Journal of Zhejiang University-SCIENCE C (Computers & Electronics)
ISSN 1869-1951 (Print); ISSN 1869-196X (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

Effective operation and performance improvement methods for OMTP BONDI-based mobile Web widget resources*

Jiwoong BANG[1], Daewon KIM‡[2]

([1]*Department of Computer Science, Dankook University, Juk-jeon 440-701, Korea*)

([2]*Department of Multimedia Engineering, Dankook University, Cheon-an 330-714, Korea*)

E-mail: muse@dankook.ac.kr; drdwkim@dku.edu

Received Nov. 1, 2010; Revision accepted May 20, 2011; Crosschecked Sept. 1, 2011

Abstract: The Open Mobile Terminal Platform (OMTP) is a global forum made by telecommunications providers to promote user-oriented mobile services and data businesses. Devised by OMTP, BONDI is a browser-based application or a mobile Web run-time platform to help widgets make good use of the functions of mobile devices in a secure way. BONDI enables applications programmed with Web standard technologies such as HTML, JavaScript, CSS, and AJAX to reach the internal functions of mobile devices. Since BONDI, which is not just a simple network application, can reach the internal resources of devices in standard ways, it enables the application and widgets to be developed regardless of the operating system (OS) or platform. Web browser-based widgets are vulnerable to the network environment, and their execution speed can be slowed as the operations of the widgets or applications become heavy. Compared with the native widgets inside the device, however, those Web widgets will be continuously used thanks to the user-friendly simple interface and the faster speed in using Web resources. This study deals with a method to effectively operate and manage the resources of OMTP BONDI Web widget and provides improved results based on running performance evaluation experiments. The experiments were carried out to improve the entire operating time by enhancing the module-loading speed. In this regard, only indispensable modules were allowed to be loaded while the BONDI widget was underway. For this purpose, the widget resource list, which can make the operating speed of the BONDI widget faster, was redefined while a widget cache was employed. In addition, the widget box, a management tool for removed widgets, was devised to store temporarily idle widgets.

Key words: Mobile, OMTP, BONDI, Web, Widget, Resources
doi:10.1631/jzus.C1000379 **Document code:** A **CLC number:** TP393

1 Introduction

In the information era, as computers have become popular based on the development of high-speed networks, a growing number of people use the Internet and mobile phones without the constraints of time and space. Exchanging information via the Internet has raised the quality of life. For example, thanks to the Internet, people can search for necessary information or post articles on Internet community sites and on their blogs. These network activities promote information exchange regardless of distance among users and have brought various multimedia content to the Internet. This is why there have recently been studies aiming to suggest useful methods for people to effectively use content via their feature or smart phones. In the past, most mobile phones were 'feature phones' specialized in voice communication and short message service (SMS).

‡ Corresponding author

* Project supported by the Research Fund of Dankook University in 2009

However, currently there is an emergence of powerful and versatile terminal devices called smart phones, which is a response to the level of use of traditional functions (telephony and SMS) approaching saturation in many regions of the world, and the search for new applications for new projects. These have resulted in the popularity of phones that combine a whole range of functions and support different specialized applications. With the improvement of 3G technology, mobile phones have been equipped with high performance chips, antennas, and batteries. Mobile phones have become convergence mobile devices by which users can enjoy various functions other than voice service and SMS. Recently, smart phones such as Apple's iPhone and Google's Android phones have been launched. These smart phones, which are able to install, remove, and update programs, have attracted a lot of users and developers using Web applications and widgets. A widget is a simple application by which users of personal computers, mobile phones, and blogs can access information about the weather, calendar, news, or stock prices without running a program (Taivalsaari *et al.*, 2008; Reynolds, 2009). Early mobile widgets were mainly provided on the initial screen of phones in consideration of exposure frequency rather than a technological approach or the service itself (Gartner, 2007). These widgets aimed to reduce the execution time of applications, where users could make good use of content on the initial screen without connecting to a WAP (Wireless Application Protocol) browser (W3C WG-1, 2010). Afterward, with the introduction of smart phones and Web 2.0, as there have been numerous studies to make good use of the Web by mobile devices, the widget has become not only a way for providing information on the initial screen, but also a Web application working within mobile devices. Web applications provide an application-like environment via Web pages through HTTP in a Web browser. To promote the use of Web applications, some standardization projects have been underway, including Device API, HTML5. In particular, the W3C MWBP (Mobile Web Best Practices) working group has been trying to standardize technical model cases for establishing the mobile Web (W3C WG-3, 2006; W3C WG-2, 2009). Amid the growing smart phone market and opening mobile platforms, widgets have been allocated in various forms such as Web applications and Web widgets, running on the home screen, providing information to users in need for updates. There are some problems, however, due to the fact that smart phones are based on different platforms and Web applications in accordance with the phone's manufacturer or the telecommunications operator. Therefore, several development groups, including OMTP BONDI, JIL (Joint Innovation Lab), and Phone Gap are developing the mobile Web and standardizing projects aimed at creating hybrid applications (Ballon, 2009; JIL, 2009; OMTP BONDI-1, 2009). This study establishes OMTP BONDI widgets compatible with W3C in attempt to mitigate the problems of overflow in memory and slow execution speed of widgets due to excessive Web content operations when a Web widget or Web application is running. For this reason, only indispensable components of a widget are loaded to reduce execution time, memory requirement, and to increase the value and security of information handled by the widget. This study also aims to effectively manage downloaded Web widgets, by testing the OMTP BONDI widget and analyzing the results.

2 OMTP BONDI

One of the most noticeable aspects of Mobile 2.0 is the Web application technology. Thanks to the introduction of Web 2.0, there have been changes in the ways of selling content and browsing information, to effectively make good use of Web widget applications. The Web 2.0 uses network as a platform as it delivers or receives applications thoroughly via a browser. Users can obtain, manipulate, and control the data on the site. In addition, the Web 2.0 has a participatory architecture in which users can add or edit value to the application according to their needs. The Web 2.0 also has a rich, interactive, and user-friendly interface based on Ajax or similar frameworks.

Finally, there are some social-networking aspects and enhanced graphical interfaces such as gradients and rounded corners which were absent in the so-called Web 1.0 era. The mobile Web application is divided into three parts: Native App, Web App, and Hybrid App (Tredinnick, 2006; Chetan, 2009). In Table 1, the main features of these three parts are compared. In terms of the iPhone OS of Apple, Android of Google, and WebOS of Palm,

Table 1 Comparison of Native, Web, and Hybrid App

Functionality	Value/Description		
	Native	Web	Hybrid
Graphics	High	Low	High
Marketing	OK	N/A	OK
Off-line	OK	OK	OK
Mash-up	N/A	OK	OK
Multi-platform	Hard	Easy	Medium
Storage	Local	Server cloud	All
Device	Easy	N/A	Easy
Multi-tasking	N/A	OK	OK
S/W update	Re-install	Modify	Re-install
App cycle	Source/Lip	Source/SaaS	All
UI structure	High	Low	Medium
UI expression	High	Low	Medium

development projects have recently been carried out to create run-time engines compatible with mobile Web platform standards such as OMTP BONDI and JIL (Hoegg *et al.*, 2006; O′Reilly, 2007). The DAP (Device API and Policy) working group was proposed in Dec. 2008 and established in June 2009. As a result, after the standard content of OMTP BONDI and JIL was gathered and submitted to W3C, there has been a standardization project for Web applications and widgets. BONDI, a part of OMTP activities, is a mobile Web run-time platform that helps browser-based applications or widgets approach mobile phones in a secure way. Its first version, 1.01, was announced on May 26 2009, applying the W3C standard widget (OMTP BONDI-2, 2009; OMTP BONDI-3, 2009). The architecture of BONDI Version 1.1, the research source of this study, is shown in Fig. 1, where the widget is a conversational Web application or Web widget compatible with BONDI.

The widget resource of Web widget is downloaded from a remote widget resource provisioning server into a mobile device or a computer. A user can install, remove, run, and close it, using the widget user agent. Before running, the widget exists as a widget resource inside the server or the device. The widget resource is a package-like compressed file in which there are Web documents such as HTML, SVG (scalable vector graphics), JavaScript, and information including the configuration document and digital signature. With Web-based programming languages, a website is constructed, storing information on Web servers to give users information if necessary. A Web application accesses the network like an application program, and a Web engine is one of the key elements in a Web browser and the widget user agent. Since the widget engine basically includes a Javacript engine, it can provide rendering for markup documents such as JavaScript, CSS, HTML, and SVG. Most popular Web engines are Web-kit and Gecki. The BONDI Web engine supports JavaScript extension, API access control, native applications, and extension API, in which an expandable BONDI module exists. A browser is an application using a Web engine by which it deals with content on a website. Generally speaking, in a browser, a user can use a bookmark management, and check searched records, changing Web engine settings and user functions. The widget user agent, a management tool for widget resource, provides functions for installing, removing, running, and closing of a widget installed on a widget engine. The widget user agent requires the functionalities of a widget downloader and a widget manager application. The widget downloader application possibly exists as a widget or a Web page while the widget manager provides functions for installing, removing, running, and closing of a widget (OMTP BONDI-4, 2009; OMTP BONDI-5, 2010).

3 Effective methods for achieving effective operation and management of widget resources

3.1 Improving widget-processing time by adding the widget configuration document element

The OMTP BONDI-based widget is basically embodied based on Web documents. The widget contains various files such as HTML text, JavaScript code, CSS, and image files necessary to run the widget itself. BONDI compresses the widget package into ZIP files, whose file extension is ‘wgt’. These wgt files are called widget resources. Table 2 provides information about data organizing the widget resources.

Other than the files shown in Table 2, more files can be added and compressed if necessary. For example, in accordance with the characteristic of the widget, various files such as images of the widget, CSS, JavaScript, and digital signatures can be optionally selected. Digital signatures, being used mainly for

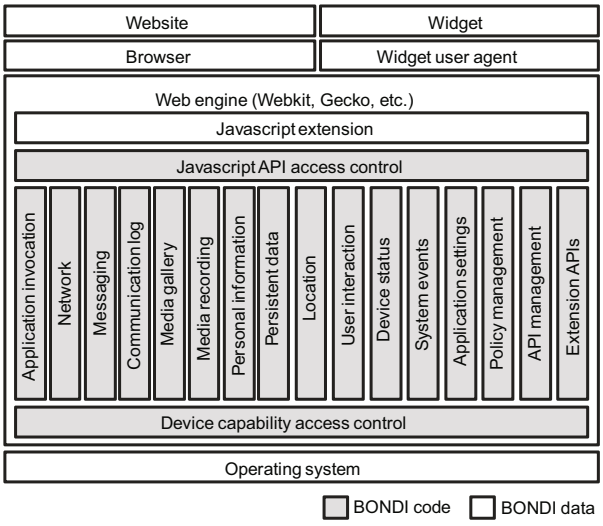


Fig. 1 OMTP BONDI Version 1.1 architecture

Table 2 Information about files in widget resources

Item	File name	Description
Configuration document	config.xml	Including widget information such as widget name, author, and license
Digital signature	signature.xml	Including digital signature information for widget security
Start file	index.htm	Widget starting Web document & firstly loaded material when executed
Widget icon	icon.png	Widget icons

e-commerce security, serve as a process to guarantee the authenticity, integrity, and non-repudiation of online messages. The most indispensable files in a widget resource are 'config.xml' and 'index.htm'. 'Index.htm', a start file, is a document to be first loaded when the widget runs, forming the first page of the widget. Before running the start file, it is necessary for the user agent to recognize information about the widget. The information is included in 'config.xml', a configuration document. The configuration document defines the widget information by means of element and attribute of XML. Table 3 shows a 'config.xml' of a clock widget; this case can be seen in the BONDI widget gallery. In Table 3, <widget>, a higher element, defines the values of attributes such as xmlns (appearance of widget), id (widget identity value), version, height and width, and mode. The next element <name> denotes the name of widget while <description> provides an explanation of the widget. <author> shows the developer's name and e-mail. In <content>, 'src' implies the path of the start file. In <icon>, 'src' shows the path of icons. In <feature>, there are 'name', the name of the module on which the widget is based, and 'required' that judges whether or not the approach to the module is available.

Table 3 'config.xml' used in a clock widget

```
<?xml version="1.0" encoding="utf-8" ?>
<widget xmlns="http://www.w3.org/ns/widgets"
id="http://bondi.omtp.org/widgets/clock" version=
"1.02" height="92" width="100" mode="window">
<name>clock</name>
<description>Sample widget which tests the user
interaction interface, including menus, activation,
deactivation, screen orientation, etc.</description>
<author email="toby.ealden@gmail.com">OMTP
</author>
<content src="index.htm"/>
<icon src="images/icon.png"/>
<feature name="http://bondi.omtp.org/api.ui"
required="true"/>
<license>Licensed to OMTP Ltd. (OMTP) under
one or more contributor license agreements. See the
NOTICE ... You may obtain a copy of the license at
http://bondi.omtp.org/BONDI</license>
</widget>
```

Finally, <license> includes the license information of the widget. Among the configuration document elements, <feature> provides information about the module that the widget is willing to employ. The module means the API provided to use sound devices, display devices, the file system, and the memory in the mobile device. Hence, to use BONDI APIs defined inside the widget, <feature> should include the information requiring the authority on APIs that the widget is willing to employ. In addition, the name of the module is shown as a URL, a file address on the Internet. Table 4 presents API authority information, available in <feature>, according to the modules. This authority information demands that the widget should access only the appropriate APIs that the widget engine reads or the internal system calls. The information sometimes bans APIs that have no access authority. The widget runs by the widget engine that basically loads all the modules provided by BONDI. If the running widget requires access to an API, the engine checks the API authorities recorded in the <feature> element and then allows the widget to access particular APIs. This process of BONDI has an unreasonable characteristic; it increases the widget-processing time as

Table 4 The list of API authorities available in <feature>

BONDI module	Module URL	Functions of API use authority
App launcher	http://bondi.omtp.org/api.applauncher.launch	Widget & native applications
Messaging	http://bondi.omtp.org/api.messaging.email.attach & 5 sites	SMS, MMS, E-mail resources in terminals
UI	http://bondi.omtp.org/api.ui	Graphic resources
File system	http://bondi.omtp.org/api.io.file.read & 1 site	File input/output
Device	http://bondi.omtp.org/api.devicestatus.get & 2 sites	Accessing terminal resources
App configuration	http://bondi.omtp.org/api.appconfig.get & 1 site	Setting widget & application environment
Geolocation	http://bondi.omtp.org/api.location.position	GPS tools
Camera	http://bondi.omtp.org/api.camera.get & 1 site	Camera
Communication log	http://bondi.omtp.org/api.commlog.sms.get & 3 sites	SMS, MMS, E-mail call-log data
PIM	http://bondi.omtp.org/api.pim.contact.read & 5 sites	Personal information management

it causes redundant memory and loads unnecessary modules. To solve such problems, we devise ways to load actually necessary modules to improve effectiveness while the widget is running. We add a new attribute value 'loaded' into the <feature> element in the configuration document. An example of single declaration in the existing <feature> is given as follows:

<feature name="http://bondi.omtp.org/api.ui" required="true"/>

where 'name' represents the first attribute recording URL of the module to be used, and 'required' represents 'true' or 'false' as permission or rejection, respectively, of access to the API. To improve its operation, add a new attribute 'loaded' by which only allowed modules are loaded in the engine. In the attribute 'loaded', use 'true' and 'false' as signs to load the module or not. If the value of 'loaded' is not defined in advance, then the default value is 'false'. A declaration is shown below with 'loaded' in <feature>:

<feature name="http://bondi.omtp.org/api.ui" required="true" loaded="true"/>

This way, with the new attribute 'loaded' added to the <feature> element, the widget engine becomes able to load only the necessary modules.

In the widget engine, accordingly, unnecessary memory resources and widget-processing time can be reduced. Moreover, in the device where the widget is installed, much personal and important information is stored. By running only the necessary modules, it is possible to significantly lower the possibility of damage and the threat of personal information exposure caused by widgets created by malicious developers.

3.2 Improving widget-processing time with the redefinition of the widget resource list

BONDI uses the widget resource list (WRL) to manage information about installed widgets. WRL is a kind of database that is based on a database application and stored in files or the device. When a widget is installed by the widget installer of the user agent, the widget manager records the installation path and time information on the WRL. Table 5 presents the structure of WRL defined in OMTP BONDI.

Table 5 Widget resource list defined in BONDI

Item	Content
Widget URL	Widget installation path
Time stamp	Widget installation time information

Since the information for running the widget is recorded in 'config.xml' inside the widget resources (*.wgt), the WRL records and manages only the two types of information about the widget: installation path and time. The widget manager gives the widget URL in the WRL to the engine when the widget starts to run. Then the widget engine reads 'config.xml' in the resource in the widget URL to obtain information necessary for running the widget.

Fig. 2 is a signal flow showing the running process of a widget. According to the figure, there are 11 stages required for the widget to display information on the screen. Due to the numerous stages, widget-processing time is increased. In particular, stages

6–9 account for most of this processing time. The reason is that the widget resources are compressed in ZIP form. On stage 6, the widget resources are decompressed to determine whether or not the widget is compatible with BONDI definitions. During the process, if a lot of data is included in the widget resources, it takes a longer time to complete the check-up. If there is a digital signature on stage 7, then it is examined on stage 8 through the processes of encryption or decryption methods. Finally, on stage 9, 'config.xml' is loaded to obtain information about running the widget. It takes some time because the configuration document based on XML is read through the XML parser. As explained previously, during stages 6–9, the running time of widget is prolonged. To solve the problem and simplify the widget-running process ruled in the existing BONDI, this study deals with a new WRL. On stage 9, when the widget runs, because its necessary information is put inside the resource, 'config.xml' is repeatedly loaded. 'Config.xml' contains information necessary for running the widget. If the information is recorded in WRL while the widget is installed, then it is possible to reduce the entire running time by skipping the XML parsing process that would be carried out on stage 9. Table 6 shows the redefined WRL structure to record such widget information.

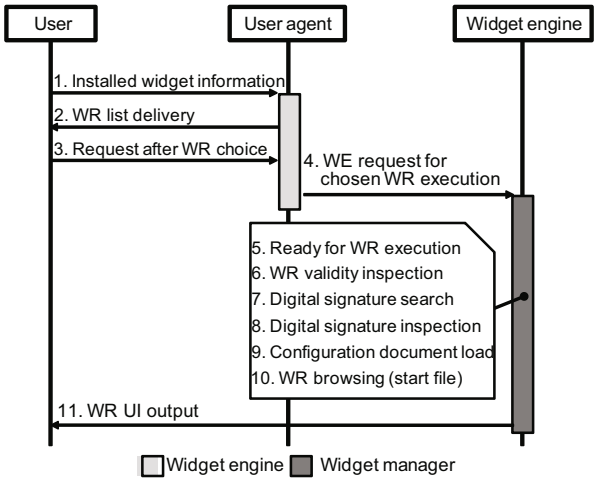


Fig. 2 A flow diagram of widget operation defined in BONDI

Table 6 shows newly added information including the widget identity value, name, explanation, version, output mode, output width and height, the path of the start file, and the path of icon files. Also, necessary authority information for the improvement

Table 6 The redefined architecture of the widget resource list

No.	Item	Note
1	Widget ID (identification value)	
2	Widget name/title	
3	Widget description	
4	Widget version	
5	Widget output mode	Newly
6	Widget output width	added
7	Widget output height	information
8	Widget content URL	
9	Widget icon URL	
10	BONDI API permissions	
11	BONDI module permissions	
12	Widget installation URL	BONDI
13	Widget installation time stamp	BONDI

of widget-processing time by adding elements of the widget configuration document in Section 3.1 was recorded on the module-permissions to quickly apply the information when the widget engine runs. Fig. 3 shows a flow diagram where the widget-installing procedure was changed to add the redefined widget information to the WRL.

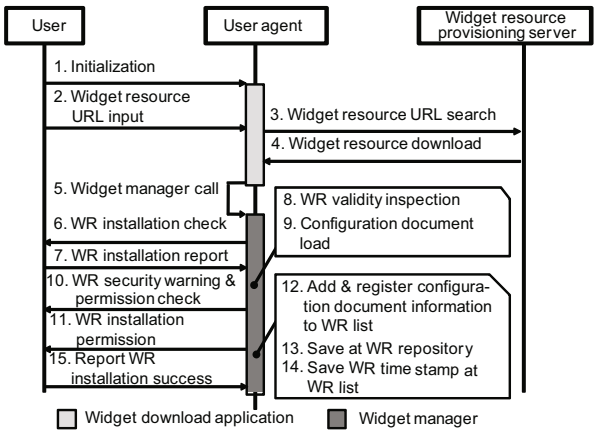


Fig. 3 A flow diagram of the widget-installing process to record the widget information on the redefined widget resource list

In comparison with the procedure defined in BONDI, stages 9–12 were added. Even though this new method takes a longer time in installing a widget than the existing method, because there are additional stages, the new method is able to reduce running time when a user starts the widget. In other words, once the widget resource is installed, its running process occurs whenever the user requires the operation. This is an effective time-saving widget operation. Furthermore, stage 10 shows the user all

the resources accessed by the widget inside the user's device. This process strengthens stability in terms of personal information protection, possibly because of stage 9 where 'config.xml' is loaded. To apply the redefined WRL information, the widget-executing process, based on the existing BONDI, was changed. The flow diagram is shown in Fig. 4. In contrast to the existing procedure defined in the existing BONDI, Fig. 4 does not include a validity check, originally conducted on stage 6, because the validity check was already completed during the widget-installing process. Also, to obtain the widget information from 'config.xml', the XML parsing process was abridged and the information was given directly from the WRL. In this way, the widget-starting time was shortened while the widget-installing and its operating management became more effective and faster with the security warning process during the widget resource-installing process.

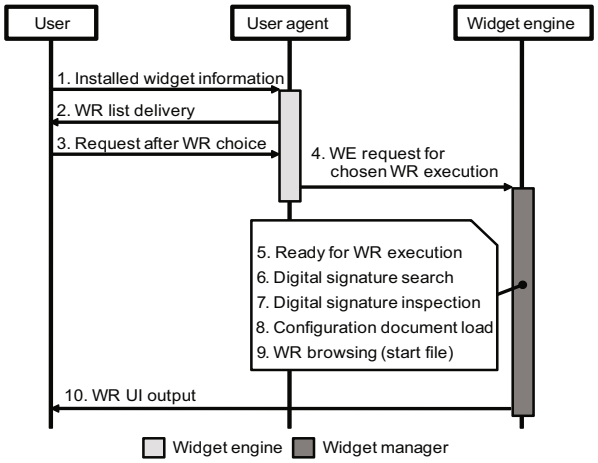


Fig. 4 A flow diagram of the widget-running process with the redefined widget resource list

3.3 Improving widget execution time with the virtual widget cache

One of the most affective elements of the widget-starting procedure is decompressing the widget resources that are generally compressed as ZIP files which have to be decompressed each time a widget is running. If it contains large-capacity files, the time required for decompression becomes longer. To reduce the decompression time, we devise the virtual widget cache like virtual memory used in computers. In short, a certain memory space in an external storage was used as a main memory. BONDI can appoint an SD card or SDRAM as storage for widget resources, creating a temporary folder where a widget can run. The temporary folder is repeatedly created and removed as the widget runs. The virtual widget cache saves widget-starting time by decompressing and keeping frequently used widget resources in the resource storage instead of removing them. For managing the virtual widget cache, the widget cache manager was added in the user agent, aiming to manage information of frequently used widgets and stored data in the virtual widget cache. The widget cache manager uses the widget cache table to effectively manage widget information (Table 7).

Table 7 Widget cache table structure

Item	Description
Widget number	Number for distinguishing widgets
Widget ID	Widget identification value
Widget last execution time	Last running time of widget
Widget frequency count	Number of widget executions

Table 7 shows the widget number that denotes the number of managed widgets in the virtual widget cache. The widget number limits the maximum number of widgets to be managed, changeable by using the widget cache manager. Widget ID represents the widget ID on WRL. It is used to find out whether there is widget data to run within the virtual widget cache. Widget last execution time implies the time when the widget is conducted lastly; the time is recorded when the widget is closed. Widget frequency count means the conducted frequency except the case in which a widget is overlapped. The widget cache manager manages the virtual widget cache in accordance with the widget last execution time and widget frequency count. For instance, in the virtual widget cache, suppose that the maximum number of widgets able to be stored is five and that there are five widgets inside the cache. If the user runs a new widget, the widget cache manager first finds that it is impossible to record it in the current cache and then removes one of those five widgets to use the cache. At the moment, a widget with the lowest widget frequency count is first removed. Then, if there are widgets with the same values, a widget with the longest widget last execution time is re-

moved. Fig. 5 shows two kinds of widget-starting processes with the virtual widget cache. Fig. 5a is a process where the start file is loaded and printed out without decompressing the widget resource if there is widget data to run in the virtual widget cache on stages 1–8. In case of digital signatures, it is possible to dispense with the process because stage 9 in Fig. 5b already took the digital signature process before the digital signature was recorded in the cache. If there is no widget data inside the virtual widget cache, then the process in Fig. 5b should be applied so that there is no time-reducing effect. However, if a certain application is frequently used, the virtual widget cache can be efficiently employed. Owing to the simplified running procedures, it is possible to reduce the widget-starting time.

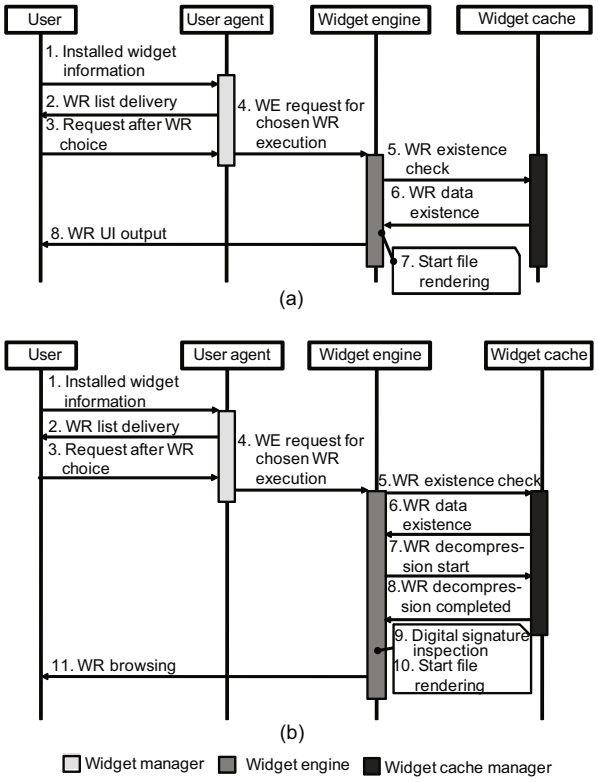


Fig. 5 A flow diagram of widget-running process with the virtual widget cache. (a) Resources exist in the widget cache; (b) Resources do not exist in the widget cache

3.4 Storage and management of widget resources with the widget box

Due to the small-sized memory of devices, sometimes there is no additional space for a new widget. Sometimes a widget that is not frequently used must be removed, or a widget that will be needed in the future must be kept even though it is unnecessary at the present time. The widget box, a personalized storage space, helps with removing this widget from the device but stores it on a remote server. Before removing the widget, the user can transmit its resource through the widget uploader to the widget box in a remote server and then remove the widget from the device. Fig. 6 is a flow chart where the widget resources are uploaded to the widget box. The widget resources stored in a widget box are available and downloadable any time the user accesses the server. The widget box is located in a widget resource provisioning server. These processes of downloading a stored widget from the user's widget box and installing it on their device in BONDI's process are the same.

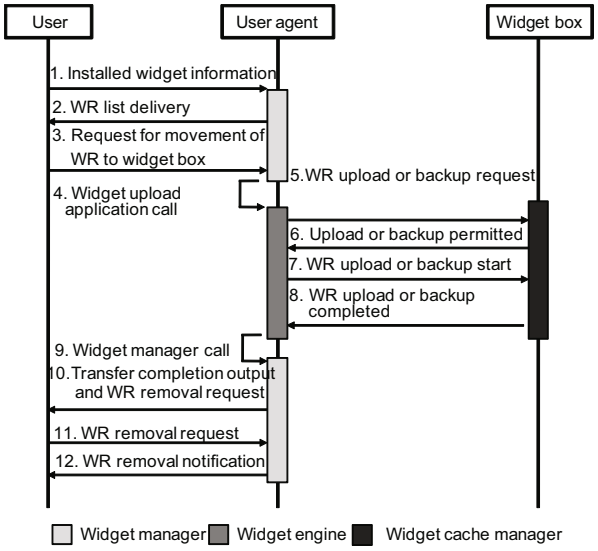


Fig. 6 A flow diagram of the procedures of uploading resources on the widget box and of removing widgets

A widget box helps a user secure sufficient memory and keeps the user from being financially burdened when his/her charged widget resource is removed.

4 Experiments and performance evaluation results

4.1 Simulation system

Fig. 7 is a structure of the simulation system established to find the difference between the wid-

get resource management and operation method suggested in this study and the standardized method of BONDI and to evaluate the performance of the two methods. In Fig. 7, the simulation system is divided into a mobile client, where a widget actually runs, and a widget resource provisioning server, where the user can download a widget package. In the mobile client, there is a browser running a widget based on Web documents. The browser includes a widget view displaying a Web widget on the screen, an HTML engine parsing HTML codes, a CSS engine dealing with style sheets in which general styles of Web documents are stored, a JavaScript engine handling JavaScript codes used in Web documents, and JavaScript extension using widget-related APIs provided from BONDI.

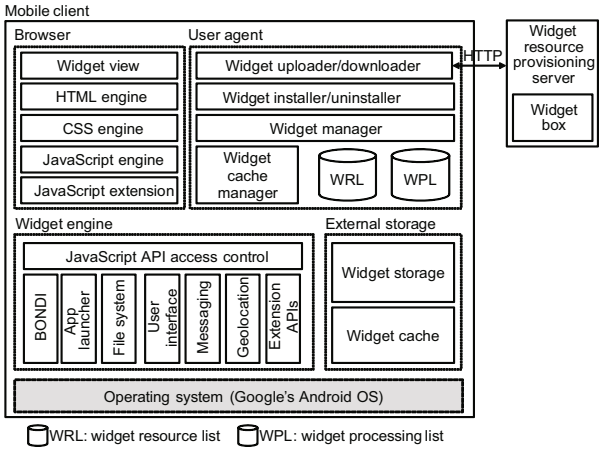


Fig. 7 Simulation system structure

In the user agent, there are a widget downloader downloading widget packages from a remote server, a widget uploader uploading widgets on a widget box in a widget resource provisioning server, and a widget installer/uninstaller dealing with installing and removing the downloaded widget packages. Additionally, the widget manager runs and sets the installed widget, dealing with addition, removal, and adjustment of the widget in WRL or WPL (widget processing list). WRL is a database managing the list of installed widgets. WPL is a database containing information of currently running widgets. The widget cache manager, an additionally devised module to test the performance of the method suggested in this study, is a module by which some frequently used data is recorded in an external storage and referred to later when a widget runs based on the widget manager. An external storage consists of a widget storage, an external memory storage storing downloaded or already installed widget packages, and the widget cache storing cache data. The widget engine, which plays one of the most important roles in running a widget, includes JavaScript API access control supporting APIs provided by BONDI in its high level and modules for BONDI API such as BONDI, App launcher, file system, user interface, messaging, geolocation, and extension APIs in the low level. Finally, there is the widget resource provisioning server that manages the widget's version to let the user download or update his/her widget package from the remote server. The mobile client and server, using HTTP, provide information about the widget updates and new widgets free of charge, by which a user can download or upload the user's widget on the widget box.

The device where tested applications worked was 'Galaxy S' of Samsung based on Android platform 2.1. Java language was used to create software. Fig. 8 shows the target device and applications actually used for the experiment. For the test, 10 sample BONDI widget packages were selected, and their performance was tested through a simulation program in accordance with the methods proposed in this study. The selected 10 widgets were Aplix Web Commander, GPS Monitor, GPS Speedometer, Simon 0870, Ajaxdemo, API Diagnostics, WWMM, BikeSharing, Send SMS, and App Manager. They were downloaded from OMTP BONDI website's widget gallery (http://bondidev.omtp.org/widget-gallery/default.aspx). For reference, those sample widgets were not used for any commercial purpose but for module development or research, and they were released to test and develop APIs supported by BONDI APIs 1.1.

4.2 Performance evaluations and results analysis

To minimize the influence from diverse widgets and various modules on the performance test and measurement, five modules and 10 widgets were selected and tested. The performance and execution speed of each module and widget were tested based on two different methods: the existing method that BONDI suggested and the new method proposed in this study.

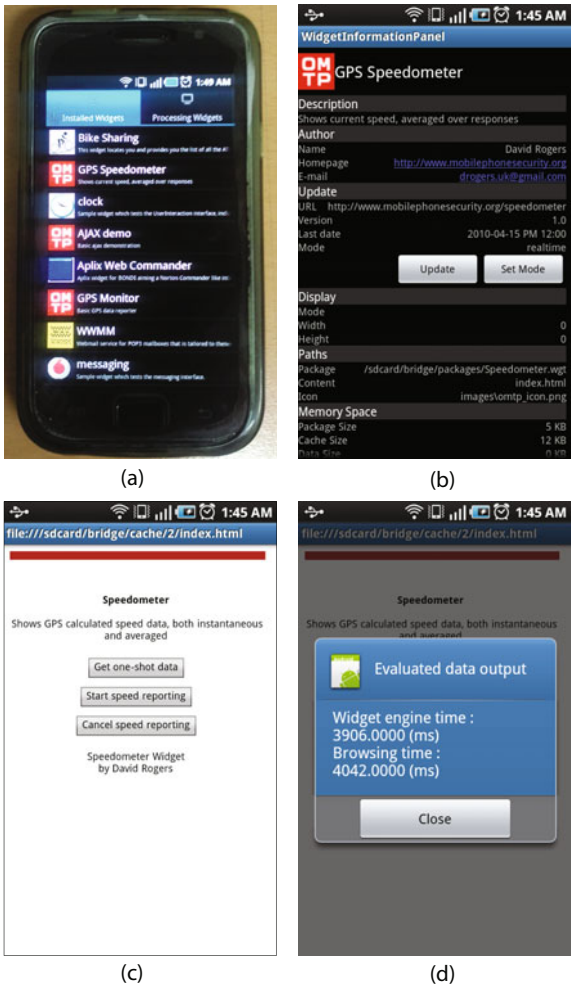


Fig. 8 Android 2.1 target device and running applications. (a) Installed widget list; (b) Widget package information; (c) Widget running screen; (d) Performance evaluation result

4.2.1 Performance benchmark with the added widget configuration document element

Fig. 9 compares 10 BONDI widgets in terms of the result based on BONDI's method and the result based on our proposed method in which the widget configuration document element was added to reduce execution time.

In Fig. 9, although there were differences in loading time because individual widgets require different memory share or modules, it was observed that each widget's module-loading time greatly decreased. Ajaxdemo shows the largest change because it does not use a BONDI module. In contrast, Aplix Web Commander shows little change because it uses a UI module and a file system module that account for large sizes.

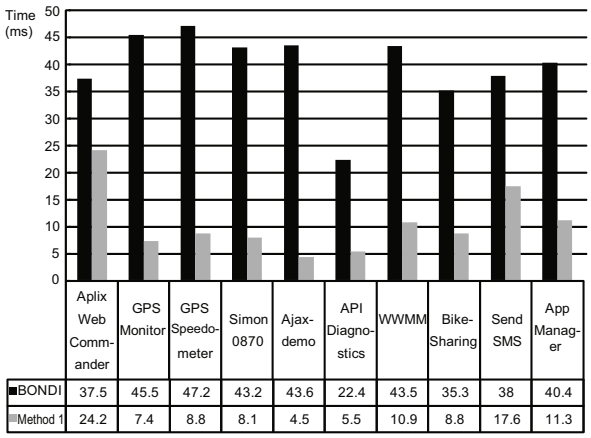


Fig. 9 Performance benchmark on the module-loading (Method 1) speed of BONDI widgets

Fig. 10 presents the results in terms of time by which each widget was fully loaded on the screen through the browser. In this case, there were no significant differences. Sections 4.2.2 and 4.2.3 show the results after applying a method that improves the execution speed of widgets. According to Fig. 10, GPS Monitor, Simon 0870, Ajaxdemo, and Send SMS took longer time in changing into full-screen. The reason is that these widgets had large-sized resource files (*.wgt) and contained a lot of files such as images, CSS, and JavaScript.

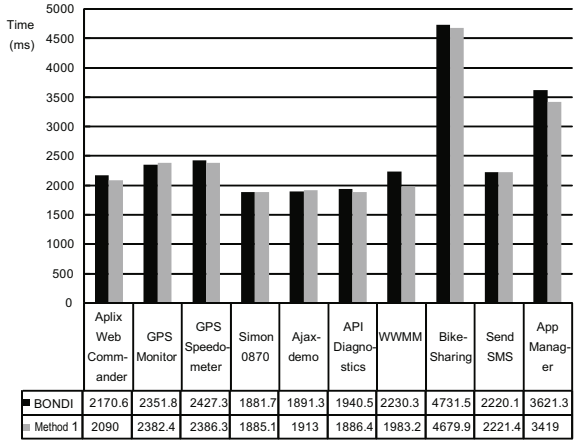
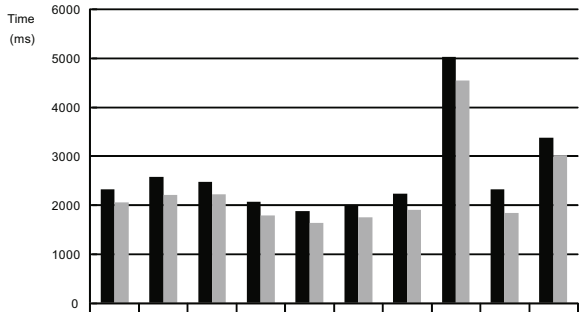


Fig. 10 Performance benchmark of BONDI widgets on loading full-screen (Method 1)

4.2.2 Performance test through redefining the widget resource list

After adding the widget configuration document element to each module, via the test result, improved speed was witnessed according to the number of

modules ruled in BONDI. In addition, a performance test was conducted after a method simplified in terms of installing and loading was applied. The method read and stored the configuration document information in advance when the widgets were installed in the WRL. BONDI's WRL is possible only to obtain URL information of the widget resources. Therefore, when the widget user agent installs a widget, the widget engine has to always refer to 'config.xml' in the widget resources as the agent acquires information such as widget ID, widget version, and widget icon from the engine. Using the method of redefining WRL devised in this study, measured values were analyzed in Fig. 11. All the 10 widgets tested showed improved execution speed (200–500 ms faster). Bike-Sharing and Send SMS recorded changes over 400 ms because they contained a large amount of files including images, CSS, and JavaScript.



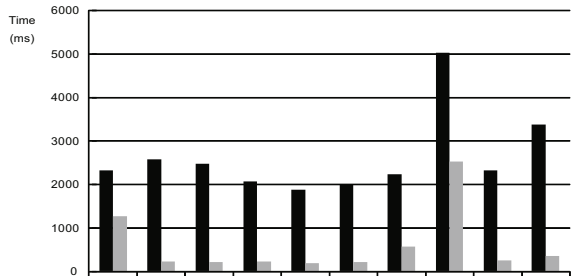
	Aplix Web Commander	GPS Monitor	GPS Speedometer	Simon 0870	Ajax-demo	API Diagnostics	WWMM	Bike-Sharing	Send SMS	App Manager
BONDI	2329.6	2583.2	2473.6	2070	1881.8	2003.5	2232.8	5029.8	2328.5	3380.8
Method 2	2058	2209.3	2225.9	1790.8	1641	1757.4	1910	4557.1	1846.9	3021.2

Fig. 11 Execution performance evaluations of BONDI widgets through redefining the widget resource list (Method 2)

It is possible that there were very small changes in accordance with the device state such as using the network during the test.

4.2.3 Performance evaluations on widget execution time with the virtual widget cache

The existing BONDI reads information in the configuration document, referring to the URL on the WRL when a widget is installed or loaded. In this study we devise a way to more effectively make good use of the WRL by using the virtual widget cache in order to more quickly execute the most-used widgets. The compressed widget resources should be decompressed to read the information of the configuration document. In this regard, BONDI's method is not efficient because it demands that decompressed widget resources should be removed when the widget is closed. Fig. 12 is the performance test result based on a new improved method with the virtual widget cache. Fig. 12 shows, in case of every widget, improved results compared with the previous evaluations based on the method of redefining the WRL and the existing BONDI method. The final result was obtained by comprehensively evaluating and analyzing the configuration document element addition method according to the modules (Method 1), WRL-redefining method (Method 2), and virtual widget cache method (Method 3).

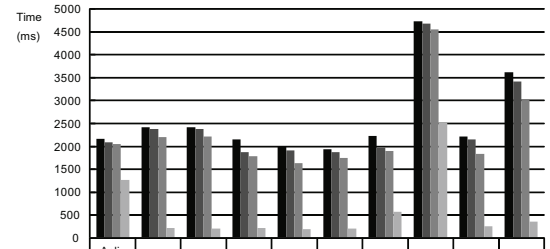


	Aplix Web Commander	GPS Monitor	GPS Speedometer	Simon 0870	Ajax-demo	API Diagnostics	WWMM	Bike-Sharing	Send SMS	App Manager
BONDI	2329.6	2583.2	2473.6	2070	1881.8	2003.5	2232.8	5029.8	2328.5	3380.8
Method 3	1270.5	230	211	230.4	196.8	219.4	576.8	2524.7	259.7	361.1

Fig. 12 Performance evaluations on widget execution with the virtual widget cache (Method 3)

For the control group, BONDI's method of loading all the previous modules was applied. For the experimental group, three methods were conducted: first, the configuration document element addition method loading only necessary modules; second, the WRL-redefining method added to the first method; and third, the virtual widget cache method added to the second method. Fig. 13 shows the final result of these methods.

Fig. 13 shows that new methods suggested in this study greatly improved the execution speed of the widgets. In particular, the virtual widget cache method (Method 3) showed a significant difference. Even though there were little differences according to widget size or the used module, all 10 widgets showed significantly improved execution speed. The same was true for the test based on an actually commercialized device, 'Galaxy S' of Samsung. It is expected there will be additional research on effective cache-



	Aplix Web Comm-ander	GPS Monitor	GPS Speedo-meter	Simon 0870	Ajax-demo	API Diagno-stics	WWMM	Bike-Sharing	Send SMS	App Manager
BONDI	2170.6	2427.6	2427.3	2163.2	2007.7	1940.5	2230.3	4731.5	2220.1	3621.3
Method 1	2090	2382.4	2386.3	1885.1	1913	1886.4	1983.2	4679.9	2154.7	3419
Methods 1+2	2058	2209.3	2225.9	1790.8	1641	1757.4	1910	4557.1	1846.9	3021.2
Methods 1+2+3	1270.5	230	211	230.4	196.8	219.4	576.8	2524.7	259.7	361.1

Fig. 13 Performance evaluations on widget execution speed based on the combination of three methods. Method 1: module-loading; Method 2: widget resource list redefining; Method 3: virtual widget cache

operating algorithms to manage the cache memory in mobile devices. Furthermore, more studies are needed on how to not only increase widget-executing speed and reduce the memory used, but also raise both stability and creditability.

5 Conclusions and further research

This study deals with methods to improve widget performance for achieving effective management and operation of mobile Web widget resources based on OMTP BONDI by suggesting and analyzing the results. OMTP BONDI widgets use designated modules to use the hardware and software of mobile devices. Being able to define the module to be used by means of the <feature> element, the BONDI widgets load unnecessary modules. If a user downloads, installs, and executes a widget programmed by a malicious user, then the user is open to a security risk such as personal information exposure while the execution speed becomes slower. To solve these problems, this paper deals with methods to improve the module-executing speed by loading BONDI modules and ways to effectively operate and manage BONDI widgets under standardization, conducting evaluation tests based on a combination of methods, and analyzing and comparing the results. The evaluation tests were conducted using three methods: the configuration document element addition method according to modules, the widget resource list redefining method, and the virtual widget cache method. As a result, in terms of practice, these suggested methods achieved improved results. In the end, performance evaluation tests were carried out with combinations of these three methods, and the results in terms of performance were positive. Also, the widget box was devised where the widgets being used can be uploaded with important information on a remote server. If the user wants to use one of these widgets again, this is possible anytime. Additional adaptive cache-operating algorithm studies are necessary to manage the memory used in mobile devices. Studies on improving the widget-execution speed and reducing the size of resources used are expected to achieve more security and credibility.

References

Ballon, P., 2009. Control and Value in Mobile Communications: a Political Economy of the Reconfiguration of Business Models in the European Mobile Industry. PhD Thesis, Vrije Universiteit Brussel, Brussels, Belgium.

Chetan, S.C., 2009. The Untapped Mobile Data Opportunity. Technical Report, Chetan Sharma Consulting. Available from http://www.chetansharma.com/The%20Untapped%20Mobile%20Data%20Opportunity%20%20Chetan%20Sharma%20Consulting.pdf

Gartner, R., 2007. Nokia Widgets Will Encourage S60Mobile Services. Technical Report, Gartner Research. Available from http://www.gartner.com/resources/148000/148087/nokia/148087.pdf

Hoegg, R., Martignoni, R., Meckel, M., Stanoevska-Slabeva, K., 2006. Overview of Business Models for Web 2.0 Communities. Technical Report. Available from http://www.alexandria.unisg.ch/export/DL/31412.pdf

JIL (Joint Innovation Lab), 2009. JIL Widget System High Level Technical Specification, Version 1.2.1. Available from http://www.jil.org/c/document_library/get_file?uuid=190b2f2a-061c-45ae-ad1f-a72fefe9d0b3&groupId=10158

OMTP BONDI-1, 2009. BONDI Architecture and Security Application Lifecycle v1.0. Available from http://bondi.omtp.org/1.0/security/BONDI_Architecture_and_Security_v1.0.pdf

OMTP BONDI-2, 2009. BONDI 1.1 Approved Release. Available from http://bondi.omtp.org/1.1/

OMTP BONDI-3, 2009. BONDI 1.5 APIs Public Working Draft v1. Available from http://bondi.omtp.org/1.5/pwd-1/

OMTP BONDI-4, 2009. The BONDI API Design Patterns, Version 1.1. Available from http://bondi.omtp.org/1.1/apis/BONDI_Interface_Patterns_v1.1.html

OMTP BONDI-5, 2010. BONDI API Specification, Version 1.1. Available from http://bondi.omtp.org/1.1/apis/index.html

O′Reilly, T., 2007. What is Web 2.0? Available from http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html

Reynolds, F., 2009. Web 2.0-in your hand. *IEEE Perv. Comput.*, **8**(1):86-88. [doi:10.1109/MPRV.2009.22]

Taivalsaari, A., Mikkonen, T., Ingalls, D., Palacz, K., 2008. Web Browser as an Application Platform. 34th Euromicro Conf. Software Engineering and Advanced Applications, p.293-302. [doi:10.1109/SEAA.2008.17]

Tredinnick, L., 2006. Web 2.0 and business: a pointer to the Intranets of the future. *Business Inf. Rev.*, **23**(4):228-234.

W3C WG-1, 2010. HTML5, a Vocabulary and Associated APIs for HTML and XHTML. Available from http://www.w3.org/TR/html5/

W3C WG-2, 2009. HTML 5 Differences from HTML 4. Available from http://www.w3.org/TR/2009/WD-html5-diff-20090423/

W3C WG-3, 2006. Widgets 1.0. Available from http://www.w3.org/TR/2006/WD-widgets-20061109

JZUS (A/B/C) latest trends and developments

➢ *JZUS-A* wins the "China Government Award for Publishing" for Journals

This prize is the highest award for the publishing industry in China. It has been award to journals for the first time, and only 20 journals in China won the prize, 10 science and technology journals and 10 social science journals.

➢ In 2010 & 2011, we opened a few active columns on the website *http://www.zju.edu.cn/jzus*

- Articles in Press
- Top 10 cited papers in parts A, B, C
- Newest cited papers in parts A, B, C
- Top 10 DOIs monthly
- Newest 10 comments (Open peer review: Debate/Discuss/Question/Opinions)

➢ As mentioned in correspondence published in ***Nature*** *Vol.* 467: p.167; p.789; 2010, respectively:

JZUS (A/B/C) are international journals with a pool of more than 7600 referees from more than 67 countries (http://www.zju.edu.cn/jzus/reviewer.php). On average, 64.4% of their contributions come from outside Zhejiang University (Hangzhou, China), of which 50% are from more than 46 countries and regions.

The publication, designated as a key academic journal by the National Natural Science Foundation of China, was the first in China to sign up for CrossRef's plagiarism screening service CrossCheck.

➢ *JZUS (A/B/C)* have developed rapidly in specialized scientific and technological areas.

- *JZUS-A (Applied Physics & Engineering)* split from *JZUS* and launched in 2005, indexed by SCI-E, Ei, INSPEC, JST, etc. (>20 databases)
- *JZUS-B (Biomedicine & Biotechnology)* split from *JZUS* and launched in 2005, indexed by SCI-E, MEDLINE, PMC, JST, BIOSIS, etc. (>20)
- *JZUS-C (Computers & Electronics)* split from *JZUS-A* and launched in 2010, indexed by SCI-E, Ei, DBLP, Scopus, JST, etc. (>10)

➢ In 2010 JCR of Thomson Reuters, the impact factors:

JZUS-A 0.322; *JZUS-B* 1.027