Hindawi Publishing Corporation
Mobile Information Systems
Volume 2016, Article ID 3038515, 20 pages
http://dx.doi.org/10.1155/2016/3038515

Research Article

Wearable Device Control Platform Technology for Network Application Development

Heejung Kim,[1,2] Misun Ahn,[2] Seunghyun Hong,[2] SeungGwan Lee,[3] and Sungwon Lee[2]

[1]*Korea Telecom, 206 Jungja-dong, Bundang-gu, Seongnam-si, Gyeonggi-do 463-711, Republic of Korea*
[2]*Department of Computer Engineering, Kyung Hee University, 1 Seocheon-dong, Giheung-gu, Yongin-si, Gyeonggi-do 446-701, Republic of Korea*
[3]*Humanitas College, Kyung Hee University, 1 Seocheon-dong, Giheung-gu, Yongin-si, Gyeonggi-do 446-701, Republic of Korea*

Correspondence should be addressed to Sungwon Lee; drsungwon@khu.ac.kr

Received 4 August 2015; Revised 24 November 2015; Accepted 6 January 2016

Academic Editor: Yassine Hadjadj-Aoul

Application development platform is the most important environment in IT industry. There are a variety of platforms. Although the native development enables application to optimize, various languages and software development kits need to be acquired according to the device. The coexistence of smart devices and platforms has rendered the native development approach time and cost consuming. Cross-platform development emerged as a response to these issues. These platforms generate applications for multiple devices based on web languages. Nevertheless, development requires additional implementation based on a native language because of the coverage and functions of supported application programming interfaces (APIs). Wearable devices have recently attracted considerable attention. These devices only support Bluetooth-based interdevice communication, thereby making communication and device control impossible beyond a certain range. We propose Network Application Agent (NetApp-Agent) in order to overcome issues. NetApp-Agent based on the Cordova is a wearable device control platform for the development of network applications, controls input/output functions of smartphones and wearable/IoT through the Cordova and Native API, and enables device control and information exchange by external users by offering a self-defined API. We confirmed the efficiency of the proposed platform through experiments and a qualitative assessment of its implementation.

1. Introduction

The market for smartphones has experienced explosive growth since the development of Apple's iPhone and Samsung's Omnia2 to the extent that smart devices have now become vital to daily life. Furthermore, wearable devices have attracted considerable attention as the next generation of mobile technology that will replace smartphones. The research service BI Intelligence has predicted that the market for wearable devices will grow to approximately $12 billion by 2018 [1].

A "wearable device" refers to a small electronic device that can be worn on the body so that a user can freely use it even when moving. Google Glass, Samsung Galaxy Gear, and Sony SmartWatch belong to this category. Furthermore, even sports equipment companies such as Nike and Adidas have lately begun introducing innovative products and services in the wearable device market [2].

The Internet of Things (IoT) is a recent technology for collecting data and transferring data through sensors and adding a communication function to every object. The market size of IoT was estimated to be $203.1 in 2013 and is expected to reach $1 trillion, with an average annual growth of 21.8%, by 2022 [3]. In December 2013, subscribers of smartphones exceeded 37.5 million in Korea [4]. Along with the popularity of smartphones and wearable devices, the market for mobile applications is steadily growing as well [5].

Cross-Platform Development. Due to the emergence of smartphones and the existence of multiple platforms, developers in the stage of application development have to build platform-specific environments, use multiple programming languages,

and learn the relevant supporting application programming interfaces (APIs). These constraints lead to wasted time and effort and increase the cost of application development. In order to solve such problems, mobile programming is in the process of standardization. Cross-platform development frameworks, such as Cordova [6] and Titanium [7], have garnered considerable attention as a solution. The Association for Computing Machinery (ACM), the Institute of Electrical and Electronics Engineers (IEEE), and related organizations are publishing an increasing number of studies on platform research that analyze the characteristics of cross-platform development frameworks. This suggests that application development on a single, unified platform will become possible in the near future [8, 9].

Research Challenges and Contributions. Cross-platform development is an attribute that assists the development of applications using web languages such as HyperText Markup Language 5 (HTML5), Cascading Style Sheets (CSS), and JavaScript. Its greatest benefit is that it provides an application with a single source that is feasible on multiple mobile platforms, which increases development productivity. However, the range of APIs in Cordova and Titanium is not sufficiently wide, especially with regard to supporting wearable devices and IoT devices, since they are still in their developmental stages. Developers are thus inevitably required to learn the native language supported by the platform of the relevant device and initiate follow-up development because cross-platform development alone is insufficient for application development.

Furthermore, wearable/IoT devices support Bluetooth technology for communication among them. Bluetooth is a communication technology over short distances, due to which it is impossible to communicate with wearable/IoT devices beyond a limited range [10].

In this regard, we propose a "Network Application Agent" (NetApp-Agent) platform that integrates a development environment for wearable devices and supports Internet Protocol-(IP-) based communication. NetApp-Agent is a smart device platform for network application development that allows outside users to control input/output (I/O) functions and exchange data. This is because it is based on the Apache Cordova platform and uses Cordova API and a Native API that enable the I/O function in smartphones and wearable devices as well as the I/O function control of IoT devices by providing a self-defined API.

The potential benefits of our proposed platform are as follows. First, it eases application development by supporting integrated development environments that supply essential APIs or facilitate device development. Second, it enables wearable devices that use Bluetooth to communicate with the outside by supporting IP communication based on WebSocket.

The outline of the remainder of this paper is as follows. Section 2 examines the benefits and drawbacks of the existing development platforms. In Section 3, we introduce NetApp-Agent, our proposed integrated development platform, together with its structure and features. Section 4 presents the results of our experiments involving NetApp-Agent as well as a qualitative assessment of its implementation. Finally, we offer our conclusions and recommendations for further research in Section 5.

2. Review of Existing Approaches and Issues

Approaches to mobile application development can be divided into three major categories: native applications (native apps), web applications (web apps), and hybrid applications (hybrid apps). Native apps involve application development on the platform of each device, whereas web apps utilize HTML, JavaScript, and CSS. Hybrid apps assume the form of native apps, but all or part of its internal configuration is developed in a web app environment [11–13].

In this section, we examine the development of native and hybrid apps together with the features and challenges of a network application development system that provides an IoT development platform and cloud service.

2.1. Native Development Platform. A native development platform involves developing applications on the platform of each device, such as the iPhone, the Android phone, and the Windows Phone, which operate in machine language code. It ensures optimized application performance. However, it has a few major disadvantages: it builds a different development environment for each platform, and the developer needs to learn the relevant development language and software development kit (SDK). We examine the native development environments for both the Sony and the Pebble SmartWatch, which are representative of wearable devices.

2.1.1. Sony SmartWatch. Sony SmartWatch is based on Android 4.0 operating system, and Java is used as its development language. A developer cannot check his/her developed screen on the actual device, but on a computer with a separate emulator. The procedure for developing applications for the Sony SmartWatch is as follows [14].

A development environment must first be constructed. Sony SmartWatch provides the Sony Add-on SDK as an additional installation to the existing Android SDK. Therefore, the Android's development environment needs to be built in advance for SmartWatch development. The Java Development Kit (JDK) and the integrated development tool Eclipse are installed in order to create the Java Runtime Environment.

Following this, the Android Development Tools (ADT) plugin is installed on Eclipse to support the Android system together with the Android SDK. The Android development environment is then constructed [15]. The development environment for the SmartWatch is finally constructed after downloading the Add-on SDK from Sony's developers' website and installing it on Eclipse.

Furthermore, the system structure and the API supporting the development need to be learned. The system architecture of Sony SmartWatch is shown in Figure 1 and can be divided into three major components: Smart Extension, Host Application, and Accessory. Smart Extension refers to the application to perform in wearable device.

Table 1: Smart Extension API of Sony SmartWatch.

API	Description
Registration and Capabilities API	Provide API data of SmartWatch/Smart Extension
Notification API	Notify the event that occurred in smartphone to the device
Control API	Control the display of device
Sensor API	Transmit sensor data
Widget API	Preview contents

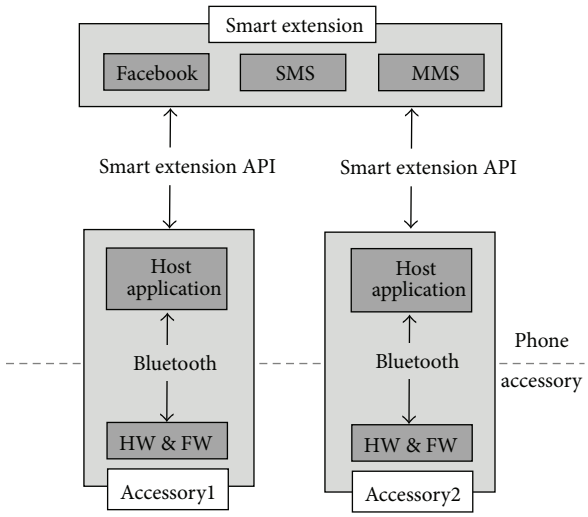

Figure 1: System architecture of Sony SmartWatch.

This application utilizes the Smart Extension API to communicate with the Host Application of the wearable device. The Host Application is installed in smartphones and connected to the wearable device using Bluetooth technology. The Smart Extension API for the development of Sony SmartWatch must be learned in sequence.

Table 1 lists various supporting APIs. The Registration API and the Capabilities API are employed when the Host Application provides data for the API of the SmartWatch or Smart Extension provides its data on the API to the Host Application. The Notification API is used when the Host Application notifies the event occurred in smartphones to device, whereas the Control API enables Smart Extension application to the control display of the device. Control API in particular is crucial to controlling the display or the light-emitting diode (LED) of the device and to processing key events or touching events that require close attention. The Sensor API transmits data from the accelerometer and the illumination sensor of the device to the Smart Extension application. The Widget API affords content preview.

The lengthy and complex procedure described above concludes the preparation for the development of an operating application for Sony SmartWatch. However, information regarding implemented classes, functions, and variables still needs to be checked, and extensive research needs to be conducted on the API implementation code and the sample code, along with code analysis, by using API reference documents for application development.

2.1.2. Pebble SmartWatch. The Pebble SmartWatch functions on an independent Pebble operating system (OS) that is compatible with both Android and iOS. Pebble supports different types of languages, such as JavaScript and Objective-C, where the latter is primarily used for application development. Moreover, a Pebble SDK is provided for development. The procedure for developing applications for Pebble SmartWatch can be divided into two parts.

First, the application development environment must be constructed. Pebble SDK can be installed on Mac OS X and Linux. We assume the construction of a development environment conducted on Mac OS X. The Pebble SDK is downloaded from the Pebble developers' website and the Pebble ARM Toolchain is installed after the installation of Xcode Command Line Tools [16], which is a developers' command line tool. Development environment construction is completed after building in Python library because Pebble SDK is based on Python. Pebble is compatible with smartphones that use Android and iOS platforms. Therefore, the development environment of a smartphone application should be built and an SDK called PebbleKit should then be installed to create an application that is in sync with the smartphone [17].

Second, a development support API must be learned. The Pebble API consists of Pebble Watch App SDK (for SmartWatch applications), PebbleKit Android (for Android), and PebbleKit iOS (for iOS). In order to develop an application that is compatible with smartphones that use Android OS, one needs to learn Watch App SDK and PebbleKit for Android. The scope of the supporting SDK is presented in Table 2.

When developing a Pebble application, two issues need careful scrutiny in addition to the construction of the development environment and the examination of the supporting APIs. First, the Pebble SmartWatch does not support Korean characters. Hence, expressions in Korean need to be considered when developing an application that sends text messages or notifications for Social Network Services (SNS) to Pebble devices in Korean. Second, the image format of Pebble is problematic. Pebble uses its own image format, called Pebble Binary Image (PBI), when displaying an image on the screen. PBI represents each pixel using one bit that contains image information in the header file. Thus, a developer must create a tool for image conversion to enable images in Pebble Watch.

The relevant Pebble Watch application is then installed on a computer terminal by the way of inputting build and install command. Pebble Watch should be connected to a smartphone through Bluetooth for application installation, following which the computer and the smartphone that have already progressed in development should be connected to the same Wi-Fi network. Therefore, developers should pay particular attention to network configuration when installing the application.

Table 2: Watch App SDK of Pebble SmartWatch.

Watch App SDK Pebble SmartWatch	
Foundation	(i) App (ii) Media Utilities (iii) Timer (iv) Wall Time (v) Math (vi) Dictionary (vii) AppMessage (viii) Resources (ix) AppSync (x) Logging (xi) App Communication
Graphics	(i) Graphics Context (ii) Drawing Primitives (iii) Graphics Types (iv) Drawing Text (v) Fonts
User interface	(i) Layers (ii) Animation (iii) Window (iv) Vibes (v) Light
Standard C	(i) Math (ii) Memory (iii) Format (iv) String (v) Time

2.1.3. Problems with Native Development Platforms. In the current mobile market, several mobile platforms exist, for example, Android, iOS, and Windows Mobile. Thus, manufacturers produce smartphones on a variety of platforms. The coexistence of different platforms has led developers to establish suitable development environments for each platform when creating applications and learn the relevant development languages, the SDK, and the API.

Table 3 shows diverse development environments according to types of wearable device. As shown in the table, a developer needs to learn ten programming languages and seven APIs for an application adaptable to four devices. With the growing trend of wearable devices, more and more devices are expected to be introduced. Moreover, time spent on application development will increase in proportion to the number of devices. To solve these kinds of problems, an integrated development platform is required.

2.2. Cross-Platform Mobile Development Framework. Table 4 shows that the development of mobile applications can be divided into three types: native app, web app, and hybrid app.

As shown in Section 2.1, native apps guarantee optimized application performance. However, they have a few constraints given that they require building a different development environment for each platform and that the developer needs to learn the relevant development language and the SDK.

Application development in web apps is based on widespread Internet technologies, such as HTML, JavaScript, and CSS. The advantage of web apps is that they can attract and train developers, since learning a development language is relatively easy. On the other hand, difficulty in hardware control, slow speed of applications, and vulnerability on networks are the major weaknesses of web apps. Applications developed by hybrid apps assume the form of native apps, but all or part of their internal configuration is developed in web app. The final form of the application is a binary file, which has the same file extension but is developed using web languages, such as HTML5, CSS, and JavaScript.

Hybrid apps improve development productivity because they can be operated in various mobile platform using a single source. Hardware control is also possible. The appropriate incorporation of two apps' advantages brings forth a new development strategy. Cross-platform is applied as a development tool for hybrid apps. In this section, we discuss Cordova and Titanium, two typical instances of cross-platform development.

2.2.1. Cordova. Cordova is an open-source framework that enables hybrid application development. It was first developed as "Phone Gap" by Nitobi and was subsequently taken over by Adobe in October 2011. Following the takeover, it reinforced the open-source policy with the development of the Apache license, and then he changed the name of the application to Cordova from version 1.4 onward. Additional functions for Cordova are developed as plug-ins that are shared in open-source communities. Moreover, Cordova supports seven smartphone platforms, Android, BlackBerry, Firefox OS, iOS, Windows Phone, Windows 8, and Tizen, with high product quality that renders it the most competitive among cross-platforms [18, 19].

The structure of Cordova's applications is shown in Figure 2. The developer creates applications using HTML5, CSS, and JavaScript. The completed codes are then packaged through the Cordova library. The application in packaging is distributed to the device in which web-kit provided browser is equipped.

Cordova provides APIs shown in Table 5. The APIs are called by JavaScript, whereas the JavaScript engine exchanges data with the native engine using string type. However, Cordova has a limited range of APIs because of its incomplete platform. With regard to network APIs, it only checks the status of the connection to Wi-Fi or cellular data. APIs related to Bluetooth are not yet available. As a consequence, implementation through the native language is inevitable in order to call particular functions, even though the application is developed using the Cordova platform [6].

2.2.2. Titanium. Titanium is a cross-platform development framework created by Appcelerator. Unlike Cordova which concentrates on the mobile application development, it is possible to develop desktop applications using Titanium Studio in addition to mobile applications. We focus on cross-platform mobile application development in this paper.

Figure 3 shows the mobile application development process using Titanium Studio. Titanium delivers a development tool called Titanium Studio. Therefore, application

Table 3: Development environments for each wearable device.

Types of device	Operating system	Supporting languages	SDK/API
Google Glass	Android 4.0.4	Go, Java, .NET, PHP, Python, Ruby	Google Mirror API
Sony SmartWatch	Android 4.0	Java	Sony Add-on SDK/Smart Extension API
Pebble SmartWatch	Pebble OS	C, JavaScript (Objective-C/Java)	Pebble SDK PebbleKit
Samsung Gear	Tizen OS	Java, HTML, JavaScript	Tizen SDK Samsung Mobile SDK
Total	**4**	**10**	**7**

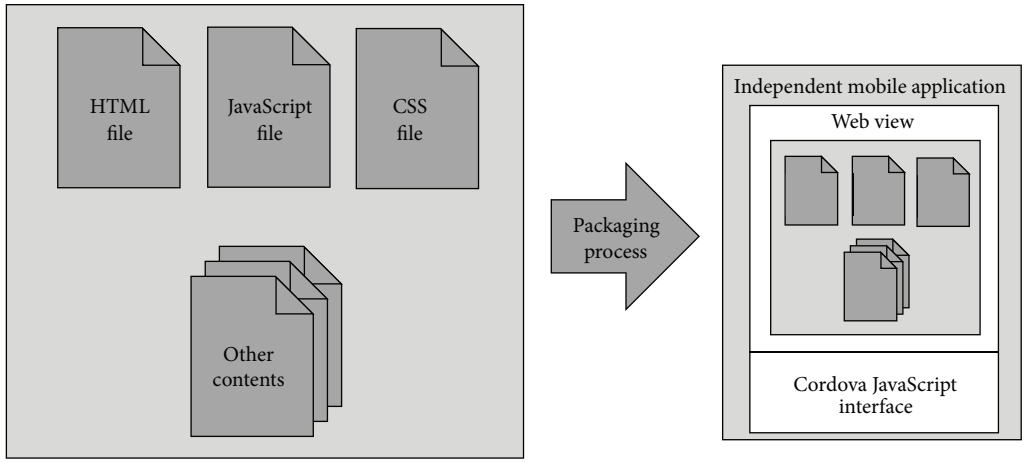

Figure 2: Structure of Cordova.

Table 4: Features of each mobile application development method.

Feature	Native App	Hybrid App	Web App
Development language	Native only	Native and web or web only	Web only
Code portability and optimization	None	High	High
Access device-specific features	High	Medium	Low
Leverage existing knowledge	Low	High	High
Advanced graphics	High	Medium	Medium
Upgrade flexibility	Low	Medium	High
Installation experience	High	High	Medium

development begins after the construction of the development environment by installing Titanium Studio. The application development code is written in HTML and JavaScript and utilizes the interpreter method in case of code translation. In short, Titanium Bridge, which is embodied in Titanium SDK, substitutes Titanium API to a native development language when the developer calls a Titanium API in an application written in JavaScript. Thus, it creates an almost identical product with the application written in native code. Nonetheless, the spectrum of the supporting platform is quite narrow because it is difficult to format a JavaScript engine for the substitution of the development language into each platform [7, 20].

Table 6 presents the list of Titanium mobile APIs. Titanium has a limited domain of supporting APIs, like Cordova. From a communication aspect, it only supplies APIs for socket and HTTP client production and communication. The range of the hardware module control is restricted to camera, audio, and video control [21].

2.3. Internet of Things (IoT) Development Platform. IoT refers to a technology that collects and transfers data by installing sensors and adding network connectivity to every object. To satisfy rising user demand, it generates massive amounts of information through smart sensors installed in smart devices such as smartphones, tablet PCs, and smart TVs. The development of mobile devices of every kind together with built-in sensors has ushered in the age of IoT [22].

Smart devices, such as smart TVs, smartphones, and wearable devices, communicate by forming a network structure in IoT environment. Bluetooth, Wi-Fi, ZigBee [23], and Near Field Communication (NFC) [24] are typical close-range wireless communication technologies used for communication among IoT devices. Bluetooth Smart, widely known as Bluetooth Low Energy (BLE), is particularly widely used.

Table 5: Cordova plug-in API.

API	Description
Battery Status	Check battery status of device
Camera	Take picture and browse gallery
Contacts	Search contacts and add/edit contacts
Device	Provide information about device
Device Motion (Accelerometer)	Provide information on accelerometer sensor
Device Orientation (Compass)	Provide information on compass sensor
Dialogs	Show notification of device
FileSystem	Access file system of device
File Transfer	Receive and transfer file
Geolocation	Provide information on location
Globalization	Provide international expressions
InAppBrowser	Run new application browser
Media	Record and play voice file
Media Capture	Capture media files
Network Information (Connection)	Provide information on network status and connectivity to cellular data and Wi-Fi
Splashscreen	Show/hide start screen of application program
Vibration	Generate vibration on device
StatusBar	Hide/configure status bar background
Whitelist	Whitelist network requests
Legacy Whitelist	Use the old style of whitelist

Table 6: Titanium mobile API.

API	Description
App	Provide information on application and system event
Calendar	Access native calendar
Cloud	Access ACS (Appcelerator Cloud Services)
Contacts	Search contacts and add/edit contacts
Database	Access to SQLite DB within application and produce DB
Facebook	Support application connection to Facebook
Filesystem	Access device's file and folder
Geolocation	Provide information on device's location
Map	Produce native map
Media	Call out media-related function of device (Audio, Video, ImageView, Camera, and Photo Gallery API)
Network	Produce Socket, HTTPClient, TCPSocket, and support communication
Platform	Access function per device's platform (check battery status)
UI	Form UI of application
XML	XML-based content parsing

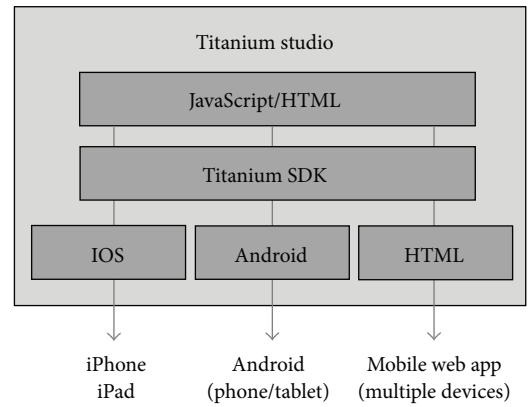

Figure 3: Mobile application development process using Titanium Studio.

The structure of Bluetooth Smart consists of traditional Bluetooth, Bluetooth Smart Ready, and Bluetooth Smart, as shown in Figure 4. Bluetooth Smart Ready is in the form of a hub device that can be connected to both traditional Bluetooth and Bluetooth Smart devices. Yet, Bluetooth communicates through Mac addresses when a device does not have an IP. This leads to a problem where deviating from the given range renders communication among devices impossible.

2.4. Network Application Development. Cloud computing refers to a structure of computing systems where I/O operations are carried out through a user's device, but information analysis, process, storage, management, and distribution are accomplished in another space called a "cloud." Figure 5 shows an outline of this structure [25].

Cloud technology can be divided into three parts, Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS), from the perspective of service provision. A representative service of IaaS is Amazon's EC2, which provides server computation and hardware storage online [26]. PaaS represents platform supply for host and enterprise services. PaaS makes use of resources of IaaS to support certain platforms, including Microsoft's Azure and Google's App Engine [27, 28]. SaaS is a basic technology that supplies user-driven service. It can be divided into applied software service, web-based service, and component-based service. The sole function of several SaaS companies that research maps, images, videos, documents, mail, and so forth is service, and they distribute representational state transfer- (REST-) based Open APIs, numerous platforms, and development languages supporting SDK. IT companies, such as Google, Facebook, Baidu, Kakao, and NAVER, cater to many types of Open APIs for developers [29, 30].

For cloud service development, Amazon offers smartphone platforms for Android and SDKs for different programming languages (Java, .NET, PHP, Ruby, etc.) at the same time [31]. The Azure platform also provides program APIs in the structure of .NET, Node.js, Java, and PHP [32]. SaaS companies distribute REST-based Open APIs and development languages supporting SDK. However, most SDKs and

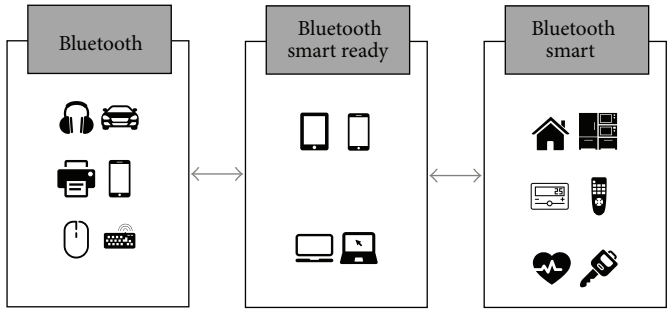

Figure 4: Structure of Bluetooth Smart.

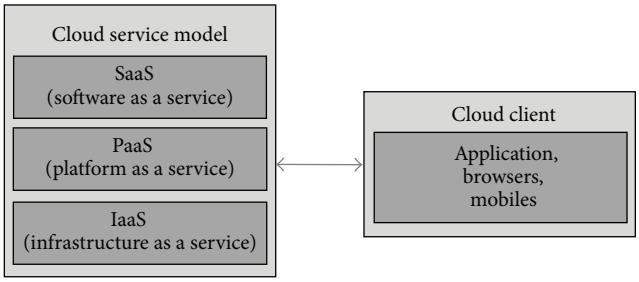

Figure 5: Structure of cloud service model.

APIs focus on establishing cloud servers. Thus, the client application needs to be separately implemented on each platform of a service-providing device where a native app is used. This implies that a network application that provides cloud service must construct a development environment for each platform and learn the development language to create an application, in a similar manner to the native app development process.

3. Proposal

Sections 1 and 2 examined the characteristics of and challenges faced by the existing development platforms. In Section 3, we suggest a solution for the foregoing problems and discuss the detailed structure of our proposed platform, followed by a discussion of scenarios to which the technique is applicable.

3.1. Wearable Device Control Platform

3.1.1. Network Application Agent (NetApp-Agent). In this paper, we propose a wearable device control platform called Network Application Agent (NetApp-Agent) for network application development. Figure 6 shows our proposed platform diagram. NetApp-Agent, in the form of a smartphone application, controls the I/O function of the smartphones, its connected wearable devices, and IoT devices by following commands from a remote controller. In order to do this, a self-defined JavaScript Object Notation- (JSON-) based NetApp-Agent API equipped with WebSocket is provided to the user for remote control.

3.1.2. Network Application Agent API (NetApp-Agent API). The user connects to NetApp-Agent through an IP. NetApp-Agent controls the I/O functions of smart devices or linked wearable devices and requests the necessary information through supporting APIs. Therefore, the proposed platform allows the developer to easily create a device without background knowledge of wearable or IoT devices. Outside users can be any programmable device that wants to communicate through IP-based cloud server, tablet PC, or smartphone using a supporting API.

3.1.3. Structure of Proposed Platform. Figure 7 shows the detailed structure of our proposed platform. NetApp-Agent aims to be a cross-platform development framework that can run applications regardless of the type of mobile platform in question. However, as mentioned before, Cordova's API is limited in its range of support. Our proposed platform amplifies the range of APIs by the binary use of Cordova and Native device wrappers. Functions such as battery check, vibration, and acquisition of global positioning system (GPS) information are developed with the API, whereas I/O functions such as sensors and display are developed with the Native API. For network connection control, Cordova provides an API for receiving cellular and Wi-Fi connection information. Thus, cellular communication and Wi-Fi connection information is implemented using Cordova API, and connection control, such as obtaining Bluetooth connection information and simple On/Off functions of the network, is implemented by using Native API. Wearable devices and IoT devices do not support Cordova API, and thus related functions are established through the Native API. The established functions are controllable by a self-defined JSON-based NetApp-Agent API, where NetApp-Agent provides its implemented API to the outside user.

The WebSocket in NetApp-Agent allows IP-based communication between the device and the user. The outside user thus has remote access to control I/O functions of the device. The inventory database saves the device's connectivity and specification information related to the smartphone. If remotely requested to connect, NetApp-Agent provides its stored information to the outside user. The user then confirms the desired device to control. An intelligence-processing module supports the establishment of intelligible operations in the device. Pebble's SmartWatch does not

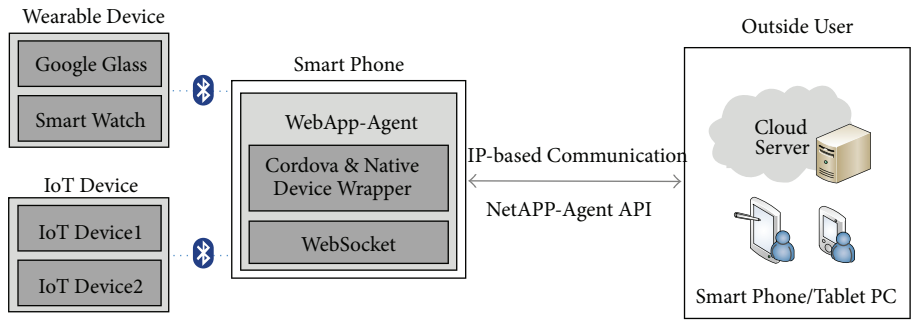

Figure 6: Proposed platform diagram.

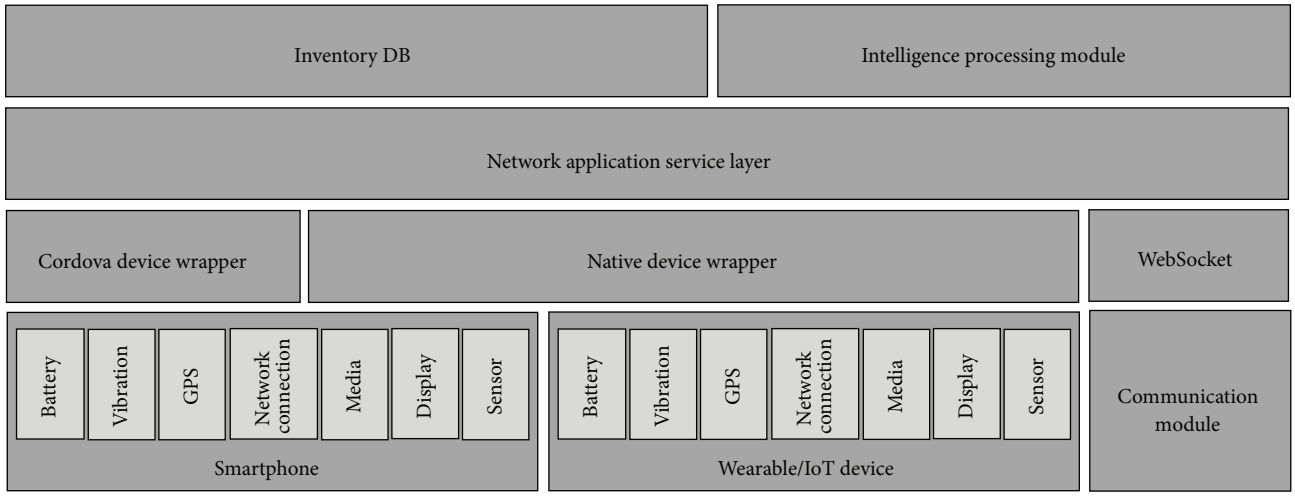

Figure 7: Detailed structure of proposed platform.

support Korean characters and uses a self-defined Perceptual-Backdrop Image (PBI) format for images. As a consequence, if an outside user sends a message containing Korean characters, NetApp-Agent automatically transmits it after translating it into an image. NetApp-Agent does not simply read or write the information of the device connected to the smartphone, but it provides intelligent service through the intelligence-processing module.

3.2. Cloud Service Application Scenario for Proposed Platform. Figure 8 shows the interface of interaction between NetApp-Agent and the cloud server. There is a network application in the cloud server for communication with NetApp-Agent.

The network application uses the NetApp-Agent API for the cloud server to supply the service demanded by the user without modification. We examine here the application of the proposed platform to a cloud service scenario. Possible scenarios are divided into three parts according to their function.

3.2.1. Acquisition of QR Code Information Using Cloud Server. Figure 9 shows a scenario concerning the analysis of Quick Response (QR) code spotted in the camera of a wearable device. The code is processed in the cloud server, and the extracted information is sent back to the wearable device for display.

The video containing the QR code is sent to the cloud server. Images are extracted from the video and form the basis of QR code recognition. The cloud server simultaneously runs various code extraction methods to improve QR code recognition rate. There are three methods to recognize QR code: the traditional method, recognition using the image of the object itself, and an analysis of the similarity of images. In the analysis of similarity, information related to previously stored QR images and location information on the product by beacon are used together.

The information extracted from the QR code goes through a series of processes to be displayed on the screen of the wearable device. In this scenario, the wearable device is only used as a tool for I/O, while the extraction and processing of information from the QR code are carried out in the cloud server. The cloud server can implement an application that displays the information on the wearable device using the NetApp-Agent API without having to develop a native app.

3.2.2. Voice and Video-Sharing Scenario Involving Smart Device and Cloud Server. Figure 10 shows a scenario for voice and video sharing between a smart device and the cloud

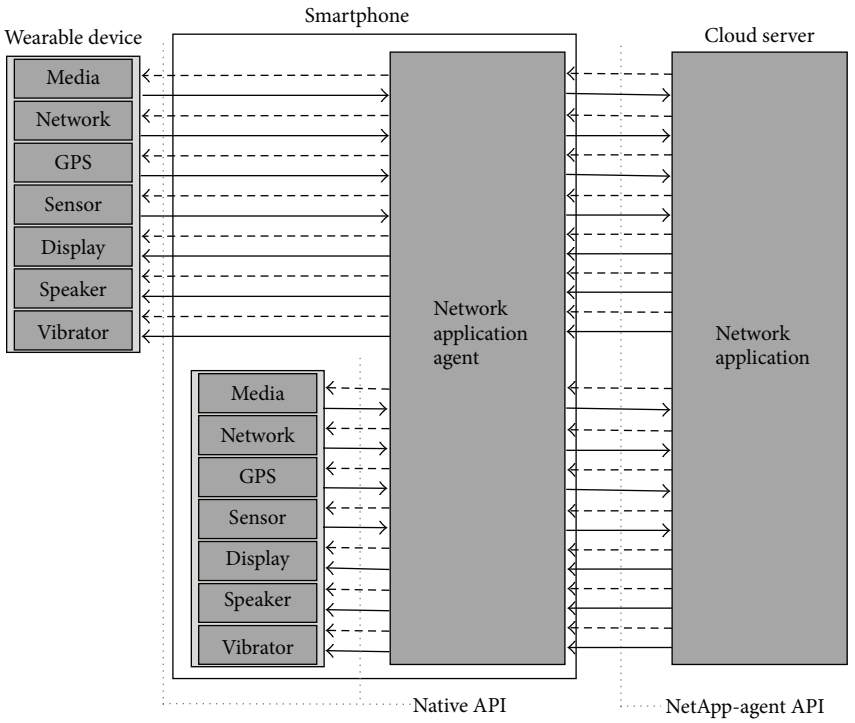

Figure 8: The NetApp-Agent interface.

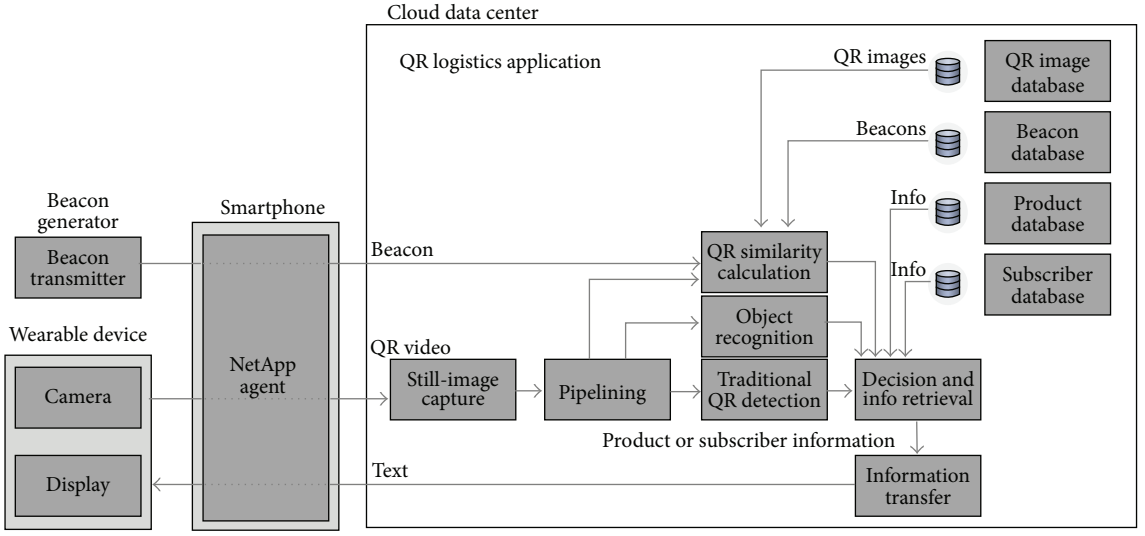

Figure 9: QR code analysis scenario using cloud server.

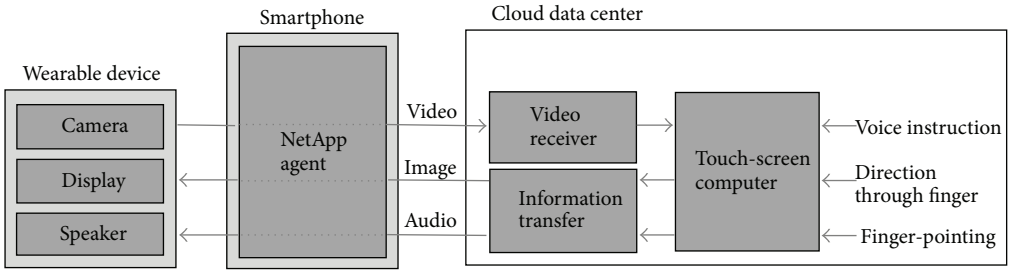

Figure 10: Voice and video-sharing scenario involving smart device and cloud server.

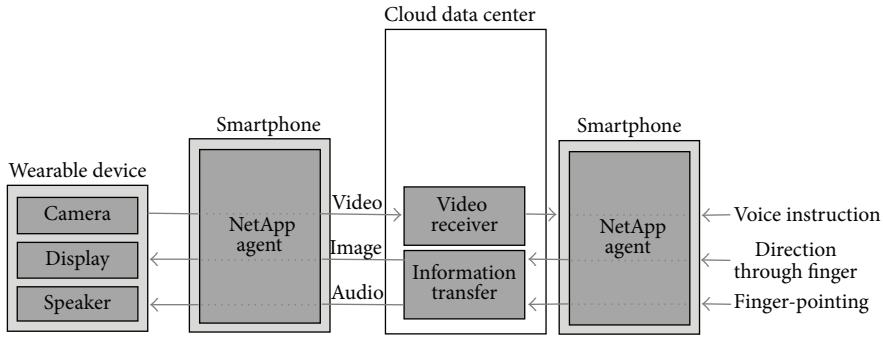

Figure 11: Voice and video data-sharing scenario involving smart devices.

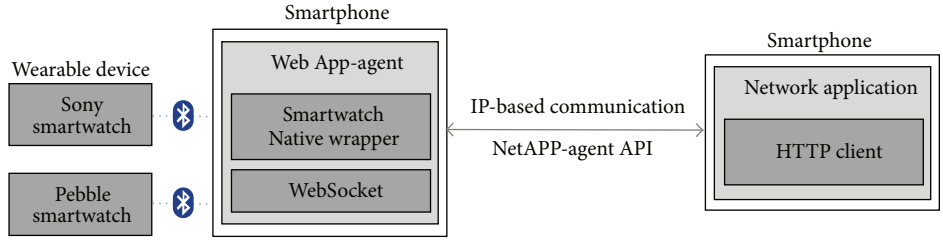

Figure 12: Implemented NetApp-Agent.

server. A user wearing Google Glass transmits information in real time to the cloud server, which shares the transferred video with the user again and issues an order. An order can be issued by pointing at the transferred video. The cloud server once again utilizes the NetApp-Agent API to implement an application to display information on the wearable device. Thus, modification of the wearable device is unnecessary.

3.2.3. Voice and Video Data Sharing among Smart Devices. Figure 11 shows a more detailed scenario than the previous one. Two smart devices capable of IP-based communication share video, voice, and image through a camera in real time with the cloud server acting as mediator. The realization of this scenario can lead to innovative services in the market. For instance, suppose User A shares his/her location with User B through an image as User B guides User A through vocal instructions or ostensive guides using the shared image.

4. Experiment and Qualitative Assessment

In this section, we report an experiment to show the feasibility of the suggested platform and discuss the results. The efficiency of the proposed platform is also confirmed through a comparative analysis with the existing development platforms.

4.1. Empirical Research of the Proposed Platform

4.1.1. NetApp-Agent Implementation by Utilizing Native Development Method. The proposed platform was implemented in order to demonstrate how proposal works as the prototype. The setup consisted of a smartphone equipped with NetApp-Agent, another smartphone with a network application communicating with NetApp-Agent, and controllable wearable devices, as shown in Figure 12. NetApp-Agent was implemented in Android 4.1.2, and a web server function was added to support Hypertext Transfer Protocol (HTTP) communication based on IP. The two wearable devices used were a Sony SmartWatch and a Pebble SmartWatch. Sony Add-on SDK 2.1 and Pebble SDK 2.0 were installed to develop the SmartWatch application. The user application was implemented in Android 4.1.0 version and the HTTPClient class in Android was used to support HTTP communication. Both NetApp-Agent and the user application exchanged JSON data using a self-defined NetApp-Agent API and sending HTTP communication based on IP.

Table 7 lists the NetApp-Agent APIs. NetApp-Agent enabled the transmission of texts and images to wearable devices. Accordingly, the four former APIs were used, Connected Device List (which offered the list of connected wearable devices and detailed information regarding each device), Select Device (which selected the messaging terminals), Send Text (to send texts), and Send Image (to send images to devices). The four latter APIs will soon be added to NetApp-Agent APIs.

The detailed working structure of the implemented platform is shown through a sequence diagram. Figure 13 shows the process of acquiring information related to the detailed specifications and connections of wearable devices using the network application. The network applications transmit an HTTP GET method to set connections with NetApp-Agent. Upon receiving requests, the NetApp-Agent sends messages containing information regarding connected wearable devices and detailed specifications of each device to the network application, which follows a JSON type. On receiving

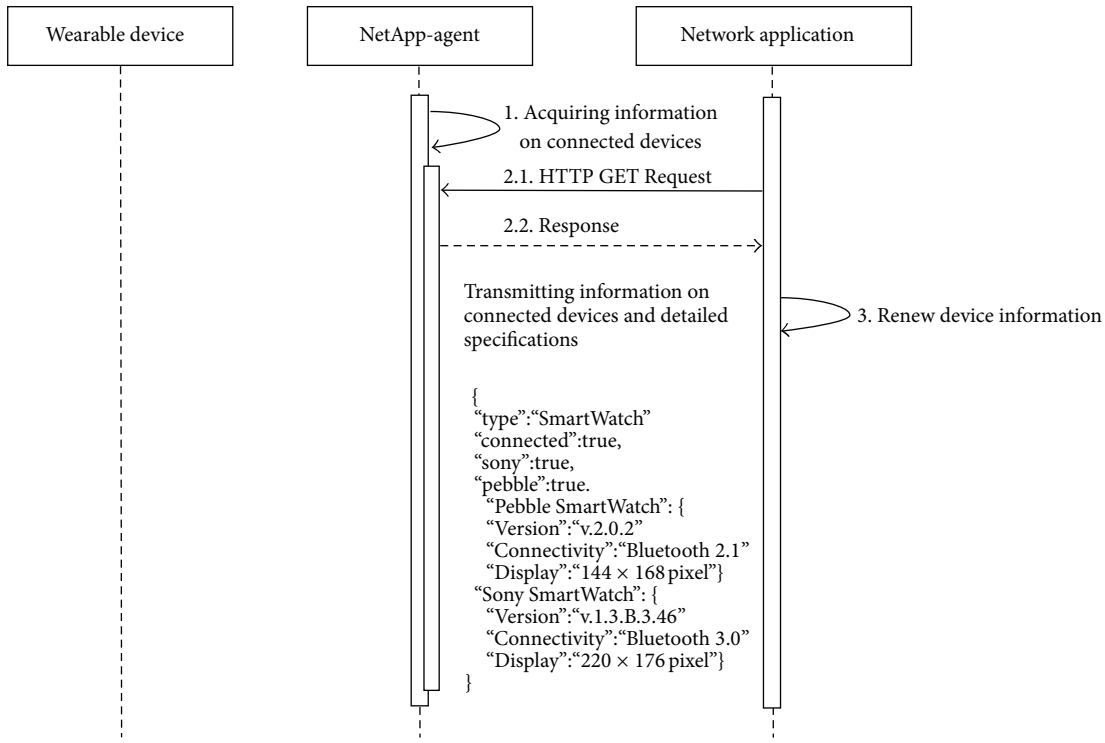

Figure 13: Acquiring information on wearable devices using network application.

Table 7: List of NetApp-Agent APIs.

API	Description
Connected Device List	Provide detailed information on each device and the list of wearable devices connected to smartphone through Bluetooth
Select Device	Select the kinds of wearable devices to control
Send Text	Make the text appear on the screen by sending it to wearable devices
Send Image	Make the image appear on the screen by sending it to wearable devices
Send Voice	Make the voice and sound appear at speaker by sending it to wearable devices
Cellular	Monitor cellular state and make cellular state enabled or disabled
Wi-Fi	Monitor Wi-Fi state and make Wi-Fi enabled or disabled
Bluetooth	Monitor Bluetooth state and make Bluetooth enabled or disabled

information about wearable devices, the network application adds the device information to the List View so that users can select devices after checking related information.

Figure 14 shows the process of sending texts to wearable devices via NetApp-Agent, and Figure 15 shows the implementation of the platform. Information on devices for submission and text values selected by users are transmitted in JSON messages through HTTP POST. On receiving the messages, NetApp-Agent sends texts to the selected wearable devices and prints it on the screen. Depending on the context, NetApp-Agent supports intelligent tasks.

In case of the Pebble SmartWatch, it used to show the issue of incompatibility with Korean language. Thus, if text has to be transmitted through Pebble SmartWatch, NetApp-Agent senses the Korean characters in the messages before sending messages to wearable devices. While messages containing no Korean characters are transmitted as text, those containing Korean characters are transformed into PBI images and then transmitted to Pebble SmartWatch and vice versa.

Figures 16 and 17 show the process of sending images to wearable devices via NetApp-Agent and the results obtained on the screen. As in the case of sending texts, device information regarding the data and the images is sent to NetApp-Agent using HTTP POST. The images may need to be converted depending on the types of targeted devices when sending images. Sony SmartWatch displays images by temporarily saving them on the smartphone and sending uniform resource identifier (URI) information. By utilizing its self-defined image format, the Pebble SmartWatch converts the received bitmap images to PBI format and then sends it.

4.1.2. NetApp-Agent Implementation Utilizing Cordova Platform. NetApp-Agent aspires to be a cross-platform development framework that implements applications regardless of the type of mobile platform at hand and expands the coverage of API support by adding Native API functions to the Cordova API.

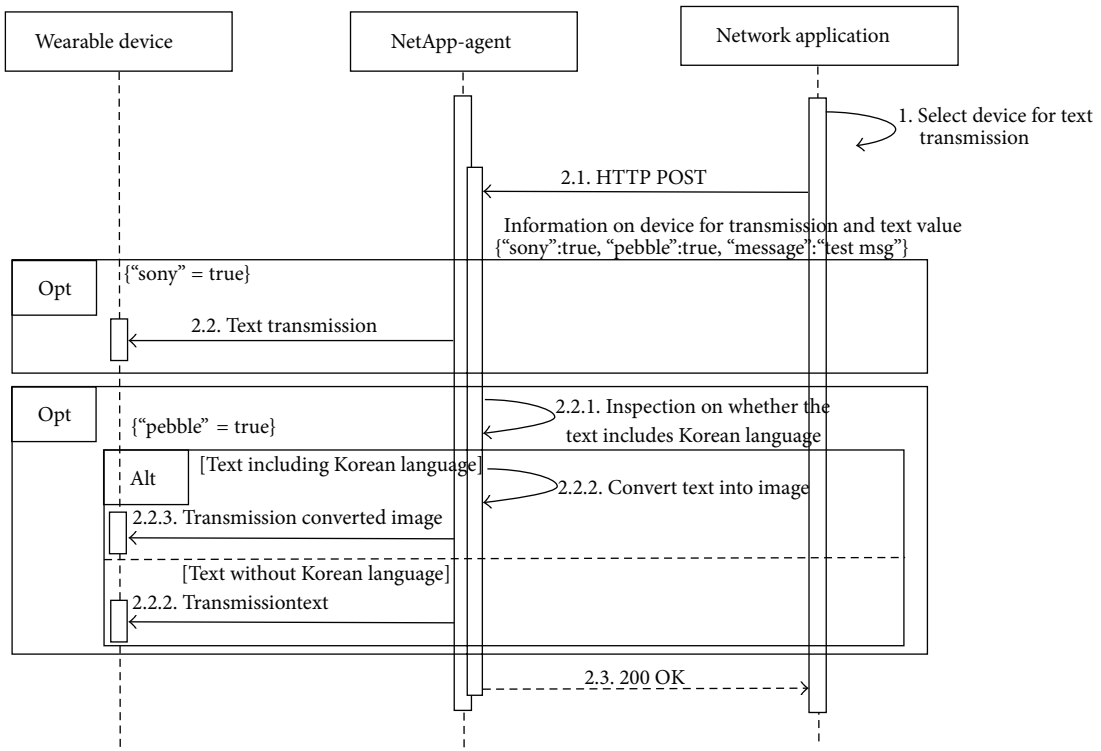

Figure 14: Sending text via NetApp-Agent.

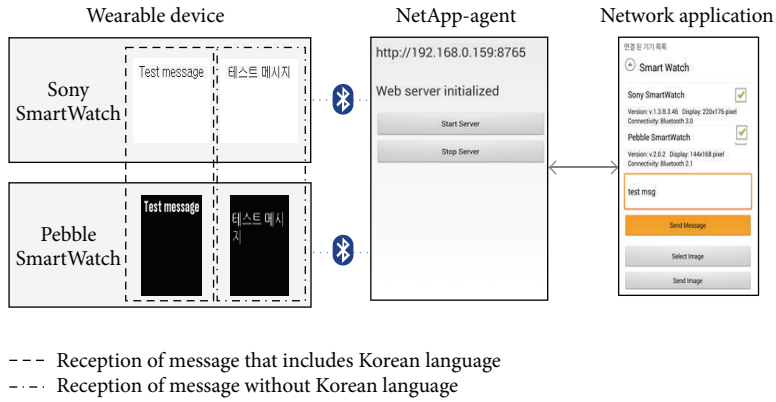

Figure 15: Result of sending text via NetApp-Agent.

Cordova supports the acquisition of the location information of devices and an API that provides status information regarding cellular connected to devices or to Wi-Fi networks. Figure 18 shows sequence of Acquiring GPS and network information. Consequently, when the location information of a smartphone and a network is requested to NetApp-Agent by network application, NetApp- Agent responds to a call in a way of acquiring information by calling Cordova API.

4.2. Qualitative Assessment of the Proposed Method. In this subsection, we conduct a functional analysis of the existing development platforms as well as our proposed NetApp-Agent and use this qualitative assessment to identify the pros and cons of the proposed platform. The efficiency of the proposed platform is tested throughout this process.

4.2.1. Provision of Integrated Development Environment for Devices. The current mobile market has several mobile platforms, such as Android, iOS, and Windows Mobile. Owing to the diversity of platforms, developers need to construct development environments according to platform and learn several programming languages, which slows down the pace of development. Mobile programming is gradually becoming standardized to cope with such issues, and cross-platforms such as Cordova and Titanium are welcome as a result. Nevertheless, cross-platforms are limited in their scope of

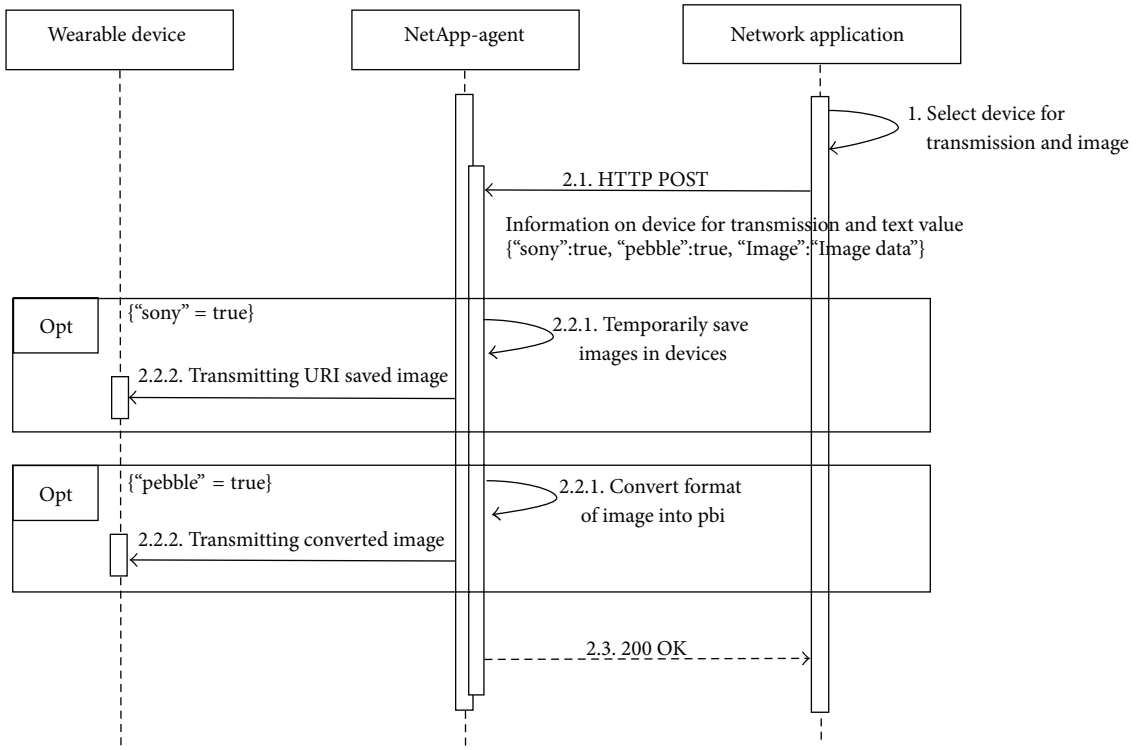

Figure 16: Sending image via NetApp-Agent.

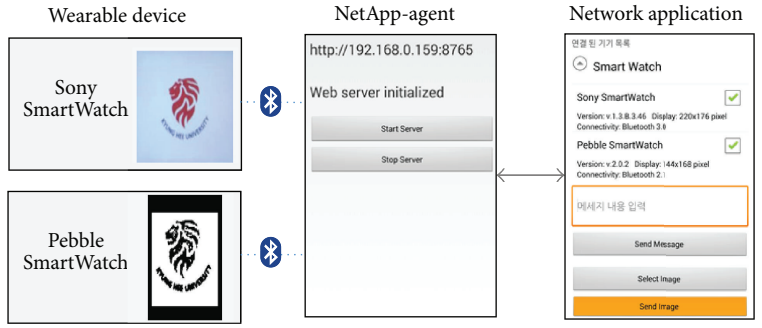

Figure 17: Result of sending image via NetApp-Agent.

API coverage and are still in the process of developing APIs to control wearable devices. All of this underscores the necessity of the acquisition and development of native languages. In other words, cross-platform by itself is inadequate for developing applications for wearable devices. A comprehension and development of applications that take into account unfamiliar development environments and API assistance for wearable devices would be extremely time-consuming.

Table 8 shows the development environment of the Cordova development platform. When developing wearable devices outside Cordova's stipulated coverage, the native development approach is preferred. Hence, the development of applications necessitates the construction of development environments depending on target devices as well as knowledge of compatible APIs. In the case of applications for Google Glass, for instance, developers need to learn both the development language, such as Python and Java, and the Google Mirror API after installing the relevant development tools.

Developing applications for Pebble SmartWatch requires that developers construct suitable development environments and learn specific languages such as Pebble SDK and PebbleKit SDK, which are built-in functions of a smartphone. Even when developing different applications for the same purpose, a considerable amount of time needs to be spent on creating different development environments and learning unfamiliar languages. The table shows development environments for four different devices, where developers had to learn 10 programming languages and 7 APIs. As the number of devices supplying applications increases, the workload and time spent on development increase proportionately.

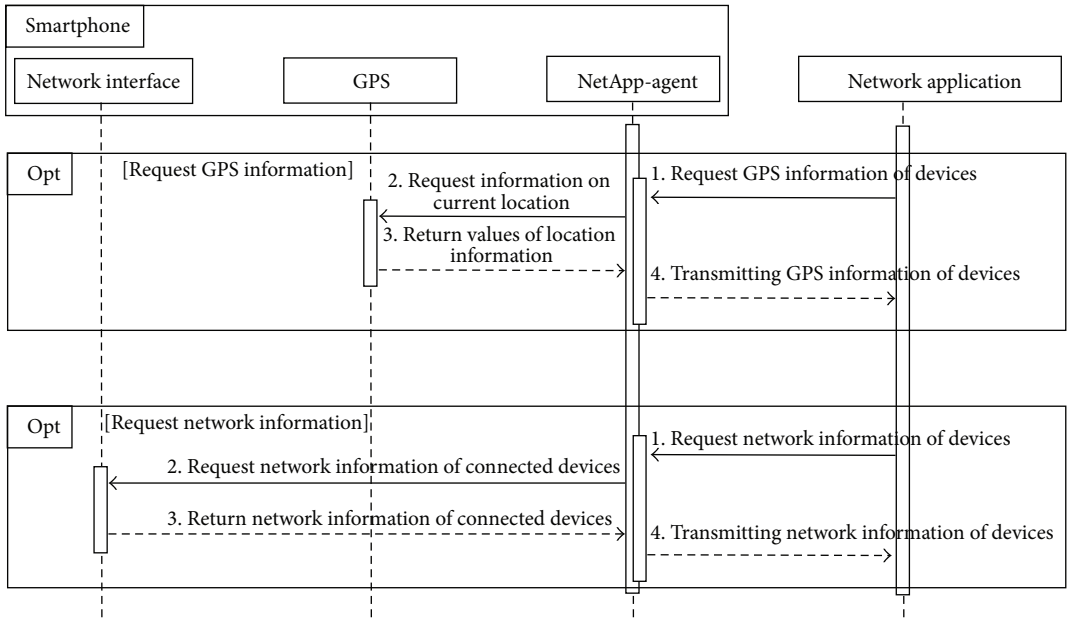

Figure 18: Acquiring GPS/network information of a device via Cordova API.

Table 8: Development environment of Cordova platform.

	Under Cordova platform		
Types of devices	Operating system	Supporting languages	SDK/API
Total	**4**	**10**	**7**
Google Glass	Android 4.0.4	Go, Java, .NET, PHP, Python, and Ruby	Google Mirror API
Sony SmartWatch	Android 4.0	Java	Sony Add-on SDK/Smart Extension API
Pebble SmartWatch	Pebble OS	C, JavaScript, and so forth (Objective-C/Java)	Pebble SDK PebbleKit
Samsung Gear	Tizen OS	Java, HTML, JavaScript, and so forth	Tizen SDK Samsung Mobile SDK

Table 9: Development environment of NetApp-Agent.

	Under NetApp-Agent		
Types of device	Operating system	Supporting languages	SDK/API
Total	**1**	**1**	**1**
NetApp-Agent supporting devices (extendable)	No limitations	No limitations	NetApp-Agent API (WebSocket/JSON-based)

Table 9 shows the application development environment based on NetApp-Agent. If developers are aware of the specific features of JSON-based NetApp-Agent APIs and the implementation of WebSocket for communication with a platform, other applications can be easily developed using programming languages with which developers are, presumably, already familiar. Thus, procedural redundancies, such as constructing a separate development environment for each platform and learning both the relevant languages and the API, can be effectively avoided. This approach can downsize the time and cost needed to develop applications, and developers can easily expand the selection of devices compatible with the applications.

4.2.2. Expansion of API Supply Coverage of Platform. Cross-platform development frameworks such as Titanium and Cordova are still in the process of development and hitherto have failed to guarantee wide API coverage. The proposed platform widens the coverage of serviceable APIs by adding a native device wrapper to the Cordova wrapper. Table 10 lists the kinds of APIs per development platform. We see that NetApp-Agent supports wider API coverage than the

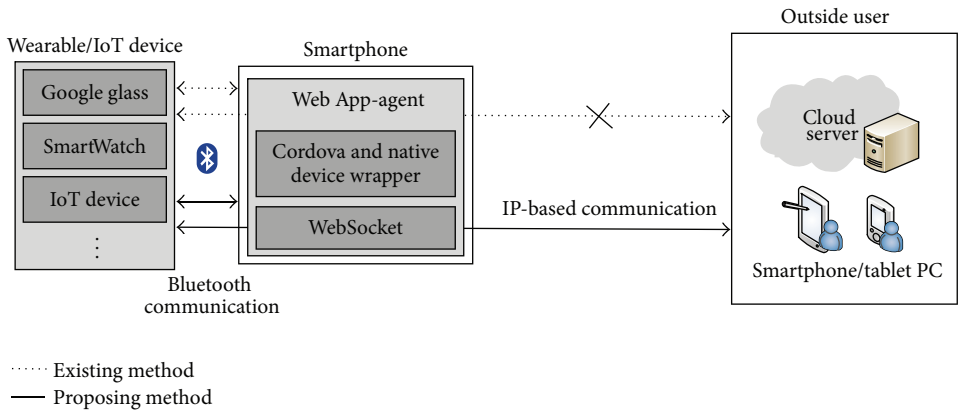

Figure 19: Comparison of communication methods between existing and proposed platforms.

Table 10: Supporting APIs per development platform.

API		Titanium	Cordova	NetApp-Agent
Battery		O	O	O
Vibration		X	O	O
Camera		O	O	O
GPS		X	O	O
Network	Cell_State	X	O	O
	Cell_ON	X	X	O
	Cell_OFF	X	X	O
	WiFi_State	X	O	O
	WiFi_ON	X	X	O
	WiFi_OFF	X	X	O
	Bluetooth_State	X	X	O
	Bluetooth_ON	X	X	O
	Bluetooth_OFF	X	X	O
	Device List	X	X	O
Media	Send Audio	X	X	O
	Recv Audio	X	X	O
	Play Audio	O	O	O
	Send Video	X	X	O
	Recv Video	X	X	O
	Play Video	O	X	O
Display	Send Text	X	X	O
	Recv Text	X	X	O
	Send Image	X	X	O
	Recv Image	X	X	O
Sensor	Compass	X	O	O
	Accelerometer	X	O	O
	Light Sensor	X	X	O

existing platforms. Consequently, the use of NetApp-Agent API allows in-depth control of hardware/software modules as well as communication without further development of the native language.

4.2.3. IP-Based Communication Service for Bluetooth Communication Devices. It is rare to find stand-alone functions in wearable devices as most functions are usually dependent on smartphones. Connections between smartphones and wearable devices are needed to guarantee the practical use of devices, and Bluetooth-based interdevice communication is utilized at this stage. Bluetooth as a communication system for short distances is implemented by referring to the Mac addresses of the paired devices. Accordingly, it is virtually limited for a wearable device that only supports Bluetooth communication to communicate with external devices or users as it is the same issue of IoT devices.

Our proposed platform forms Bluetooth connections with wearable/IoT devices and supports WebSocket-based IP communication. It creates a network by grouping together Bluetooth-based devices and enables IP communication with the outside. Thus, outside users can communicate with and control wearable/IoT devices using IP without having to form Bluetooth connections.

Figure 19 shows the differences between the existing platform and the proposed one. In case of the existing method, wearable/IoT devices only support Bluetooth communication. To use and control devices, smartphones connected to them need to be carried to within the coverage range of Bluetooth communication. The proposed platform supports remote access to devices without IP. Accordingly, devices can be controlled from outside using IP communication.

4.2.4. Provision of Network Application Development Environment. SDKs and APIs being offered to implement cloud services mostly focus on building the cloud server. Consequently, client applications need independent implementation for each platform in order to service devices. The proposed NetApp-Agent platform supports the development of network applications for cloud services.

Network applications carry out IP-based communication with NetApp-Agent and use NetApp-Agent APIs to provide services using cloud servers to users without the need for further implementation or revision of smart devices. Considering that smart devices are only implemented through the API offered, the revision of smart device programs is unnecessary even if the function of the native application is extended or a brand-new network application is developed.

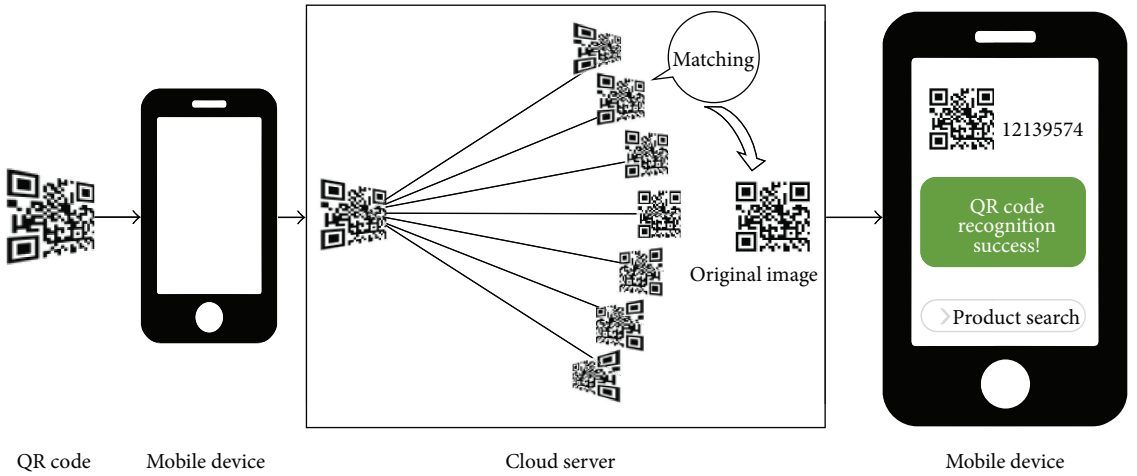

Figure 20: Operating process of the pregenerated Image Matching Method.

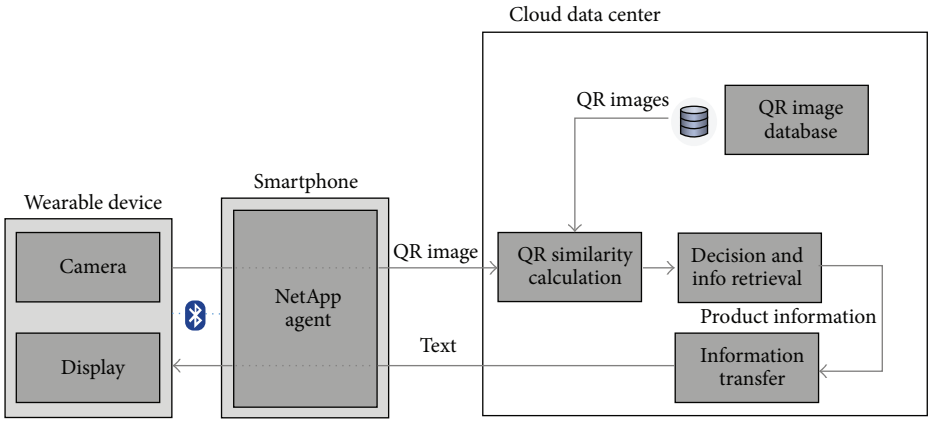

Figure 21: Expandable Cross-Platform for QR code recognizing scenario based NetApp-Agent.

Furthermore, smart devices can support intelligent functions by connecting to the cloud server, which was hitherto unavailable for smart devices. This enables the provision of advanced services to users as a result.

4.3. Experimental Implementation of the Proposed Method

4.3.1. Expandable Cross-Platform for QR Code Recognition Using the Pregenerated Image Matching Method. QR code is two-dimensional code developed in 1994, and the amount of a QR code is rapidly increasing [33]. It can handle more information than barcode about several hundreds of times and anyone can make and use it easily. Also, it enables the user to access website through recognizing code without entering URL [34]. The important key of QR code recognition is caught "Finder Pattern" of QR code. QR code can be recognized only if camera catches these three patterns. When angle bends largely, camera cannot catch three patterns so that it becomes impossible to recognize QR code [35].

The QR code recognition method is "pregenerated image matching" that finds original QR code through similarity test of shooting photograph and database's images in the server. We suggest the composition of server-device environment and operational process.

Figure 20 shows the entire operating process of QR code recognition method. At this method, wireless device sends shooting QR code image without any processing to the cloud server. After receiving QR code image, cloud server performs whole operating process. It is a contrast to traditional technology where shooting device performs whole operating process.

Before the recognition, cloud server must store QR code images in the database. This process includes transform QR code image in four directions with every single angle.

Once wireless device takes a picture of QR code image, device sends image to server. Server performs similarity test among input image and images stored into the database. Server finds an original image with the largest similarity coefficient. Finally, it sends detecting QR code image and decoding information to the wireless device.

Figures 21 and 22 show the process of QR code recognition based NetApp-Agent. We create a NetApp-Agent application using Cordova wrapper, which supports "pregenerated image matching" on various devices. A device made of web

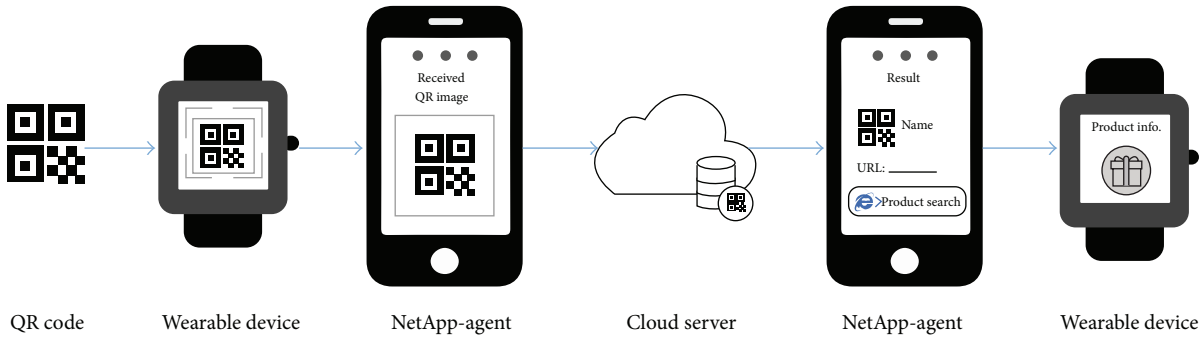

Figure 22: The process of QR code recognition using the pregenerated Image Matching Method based NetApp-Agent.

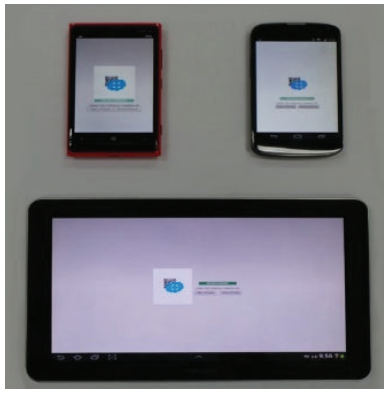

Figure 23: Operating screen of the application.

Table 11: Recognition angle using NAVER application.

Distance	Model			
	Galaxy Note 3	Galaxy S4	Nexus 4	iPhone 5
15 cm	35°	40°	45°	0°
30 cm	45°	43°	50°	X

Table 12: Recognition angle using SCANY application.

Distance	Model			
	Galaxy Note 3	Galaxy S4	Nexus 4	iPhone 5
15 cm	50°	45°	55°	45°
30 cm	55°	55°	55°	45°

application communicates with the server made of Node.js using WebSocket. Because different operating system can use the same web application, the service can be provided to various devices with separate operating system [36].

We used Google's reference phone, which is Nexus 4 installed Android 4.4 Kitkat. In addition, we had a test in Lumia 920 installed Windows Phone 8.0 and Samsung Galaxy Tab 10.1 installed Android 4.0 Ice Cream Sandwich. Figure 23 shows that application of the QR code recognition is portable to other operating systems.

4.3.2. Performance Evaluation of the Pregenerated Image Matching Method. This paper's experiment is verifying the recognition of QR code in the angle where it cannot be recognized in traditional QR code applications.

First, we implement an experiment to find the maximum recognition angle of traditional applications. We measure the maximum recognition angle with four smartphones using NAVER and SCANY application. Tables 11 and 12 are the test result of using 1.8 cm ∗ 1.8 cm QR code.

Those two applications display the maximum recognition angle at 55° so we experiment the possibility whether our proposed method can recognize the image at 60°.

Figure 24 is 1.8 cm ∗ 1.8 cm 10 QR code images for recognition. These images of the database were taken at a distance of 20 cm by using the phone camera directly. Setting the maximum angle of the QR code to 70 degrees, the images were taken 5 degrees from 0 degrees by incrementing. The images are stored in binary. We saved 15 images per QR code, and the 150 images were set as the comparison of the experiment. We examined by comparing the similarity between the 150 QR code images and we saved the 10 QR code images inclined at 60 degrees. After checking the similarity, the similarity of each QR code was output in descending rank from first to third rankings.

Figure 25 is a graph using a QR code 01 taken at 60-degree angles, representing the similarity of the 150 QR code images. It is expressed by a line of a different color for each QR code, the horizontal axis represents the value of the angle, and the vertical axis represents the degree of similarity (%).

The most significant similarity is the image at 60 degrees of QR code 01, because the QR code image has high similarity with itself. Other QR codes can be matched with target of experiment, having relatively low similarity. These experimental results show that the proposed method is superior to the traditional method, which is used by NAVER and SCANY application; this paper presents recognition of QR code over other applications' maximum limit angle.

5. Conclusion and Future Research

In this paper, we proposed a wearable device control platform to develop network applications. We also carried out an experiment to confirm the feasibility and efficiency of the proposed platform. The advantages of the proposed platform

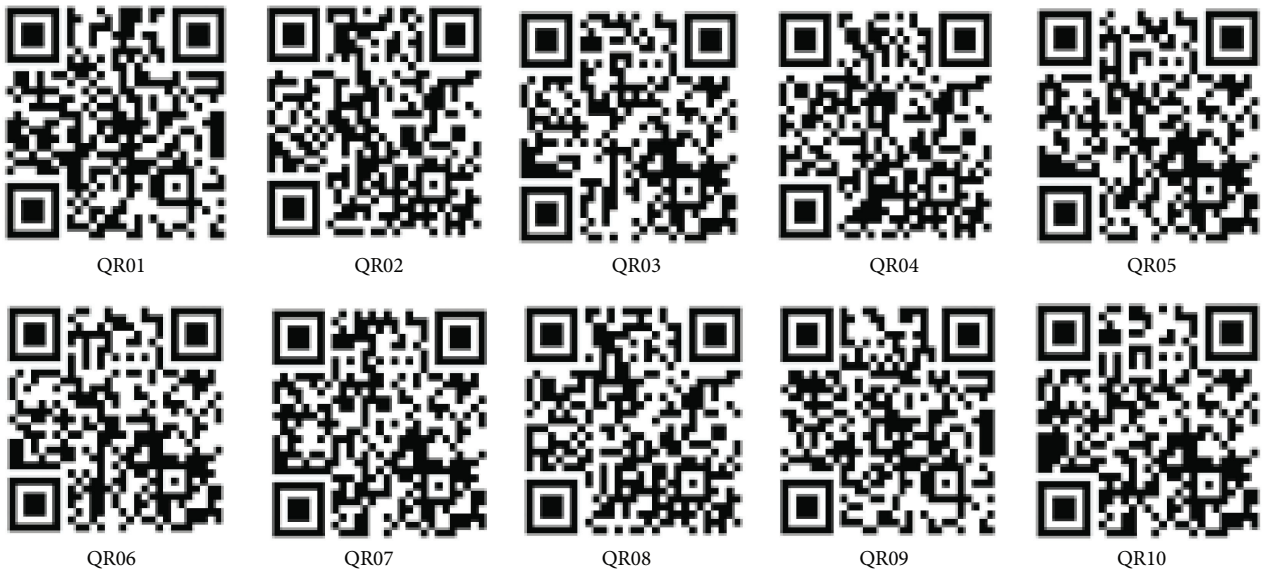

Figure 24: 10 QR code images for experiment.

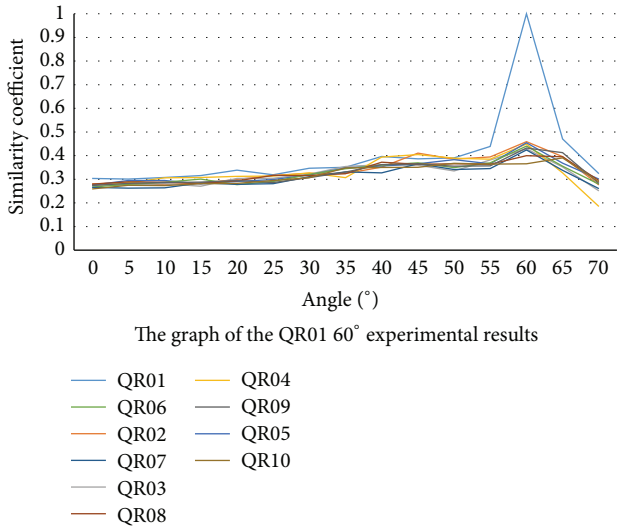

Figure 25: The graph of the experimental results.

were tested through a comparative analysis with the existing platform.

There are two main advantages of our proposed platform. First, it reduces the time and cost needed to develop applications by providing a single API to developers. In past development approaches, developers needed to construct development environments for each platform and had to learn several programming languages and APIs when developing applications. For the development of applications working on a Sony SmartWatch, an Android-based development environment had to be constructed, and, in order to install the Sony Add-on SDK, learning about Java and API had to be gone through. Furthermore, the development of applications with the same features for Pebble SmartWatch required that developers acquire relevant knowledge of the programming languages, such as C and Pebble SDK. The increase in the number of platforms led to a rise in the time and resources required to develop applications, which are falling behind the current trends in which a variety of wearable devices are consistently launched. By integrating development environments and offering a JSON-based API, our proposed platform eases the burden of application development regardless of environment and type of platform. Moreover, the issue of limited API coverage in the Cordova platform was addressed by using a Cordova device wrapper and a native wrapper together. The proposed platform enables the development of a wide variety of applications by extending API coverage to software/hardware module control of devices.

Second, wearable devices that support Bluetooth communication can communicate with the external environment in our platform using IP. Existing wearable devices support Bluetooth communication and are heavily dependent on smartphones. However, in the existing platform, a smartphone needs to be carried to control and utilize wearable devices, which need to be located within the range of Bluetooth coverage. The proposed platform allows the control of devices through remote access as well as the exchange of relevant information. Due to rising demand for IoT, the era of wider communication even with blub, TV, remote control, and vehicles is coming. The application of the proposed platform to IoT will make it possible to form an interdevice network where outside users can control devices.

At this point, in technological development, when the need for a standardized mobile programming method is pressing on account of the growing number of smart devices on a variety of platforms, our proposed platform shows advantages by unveiling its unique features which are incorporating the development methods of applications and supporting IP-based communications to external devices. Thus, since our platform facilitates the development of service applications without additional implementation or revision,

the widespread adoption of this approach is expected in developing network applications that service smart devices.

An issue to consider is that consistent attention and updates to the platform will be required for new smart devices and platforms in order to guarantee the continual use and development of NetApp-Agent. The experiment described here shows the feasibility of NetApp-Agent, which adds some limitations on the coverage of serviceable API. Hence, the proposed NetApp-Agent is in need of further implementations.

Conflict of Interests

The researchers claim no conflict of interests.

Acknowledgments

This research was supported by the MSIP (Ministry of Science, ICT and Future Planning), Republic of Korea, under the ITRC (Information Technology Research Center) support program (IITP-2015-(H8501-15-1015)) supervised by the IITP (Institute for Information & Communications Technology Promotion); the Institute for Information & Communications Technology Promotion (IITP) Grant funded by the Korea government (MSIP) (B0190-15-2013, Development of Access Technology Agnostic Next-Generation Networking Technology for Wired-Wireless Converged Networks); and the ICT R&D Program of MSIP/IITP, Republic of Korea (B0101-15-1366, Development of Core Technology for Autonomous Network Control and Management).

References

[1] M. Ballve, *Wearable Computing: From Fitness Bands to Smart Eyewear. A New Mobile Market Takes Shape*, Business Insider, 2013.

[2] K. Daegun, "Trends and implications of wearable device," *Policy of Broadcasting and Telecommunication*, vol. 25, no. 21, 2013.

[3] Korea Association for ICT Promotion, *A Monthly ICT Statistics*, Monthly ICT Item Trend Investigation, ICT Business Survey Index (BSI), 2013.

[4] Machina Research, *Strategy Report—M2M Communication Service Provider Benchmarking*, 2012.

[5] Gartner, *Forecast: Mobile App Stores, Worldwide, 2013 Update*, Gartner, 2013.

[6] A. Zibula and T. A. Majchrzak, "Cross-platform development using HTML5, jQuery mobile, and phonegap: realizing a smart meter application," in *Web Information Systems and Technologies*, vol. 140 of *Lecture Notes in Business Information Processing*, pp. 16–33, Springer, Berlin, Germany, 2013.

[7] Titanium, http://www.appcelerator.com/titanium/.

[8] B. Zhang, T.-G. Xu, W. Wang, and X. Jia, "Research and implementation of cross-platform development of mobile widget," in *Proceedings of the 3rd International Conference on Communication Software and Networks (ICCSN '11)*, pp. 146–150, IEEE, Xi'an, China, May 2011.

[9] S. Xanthopoulos and S. Xinogalos, "A comparative analysis of cross-platform development approaches for mobile applications," in *Proceedings of the 6th Balkan Conference in Informatics (BCI '13)*, pp. 213–220, Thessaloniki, Greece, September 2013.

[10] N. Fernando, S. W. Loke, and W. Rahayu, "Mobile cloud computing: a survey," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 84–106, 2013.

[11] J. Donggeun, *Web Application Hybrid Application Programming*, Answerbook, 2013.

[12] C. Jaekyu, *PhoneGap for Development of Hybrid Mobile Application*, WannaBooks, 2012.

[13] IBM, *Native, Web or Hybrid Mobile-App Development*, Software Thought Leadership White Paper, 2012.

[14] Sony SmartWatch Developer World web site, 2013, http://developer.sonymobile.com/tag/smartwatch/.

[15] K. Sanghyung, *Android Programming Complete Guide*, Hanbit Media, 2010.

[16] Xcode web site, https://developer.apple.com/xcode/.

[17] Pebble, http://developer.getpebble.com.

[18] L. Tian, H. Du, L. Tang, and Y. Xu, "The discussion of cross-platform mobile application based on Phonegap," in *Proceedings of the 4th IEEE International Conference on Software Engineering and Service Science (ICSESS '13)*, pp. 652–655, IEEE, Beijing, China, May 2013.

[19] J. M. Wargo, *Phonegap Programming*, Acorn, 2013.

[20] B. Pollentine, *Titanium Mobile Application Programming*, Acorn, 2012.

[21] H. Heitkötter, S. Hanschke, and T. A. Majchrzak, "Evaluating cross-platform development approaches for mobile applications," in *Web Information Systems and Technologies: 8th International Conference, WEBIST 2012, Porto, Portugal, April 18–21, 2012, Revised Selected Papers*, vol. 140 of *Lecture Notes in Business Information Processing*, pp. 120–138, Springer, Berlin, Germany, 2013.

[22] J. Daeyoung and K. Jongki, *Creative Convergence Visualization Method of IoT*, KIET Industrial Research, 2014.

[23] M. Younis, I. F. Senturk, K. Akkaya, S. Lee, and F. Senel, "Topology management techniques for tolerating node failures in wireless sensor networks: a survey," *Computer Networks*, vol. 58, no. 1, pp. 254–283, 2014.

[24] K. Curran, A. Millar, and C. Mc Garvey, "Near Field Communication," *International Journal of Electrical and Computer Engineering*, vol. 2, no. 3, pp. 371–382, 2012.

[25] A. Jula, E. Sundararajan, and Z. Othman, "Cloud computing service composition: a systematic literature review," *Expert Systems with Applications*, vol. 41, no. 8, pp. 3809–3824, 2014.

[26] A. Marathe, R. Harris, D. K. Lowenthal, B. R. de Supinski, B. Rountree, and M. Schulz, "Exploiting redundancy for cost-effective, time-constrained execution of HPC applications on amazon EC2," in *Proceedings of the 23rd International Symposium on High-Performance Parallel and Distributed Computing (HPDC '14)*, pp. 279–290, ACM, Vancouver, Canada, June 2014.

[27] Z. Sanaei, S. Abolfazli, A. Gani, and R. Buyya, "Heterogeneity in mobile cloud computing: taxonomy and open challenges," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 369–392, 2014.

[28] A. Bedra, "Getting started with Google app engine and Clojure," *IEEE Internet Computing*, vol. 14, no. 4, pp. 85–88, 2010.

[29] DNA Developer Network web site, http://developers.daum.net/services.

[30] C. Jaekyu, *Understanding of Cloud Technology and Development Platform*, MicroSoftware Article, 2012.

[31] Amazon Webservice SDK, http://aws.amazon.com/ko/tools/.

[32] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, 2010.

[33] Y. Liu, J. Yang, and M. Liu, "Recognition of QR Code with mobile phones," in *Proceedings of the Chinese Control and Decision Conference (CCDC '08)*, pp. 203–206, IEEE, Yantai, China, July 2008.

[34] L. F. F. Belussi and N. S. T. Hirata, "Fast QR code detection in arbitrarily acquired images," in *Proceedings of the 24th SIB-GRAPI Conference on Graphics, Patterns and Images (Sibgrapi '01)*, pp. 281–288, Maceió, Brazil, August 2011.

[35] M. Ahn and S. Lee, "A research on QR Code recognition enhancement using pre-constructed image matching scheme," in *Proceedings of the International Conference on Information and Communication Technology Convergence (ICTC '14)*, pp. 82–83, IEEE, Busan, South Korea, October 2014.

[36] M. Ahn, S. Hong, and S. Lee, "A research on the QR Code recognition improvement using the cloud-based pre-generated image matching scheme," in *Proceedings of the International Conference on Information Networking (ICOIN '15)*, pp. 356–357, IEEE, January 2015.