Hindawi Publishing Corporation
Mobile Information Systems
Volume 2016, Article ID 7263729, 16 pages
http://dx.doi.org/10.1155/2016/7263729

Research Article

ISOMP: An Instant Service-Orchestration Mobile M2M Platform

Cholhong Im and Changsung Jeong

Department of Electrical Engineering, Korea University, 145 Anam-ro, Seongbuk-gu, Seoul 02841, Republic of Korea

Correspondence should be addressed to Changsung Jeong; csjeong@korea.ac.kr

Received 19 October 2015; Revised 30 December 2015; Accepted 14 February 2016

Academic Editor: Lazaros Gkatzikis

Smartphones have greater computing power than ever before, providing convenient applications to improve our lives. In general, people find it difficult to locate suitable applications and implementing new applications often requires professional skills. In this paper, we propose a new service platform that facilitates the implementation of new applications by composing prebuilt components that provide the context information of mobile devices such as location and contacts. Our platform introduces an innovative concept named context collaboration, in which smartphones exchange context information with each other, which in turn is used to deduct useful inferences. The concept is realized by instant orchestration, which assembles some components and implements a composite component. The interactive communication interface helps a mobile device to communicate with other devices using open APIs, such as SOAP and HTTP (REST). The platform also works in heterogeneous environments, for example, between Android and iOS operating systems. Throughout the platform, mobile devices can act as smart M2M machines with context awareness, enabling intelligent tasks on behalf of users. Our platform will open up a new and innovative pathway for both enhanced mobile context awareness and M2M, which is expected to be a fundamental feature of the next generation of mobile devices.

1. Introduction

Smartphones are now common and tightly interlinked with our lives. An increasing number of people have an interest in mobile technology and expect it to be continuously improved. Currently, the hardware specifications of smartphones are almost equivalent to those of PCs and smartphones have greater computing power than ever before. Furthermore, smartphones have convenient applications that improve our lives. However, they still need more capabilities in terms of gathering, analyzing, and inferring the user's environment and intentions. Those capabilities can be represented by context awareness. Context is any information that can be used to characterize the situation of an entity [1]. To achieve a tangible context-awareness capability, we design and implement an instant service-orchestration platform based on M2M architecture. Thus, the platforms are distributed and they can collaborate with each other. Even though some simple types of context-awareness service have been investigated, such as the retrieval of another user's location and contacts, users cannot get location and contact information on other devices without making use of exclusive applications. Furthermore, the context information of one application cannot be exchanged with other applications. In turn, our platform can support the gathering of context information from devices as well as determining what assistance the users need by analyzing the gathered context information. Orchestration is the process of arranging multiple services or functions to act in a predefined sequence [2]. Our research accomplishes the aim of service-oriented computing to implement new services easily by connecting some readymade components. From now on, mobile devices will cease being merely clients and will become servers or providers. Our platform can also collaborate with other devices such as PCs and workstations. It realizes flexible component architecture for interactive service-oriented middleware to offer enhanced context awareness on common smartphones.

2. Related Work

Researchers have proposed various context-provisioning systems that can acquire, analyze, and manage context information. The context toolkit [3] has a layered architecture that separates out the context acquisition, representation, and adaptation processes. The context management framework (CMF) [4] allows for semantic reasoning of the context in real time and even in the presence of noise, for uncertainty, and a rapid variation in the context. Service-oriented context-awareness middleware (SOCAM) [5] constitutes an architecture of service-oriented context-aware middleware for the building and rapid prototyping of context-aware mobile services. However, the authors handle generic context information with no context-aware services. Other researchers have managed to apply context information to mobile services, as smartphones provide various kinds of context information such as location, contacts, and media. Context-Aware Pervasive Networking (CAPNET) [6] constitutes context-aware middleware for mobile multimedia applications. The middleware offers functionality for service discovery, asynchronous messaging, publish-and-subscribe event management, management of context information, and the handling of both local and network resources. Context Phone [7] is middleware for context-based mobile services and it can provide context information as a resource. Therefore, developers can gain context information from smartphones and implement context services such as context contacts and context media. LifeMap [8] is a smartphone-based context provider operating in real time, fusing an accelerometer, digital compass, Wi-Fi, and GPS to track and automatically identify points of interest with room-level accuracy. However, these types of context-aware middleware cannot offer context information to other devices and their applications cannot be implemented instantly with service-oriented components. Some SOA-based context-aware middleware, such as A-MUSE [9], MobileSOA [10], and ContextServ [11], has been developed, adopting the representational state transfer (REST) protocol or model-driven architecture (MDA). Our research proposes instant orchestration, which can implement applications on demand and provide context information to other devices with open APIs based on SOA. ContextDroid [12] proposed a context service from sensors to applications, but other devices cannot integrate the information. Context Aware Machine Learning Framework (CAMF) [13] proposed context-aware machine learning architecture and context information interfaces that can provide location proximity, face recognition, and so on. Consequently, our research can make full use of the context information of a mobile device and exchange this piece of information with other devices using an open API. Therefore, mobile devices collaborating with each other are able to perform orchestrated processes. Since we adopt SOA-based event-driven architecture, our composite components can be implemented by orchestrating various context components and external components including other composite components. We first illustrate that a mobile device can be a great server machine, providing a useful intelligent service based on mobile context information and that our platform can make a flexible mobile application structure based on SOA.

3. The Capability of Context Awareness

In the work that first introduces the term "context-aware," the context is defined as the location, the identities of nearby people and objects, and changes to those objects [14]. The new definition of "context" includes any information that can be used to characterize the situation. If a piece of information can be used to characterize the situation of a participant in an interaction, then that information is context [1]. We assume contacts, calendar, and phone state as the context information, since the information gives important clues regarding the inference of user situations. Various kinds of context information can be gathered from mobile devices. With the help of the mobile platform SDK (e.g., Google Android and Apple Cocoa), the information can easily be retrieved from mobile devices. In existing research works, researchers have inquired about simple forms of context information such as current position, schedule, and friends' phone numbers, without integrating mutual context information. However, our platform focuses on the mutual and simultaneous collaboration of context information. This action is termed "context collaboration." Due to this context collaboration, mobile devices can provide more intelligent services to satisfy user demands. Our platform can undertake context collaboration according to the phone state such as unavailable or busy. Generally, it is not easy to call someone at a time when he/she is busy or unavailable, in which case the phone needs to keep calling until the recipient is on the line. Even though a mobile device has its own phone state information such as busy, unavailable, and available, there have not yet been any trials to acquire and make use of this aspect properly. On the other hand, our platform automatically queries the phone state information on the other device, gathers the device state information, and decides what to do according to whether the line is busy or not. Hence, it would be possible for users to determine whether they could make a call or not by using the context-aware function of our platform. However, such context awareness might cause privacy and energy consumption issues.

4. The Instant Service-Orchestration Platform

4.1. The Basic Concept. Our platform consists of a provider service, client service, and an external service. The provider service includes instant orchestration, context components, and an interactive communication interface. The context components are implemented via an Android SDK and they provide device information such as the location, phone state, and contacts. The interactive communication interface supports a bidirectional connection using an open API that is independent of network protocols. Instant orchestration creates composite components that consist of context components and external services, and it controls the execution process of the composite components conforming to orchestration specifications. Any external service with SOAP or HTTP (REST) interface can be a part of the composite components. The client service sends a request to execute a composite component to the provider service and receives its results. Figure 1 shows the basic concept of our platform.

4.2. Event-Driven Architecture. An event is a notable thing that happens inside or outside our platform. An event may signify a problem or impending problem, an opportunity, a threshold, or a deviation [15]. Event-driven architecture embraces mechanisms for coordinating the callers and

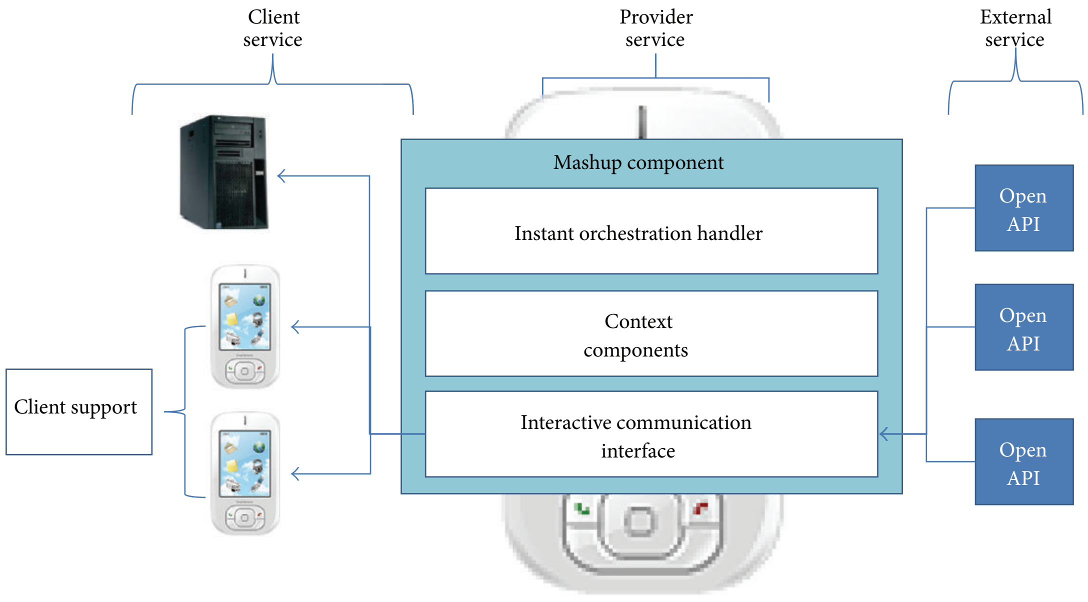


FIGURE 1: The basic concept of our platform.

providers of a service, the producers and consumers of data, and the sensors of and responders to software events with a variable level of communication coupling, a variable spectrum of message correlation, and variable options to deliver quality of service [16]. The enterprise service bus (ESB) is one of the typical solutions that is implemented based on event-driven architecture [17]. Our platform adopts event-driven architecture to work intelligently with context awareness. First, events are generated whenever the platform receives a new request message, interprets the message, and generates additional events such as an invocation, skipping, or termination. Our platform controls some actions according to the generated events. Generally, the actions and events are described by standard process notation such as Business Process Execution Language (BPEL [18]) or Business Process Modeling Notation (BPMN). Figure 1 shows the concept of event-driven architecture for an inventory management system. If a system starts working on some conditions, then those conditions should be defined as events. When one of the events is generated, an event-processing engine starts performing actions such as publishing, notifying, invoking service, or starting another business process. In this process, low inventory generates an event, and the reorder inventory process is started to fill the inventory automatically.

4.3. Provider Service

4.3.1. Instant Orchestration Handler. A composite component is made by composing services from other services, which is an increasingly common service design goal [19]. Using instant orchestration, composite components are implemented in an easy and quick way on a mobile device, and it accomplishes context collaboration among mobile devices. Figure 2 shows the orchestration example of a composite component interacting between two devices.

We assume that a user has to contact another user whose contact information is unknown. Usually, people make a call to another acquaintance who might know the contact information, and then they write down or memorize the phone number. Then they call again with the given number, but he/she may not be available or may be busy, so they need to call again later. This situation makes people continue to call until the call is answered. Our platform solves this problem by invoking a composite component to perform context collaboration. In particular, an individual can gain another person's contact information and the other user's device state through context collaboration. The interactive communication interface makes this possible. It does not depend on radio technology as long as the mobile devices are connected to the internet. However, the person seeking the information needs to point out one of his/her acquaintances' devices that is likely to have the contact information. The platform finds that contact information by invoking the "GETCONTACT" component and checks the phone state of the receiver by invoking the "GETTELSTATE" component in the other device. If available, the platform automatically calls via the "TELLCALL" component and if not, as an alternative, the device sends an SMS via "SENDSMS." The instant orchestration handler interprets the orchestration specifications in XML form and controls the execution process of the composite components conforming to the orchestration specifications, as shown in Figure 3. We have already defined the orchestration specification as simple XML language, which is termed "Instant Orchestration Description Language" (IODL) [20] in prior research. IODL is simple and describes the necessary information for the composite components. It does not have the "assign" tag in BPEL. Instead, the same variable can be used for assigning a function. In Figure 3, the initial value of "contactname" is used as the input value

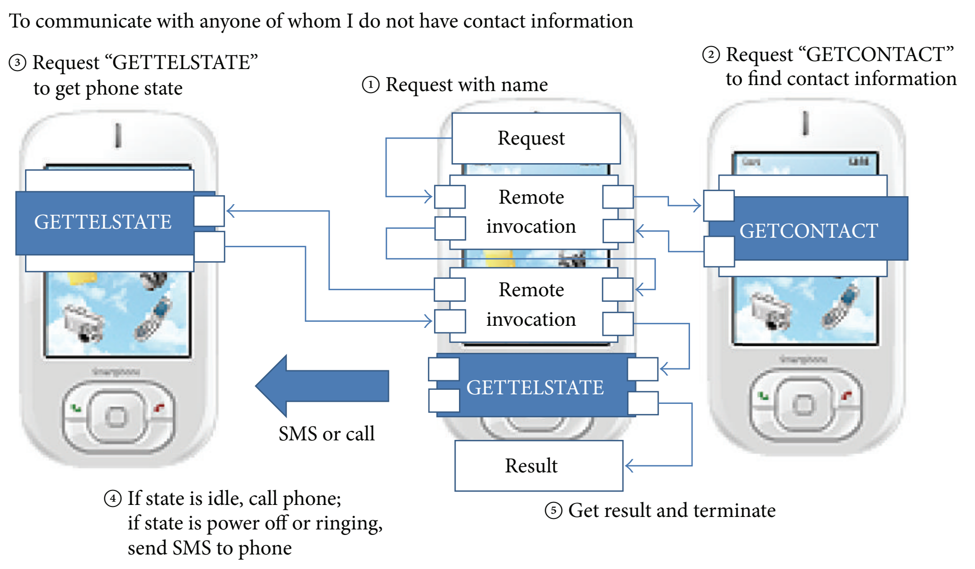
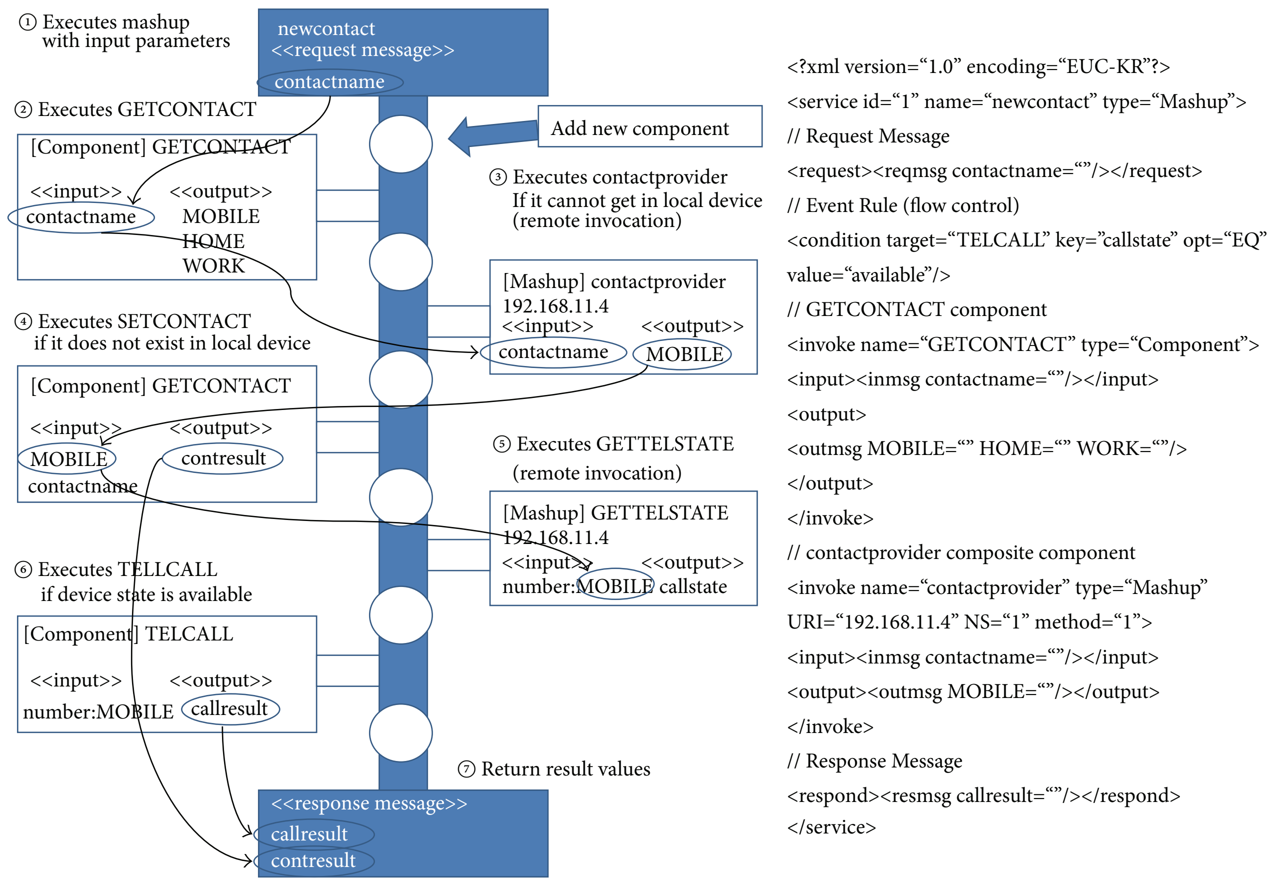

To communicate with anyone of whom I do not have contact information
③ Request "GETTELSTATE" to get phone state
① Request with name
② Request "GETCONTACT" to find contact information
Request
Remote invocation
Remote invocation
GETTELSTATE
GETCONTACT
GETTELSTATE
SMS or call
Result
④ If state is idle, call phone; if state is power off or ringing, send SMS to phone
⑤ Get result and terminate

Figure 2: A composite component implemented by instant orchestration.



```
<?xml version="1.0" encoding="EUC-KR"?>
<service id="1" name="newcontact" type="Mashup">
// Request Message
<request><reqmsg contactname=""/></request>
// Event Rule (flow control)
<condition target="TELCALL" key="callstate" opt="EQ"
value="available"/>
// GETCONTACT component
<invoke name="GETCONTACT" type="Component">
<input><inmsg contactname=""/></input>
<output>
<outmsg MOBILE="" HOME="" WORK=""/>
</output>
</invoke>
// contactprovider composite component
<invoke name="contactprovider" type="Mashup"
URI="192.168.11.4" NS="1" method="1">
<input><inmsg contactname=""/></input>
<output><outmsg MOBILE=""/></output>
</invoke>
// Response Message
<respond><resmsg callresult=""/></respond>
</service>
```

Figure 3: The instant orchestration execution process and IODL specification.

in the orchestration process and maintains its value to the end of the orchestration process. The result values such as "callresult" and "contresult" store the value of the invocation result and are returned to the client service. If a value were to be replaced by another one, then IODL would simply provide an assignment method. Wherever "MOBILE" is used, it has the same value and can be replaced by other variables also. If we specify this as "number: MOBILE," then the "number" variable is replaced with "MOBILE." Additionally, the event rule contains control information such as the condition and

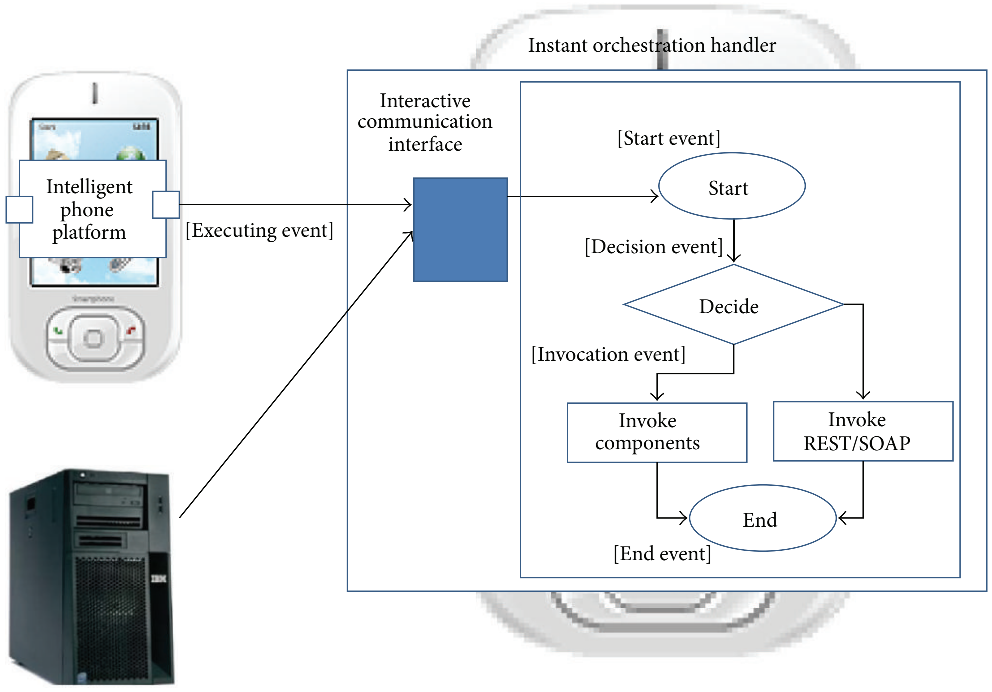


Figure 4: The event-driven process of the instant orchestration handler.

Table 1: Event-processing functions.

Function	Description	Component name
Event handler	Edit and control an event flow	Event flow configuration
		Event flow handler
Event invocation	Invoke remote APIs offering various kinds of interface	Composite endpoint
		SOAP endpoint
		REST endpoint
		Component endpoint
Event rule	Edit and interpret an event rule	Event rule configuration
		Event rule resolver

target, and it handles event processing such as invoking, skip, and beginning process. Surprisingly, direct manual editing in IODL form is not needed, since it can be generated automatically as users enter some information on the edit screen provided by our platform.

The instant orchestration handler adopts SOA-based event-driven architecture and has three event-processing functions. They receive events from the instant orchestration handler and perform various kinds of actions such as invocation, skip, and starting other composite components. Table 1 shows the event-processing functions provided by the instant orchestration handler.

The event handler edits and controls the described event flow in IODL. Event invocation executes components with various kinds of communication protocol. The interactive communication interface starts during the boot sequence and waits to receive a request message. The messages contain the type of events that are generated and which actions should be performed. As a message containing a context collaboration specification is received at a waiting socket, the instant orchestration handler starts interpreting the message automatically and generates various kinds of events such as invocation, skip, and starting other composite components. The event flow handler controls actions according to the received events, as shown in Figure 4.

4.3.2. Context Components. Our platform has eleven context components such as location, contact, and phone state. They are implemented with a native Android SDK including the Google Map API, content provider API, and Android system API [21]. The components provide context information on mobile devices. A composite component is implemented by assembling them, in other words, by orchestration. With the help of the interactive communication interface, a device gathers context information from other mobile devices, and a composite component can be invoked from both one's own device and other devices. Supporting both SOAP and the HTTP (REST) protocol, our platform can exchange information via a message-driven route even in heterogeneous environments. Table 2 shows the context components of our platform. In the location function group, "GETLOCATION" retrieves the latitude and longitude values of the current position [22]. "GETDISTANCE" calculates the distance between two positions. "VIEWMAP" displays a Google Map of the requested position. In addition, the contact and calendar

Table 2: Context components.

Group	Component name	Remarks
Location	GETLOCATION	Android service (AIDL)
	GETDISTANCE	
	VIEWMAP	Google Map API
Contact	GETCONTACT	Content provider API
	SETCONTACT	
Calendar	GETCALENDAR	
	SETCALENDAR	
Device system	GETTELSTATE	Android system API
	SENDSMS	
	TELCALL	
	NOTIFICATION	

Table 3: Communication protocol.

Function	Protocol
Service list (socket/HTTP)	REQUEST|LISTSERVICE RESPONSE|LISTSERVICE|[result] [http://url:port/]servicelist (ex) http://192.168.1.2:7000/servicelist
Service information (socket/HTTP)	REQUEST|GETSERVICE |[id] RESPONSE|GETSERVICE |[result] [http://url:port/]servicedetail?id=
Service execution (socket/HTTP)	REQUEST|EXECSERVICE|[id] RESPONSE|EXECSERVICE|[result] [http://url:port/] Execution?id=&key=

information is handled by the context components "GETCONTACT," "SETCONTACT," and so forth. Furthermore, there are various device functions such as sending an SMS and calling. "GETTELSTATE" finds out the current phone state such as busy, power off, or available.

4.3.3. Interactive Communication Interface. Many researchers have attempted to make decisions via inference. Unfortunately, inference can require considerable computation time, from a few seconds to some days. Furthermore, it only provides a probability instead of an accurate result. The intelligence of our platform depends on explicit context information. Our platform needs to locate a device with useful information and it asks the device about the context information. The platform should interact in a bidirectional manner, and mobile devices can interact actively with each other as if they are talking with each other. Context collaboration can be performed via the interactive communication interface, using network protocol adaption and an interface service. Due to network protocol adaption, our platform works in any network environment such as 3G, Wi-Fi, and Bluetooth, and it has an interface service that can support both socket and HTTP (REST) services. Both the socket and HTTP (REST) capabilities are used for communication among our platforms. The socket protocol outperforms HTTP (REST) because the connection can be maintained for a long time, state information can be stored, and both devices can communicate with each other in any direction whenever they need information. On the other hand, HTTP (REST) communication does not retain a connection and cannot communicate in a bidirectional manner. Socket communication is more suitable for use among our platforms. However, when communicating with an external system involving open APIs, HTTP (REST) should be used for context collaboration. We implemented a server socket function that was provided by web application servers on Android [23]. The operation of a server side socket on iPhones is not permitted by the iOS when iPhone devices are in background mode [24]. Thus, the iPhone needs to be woken up by a push notification in order to communicate via a socket. Table 3 shows the protocol information of our platform in comparison to the socket and HTTP (REST) protocols. Our protocol contains requesting the available service information and executing a service.

4.4. Client Service. The client service asks the provider service to return a list of available composite components by sending a request to execute a composite component. If an available service is found, the client service executes it directly on the mobile device. Our platform includes client support that communicates via a socket interface on mobile devices. The client service can also be provided without client support, since mobile devices and PCs can use the HTTP (REST) interface that is provided by the provider service directly. Composite components can be orchestrated not only with context components, but also with external services such as Google API, Programmable Web, and other open APIs. The interactive communication interface supports communicating with those open APIs via both the SOAP and HTTP (REST) interfaces. As the amount of information is not limited, too much information may cause some failures such as "out of memory" and socket errors.

4.5. Security Service. As the context information on mobile devices is critical, a security service is necessary, and the privacy concerns of users must be answered. Various security services should be provided to users in order to protect them from security threats. First, important information should be encrypted using a strong algorithm such as the Public Key Infrastructure (PKI) algorithm and the triple Data Encryption Standard (DES). Next, users need to gain permission to use context services. Authentication is the act of confirming user identity, and it allows permitted users access to information. Although a simple authentication method employing ID and a password is widely used, the method is extremely weak. Certificated authentication is a stronger option, since the information on the certificate is encrypted by PKI and verified by a public agency. Then, the users should have controlled access to their permitted information. Authorization is distinct from that of authentication. Whereas authentication is the process of verifying that "you are who you say you are," authorization is the process of verifying that

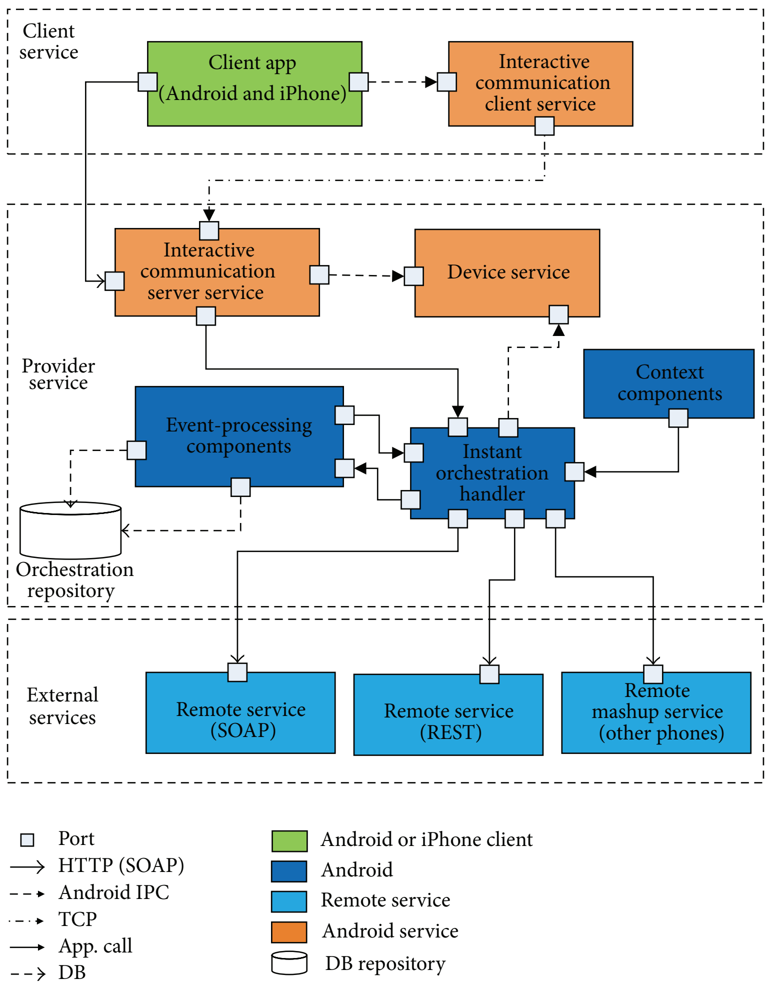


FIGURE 5: The overall architecture of the instant service-orchestration platform.

"you are permitted to do what you are trying to do." Finally, a security service must be managed by a proper security policy.

5. Instant Service-Orchestration Platform Implementation

5.1. Implementation Overview. The provider service, client service, and external service are described by a component and connector diagram [25], as shown in Figure 5. Components are principal units of runtime interactions and data stores, and connectors are interaction mechanisms. The instant service-orchestration platform has various kinds of components such as an Android service, Android activity, open APIs, and a repository. They are connected by ports and communicate with each other using various kinds of protocol. Our platform supports peer-to-peer two-way communication in order to provide both the provider and client services, works as a server for the mobile device service, and gives responses to the client service. If the client service sends a request to the server, the provider service executes its corresponding composite components. Composite components are generated by orchestrating context components, external open APIs, and even other composite components. The provider service provides two server sockets; one is executed by the client socket in the client service, and the other is invoked by the HTTP (REST) socket in a composite component. Therefore, even a PC and workstation can execute the provider service using the HTTP (REST) service. Our platform is implemented using Android SDK r21.1 for Android 4.0.3 (Ice Cream Sandwich) and xcode 4.6.1 for iOS 6. The Android version of our platform has all functions, including the provider service and client service, but the iOS version only has the client service.

5.2. Provider Service. The provider service consists of an instant orchestration handler, an interactive communication interface, and context components. The interactive communication interface has two Android services: the interactive communication server service and the device service, as shown in Figure 6. If the client service sends a request to the server, the interactive communication server service interprets it and executes the proper components. The device service inquires about context information such as the location and phone state of one's own device and receives the results.

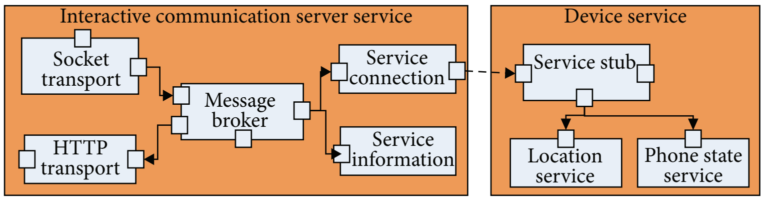


Figure 6: Interactive communication interface architecture.

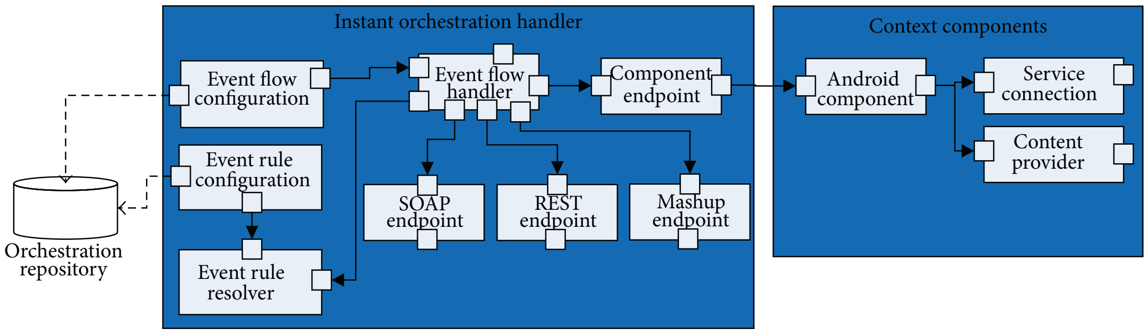


Figure 7: The architecture of the instant orchestration handler and context components.

The architecture of the instant orchestration handler and context components is shown in Figure 7.

The event flow configurator is implemented by Android activity, and it helps in the ease of configuring the XML message. Composite components are made by the event flow configuration and are controlled by the event flow handler. The event rule configurator edits some conditions concerning whether actions should be executed or not. The event flow handler reads the event rule and flow configuration and controls the execution of context components, external components, and other composite components. The service connection or content provider executes the context components. The service connection helps in performing state device functions such as the location and phone state information. Since the state changes continuously, the Android API helps by tracking it periodically every second. The period of tracking can be altered to other values when considering the application performance and battery consumption. Context components can retrieve the state via a service connection. The content provider makes context components access the complex internal databases easily such as those of the contacts, calendar, and so on. Our platform helps in implementing context-based composite applications easily via instant orchestration without complicated programming or configuration. The graphic user interface supports easily adding more components, configuring parameters, and adding event rules. By merely tapping on the screen, the configured information is shown, and an editable new screen pops up. Figure 8 shows an implementation snapshot of the instant orchestration handler.

5.3. Client Service. The client service consists of the client application and the interactive communication client service. The client application displays service information and execution results for the composite components. The communication client service takes the consumer part in the interactive communication interface and communicates with the communication server service via a socket interface. When the communication client service takes a result from the server, the broadcast receiver transfers the result to the client application. Figure 9 shows the client service architecture.

Two versions of the client service are implemented for both Android and iOS. Figures 10(a) and 10(b) show the two versions of client service-implementation snapshots. If user sets parameters and press the "Done" button, a composite component will be executed. If the server finishes executing a composite component, the result is displayed as a dialog box. More detailed information (including the execution result and response time of all the components) is written in logs on the device.

6. The Implementation and Evaluation of Composite Components

6.1. Evaluation Scenarios. Our platform can make up a number of composite components, but we implemented two composite components for evaluation purposes. One is a composite case for calling others with no contact information. It executes three internal context components and two external composite components located in other devices. If this scenario is to be provided for the public, then the contact information must be shared properly by taking into account strict security procedures. The scenario is implemented for evaluation. The other is a fully remote case for making an appointment without disturbance. In this case, the client service requests a provider service to execute a composite component and receives its result from the component. The evaluation of the two cases is implemented on two Samsung Galaxy

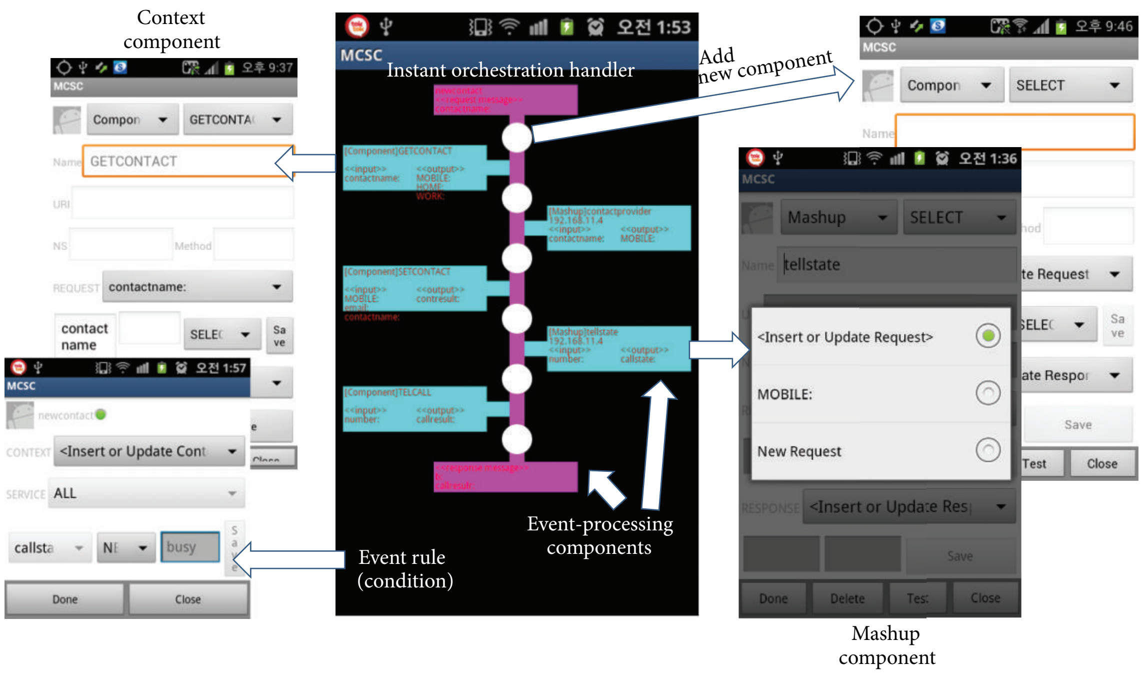


Figure 8: An implementation snapshot of the instant orchestration handler.

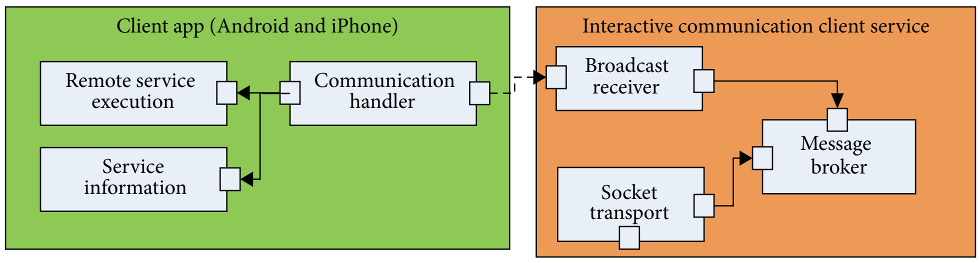
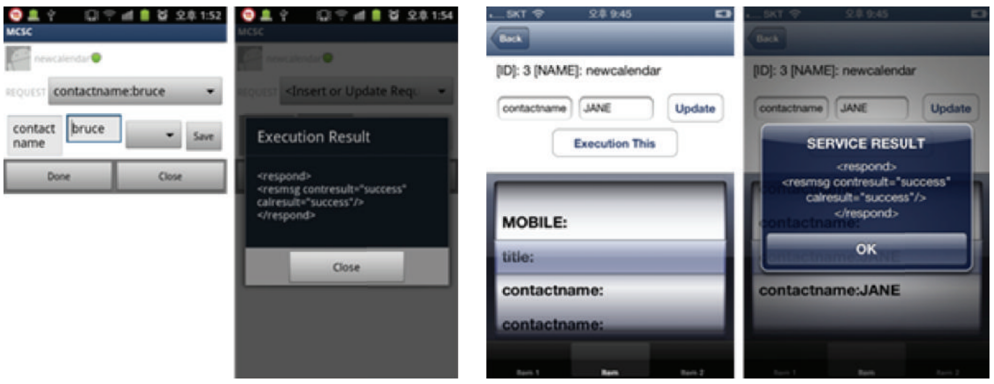


Figure 9: The client service architecture.



```
<respond>
<resmsg contresult="success"
calresult="success"/>
</respond>
```

(a) (b)

Figure 10: (a) Client service-implementation snapshots for the Android operating system. (b) Client service-implementation snapshots for iOS.

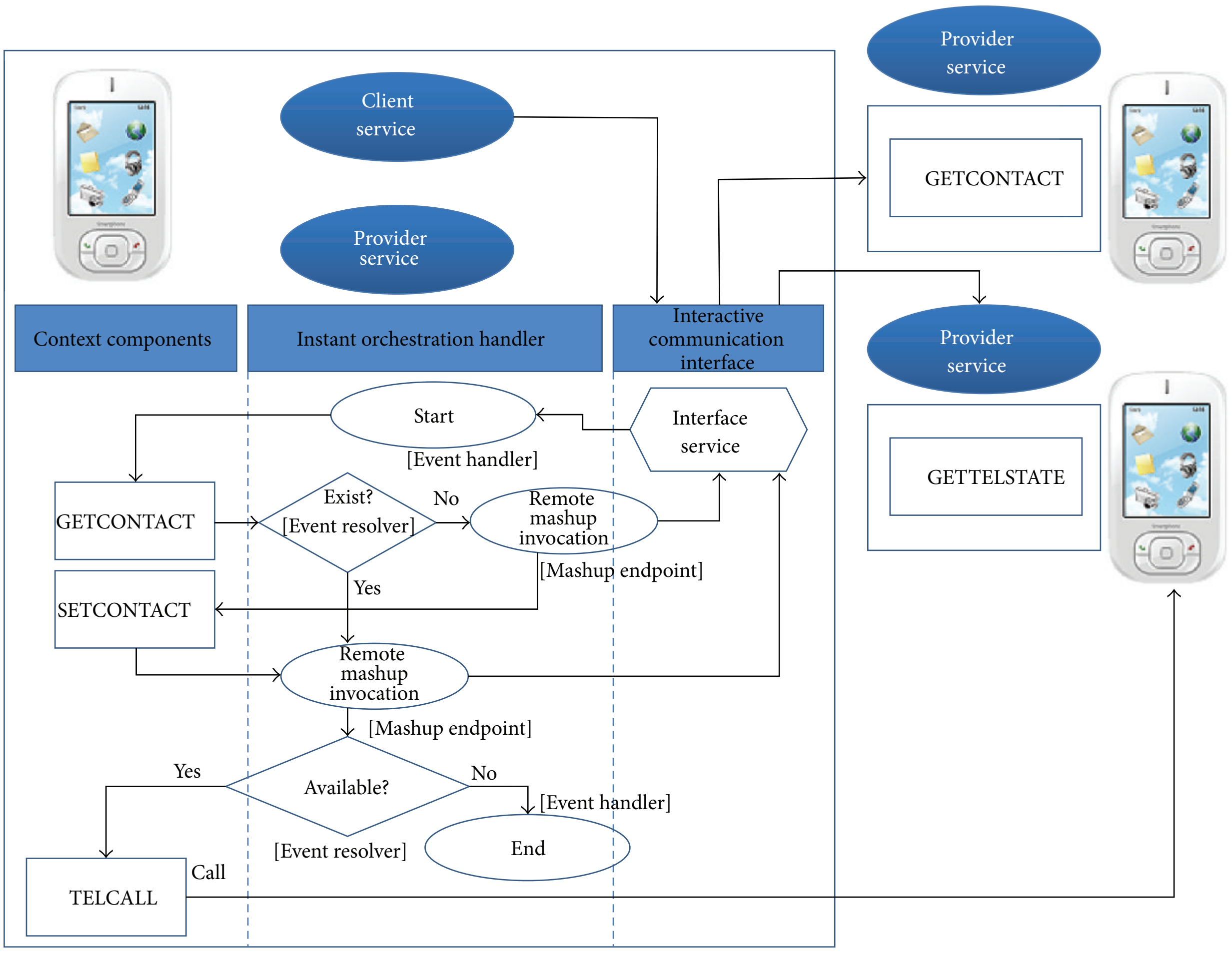
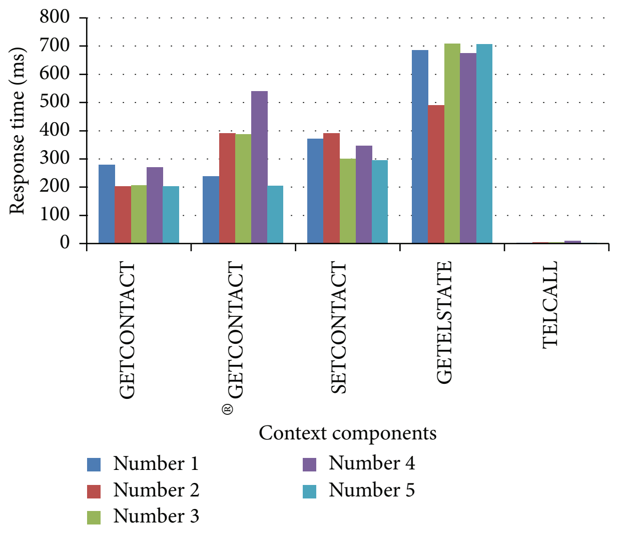


Figure 11: Orchestration flowchart for the composite case.

S2 devices and one iPhone 4. Execution and response times are measured for each case. Additionally, the performance capability is evaluated with an automatic load-test solution.

6.2. Composite Case: "Calling Others with No Contact Information". The composite component consists of three context components and two remote composite components, as shown in Figure 11. When the client service invokes it, the "GETCONTACT" context component is invoked, and it tries to find contact information on its own device. If it does not have contact information, the remote "(R)GETCONTACT" component finds contact information on another device by remote invocation and stores the newly gained contact information by invoking the "SETCONTACT" component. Then, a composite component, "GETTELSTATE," located in another device, is invoked and retrieves the device's state such as busy, unavailable, or available. Finally, if the device state is available, the "TELCALL" component is invoked and makes a call. Two event rules, as shown in the diamond in Figure 11, control orchestration flow and decide whether to execute a component or not depending on the situation. For example, if the phone state is busy, then the "TELCALL" component does not place a call. Three smartphones can be used in reality, but our implementation uses two smartphones, one of which is used for two provider services. "(R)GETCONTACT" should be performed under a privacy policy. Users may require an agreement for sharing information. Thus, an extra approval step needs to be added to the scenario, or users need to be able to control the privacy policy.

Our platform is evaluated on two Samsung Galaxy S2 smartphones. One smartphone executes the orchestration process of composite components while the other smartphone provides two remote composite components such as "GETCONTACT" and "GETTELSTATE." Figure 12 shows two evaluation results executed five times. The first describes the response time of each context component over five different experiments. The execution times of remote composite components such as "(R)GETCONTACT" and "GETTELSTATE" take much longer than those of "GETCONTACT" and "SETCONTACT." The response time of "GETTELSTATE" is the longest, since it is a remote service and is executed by invoking another internal Android service to keep track of the phone state. The execution time for the "TELCALL" component is very short, since it works asynchronously. In other words, it executes its process without waiting for any desired result. The second shows the rate of the execution times for each context component in the composite components for five different experiments. The response time



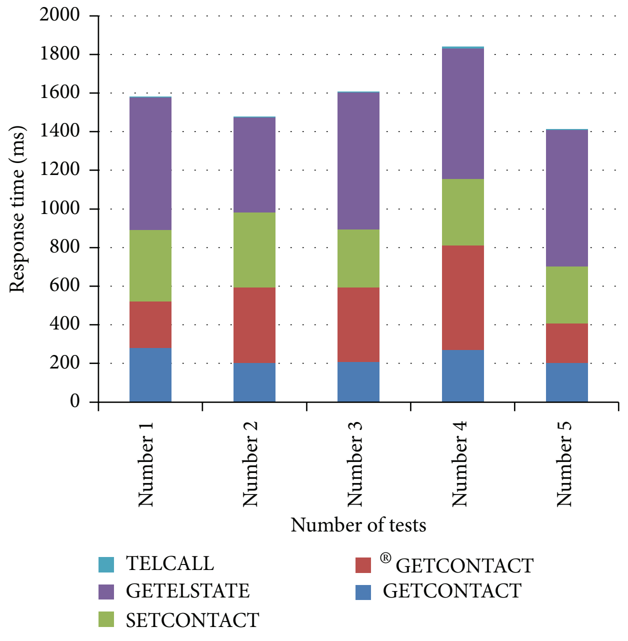


Figure 12: The evaluation results for the composite case.

varies with the experiments, since the resource allocation and task priority of the operating system can be changed continuously. It is difficult to determine whether the result values are unconditional or not.

6.3. Fully Remote Case: "Making an Appointment without Disturbance". The first thing one does when making an appointment is to call someone directly and check his or her schedule. It requires much effort to make an appointment and to remind the individual of one's contact information. However, our platform queries schedule information from other devices regarding whether someone is available or not. Then it automatically creates a new appointment on the other person's device if no appointment has been assigned at a desired time. Figure 13 shows the orchestration flowchart describing the fully remote case.

The "GETCLAENDAR" component finds any booked or available schedules, while the "SETCALENDAR" component registers a new schedule to the device planner database. Then, "GETCONTACT" finds the caller's contact information. If it does not exist, "SETCONTACT" registers new contact information on the device contacts. The remote composite component is evaluated with two devices, similarly to the previous case, but in this case, one works as a server while the other does so as a client. The client service configures some parameter values such as the schedule title, date, and content and executes the composite component. After executing the provider service, the client service receives an execution result. Client services are implemented on both Android and iOS. They are evaluated respectively on a Samsung Galaxy S2 and an iPhone 4. Figure 14 shows two evaluation results for the fully remote case on an Android server and a client

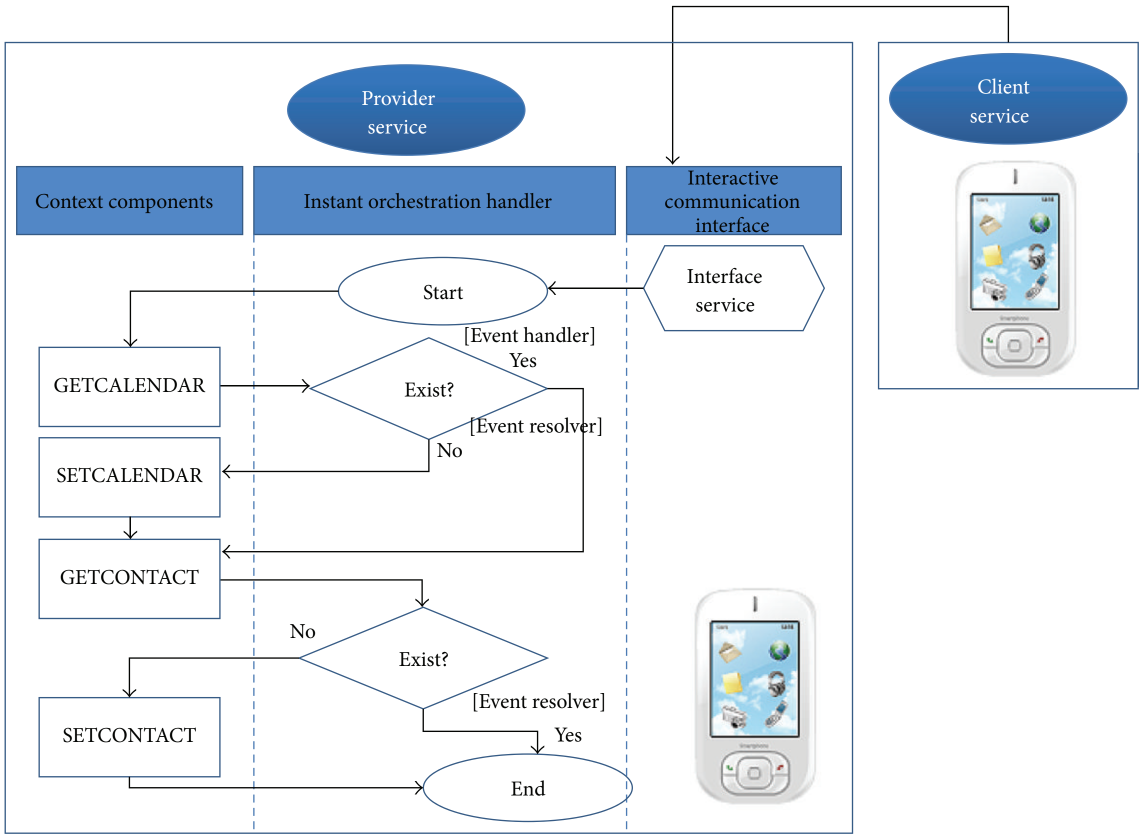


FIGURE 13: The orchestration flowchart describing the fully remote case.

device executed five times. The most execution time is taken by the client service, since the client communicates with the server remotely, but the other components such as "GETCANLENDAR," "SETCALENDAR," "GETCONTACT," and "SETCONTACT" are executed in the local server. Figure 15 shows two evaluation results for the fully remote case on an Android server and an iOS client device. The execution time for the client service is faster than that of the Android client, since the Android application has been developed using Java language and executed within a Dalvik VM in the device, while the iOS application has been developed using Objective-C language, and an interactive communication client interface is implemented via pure C socket communication code instead of by the Objective-C library.

6.4. Performance Evaluation with a Load-Test Solution. We evaluate the performance capability of composite case implementation with an automatic load-test solution, LoadUI [26]. It is an open source project, is easy to use, and has good reporting results in the form of graphs and raw data. Two laptop computers, one wireless router, and two smartphones (Samsung Galaxy S and S2) are used for this evaluation, as shown in Figure 16. One laptop computer generates and sends a heavy load to the smartphones using LoadUI, and the other one monitors the platform logs with eclipse DDMS, that is, the Android device-management plugin SDK.

LoadUI simulates a heavy user's usage with virtual users and undertakes a performance test as if hundreds or thousands of users have accessed a remote target application. It supports various virtual user-generation methods such as random, ramp, fixed rate, and other methods and controls the amount of the heavy load as transactions per second (TPS). A fixed rate maintains an unchanged amount of usage from the beginning to the end, and ramp increases the amount of usage to predefined values for a set time. Web page runner configures the URL address of target applications, displaying the evaluation status such as running, completed, or failed. The performance evaluation is carried out by measuring the response time and throughput (counted as the number of transactions) by sending 5, 10, 15, and 20 TPS, respectively, 5 TPS meaning sending a request 5 times a second. The ramp rate is selected as a virtual user-generation method. Hence, the load amount increases from 0 to 5, 10, 15, and 20. The load amount is similar in the beginning, while it is significantly different at the end of test. We measure response time and throughput with different load cases using Samsung Galaxy S and S2 and show the results in Figures 17 and 18, respectively. The response time is fluctuating due to CPU and memory scheduling based on a round robin policy; that is, more resource occupation leads to a lower response time. As the load amount gets heavier, the response time also increases. In the case of the Galaxy S2, the throughput is almost the same as the amount of input load data; in other

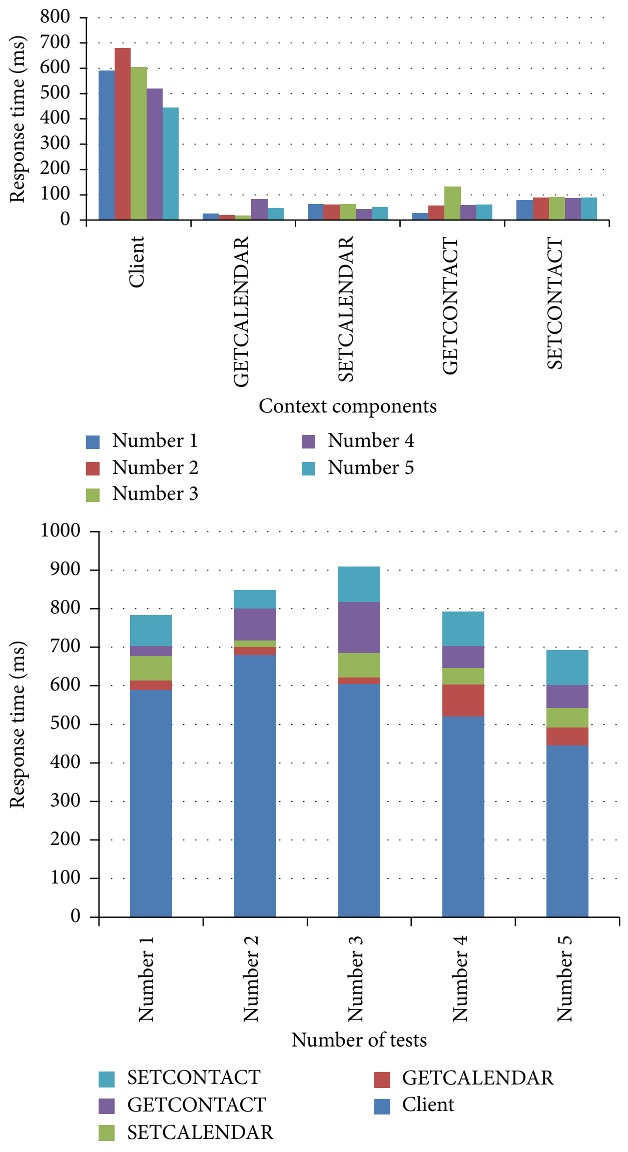


Figure 14: The evaluation results for the Android remote case.

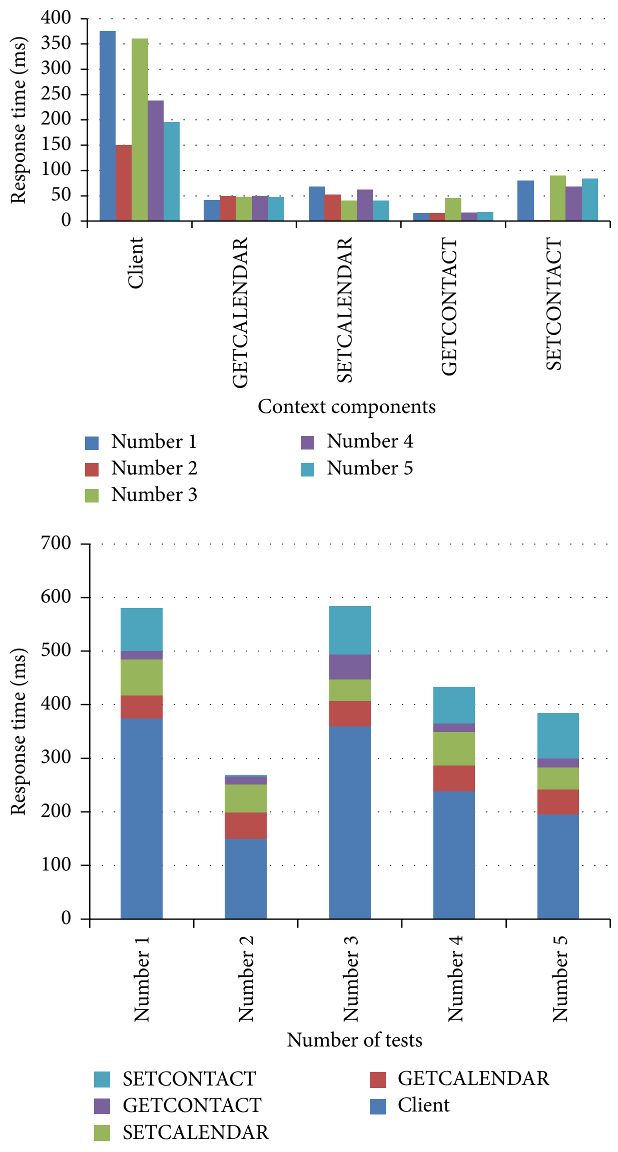


Figure 15: The evaluation results for the iOS remote case.

words, there is no bottleneck in any of the test cases. The response time is lower than 0.5 seconds and the throughput is about 20 TPS at the highest performance rate. Therefore, our platform shows good performance in terms of providing a context collaboration service.

The same performance evaluation was carried out on the Galaxy S (1 GHz Single Core and 512 MB RAM). The Galaxy S has lower CPU power and memory size than the S2 does (1 GHz Dual Core and 1 GB RAM). Even though the evaluation result is similar to that of S2 in the case of 5 and 10 TPS, the response time becomes very large for 15 and 20 TPS, and the execution time is delayed due to accumulated requests. For the 20 TPS evaluation result, the response time increases up to 2000 ms and the throughput almost approaches 0. The lower resources of the Galaxy S device lead to execution time increases and to decreases in throughput.

7. Conclusion

Most users expect that smartphones have the potential for growth and that they will be continuously improved. However, researchers still have not made full use of the useful context information on smartphones. Our research focused on the next innovatory stage for smartphones and M2M technology. In this paper, we have presented a new mobile M2M platform that provides intelligent functions such as instant orchestration, context components, and an interactive communication interface and which works based on the context information of mobile devices. In addition, the platform introduces an innovative concept named "context collaboration" that enables smartphones to exchange mutual mobile device context information with each other. In order to realize context collaboration, we have designed instant orchestration by assembling prebuilt context components, external APIs, and even other composite components. The interactive

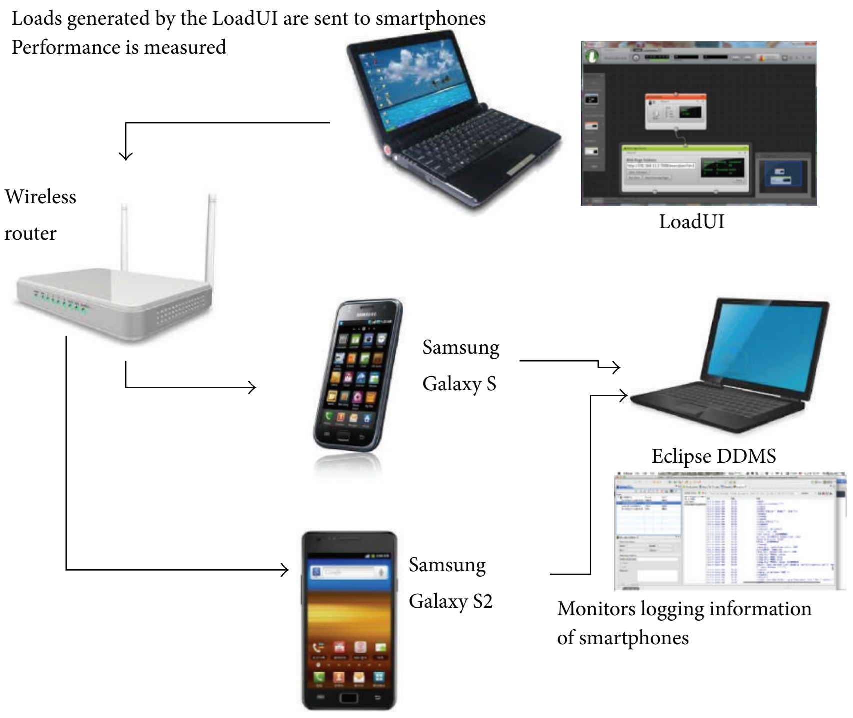

Loads generated by the LoadUI are sent to smartphones
Performance is measured
LoadUI
Wireless router
Samsung Galaxy S
Eclipse DDMS
Samsung Galaxy S2
Monitors logging information of smartphones

FIGURE 16: The performance evaluation environment.

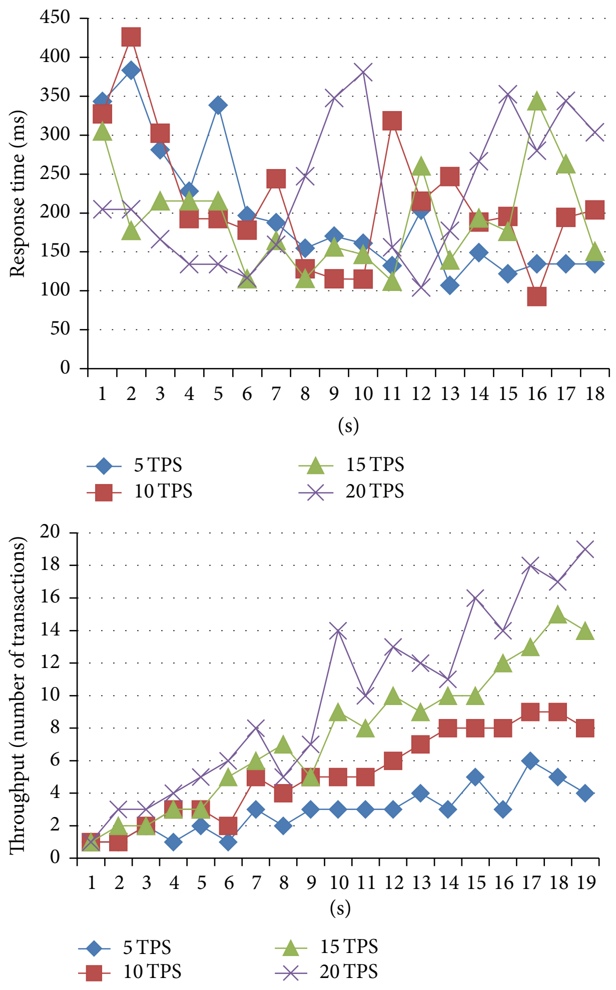
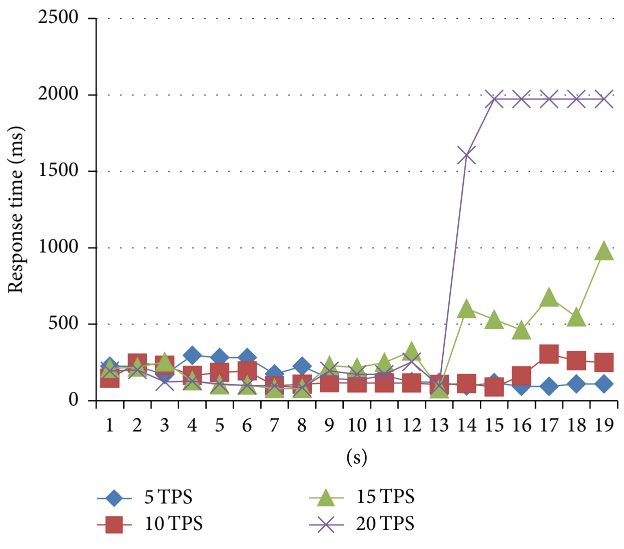


FIGURE 17: The performance test results for Samsung Galaxy S2.



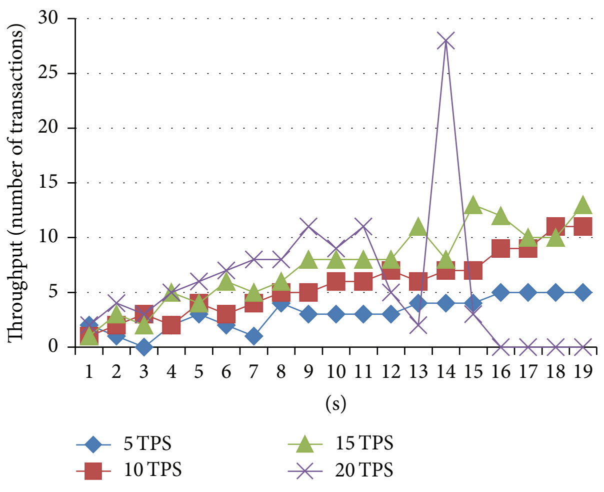


Figure 18: The performance test results for Samsung Galaxy S.

communication interface provided by our platform supports the exchange of messages heterogeneously among various devices such as a smartphone, PC, and workstation via open APIs such as SOAP and HTTP (REST). The platform can make up a number of composite components, but we have shown two composite components for evaluation purposes. In addition, we have shown the results of the performance evaluation for two cases on two Samsung Galaxy S2 devices and one iPhone 4, respectively, by measuring the response time. Furthermore, we have evaluated the performance capability of composite case implementation via an automatic load-test solution on Samsung Galaxy S and S2. We believe that our platform will open up a new and innovative way forward for enhanced mobile context-aware M2M middleware.

Competing Interests

The authors declare that there are no competing interests regarding the publication of this paper.

Acknowledgments

This research was supported by Korea University and the Brain Korea 21 Plus Project in 2016.

References

[1] A. K. Dey, “Understanding and using context,” *Personal and Ubiquitous Computing*, vol. 5, no. 1, pp. 4–7, 2001.

[2] E. Pulier and H. Tylor, *Understanding Enterprise SOA*, Manning, Greenwich, Conn, USA, 2005.

[3] A. K. Dey, G. D. Abowd, and D. Salber, “A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications,” *Human-Computer Interaction*, vol. 16, no. 2–4, pp. 97–166, 2001.

[4] P. Korpipää, J. Mäntyjärvi, J. Kela, H. Keränen, and E.-J. Malm, “Managing context information in mobile devices,” *IEEE Pervasive Computing*, vol. 2, no. 3, pp. 42–51, 2003.

[5] T. Gu, X. H. Wang, H. K. Pung, and D. Q. Zhang, “A middleware for building context-aware mobile services,” in *Proceedings of the IEEE Vehicular Technology Conference*, Milan, Italy, May 2004.

[6] O. Davidyuk, J. Riekki, V.-M. Rautio, and J. Sun, “Context-aware middleware for mobile multimedia applications,” in *Proceedings of the 3rd International Conference on Mobile and Ubiquitous Multimedia (MUM ’04)*, pp. 213–220, ACM, College Park, Md, USA, October 2004.

[7] M. Raento, A. Oulasvirta, R. Petit, and H. Toivonen, “ContextPhone: a prototyping platform for context-aware mobile applications,” *IEEE Pervasive Computing*, vol. 4, no. 2, pp. 51–59, 2005.

[8] J. Chon and H. Cha, “LifeMap: a smartphone-based context provider for location-based services,” *IEEE Pervasive Computing*, vol. 10, no. 2, pp. 58–67, 2011.

[9] L. M. Daniele, E. Silva, L. F. Pires, and M. van Sinderen, “A SOA-based platform-specific framework for context-aware mobile applications,” in *Enterprise Interoperability*, vol. 38 of *Lecture Notes in Business Information Processing*, pp. 25–37, Springer, Berlin, Germany, 2009.

[10] A. Ennai and S. Bose, “MobileSOA: a service oriented web 2.0 framework for context-aware, lightweight and flexible mobile applications,” in *Proceedings of the 12th Enterprise Distributed Object Computing Conference Workshops (EDOCW ’08)*, pp. 345–352, Munich, Germany, September 2008.

[11] Q. Z. Sheng, S. Pohlenz, J. Yu, H. S. Wong, A. H. H. Ngu, and Z. Maamar, “ContextServ: a platform for rapid and flexible development of context-aware web services,” in *Proceedings of the 31st International Conference on Software Engineering (ICSE ’09)*, pp. 619–622, IEEE, British Columbia, Canada, May 2009.

[12] B. van Wissen, N. Palmer, R. Kemp, T. Kielmann, and H. Bal, “ContextDroid: an expression-based context framework for Android,” in *Proceedings of the International Workshop on Sensing for App Phones (PhoneSense ’10)*, Zürich, Switzerland, November 2010.

[13] A. I. Wang, B. Wu, and S. K. Bakken, “CAMF: context-aware machine learning framework for Android,” in *Proceedings of the International Conference on Software Engineering and Applications (SEA ’10)*, Marina del Rey, Calif, USA, November 2010.

[14] A. K. Dey and G. D. Abowd, “Towards a better understanding of context and context-awareness,” in *Proceedings of the CHI*

Workshop on "The What, Who, Where, When, Why and How of Context-Awareness", April 2000.

[15] B. M. Michelson, *Event-Driven Architecture Overview: Event-Driven SOA Is Just Part of the EDA Story*, Patricia Seybold Group, Boston, Mass, USA, 2006.

[16] B. Sriraman and R. Radhakrishnan, *Event Driven Architecture Augmenting Service Oriented Architectures*, Sun Microsystems, Santa Clara, Calif, USA, 2005.

[17] D. A. Chappell, *Enterprise Service Bus*, O'Reilly Media, Sebastopol, Calif, USA, 2004.

[18] E. A. Marks and M. Bell, *Service Oriented Architecture*, John Wiley & Sons, New Jersey, NJ, USA, 2006.

[19] A. Alves, A. Arkin, S. Askary, C. Barreto et al., April 2007, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html.

[20] C.-H. Im and C.-S. Jeong, "MCSC: mobile collaborative service cloud using instant adaptive orchestration and mashup," *International Journal of Software Engineering and Its Applications*, vol. 6, no. 4, pp. 265–272, 2012.

[21] R. Meierm, *Professional Android 4 Application Development*, John Wiley & Sons, New York, NY, USA, 2012.

[22] G. Milette and A. Stroud, *Professional Android Sensor Programming*, John Wiley & Sons, 2012.

[23] B. Kurniawan and P. Deck, *How Tomcat Works: A Guide to Developing Your Own Java Servlet Container*, Boyd Printing Company, 2004.

[24] D. Mark, J. Nutting, J. LaMarche, and F. Olsson, *Beginning iOS6 Development*, Apress, New York, NY, USA, 2013.

[25] P. Clements, F. Bachmann, L. Bass et al., *Documenting Software Architectures, Views and Beyond*, Addison Wesley, Boston, Mass, USA, 2002.

[26] LoadUI, http://www.loadui.org.