Application Push & Play – Proposal on Dynamic Execution Environment Combined with Personal Devices and Cloud Computing. -

Hidenobu Ito †, Kazuaki Nimura †, Yousuke Nakamura †, Akira Shiba †, and Nobutsugu Fujino †

† Fujitsu Laboratories Ltd.
4-1-1, Kamikodanaka, Nakahara-ku, Kanagawa, Japan
{itou.hidenobu, kazuaki.nimura, nkmr, shiba.akira, fujino}@ jp.fujitsu.com

Abstract – Mobile devices have become essential tools in our life. To make effective use of a smartphone, a user may install many small applications and use them according to his/her circumstances. However, it is becoming impossible to ignore the time and effort a smartphone user spends to discover and install appropriate applications. In this paper, we propose a concept wherein desirable applications and data automatically descend from the Cloud and are executed on a device when a user is authorized to receive a service. Such applications are available only at the right moment and go away when a user no longer requires them. As a proof of concept, we implemented the system on an Android smartphone and confirmed that it can reduce installation steps and has a good enough response. The proposed system would be able to help users in their daily activities and provide a new user experience different from the conventional one.

Keywords: Mobile device, Cloud computing, Push services, Android

1 INTRODUCTION

A variety of personal devices such as smartphones, tablets, and personal computers are now on the market. In addition, there are many applications for such devices.

In terms of the usability, it is not necessarily easy to start using smartphone applications. The initial setup can be especially troublesome and annoying. For example, those who have purchased a smartphone tend to spend a long time finding and installing applications from an app store, etc., even from stores they frequently visit. Moreover, smartphones without any preinstalled applications would not be able to be used conveniently. Users hence have to find and install applications before use their phones. In addition, installing too many applications on local devices makes them inconvenient because it would be difficult to identify an application they want to use on the small screen of a smartphone. In short, people do not like to spend too much time setting up a smart phone because it has a short life cycle. It would thus be good to have a system where a user can receive services with minimal preparation.

There are several studies about usage of smartphone applications. Some point out that users use these applications as important tools to manage information, tasks, work, and social relationships in their busy lives. Time management is one of the most important factors for them.

According to the report of the SBE Council [7], mobile application discovery is a big problem for smartphone users.

Moreover, there are several studies showing that the retention rate of mobile applications is rather low. The Ringcentral Survey [6] found that the retention rate after six months is only 36%. Scott Kveton, CEO of Urban Airship, a mobile notifications provider, stated that there is only a 5% retention rate of free applications after 30 days.

Therefore, it is not always true that smartphone users can save time by utilizing mobile applications because they waste too much time in preparing seldom-used applications. In other words, reducing the time for discovery and installation of applications would give them a better mobile experience.

In this paper, we propose a concept of dynamic installation and execution called Application Push & Play (APnP) to resolve the above problem. In addition, we propose an architecture for a smartphone to embody the concept. We prototyped an APnP system by utilizing web applications executable on local devices and an Android smartphone. We used the results of the experiment to evaluate the feasibility and efficacy of our architecture.

2 RELATED WORK

Apple Inc. provides the App Store on iTunes [4] to distribute applications for iPhones. An iPhone user can download applications via iTunes on a PC, and synchronize and install them on an iPhone through a USB connection. A user can also download and install applications directly on an iPhone. Google Inc. provides the Android Market [3] for Android smartphones. The advantage of the Android Market is that a user can download and install applications from a PC to any android smartphone he/she owns. On the other hand, Apple has recently started to provide iCloud to synchronize all applications between devices. The problem is that Google and Apple do not care when and what applications a user wants to use though they provide a mechanism which delivers applications. In addition dedicated ids such as Google account and Apple id are required for a user to get application. It might be preferable for business purpose because some company policies does not allow to use such kind of ids in order to avoid the risk of information leakage.

In terms of applications for mobile devices, Jason Grigsby [5] compared three types of application: Native, Web, and Hybrid. 'Hybrid applications' are executable on local devices and are written in HTML, CSS and JavaScript. Though they look like web applications, they can run in a dedicated runtime environment on a local device. As far as Web applications go, HTML5 [1] specifies a disk cache that

can let web applications run on a local device even when the network is disconnected.

MIT Project Oxygen [9] is a project that addressed challenges to support highly dynamic and varied human activities. The goal of Oxygen system is to help us do what we want when we want to do it using embedded devices, handheld devices, dynamic self-configuring networks and software that adapts to changes in the environment or in user requirements. The architecture relies on control and planning abstractions that provide mechanisms for change, on specifications that support putting these mechanisms to use, and on persistent object stores with transactional semantics to provide operational support for change. For example, it realizes that when a user having listened his favorite music in his room moves to living room, it automatically continue the music from the speaker in the room. The focus of this project is maintaining continuity of services by involving various available resources. On the other hands, our approach improves user experience by narrowing down of service delivery.

3 PROPOSED ARCHITECTURE

As described in the introduction, a big issue is how to reduce the time for discovery and installation of applications. Additionally, we believe this discovery and installation should proceed without user operations in order to save time and labor. In this section, we define the requirements and describe the architecture to enable this.

3.1 Requirements

The requirements are as follows:

(R1) Applications exist on a personal device only when a user needs them.
(R2) A user need not explicitly install/uninstall applications.
(R3) A set of suitable applications is automatically identified by a user's context.

The best way to resolve the issue of application discovery and installation is that a user does not have to think about it. All user wants to do is to deal with content; that is, s/he may not want to worry what kind of application can operate on the content. Thus, the requirement is that not the user but the system should identify applications which the user needs and can deliver them to a personal device just in time. (R3) is equivalent to the requirement of identification. The requirement of delivery is a combination of adequate notification (R1) and automatic installation (R2).

3.2 Architecture

Figure **1** illustrates the architecture of Application Push & Play in order to fulfill the above requirements.

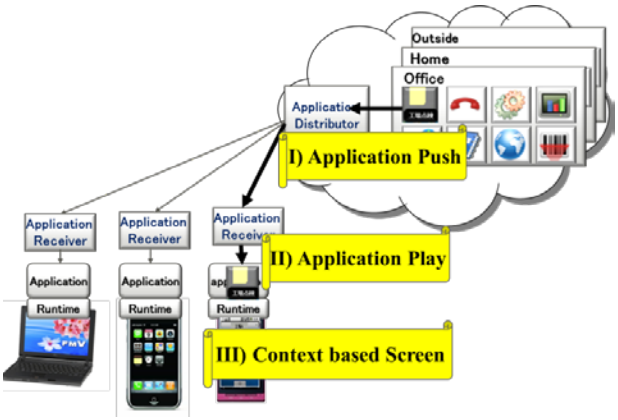


Figure 1: Architecture of APnP

The architecture consists of the following features:

(A1) Application Push
(A2) Application Play
(A3) Context-based Screen

Our approach assumes that all applications are in the Cloud environment and are downloadable and executable on local devices whether network connections are available or not.

'Application Push' (A1) handles distribution of applications and data based on a user's context determined from sensor information, user behaviors, etc. Once the system has detected a user's state as being likely to use a service, a particular application in the Cloud would be injected into the local storage of the user's device.

'Application Play' (A2) deals with automatic installation and execution of applications injected by 'Application Push'.

'Context-based Screen' (A3) provides a user with a simple and comprehensible GUI. It is a dynamically morphing home screen in accordance with the place and time of the user.

To meet requirement (R1), the system should have a method of real-time notification to deliver an application in a timely manner. In general, there are two methods of notification: "polling" and "push". In polling, a client checks a server periodically to see if there is an event. A typical example is an e-mail system in a PC. The push method is that a server notifies a client when an event occurs. A typical example is the SMS system in a cell phone. In terms of power consumption, the polling method is not as good for a smartphone. That is why the push notification of application (A1) is an important feature. Our architecture includes a server and push handler as a client.

To meet requirement (R2), the system should have a method of automatic installation (A2). This method includes a management functionality for downloading, storing data, and launching applications based on commands in a pushed message.

To meet requirement (R3), the system should have a method for a user to reach an application identified at a glance under a particular context. At a client-side's point of view, a user-friendly interface as a home screen optimized by context (A3) is important. In addition, the definition of the user's context is also important. There has been a lot of

works about it, and many people are still considering what is the best way. The performance of our architecture extremely depends on which method is applied to. Therefore, it might not be practical even if we measured the performance in a particular method. In this paper, we address how stable the performance of our architecture except context identification. The mechanism for determining it can be placed in the Cloud, and it can collect information such as sensor data about a user. A simple example of a context engine is a location-based or schedule-based system. Typically, a task server includes a context engine and invokes notification via a push server.

Figure 2 shows the APnP Framework which presents the relationship between the three layers and the components described above.

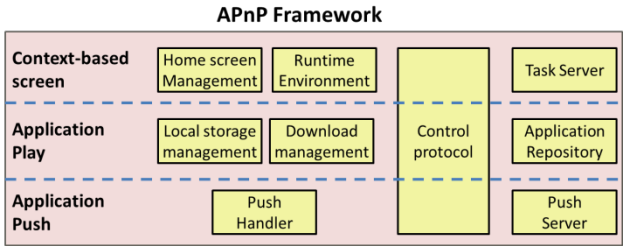


Figure 2: APnP Framework

4 PROTOTYPE IMPLEMENTATION

Figure 3 shows a prototype system to prove the feasibility of our architecture.

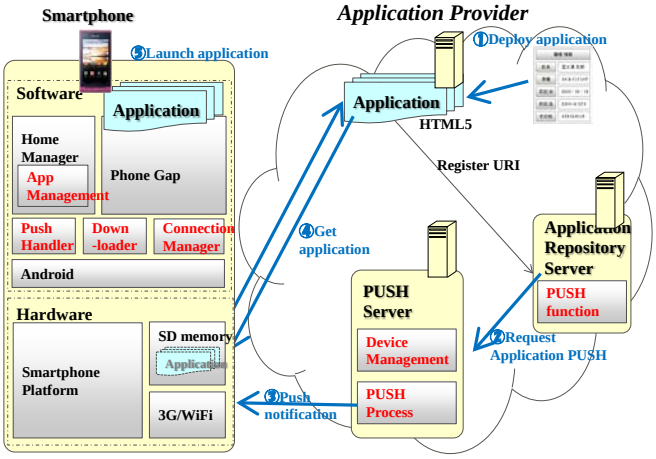


Figure 3: Implementation of the proposed system

4.1 Components

The system consists of the following components:

'Application Repository Server': This stores universal resource indicators (URIs) of applications. It picks up an appropriate URI upon an order from the 'Task Manager,' and commands the 'Push Server' to send a PUSH message with the URI.

'Application Provider': This stores an actual application and provides an interface to send it upon request.

'Task Management': This manages the user's context on the basis of calculating data from sensor information. Once it evaluates the timing to send a particular application to a particular user, it issues a command to the 'Application Repository Server' to send the application.

'Push Server': This constructs PUSH messages and sends them to client devices. Upon receiving a command from the 'Application Repository Server' it chooses an appropriate client device to send a PUSH message to with an application URI. Google already uses the PUSH mechanism in its Android Cloud Device Messaging Framework [2]. However, it is not suitable for business users because it requires a Google Account. Therefore, we implemented a dedicated PUSH Framework.

'Push Handler': This is a background process working on a client device to receive a PUSH message from a 'Push Server'. It keeps a TCP connection to 'Push Server' and can communicate with the 'Push Server' as long as physical network connectivity through 3G or WiFi is maintained. Once it receives a PUSH message and it finds an APnP command in the message, it hands the command to the 'Home Manager'. In this implementation, the 'Push Handler' is realized as an android application.

'Downloader': This is a client module to download applications from the 'Application Provider' to SD memory in the smartphone upon request from the 'Home Manager'. In this implementation, the 'Downloader' is realized as a library module included in the 'Home Manager'.

'Home Manager': This displays a home screen. It has three functionalities. First, it is an application launcher to execute applications. Second, it performs management of icons to be shown on the home screen. The home screen is dynamically changeable on the basis of APnP commands from the 'Push Handler'. It displays only icons necessary in context to provide a simple user interface. Finally, it orders the 'Downloader' to download applications. In this implementation, the 'Home Manager' is realized as an Android application.

'PhoneGap' [8]: This is a runtime environment to execute applications stored in SD memory in the smartphone.

4.2 Work flow

A typical work flow is as follows:

1. Deploy applications
   Developers distribute their packaged applications via an application server provided by an application provider. After that, they register the URI of the application with the application repository server. IT administrators might have to supervise this step for enterprise usage.
2. Request Application Push

Once the context engine, which knows the user's behavior, detects an event in which the user wants to use the application, it creates a command message and requests the push server to a notify the target device via.

3. Push notification
   After receiving the request from the application repository server, the push server sends a push message which includes an APnP command to download and invoke the application.
4. Get application
   When the push handler inside the smartphone receives the pushed message, it analyzes the APnP command and downloads zipped resources of the application by utilizing the HTTP protocol. After completing the download, it stores them in the SD card which is accessible to the home manager and PhoneGap.
5. Launch application
   Finally, the downloader sends the Intent to the home manager to tell it that the download is complete. The home manager changes the screen and launches the application by invoking PhoneGap.

4.3 APnP Commands

APnP commands in a push message consist of the following elements:

Table 1: List of APnP Commands

Command Name	Description
APCTL	Operations of the Downloader, such as download an application, update part of an application, delete an application, etc.
HMCTL	Operations of the Home Manager, which displays options on how it shows notifications to a user.
HMOP	Option values which the Home Manager use, such as context information, delay period before executing an application, etc.
URI	The URI of applications.
RESOUCES	The resource name of the application to be downloaded.

4.4 Packaged application

An application is packaged as a set of resource files, a list of resources, and a manifest file.

Figure 4 is an example of a list of resource files. All files are described in plain text.

```
Index.html
pageA.html
pageB.html
webapp-manifest.json
walkingApp.manifest
images/walking.png
images/title.jpg
```

Figure 4: Example of a list of resource files.

In this example, HTML files ('*.html') and image files ('walking.png', 'title.jpg') are the actual content of the web application. The set of files constitutes the minimum resources in a common web application. Additional resources specific to this system are the 'walkingApp.manifest' and 'webapp-manifest.json'. The 'walkingApp.manifest' is the name of this file. It is specified in HTML5 [1] as a means of local execution. The resources described in this file are cached in local storage and can be executed without communication to a web server. The'webapp-manifest.json' is a manifest file to describe the properties of the application. Figure 5 is an example of 'webapp-manifest.json'.

```
{
  "appURI":
"http://www.example.com/walkingApp/",
  "appName": "walking",
  "description": "This application recommends a
good walking place."
   "creator": "fujitsu_healthcare@example.com",
  "version": "1.0 ",
  "manifest": "walkingApp.manifest",
  "icon": "images/walking.png ",
  "toppage": "index.html"
}
```

Figure 5: Example of a manifest file.

An application package is uniquely identified by an 'appURI'. The Downloader refers to the appURL to download the application and deploys the downloaded files in the local storage in the same folder tree. For example, files downloaded from http://www.example.com/ walkingApp/ are placed in /sdcard/apps/www.example.com /walkingApp/.

'appName' is the name of the application. 'description' is a detailed explanation of the application. 'creator' is the contact address of the developer. 'version' is the version number of the application. 'manifest' is the name of the HTML5 manifest file. The Downloader can get information about where each resource is located. 'icon' is an image file to show an icon on the home screen. 'toppage' is the entry page to be displayed first after launching the application.

5 EVALUATION

5.1 System evaluation

To evaluate the feasibility of the proposed system, each component shown in Section 4 was implemented in the following hardware:

Fujitsu PRIMERGY TX100: This was used an IA server to run the Application Repository Server, PUSH Server and Applications. It had an Intel Xeon Processor (E3120 3.16 GHz), 8GB DDR2 memory, 1TB SATA HDD, and 1000Base-T for NIC. The operating system was Cent OS 5.5

Smartphone T: This was a smartphone equipped with Android 2.1 to run client-side modules, such as Push Handler, Downloader, Home Manager, and PhoneGap, as a runtime environment.

Smartphone N: This was a smartphone equipped with Android 2.3.

Figure 6 shows an example screen transition.

The home screen provides a user-friendly interface to use application. The screen is switched based on user's context and displays limited icons for each screen even though a user installed a lot of applications. It should be a good navigation by giving a user limited choice to do in a particular context.

Home screen for outside. | Notification of an business application in the meeting room. | Execution of the application.

Figure 6: Example of Context-based Home Screen

The picture on the left is a snapshot of the home screen. In this example, it displays only icons suitable to outside when a user stays out. Example of context is outside, home, work, etc.

The picture in the middle is a snapshot of a notification that a business application is in a meeting room. In this case, the home screen changes from "Outside" to "Meeting room" as soon as the user comes into the company's meeting room. After that, a new icon of the application appears in the center of the screen, and a small icon is deployed on the home screen. It is optional to display icons on the home screen without invocations of an application.

The picture on the right is a snapshot of running the notified application. After notification, the system automatically executes the application without the user having to perform an operation.

After the user stops using the application, it might disappear from the device if the user no longer needs it.

It provides a simple user interface to a user and helps to discover application icons in the particular context.

5.2 Performance evaluation

5.2.1. Test Environment

To improve the user experience, the period from when a user notices a notification till when he/she starts to operate is very important. The time from receiving the PUSH message till the icon appears or till the application is invoked may be a bottleneck to providing the user a sufficient response rate. Therefore, we evaluated the difference in performance for four kinds of implementations on smartphones T and N. The specifications of each smartphone are shown in Table 2.

Table 2: Specifications of smartphone T and N

Model	Smartphone T	Smartphone N
CPU	QSD8250 1GHz	S5PC110 1GHz
Internal Memory	512MB RAM	16GB iNAND
External Memory	2GB Micro SD	Use a partition of internal memory
OS	Android 2.1	Android 2.3

Four kinds of implementation are determined by the combination of how to download and store an application and whether the application is compressed or not. The differences between the implementations are as follows:

I) Download a zip-compressed file and store it in SD memory. After that, unzip it and store the unzipped files.
II) Download a zip-compressed file and store the unzipped files in SD memory by utilizing ZipInputStream class without storing the zipped file.
III) Download an uncompressed file packed by using a tar tool and store it in SD memory. After that, unpack it and store the unpacked files.
IV) Download the uncompressed file packed by tar and store the unpacked files in SD memory by utilizing the TarInputStream class without storing the packed file.

At this point, we can make the following hypothesis:

Hypothesis:
The best method to reduce the delay is that an application package is compressed and is stored it directly as unzipped files in local storage.

To prove the hypothesis, we should confirm the following:

A) Total throughputs in Case-II and Case-IV are higher than in Case-I and Case-III, respectively.
B) Total throughputs in Case-I and Case-II are higher than in Case-III and Case-IV, respectively.

It is expected that (A) and (B) is true if the CPU is sufficiently fast. The reason why (A) should be true is because two times the file I/O operations occur in Case-I and Case-

III, in comparison with Case-II and Case-IV. The reason why (B) should be true is because the increase in data traffic has a large impact on the total performance in comparison with a reduction in the load of decompression process.

We measured the actual time spent from the beginning of downloading till the end of storing an application package for each case. In Case-I and Case-III, we also measured the download time and unpacking (or decompressing) time. The sum of both times is the total time.

The application package to be downloaded was a 167KB file containing 25 zip-compressed files in 5 folders, the re-write is a guess. The original is unclear.>. The total size before compression was 655KB. The compression rate was 27%. The application was downloaded through a WiFi connection.

5.2.2. Test results

Table 3 shows the results of the measurement. Each time in Table 3 is the average of 10 measurements.

Table 3: Measured results

Item #	Device	Case #	Total time (msec)	Down-load time (msec)	Un-pack time (msec)
N-I	N	I	203.2	110.2	93
N-II	N	II	190.8	-	-
N-III	N	III	434.4	264.6	169.8
N-IV	N	IV	402	-	-
T-I	T	I	817.6	197.9	619.7
T-II	T	II	653	-	-
T-III	T	III	1318.6	524.6	794
T-IV	T	IV	861.4	-	-

Figure **7** compares the elapsed times for each test case.

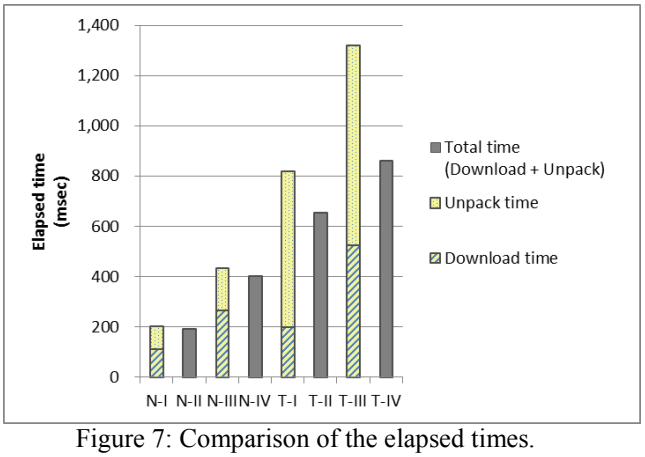


Figure 7: Comparison of the elapsed times.

Figure 8 compares the throughputs calculated from the results in Table 3.

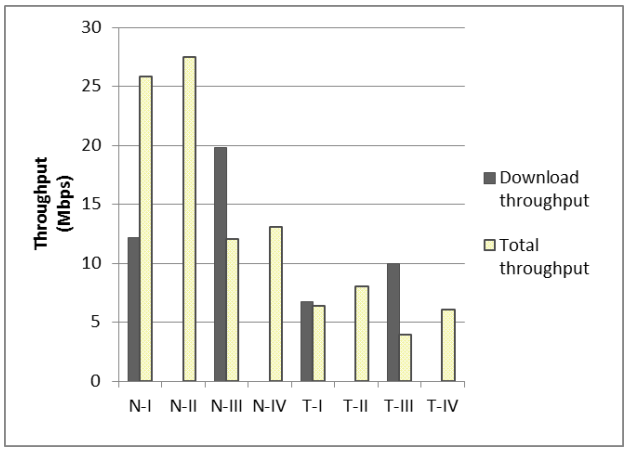


Figure 8: Comparison of throughputs.

5.2.3. Discussion on experimental results

The histograms reveal the following facts:

i). Smartphone N is approximately twice as fast as smartphone T in all cases.

ii). The fastest implementation is Case-II on both smartphones.

iii). There is little difference in time between Case-I and Case-II or between Case-III and Case-IV on smartphone N, whereas there is a significant difference on smartphone T.

iv). On smartphone N, the cases of using a compressed application package (Case-I and Case-II) are approximately two times faster than cases without compression of the application package (Case-III and Case-IV), whereas on smartphone T, there is little difference in total throughput between Case-I and in Case-IV.

The reason why smartphone N is faster than smartphone T is due to their different storage devices. The application is stored in external memory. The Micro SD memory used in smartphone T is significantly slower than the NAND flash memory used in smartphone N. Therefore, the difference in I/O performance has a large impact on total throughput. Fact-iv is due to the same reason.

Even the throughput of downloading on smartphone N is faster than on smartphone T. This result is presumably due to the difference in OS version. Android 2.3 is faster than Android 2.1 because it has the JIT (Just-In-Compiler). However, smartphone T has enough throughput because even the worst total time is less than a second if the application is zip-compressed.

In addition, the results of the cases without compression of application package were worse than expected. It is natural that data traffic is increasing, and download time is consequently increasing. However, there was difference between the unpack operations and there was supposed to be little difference between them because the performance of TarInputStream class on Android was poor. Hence, an application package should be compressed.

To summarize, the experimental results satisfied conditions (A) and (B) described in 5.2.1. Therefore, we were

able to confirm that the hypothesis in 5.2.1 is true. In addition, it took less than 1 second even in the worst case using a compressed package in smartphone T. A user would rarely be able to notice the delay. The proposed method can be used for delivering applications since the performance of smartphones will improve in the future.

The results of the experiment utilizing an actual business web application confirmed that the smartphone was notified of the application's delivery within a few seconds after an event was issued, and that an application icon was dynamically added to the home screen, and the application was invoked shortly after.

6 USE CASE

In this section, we present an effective use case of an APnP system.

To simplify the user interface further, the home screen could be removed. That is, an application could be directly executed without displaying the home screen once it is distributed to our devices.

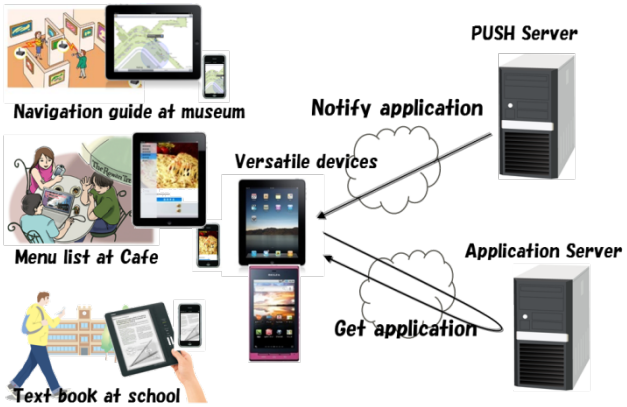


Figure 9: Example of service.

Figure **9** shows an example service. A user's device can be transformed into a textbook while he/she is at school. While he/she is in a museum, it can be a navigational guide. In such cases, a personal device with our system can behave as if it were a dedicated device and provide the user with a rather simple interface. Such a functionality might be more useful for a tablet device than for a smartphone.

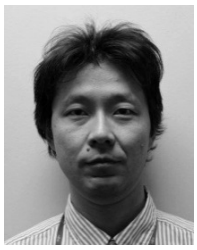
7 CONCLUSION

We proposed the concept of APnP and prototyped a system where appropriate applications are automatically distributed and executed. Such applications are made available only at the right moment and disappear when a user no longer requires them. We confirmed that APnP can reduce the installation steps and has a sufficient response. In conclusion, APnP can help a user in his/her daily activities and provide a new user experience different from the conventional one.

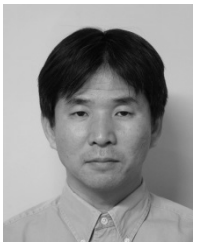
REFERENCES

[1] HTML5, http://dev.w3.org/html5/spec/Overview.html.
[2] Android Cloud to Device Messaging Framework, http://code.google.com/android/c2dm/index.html.
[3] Google Android Market, https://market.android.com/.
[4] Apple iTunes, http://www.apple.com/itunes/.
[5] Jason Grigsby, ``Native vs Web vs Hybrid Mobile Development Choices,'' http://assets.en.oreilly.com/1/event/34/Native%20Apps%20vs_%20Mobile%20Web%20vs_%20Hybrid%20Apps_%20Mobile%20Development%20Choices%20Presentation.pdf (2010).
[6] Ringcentral Survey, ``Small Business Professionals Admit to Smartphone Addiction,'' http://blog.ringcentral.com/tag/ringcentral-survey (2011).
[7] SBE Council, ``Saving Time and Money with Mobile Apps – a small business "app"ortunity –,'' http://www.sbecouncil.org/uploads/Mobile%20APP%20Final%20Report%20SBE%20Council.pdf (2011).
[8] PhoneGap, http://www.phonegap.com/.
[9] MIT Project Oxygen, http://oxygen.csail.mit.edu/.

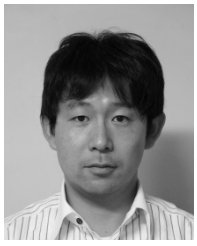
(Received February 27, 2012)
(Revised March 1, 2013)



Hidenobu Ito received the BE and ME degrees in Mathematical Sciences from University of Osaka Prefecture, Japan, in 1991 and 1993, respectively. He joined Fujitsu Laboratories Ltd in 1993. His current research includes mobile computing and human centric computing



Kazuaki Nimura received the BE and ME degrees in Graduate School of Information and Communication Engineering, Tokyo Denki University, Japan, in 1992 and 1994, respectively. He joined Fujitsu Limited in 1994 and transferred to Fujitsu Laboratories Ltd, in 1997. His current research includes advanced technology of smart device and human centric computing.



Yosuke Nakamura received the BE and ME degrees in Graduate School of Engineering, Yokohama National University, Japan, in 2000 and 2002, respectively. He joined Fujitsu Laboratories Ltd in 2002. His current research includes advanced technology of personal computer and human centric computing.

Akira Shiba received the B.S. and M.S. degrees in electronics engineering from Sophia University in 1980 and 1982, respectively. He joined Fujitsu Laboratories Ltd. in 1982. Since then he has been engaged medical electronics and mobile computing, and is currently a Research Manager of human centric computing technology.

Nobutsugu Fujino received the B.S. and M.E. degrees in electronics engineering from University of Osaka Prefecture in 1984 and 1986, respectively. He joined Fujitsu Laboratories Ltd. in 1986. Since then he has been engaged radio communication systems and mobile computing, and is currently a research manager of human centric computing and multi device interaction technology. His research interests include mobile and ubiquitous computing and network applications. He received IPSJ Industrial Achievement Award in 2003. He received Ph.D. degree in informatics from Shizuoka University in 2008.