational services that are related to the base service can also be invoked in advance, thereby enabling the caching of responses of pre-invoked services for further use. In this research, a relational service is defined as any service that accepts part of the responses of the base service as input arguments to produce subsequent data. Afterwards, access to the prefetched service data is monitored to determine whether these data are actually useful for the user. When prefetched service data is not used regularly (*i.e.*, the user tends not to browse prefetched or cached data), the linkage between the base service and the relational one is temporarily canceled. The unlinked relational service is not prefetched again until the rules of re-linking are triggered.

Multiple user-given parameters for relational services must be designated in the MAS configuration, including a brief description of the base service and its related services. Note that the description of each relational service contains the rules for the retrieval of input data from the output of the base service using JSONPath to invoke the designated relational service. For example, address information retrieved from the output of the base service for finding nearby tourist attractions is used as the input of the map service specified as a relational service.

We formulated a pattern (referred to as the *Consume and Prefetch Pattern*) to retrieve candidate prefetchable services in order to determine whether the linkage between a base service and its relational service(s) is strong enough and whether the prefetch of a relational service must be canceled or resumed. A relational service is deemed "consumed" if the user browses the prefetched data of a relational service after triggering the base service. A relational service is deemed "unconsumed" if the user does not browse the prefetched data of a relational service in either of two invocations of the base service. Consumption is checked by using Eq. (1) to determine whether the relational service needs to be pre-invoked, as follows:

$$PS(s) = \gamma + \alpha \cdot m - \beta \cdot n \geq \epsilon \tag{1}$$

where ϵ is the given threshold, γ is a given initial value, α is the increment value when a "consumed" event occurs, β is the decrement value when an "unconsumed" event occurs, m is the number of "consumed" events, n is the number of "unconsumed" events, and PS(s) is a score (called the prefetch score), which is used to determine whether the service can be prefetched.

For any specified relational service, MASA monitors the cached service data and computes the prefetch score continuously. When a given base service is invoked again, its relational service is identified as prefetchable only if its prefetch score is equal to or exceeds threshold ϵ.

3.4.2 Context-aware service prefetch (CASP)

Due to the limited resources of mobile devices, arbitrarily invoking all services in advance is inappropriate. Although we have reduced the number of services that may be prefetched by using the IPS method, the prefetch task should be conducted only if the current context of the smartphone is suitable for additional service invocation. In other words, one must consider the current context, such as network bandwidth and battery

capacity to determine whether to perform a prefetch operation in a rule-based way. MASA mainly focuses on the remaining battery capacity and the current wireless network conditions as contextual data of whether to perform context-aware service prefetch. We also formulated another concept called "*Long Response Time First*", to decide whether a service should actually be prefetched in a rule-based way. The key points of the proposed context-aware service prefetch method include the following:

1. MASA divides services into LRT (long response time) and SRT (short response time), according to the threshold θ, which is specified in the MAS configuration file. A service is identified as LRT if its response time exceeds θ, whereas it is identified as SRT if the response time is equal or less than θ. MASA attempts to prefetch LRT services first because users tend to spend more time-consuming LRT services, particularly under slow or unstable network conditions. The default θ in this study was set to 2 seconds.
2. Three categories are specified for wireless network conditions: (a) 4G/Wifi with high bandwidth; (b) 3G with medium/low bandwidth; and (c) other conditions with no network access or unidentified bandwidth. The current network conditions are detected automatically by MASA.
3. Battery capacity is categorized in the same manner: (1) high: for the capacity exceeding 70%; (2) low: for the capacity of less than 30%; and (3) medium: for the remaining cases. The current battery capacity is also detected automatically by MASA.
4. CASP is essentially naïve, in that services are pre-invoked when the resources available to the user's smartphone are enough, whereas service prefetch is avoided when the available resources are insufficient. The rules for service prefetching are listed in Table 1. For example, when the remaining battery capacity is high and the network type is 4G/Wifi, MASA conducts service prefetch for both LRT and SRT services. However, when the remaining battery capacity is low, service prefetch is skipped to conserve energy.

Table 1. Rules for context-aware prefetching.

Battery Capacity	Wireless Network Condition					
	4G/Wifi		3G		Others	
High	LRT	*true*	LRT	*true*	LRT	*false*
	SRT	*true*	SRT	*false*	SRT	*false*
Medium	LRT	*true*	LRT	*true*	LRT	*false*
	SRT	*false*	SRT	*false*	SRT	*false*
Low	LRT	*false*	LRT	*false*	LRT	*false*
	SRT	*false*	SRT	*false*	SRT	*false*

3.5 Broadcast

As mentioned above, dealing with the movement of data between MAS components is crucial to the underlying architecture. There are several common ways to enable the exchange of data between components: (1) specifying the interfaces for each component, and integrating components based on interface requirements; and (2) preparing a universal memory to allow components to exchange messages. The first strategy tends to increase complexity in composition; therefore, we adopted the second strategy and devised a corresponding Broadcast mechanism utilizing HTML5 web storage APIs to allow MAS com-

ponents to publish and subscribe to messages.

Similar to the Broadcast mechanism on the Android platform, Broadcast in MASA allows MAS components to communicate with one another. In formulating Broadcast, we followed the publish-subscribe pattern in the design of objects (called channels) to be shared among multiple MAS components. From the perspective of implementation, a channel is a group of items in the local storage whose names have the same prefix. A channel can be used to publish and subscribe to messages through the MAS API.

Fig. 4 presents the architecture of the proposed Broadcast mechanism. MAS components can freely publish and subscribe messages to and from a designate channel. Broadcast monitors all channels when MAS publishes any message to any channel and notifies all MAS components subscribing to the same channel in order to obtain newly published messages. From the perspective of App development, designers can use Broadcast to integrate MAS components, as follows: (1) Assign the channel where the message populates; (2) Assign a message name; (3) Add the channel name to the MAS configuration file; and (4) Call the message publish/subscribe APIs in the MAS code.

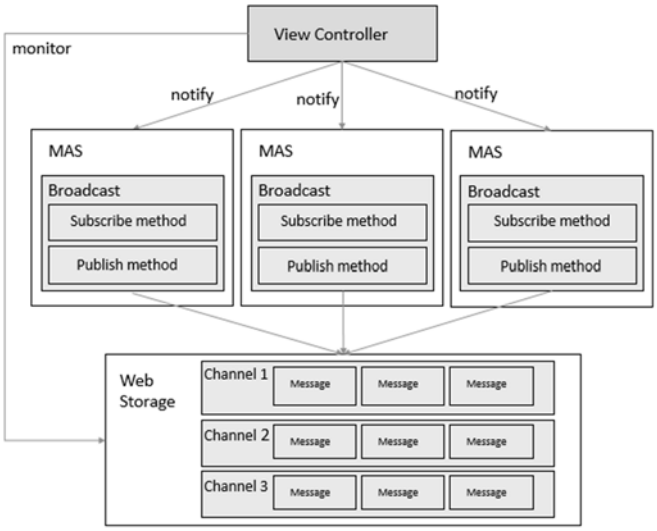


Fig. 4. Architecture for the broadcast mechanism.

4. SYSTEM PROTOTYPE

In this section, we implement a prototype of the MASA system. First, we introduce the MAS APIs. Second, we illustrate the design of the MASA Portal. Third, we present a Mobile App that is composed of three MAS components to demonstrate the features of MASA.

4.1 MAS API

To assist developers in the creation of MAS components, we developed a suite of MAS APIs in JavaScript to provide functionalities for service invocation, service cache, service prefetch, and broadcast. The specifications of MAS APIs are described in Table 2.

Table 2. MAS APIs.

Return data	API descriptions
void	*invokeService* (*serviceName, callback, parameter*) Designers can use this API to invoke RESTful services with the support of service caching. MASA invokes the designate service based on the parameters. Note that other service information, such as service URL and supported HTTP method, is already specified in the MAS configuration file and therefore does not have to be provided again. The callback is the developer-provided JavaScript function, which is triggered when a service response is returned.
void	*prefetchService* (*serviceName, parameter*) Designers can use this API to start performing service prefetch (IPS and CASP).
object	*subscribeByBroadcast* (*channel*) Enables designers to proactively obtain messages in the designate channel.
void	*publishByBroadcast* (*channel, data*) Designers can use this API to push messages to the designated channel.
void	*setCache* (*cacheName*, *data*, *TTL*) Provides the functionality required to store data in a user action cache with a specified cache name and TTL (Time-To-Live).
object	*getCache* (*cacheName*) Used to retrieve data from the designated user action cache. Note that only unexpired cached data is retrieved using this API.

4.2 MASA Portal

MASA Portal is a web-based tool for MAS designers as well as App designers that seek to build a composite MAS. The interface of MASA Portal is used to compose a MAS for assembly by different MAS designers. Fig. 5 presents the user interfaces used to design a composite MAS based on searched MAS components. The composite MAS is automatically transformed into an installable App by MASA Portal. The transformation process is realized by NPM (Node Package Manager) and the Cordova command-line manager to download the necessary libraries from various sources and wrap the composite MAS as an App. At present, MASA allows the creation of Android Apps; however, other platforms (such as iOS and Windows Phone), could easily be supported because MASA is based on cross-platform HTML5 technology.

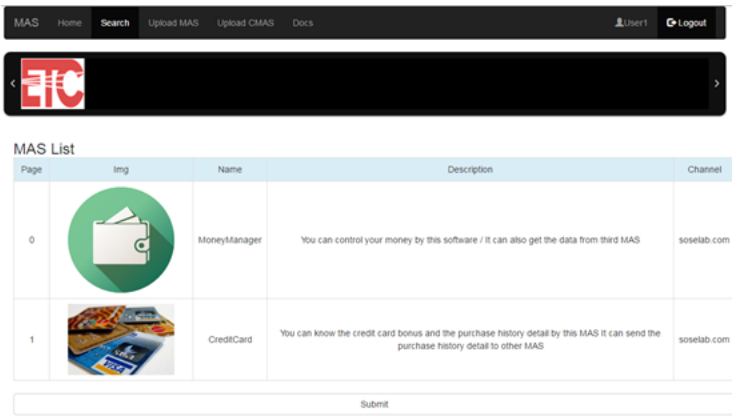
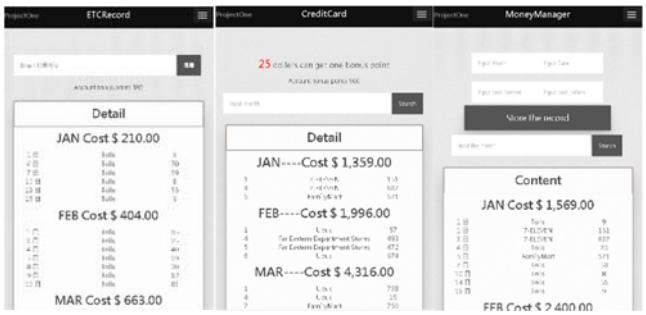


Fig. 5. MASA portal: Design of composite MAS.

4.3 MASA Applications

Fig. 6 demonstrates an example of a mobile app built via the MASA portal. This CMAS is assembled by three different MAS components, namely, *MoneyManager* MAS, *ETC* (Electronic Toll Collection) MAS and *CreditCard* MAS. All three MAS components are published in the MASA portal. In the generated CMAS, the data produced by *CreditCard* MAS and *ETC* MAS are transmitted to *MoneyManager* MAS via the proposed Broadcast mechanism for data aggregation and information presentation. Fig. 6 (a) shows the *ETC* MAS that retrieves the ETC payment records for a user. Fig. 6 (b) displays the *CreditCard* MAS that mainly lists records of credit card expenditure and gained bonus for the same user. Fig. 6 (c) is the MoneyManager MAS that stores the expenditure filled in by the user, receives the data transmitted by two other MASs via the Broadcast, and renders the aggregate expenditure information (*ETC*, *CreditCard*, and user-provided data). Three MAS pages can be switched by swiping the screen or clicking on items of the menu bar. We also used this App to be the testbed to evaluate the efficacy of the proposed IPS and CASP prefetch methods. Details of experiments are shown in Section 5.



(a) (b) (c)

Fig. 6. Example of a MASA app.

5. EXPERIMENTAL EVALUATIONS

To demonstrate the feasibility of the proposed MASA system, four requirements presented in Section 3.1 should be satisfied. We discuss how MASA realizes these requirements are as follows.

1. Communication mechanisms: the proposed Broadcast mechanism obviously realizes this requirement.
2. Supporting actions under various wireless network conditions: the proposed IPS and CASP caching and prefetch methods could meet this requirement. To further evaluate the efficacy of these two methods, we conducted two quantitative experiments. Details are shown in the following subsections.
3. Publish, find, compose, and utilize components: the MASA portal is able to provide these features via Web user interfaces.
4. Cross-platform: since MASA is based on Cordova and HTML5, this requirement is sufficiently satisfied.

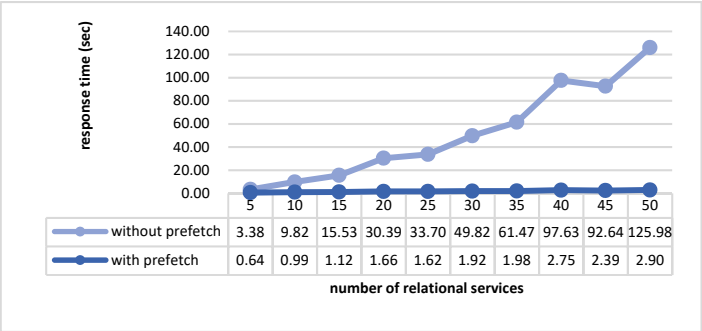
5.1 Experimental Setup

The hardware and software configurations for experiments were as follows: (1) Desktop PC (hosting the RESTful services): Intel Core™ i7-2600 CPU 3.4GHz with 8G RAM and 150G hard disk running Windows 7 (64 bit); and (2) Android virtual device (emulator): ARM (armeabi-v7a) with 1G RAM running under the Android 6.0 OS.

5.2 Performance Testing of the IPS (Identification of Prefetch Services) Mechanism

In the first experiment, we evaluated system performance when various numbers of relational services are prefetched for a base service. The performance was evaluated by conducting a comparison with normal service invocation (without a prefetch mechanism). We used the *CreditCardRecords* service (base service) and *Bonus* service (relational service) used in *CreditCard* MAS (described in Section 4.3) to conduct this experiment. The performance was evaluated in terms of response time; *i.e.*, the delay time (in seconds) between the moment the service request is issued and the moment a service response is received. The total response time was calculated by summing up all of the response times for a given base service and its relational services. The number of relational services was increased by five each time.

Fig. 7 shows the total service response times of base services and relational services. It is clear that MASA shortens the response time by pre-invoking relational services and storing service responses in advance. Conventional service invocation resulted in very long response times when the base service was associated with a large number of relational services. For example, the invocation of the base service with 50 relational services required 126 seconds, which would be unacceptable to most end users.



	5	10	15	20	25	30	35	40	45	50
without prefetch	3.38	9.82	15.53	30.39	33.70	49.82	61.47	97.63	92.64	125.98
with prefetch	0.64	0.99	1.12	1.66	1.62	1.92	1.98	2.75	2.39	2.90

Fig. 7. Total response time with various numbers of relational services.

5.3 Performance Tests for the CASP (Context-Aware Service Prefetch) Mechanism

The performance of the proposed service prefetch mechanism was evaluated using the *CreditCardRecord* service used in the mentioned *CreditCard* MAS. We implemented one version with prefetch (WP) and one version without prefetch (WOP), and conducted experiments in three contexts under high battery capacity: 4G network, 3G network, and no network access. To simplify the experiment, we assumed that the *CreditCardRecord*

service would always be identified as a prefetchable service. Experiments were conducted 50 times per second under the three conditions to obtain the rendering time; *i.e.*, the delay (in seconds) between the moment the service request was issued and the moment the service response was displayed in the page. The rendering time is the sum of the service response time and the time required to display the service results. The TTL (time to live) was set to 30 seconds. The experiment results are presented in Figs. 8-10 for the 4G network, 3G network, and no network access, respectively.

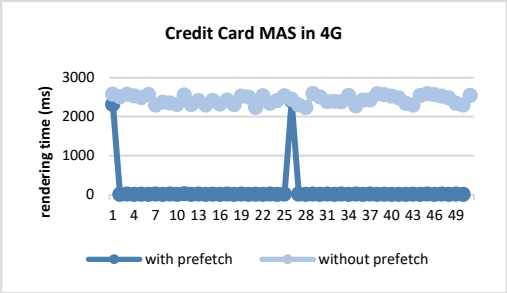


Fig. 8. Rendering time under the 4G condition.

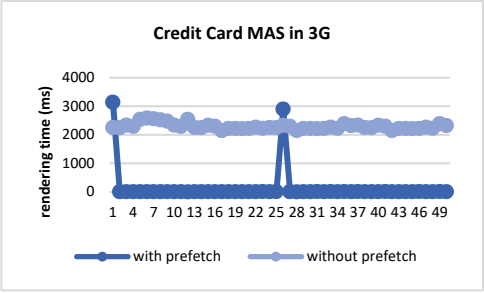


Fig. 9. Rendering time under the 3G condition.

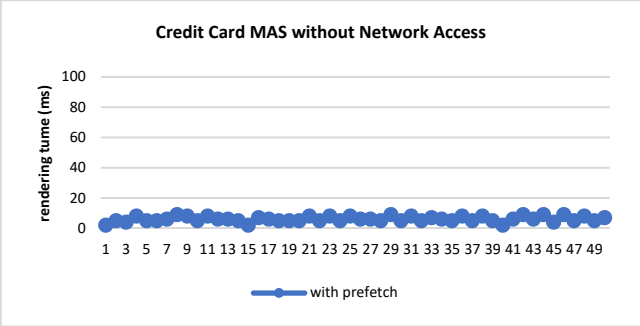


Fig. 10. Rendering time without network access.

Our experiment results demonstrate that the proposed prefetch mechanism (WP solution) is able to trigger the pre-invocations for the *CreditCardRecord* service every 30 seconds, thereby allowing further browsing. In contrast, the WOP solution was required to invoke a service for each service request. The *CreditCardRecords* service had a long response time (LRT), and prefetch was conducted for either a 4G/Wiki network or 3G network in accordance with the rules specified in Table 1. Under these conditions, the results under 4G (Fig. 8) were very similar to those obtained under 3G (Fig. 9). When there was no network access (Fig. 10), the WP solution continued to provide service results for the previous invocation, whereas the WOP solution showed an error message. Our experiment results demonstrate the efficacy of MASA in reducing latency (by 93%~98%) and partially maintaining the availability of applications even when there is no network access. In other words, for any RESTful service, as long as its responses are cached, its service client (a MAS) is able to read cached data to maintain the availability even in the condition of no network access.

5.4 Limitations of MASA

Although MASA could provide numerous important features to help users build cache-enabled composite Mobile Apps, there are three limitations when using the proposed MASA:

1. Component designers are expected to be familiar with HTML, CSS, JavaScript, and JSON, as well as the MAS APIs to develop MAS components.
2. Although an App designer does not need to write code, he (she) must know the cached data that each MAS component accesses, and has the ability to compose MAS components based on the publish/subscribe behaviors of each MAS component.
3. Since the Broadcast mechanism does not consider authentication and authorization, malicious MAS components may steal sensitive data offered by other MAS components. The security issue is not in the scope of this paper, but will be taken into accounts in the future.

6. CONCLUSIONS

This paper introduces a novel framework (MASA) for the use in the composition and assembly of mobile components with the support of context-aware service caching and prefetch. The MASA framework has three main features: (1) A programming model for building MAS components based on mobile web techniques to produce cross-platform Mobile Apps; (2) the ability to invoke, cache, and prefetch RESTful services efficiently and thereby enhance the user experience under various network conditions; and (3) a newly-devised Broadcast mechanism to enable the exchange of data among MAS components to facilitate the integration of MAS components. The proposed prefetch mechanism was also shown to reduce response times considerably in various contexts.

The future plans of this research are two-fold: (1) MASA will combine MASA with PWA (Progressive Web Apps [20]), a new web technology that offers the functionality of working offline and push notifications, to enhance the features of the relational and context-aware service prefetch on PWA Apps; (2) MASA will support better authentication and authorization to block possibly malicious programs.

REFERENCES

1. R. Jabangwe, H. Edison, and A. N. Duc, "Software engineering process models for mobile app development: A systematic literature review," *Journal of Systems and Software*, Vol. 145, 2018, pp. 98-111.
2. B. König-Ries, "Challenges in mobile application development," *IT-Information Technology*, Vol. 51, 2009, pp. 69-71.
3. 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.
4. Z. Sanaei, S. Abolfazli, A. Gani, and R. Buyya, "Heterogeneity in mobile cloud computing: taxonomy and open challenges," *IEEE Communications Surveys & Tutorials*,

Vol. 16, 2014, pp. 369-392.
5. D. D. Terry and V. Ramasubramanian, "Caching xml web services for mobility," *Queue*, Vol. 1, 2003, pp. 70-78.
6. 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.
7. 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.
8. C. Bouras, A. Papazois, and N. Stasinos, "Cross-platform mobile applications with web technologies," *International Journal of Computing and Digital Systems*, Vol. 4, 2015, pp. 153-163.
9. H. Heitkötter, S. Hanschke, and T. A. Majchrzak, "Evaluating cross-platform development approaches for mobile applications, in Web information systems and technologies," in *Proceedings of International Conference on Web Information Systems and Technologies*, 2012, pp. 120-138.
10. J. Fernandez, A. Fernandez, and J. Pazos, "Optimizing web services performance using caching," in *Proceedings of International Conference on Next Generation Web Services Practices*, 2005, pp. 6-11.
11. A. Papageorgiou, M. Schatke, S. Schulte, and R. Steinmetz, "Enhancing the caching of web service responses on wireless clients," in *Proceedings of IEEE International Conference on Web Services*, 2011, pp. 9-16.
12. X. Liu and R. Deters, "An efficient dual caching strategy for web service-enabled PDAs," in *Proceedings of ACM Symposium on Applied Computing*, 2007, pp. 788-794.
13. A. Liyanaarachchi and S. Weerawarana, "An end-to-end caching protocol for web services," in *Proceedings of International Conference on Advances in ICT for Emerging Regions*, 2012, pp. 96-102.
14. J. Spillner, A. Utlik, T. Springer, and A. Schill, "RAFT-REST – A client-side framework for reliable, adaptive and fault-tolerant RESTful service consumption," in *Proceedings of European Conference on Service-Oriented and Cloud Computing*, 2013, pp. 104-118.
15. I. Ollite and N. Mohamudally, "Performance analysis of a 2-tier caching proxy system for mobile RESTful services," in *Proceedings of IEEE International Conference on Computer as a Tool*, 2015, pp. 1-7.
16. T. Nestler, L. Dannecker, and A. Pursche, "User-centric composition of service front-ends at the presentation layer," in *Proceedings of Service-Oriented Computing Service Wave Workshops*, 2010, pp. 520-529.
17. S.-P. Ma, P.-Z. Chen, Y.-S. Ma, and J.-S. Jiang, "Building mobile apps by ordinary users: A service-brick-based approach," *Journal of Information Science and Engineering*, Vol. 34, 2018, pp. 611-629.
18. M. Mikowski and J. Powell, *Single Page Web Applications: JavaScript End-to-End*, Manning Publications Co., NY, 2013.
19. J. Friesen, *Extracting JSON Values with JsonPath, in Java XML and JSON*, 2016, Apress, Berkeley, CA, pp. 223-239.
20. Progressive Web Apps, https://developers.google.com/web/progressive-web-apps/.

Shang-Pin Ma (馬尚彬) received his Ph.D. degree in Computer Science and Information Engineering from National Central University, Taiwan, in 2007. He is currently a 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.

Chi-Chia Li (李啟嘉) received his Bachelor (2014) and Master's (2016) degrees from the Department of Computer Science and Engineering, National Taiwan Ocean University, Taiwan. His research interests include software engineering, service computing, and mobile application architecture.

Shin-Jie Lee (李信杰) is an Associate Professor in Computer and Network Center at National Cheng Kung University (NCKU) in Taiwan and holds joint appointments from Department of Computer Science and Information Engineering at NCKU. His current research interests include software engineering and service-oriented computing. He received his Ph.D. degree in Computer Science and Information Engineering from National Central University in Taiwan in 2007.

Hsi-Min Chen (陳錫民) received the B.S. and Ph.D. degrees in Computer Science and Information Engineering from National Central University, Taiwan, in 2000 and 2010, respectively. He is currently an Associate Professor with the Department of Information Engineering and Computer Science, Feng Chia University, Taiwan. His research interests include software engineering, object-oriented technology, service computing, and distributed computing.

Wen-Tin Lee (李文廷) received his Ph.D. degree in Computer Science and Information Engineering from National Central University, Taiwan, in 2008. Lee is currently an Associative Professor in the Department of Software Engineering and Management at National Kaohsiung Normal University. His research interests include software engineering, service-oriented computing and software process management.