IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received December 12, 2019, accepted December 22, 2019, date of publication December 26, 2019, date of current version January 6, 2020.

Digital Object Identifier 10.1109/ACCESS.2019.2962393

Enabling Ordinary Users Mobile Development With Web Components

ZHAONING WANG, BO CHENG, AND JUNLIANG CHEN
State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China

Corresponding author: Bo Cheng (chengbo@bupt.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61772479, and in part by the National Key Research and Development Program of China under Grant 2017YFB140.

ABSTRACT The rapid progress of the mobile internet has been promoting the popularity of mobile devices, and mobile application development is getting more pervasive. However, the state of the art development environments has a high learning barrier for users' lack of programming experience. In this paper, instead of traditional programming environments, we take consideration of ordinary users' requirements and propose a WYSIWYG cross-platform web-component-based mobile application creation environment for ordinary users. This environment has a visual editor with a drag-and-drop web component. A web component library model is proposed to standardize customized libraries. A cross-platform application model based on web components is implemented to build applications rapidly. It helps ordinary users generate installing packages within simple operations for multiple platforms. A native plugin model is proposed to assist web components to invoke native functionalities. The experiment result shows that ordinary users could quickly start to create mobile applications in our environment.

INDEX TERMS Mobile service, web components, cross-platform, ordinary users, visual development.

I. INTRODUCTION

Along with the global popularity of smartphones, mobile technologies have been advancing at full speed these years. A large number of mobile devices were designed and produced in a few years. Following the trend of portable, tradition devices, such as laptops and tablets, have become thinner and smaller. Smartphones have become an irreplaceable part of ordinary people's daily lives [6], [7], [11], [15], [17], [20], [28], [31]. Meanwhile, the requirements of mobile applications have been increasing continuously. Nowadays, smart mobile devices provide abundant hardware tools, multiple sensors, Bluetooth, HD cameras, etc. They cooperating, several imaginative mobile applications are generated by developers to change the lives of human beings [21]–[23], [27].

Historically, application development is professional programmers' responsibility, namely rare people have programming expertise or get involved in an application development process. In this paper, we call those people that have no or little programming skills ordinary users. Due to the dominant position which the programmers have, most development tools are designed following their habits and knowledge structure. Based on the assumption that only professional programmers can create applications, traditional development tools designed and optimized surrounding the programming experience. Programming is a complicated and challenging work that demands programming skills, logic capability, professional knowledge, and long-term training. In this traditional perspective, it seems that applications creation equals to programming. However, this point of view is not real in every circumstance. Although software development ability is has existed in limited groups, creating capacity does not exist. On the contrary, a significant number of ordinary people who have experienced mobile applications might have their own opinion creating. However, the difficulties of obtaining programming skills hold them back. Ordinary users' requirements are different from the programmers. First, due to their knowledge, they prefer easy-operating and integrated GUI other than complicated functionalities. Second, they are not sensitive of the applications' performance, since their products are prototypes that are used to test functionalities and primary learning.

Ordinary users requirements could be represented by a typical situation we have considered. A designer is a typical kind of ordinary users and developers. In traditional software engineering, designers could get involved in the process of design prototypes. The method of Designing and implementation

The associate editor coordinating the review of this manuscript and approving it for publication was Honghao Gao.

process is arranged for different staff. Designers or product managers are responsible for designing work, and they usually do not have development skills. They use designing software like Photoshop and deliver prototypes to developers.

As a consequence, these people may design the applications which are impossible to implement, or the final products are worse than expectations. The gap between designing and implementing has been preventing some creative ideas into actual applications. These problems are have not been fresh in recent years [17]. Far earlier, before the mobile internet is popular around the world, tools are generated to solve this kind of problem. FrontPage of Microsoft and Dreamweaver of Macromedia are two standard solutions aiming at users who do not code. They provide convenient and visual environments to help users get rid of complicated codes and create good-looking web pages. As the mobile internet times come, old features of these tools could not satisfy the new requirements due to the continual progress.

One of the most crucial challenges in mobile development is the fragmentation of platforms [25]. In the current situation of the mobile internet, unlike the Windows system, which has a leading role without a doubt in the desktop OS market, there does not exist an absolute winner in the mobile OS market. According to the latest statistics of netmarketshare.com [4], in the report of desktop operating system market share, Android accounts for the most significant proportion (70.24%) in the report of mobile operating system market share pie chart. Next comes iOS, which makes up 28.34%. Windows Phone, Java ME, BlackBerry, Symbian, and others take up less than 2%, respectively. These statistics respect the reality that creating applications one time covering all mobile devices is a complicated project. Different platforms require different development frameworks that are incompatible with each other. Even for professional programmers, it is nearly impossible to handle all the development frameworks at the same time.

Consequently, the development tools field shows the fragmentation situation as mobile platforms do. To address these issues, researchers have proposed several solutions. The most popular one recently is based on modern web technologies aiming at professional programmers and not friendly enough for ordinary users to learn and use. Cross-platform is a non-negligible requirement for an ordinary users' development environment [29].

In this paper, we study the preferences of the ordinary users and propose a WYSIWYG web-component-based cross-platform mobile application development environment. The main contributions of us are:

1) We identify the present problems of mobile development for ordinary users and specify requirements, which are graphical development, automatic tools, and cross-platform.

2) We propose the cross-platform mobile development environment based on web component assemble and relative models to address these issues. The visual editor helps users assemble and configure web components visually. A novel component library model is included to standardize customized libraries. The native plugin model, as a functional extension of web components, supports invoking native functionalities of operating systems. The automatic application creation approach supports users rapidly building applications for multiple platforms.

3)We conduct practical experiments in a group of ordinary users, and the results show the outperforming usability and convenience of our environments.

II. RELATED WORK

A. PROGRAMMING-BASED DEVELOPMENT ENVIRONMENT

Traditional development environments have been historically designing aiming at professional developers based on programming. The programming-based development environment follows the developer-centered designing principles on the base of which a series of measures have been taken to optimize the user experience. Mobile platforms' mainstream development environments are typical developer-centered designed. Android development environments have two optional choices, which are Eclipse and Android Studio, which are both inherited from traditional JAVA development environments and embedded with Android SDK relative tools [1]. Moreover, the iOS development environment directly continues the macOS development environment, XCode, with relative tools and compilers integrated. These continuous design above ensures that developers with relative languages' coding experiences could rapidly handle the mobile applications development, at the same time, keeps the limitations of desktop development environments [36].

The limitation is that programming skill and development operations are too complex to cultivate for ordinary users. Programming is a systematic ability that requires not only expertly handling corresponding programming language but abundant knowledge of data structure and operating system, while ordinary users do not have enough relevant experiences to understand the concepts. Traditional development Environments are not appropriate for ordinary users to use since their operation processes do not conform to ordinary users' operating habits and knowledge structure. Code template, project generators, configuration forms, and other automatic code generation have been proposed to facilitate and speed up the application development process. These approaches lower the level of development difficulty to some degree by helping developers reduce repetitive work and automatically generating code blocks; however, they can not solve the fundamental problems.

B. VISUAL DEVELOPMENT ENVIRONMENT

To address these issues, several solutions are proposed to improve traditional development environments [32]. Visual designer, a graphical drag-and-drop tool or plugin, has been widely used in multiple mobile development environments to facilitate designing the GUI process. Based on the prototype of the visual designer, some products have gone further in the graphical programming field. Current graphical programming tools use the visual component as a basic designing unit in typical [16]. Current graphical development products use a visual component as a basic designing unit to make up an application. A visual component is an ordinary users' conceptualization of an interactive single purpose application with one or more functionalities for displaying and updating local or remote data, packaged in a way to allow discovery, instantiation, and combination in a unified runtime environment. In order to support the application development via components, a component could not only run as a stand-alone application but has a communication interface to compose together and operate with each other.

In practical research, visual components are widely applied to composite services and applications [18]. Reference [8] implements MashMaker, which is a visual service orchestration tool. Reference [13] proposed a semantic-based composition platform for various services and applications. Reference [12] proposed a visual components composition environment to development composite service. Reference [10] implements a service creation environment on mobile devices named MicroApp. MIT App Inventor [14] is an intuitive, visual programming environment that allows everyone, even children, to build fully functional applications for Android smartphones and tablets. These products have common problems unsolved. Firstly, only single specific platforms are supported. Secondly, functionalities of visual components are limited in GUI designing.

C. WEB COMPONENTS

As mentioned below, development environments only support a single specified operating system. This situation leads to a result of adapting to all the platforms; repeated works are required even for applications with the same appearances and functionalities. Different mobile platforms demand an utterly different SDK and developing framework. This work is far beyond ordinary users' abilities. Cross-platform is proposed to address the fragmentation issue of mobile platforms. It is a development concept indicating a rapidly developing process without considering platform differences [30]. Web components are custom, reusable, encapsulated as HTML tags to use in web pages and web applications. Custom components will work across modern browsers and can be used with any JavaScript library or framework that works with HTML. A web application is a kind of mobile application based on pure web technologies running in a browser to simulate native application known as native development following the traditional framework. The web application is a popular solution to cross-platform [26].

AppGyver Composer [2] is an online cross-platform mobile development tool that combines graphical programming and with the web-based cross-platform solution. It provides users visual development editor for GUI and its logic. Draggable components are supported, and users could custom their HTML tags and JavaScript. While it has solved the available issue, it exposes a critical negative point of Web App, which is, lacks support for native APIs invocation. Native APIs are a series of service interfaces providing hardware functionalities of the mobile platforms, such as camera, sensors, Wi-fi, Bluetooth, and so on, with-out which would severely limit the development of diverse mobile applications. Since the limitation of the browser interface, Web Apps could not invoke all the functionalities of native APIs.

PhoneGap is a solution to bridge the gap between JavaScript interface and native functionalities of the mobile application platforms by proposing a hybrid application framework to enable web technologies supporting enable we techniques developing cross-platform mobile applications. By using PhoneGap, developers can create an application for each of the platforms using a single code base. It allows developers to focus on creating a great experience instead of authoring complex platform compatibility layers. Based on this framework, several integrated development environments are proposed to support web-based cross-platform mobile application development. DCloud HBuilder [3] is a mobile hybrid application development tool aiming at programmers. It integrated various functionalities to accelerate the development process and simplify the programming work with cross-platform application creation supported. Same as other development environments, it does not go further of supporting ordinary users' development.

III. PROPOSED ENVIRONMENT AND MODELS

In this section, we introduce the overall architecture and the detailed principle of our proposed models including the web component library model, native plugin model, cross-platform application model.

A. ARCHITECTURE OVERVIEW

The proposed environment is developed as a web application that could be accessed on any device with browsers. Figure 1 shows the overview architecture, which is divided into three functional components: library repository, visual editor, and server module. These components are based on the OSGi framework, a dynamic module system for Java, enabling a model where an application is composed of several components packaged in bundles, communicating with each other through service interfaces of the framework, reducing system complexity and achieving loose coupling.

1) LIBRARY REPOSITORY

The library repository is characterized as a series of web component libraries. A library is a web component container encapsulated in an independent bundle of the OSGi framework maintaining the essential data that defines a web component collection. As shown in figure 2, all available libraries are dynamically loaded and registered to the service registry by the OSGi framework. The library manager in the server module discovers libraries from the service registry during the execution time. The component palette in the visual editor determines web components' descriptions and renders them on GUI. An API library provides a service interface for other

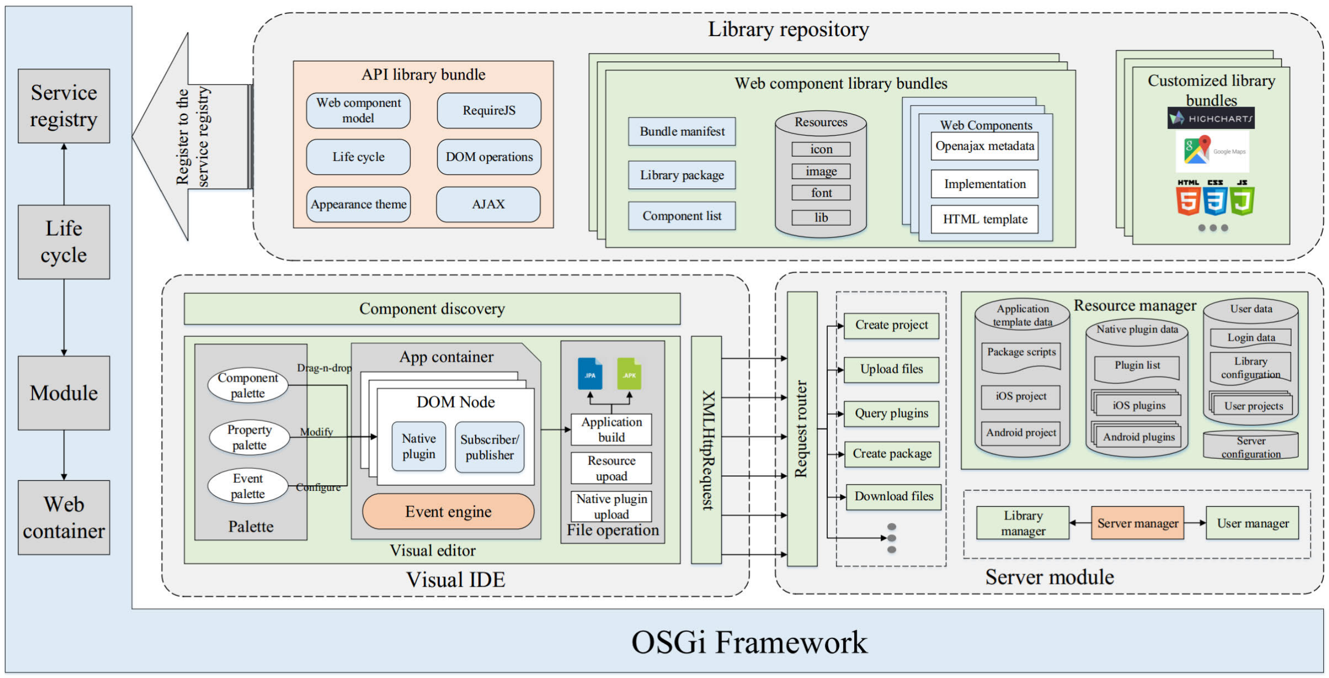


FIGURE 1. The architecture overview of the environment.

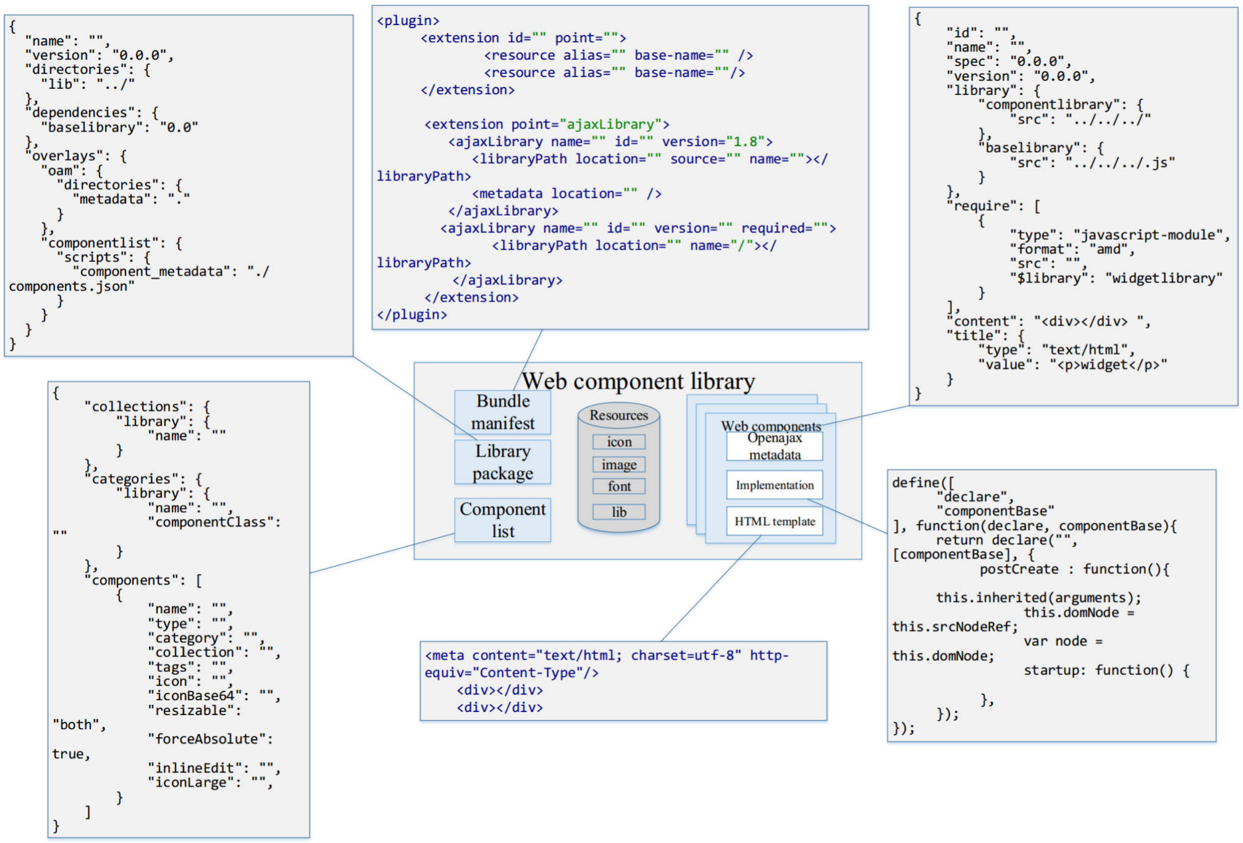
libraries, such as web component model generating new web components, life cycle management, RequireJS importing extended modules, DOM operation interface, AJAX interface, appearance theme, etc.

2) VISUAL IDE

The visual IDE component provides a drag-and-drop development environment where web components assemble. The significant component is visual editor, which includes three sub-components, app container, palette, and le operation. The component discovery module collects web component information. XMLHttpRequest provides interface communicating with the server module. The visual editor provides a WYSIWYG working area. Each time users create a new project, the visual editor generates an app container to establish an execution environment for a user's application. Users drag web components to the app container and visually modify their position and size to design their applications' GUI. The app container real-time resolves them in real-time and shows users the previews. Web components communicate through an event-driven model. The app container has an event engine provide a set of pub/sub interface to send or monitor events. The palette offers a toolkit that helps assemble and configure web components. The palette is composed of three sub-components, component palette, property palette, and event palette. The component palette shows users' available web components' icons and brief descriptions. When users drag an icon to the visual editor, it accesses the implementation's location from metadata and generates an instance. The property palette provides a visual configuration for appearance properties. The event palette allows users to configure node event functions, including native plugins, which provide interfaces to invoke native functionalities, such as Bluetooth, camera, gyroscope, and so on. The file operation module offers a series of visual tools to operate users' projects, including le uploading, native plugin upload, and application build. File up-load provides a simple approach to add resources like image and multi-media to applications. Native plugin upload allows users to customize functionalities with native interface following the native plugin model, which will be described in detail below. Application build is the core component to realize the last step of the fast development process, creating installing package, and cross-platform compatibility. It allows end-users to generate installing packages of multiple mobile platforms for one user project via simple visual operations.

3) SERVER MODULE

The server module includes a series of bundles that implement the back-end business logic including server manager, user manager, library manager, request router and resource manager. The server manager reads server configurations in order to start a web container and instantiates other managers. The user manager is responsible for user authentication and modification. The library manager discovers registered web component libraries from the OSGi service registry and loads them into the web container. The request router accepts the requests from the visual IDE and distributes them to corresponding handlers. The resource manager stores user data, native plugin data, application template data, and server configuration. User data records the metadata and project les of every user. Plugin les are resource les and basic information,



```
{
  "name": "",
  "version": "0.0.0",
  "directories": {
    "lib": "../"
  },
  "dependencies": {
    "baselibrary": "0.0"
  },
  "overlays": {
    "oam": {
      "directories": {
        "metadata": "."
      }
    },
    "componentlist": {
      "scripts": {
        "component_metadata": "./
components.json"
      }
    }
  }
}
```

```
<plugin>
     <extension id="" point="">
              <resource alias="" base-name="" />
              <resource alias="" base-name=""/>
     </extension>

      <extension point="ajaxLibrary">
         <ajaxLibrary name="" id="" version="1.8">
            <libraryPath location="" source="" name=""></
libraryPath>
            <metadata location="" />
         </ajaxLibrary>
        <ajaxLibrary name="" id="" version="" required="">
              <libraryPath location="" name="/"></
libraryPath>
         </ajaxLibrary>
      </extension>
</plugin>
```

```
{
     "id": "",
     "name": "",
     "spec": "0.0.0",
     "version": "0.0.0",
     "library": {
         "componentlibrary": {
             "src": "../../../"
         },
         "baselibrary": {
             "src": "../../../.js"
         }
     },
     "require": [
         {
             "type": "javascript-module",
             "format": "amd",
             "src": "",
             "$library": "widgetlibrary"
         }
     ],
     "content": "<div></div> ",
     "title": {
         "type": "text/html",
         "value": "<p>widget</p>"
     }
}
```

```
{
     "collections": {
         "library": {
             "name": ""
         }
     },
     "categories": {
         "library": {
             "name": "",
             "componentClass":
""
         }
     },
     "components": [
         {
             "name": "",
             "type": "",
             "category": "",
             "collection": "",
             "tags": "",
             "icon": "",
             "iconBase64": "",
             "resizable":
"both",
             "forceAbsolute":
true,
             "inlineEdit": "",
             "iconLarge": "",
         }
     ]
}
```

```
<meta content="text/html; charset=utf-8" http-
equiv="Content-Type"/>
    <div></div>
    <div></div>
```

```
define([
      "declare",
      "componentBase"
], function(declare, componentBase){
      return declare("",
[componentBase], {
            postCreate : function(){

      this.inherited(arguments);
                  this.domNode =
this.srcNodeRef;
                  var node =
this.domNode;
                  startup: function() {

            },
      });
});
```

FIGURE 2. The web component library model.

including plugin name and function list of users' plugins. Particularly, for native plugins, some other resources are required. Application template data is the essential resources of building installing packages. The major resources of building resource include execution script, template project, and log les. For native plugins data and template project data, Android version and iOS version are separately stored, which is available adapting to users' different choices.

B. WEB COMPONENT LIBRARY MODEL

A web component is implemented via web technologies instantiated as a customized reusable DOM node, namely an HTML tag, running in the web browser environment standardized by W3C through all platforms, from desktops to mobile devices, so it is the foundation of cross-platform development. A web component could be a functional control or a page template that could make up a mobile application. Figure 2 shows a standard web component library model.

A typical web component consists of an OpenAjax metadata (OAM) file [5], and implementation le, and HTML templates. The OAM represents a set of industry-standard metadata defined by the OpenAjax Alliance that enhances the interoperability across AJAX toolkits and AJAX products [19]. Each web component corresponds to an individual OAM file that describes the metadata, including a unique ID, library name, requiring libraries, context, and other optional properties, to instantiate a web component on the visual editor. The implementation le is associated with its metadata le that defines AMD(Asynchronous Module Definition) module inherited the web component model in the API library, providing the base interface. It asynchronously imports remote module via RequireJS interface and accesses web resources through XMLHttpRequest. The HTML template offers a frame in the rendering phase.

A standard library is implemented as an OSGi bundle consisting of metadata, resources, and owned web components. The metadata part maintains three primary components, bundle manifest, library package, and component list. The bundle manifest file records the configuration data, including library location, identifier, etc., to define an OSGi bundle that is registered to the service registry. The OSGi service registry loads libraries to the server according to this le. The component lists le records all available web components in this library, declaring the essential information to identify and locate them, including corresponding name, type, icon, properties, and a brief description that would be rendered in the component palette. The library package le describes the essential information of the library, including library name, version, dependencies, overlays, and relative

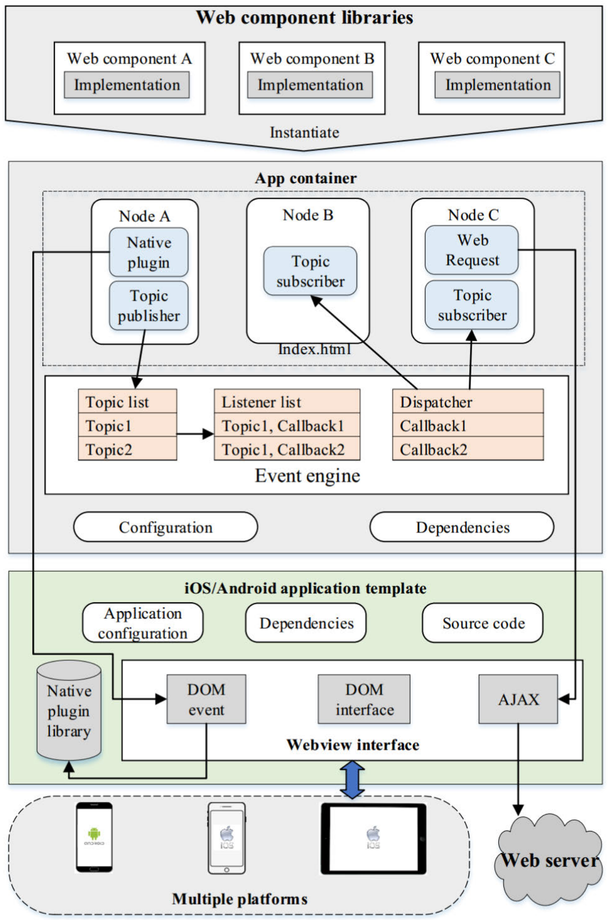


FIGURE 3. The cross-platform application model.

directories. Resources of the library offer all the necessary dependencies, images, media files, and font libraries used for instantiating web components.

C. NATIVE PLUGIN MODEL

Native plugins provide interfaces to web components while having a channel accessing OS service interfaces, which set up a bridge between web components and native functionalities. A typical native plugin consists of the following five essential components:

- An identifier represents a platform this plugin supports, such as iOS or Android;

- A function list indicating the available interfaces for web components to invoke;

- A configuration le for the package template project: It varies according to different platform standards. For the Android platform, it represents the properties of the permission and project plugin. For iOS, it represents the configuration of the source class.

- Resource files including essential JavaScript libraries, images, CSS files and other necessary JavaScript files executing in the Webview end;

- Prime class source files running in the OS end. JavaScript libraries in the resource files communicate with class and provide interfaces via JSBridge protocol [9] for web components that implement the data exchange between web components and OS.

Native plugins could be configured via binding corresponding JavaScirpt inter-face to specified web components and triggered by the event engine. The native plugin data is stored in the server layer managed by the resource module. When rendering environment GUI, the view layer sends requests to get the plugin list and show it in the visual editor. When building a mobile application, the corresponding native plugins are copied to the application package template according to users' configurations.

D. CROSS-PLATFORM APPLICATION MODEL

In this section, we introduce our proposed cross-platform application model (CAM) and its internal principle.

1) CAM OVERVIEW

As figure 2 shows, a CAM consists of an app container and an application template corresponding to free platforms. The app container provides an instantiation and execution environment for web components. An app container has an HTML file where web components' instantiations reside, an event engine and resource files including a configuration le, recording a native plugin list and application properties and runtime dependencies.

The application template is a native application wrapper for app containers. According to different platforms, templates are divided into multiple versions, but there are common parts for each template. It provides a WebView environment with page parser and the web interface for executing app containers. It maintains a configuration le ruling application properties and permissions. A native plugin library is included to provide web components with interfaces invoking native functionalities.

2) WEB COMPONENT COMMUNICATION MECHANISM

In this section, we describe the web component communication mechanism in the cross-platform application framework in detail. The communication mechanism defines data transmission approaches to web components' instantiations. Figure 2 shows the primary communication mechanism during runtime. Web components' implementations generate instantiations via creating HTML nodes in the app containers, and multi-data channels are available to obtain data from other components, web service, or operating systems. Communication mainly includes the following three categories:

Component-to-Component: Web components could communicate with each other through a pub/sub interface embedded in the corresponding app container and provides a pub/sub interface to exchange data.

Component-to-OS: Web components could access OS interfaces via native plugins, as mentioned above.

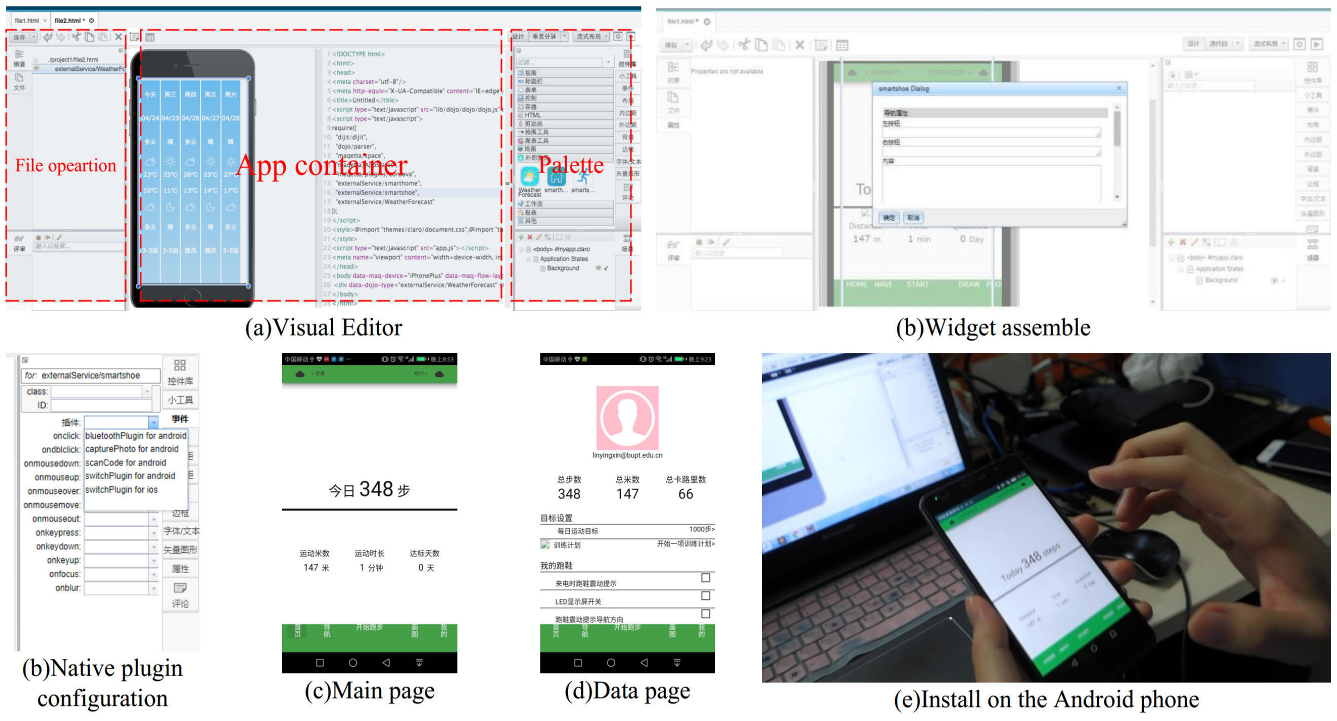


FIGURE 4. **Implementation and demonstration.**

Through this channel providing by native plugins, the OS end could receive, resolve and respond to the data from the WebVeiw end.

Component-to-Web: Web components could send RESTful requests to remote servers for data via AJAX interface providing by the dependent libraries and the WebView interface.

Event-Driven Composition Web component composition is a process of assembling instantiations and transforming them into mobile applications [24]. The composition process has two steps.: The First first one is the GUI assemble, namely to place the individual web components together via the drag-and-drop operations according to the conventional logic and users' requirements; The second one is the connection establishing, building connections among web components to make mobile applications working properly using web component communication mechanisms [33]–[35].

Connection establishing is according to an event-driven model based on the event engine. It provides a pub/sub communication channel. An event is a message with a topic name produced by a web component's instances and other instances could monitor this topic to handle it. A standard event could de ne as a tuple:

$$< Target; Topic; Object >$$

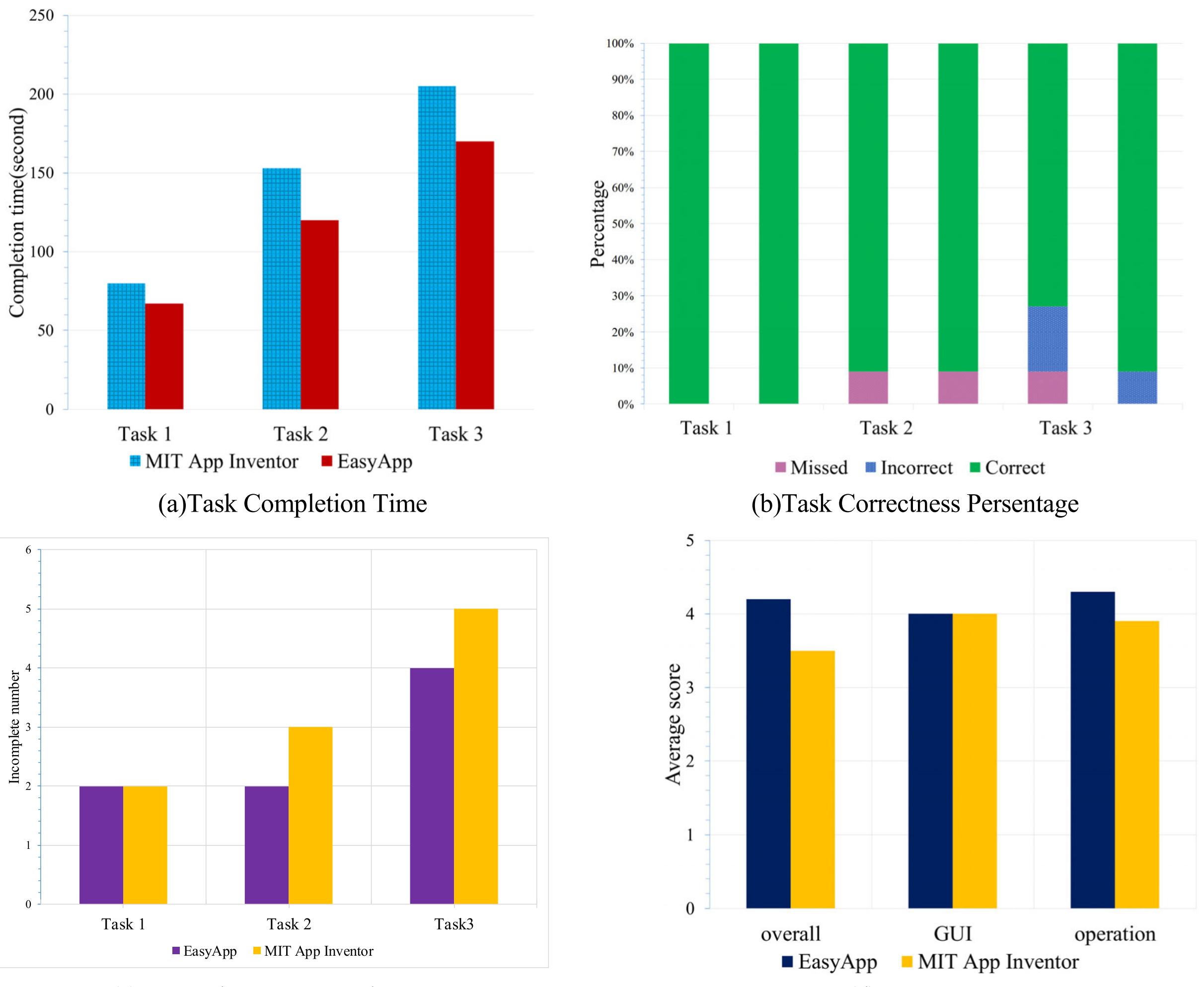
Target is a web component instance, the publisher, which generates an event. The topic is a string to describe the event type according to which subscriber's monitor the event and the event engine dispatch message. The object is a JavaScript object carrying data that Target supposes to transmit. The event engine is a JavaScript object with the pub/sub interface in charge of message transmission and management. The interface allows web components to generate topics and dispatch to subscribers or establish listeners to subscribe to specific topics and create corresponding handler functions. These two interfaces could describe as two JavaScript functions:

$$Publish(topic; object); \; subscribe(topic; callback)$$

The event engine maintains a topic list and a listener list. When the specific event comes up, that is, the target web component invokes publish interface to trigger a topic, the event engine checks the topic list, then traverses the listener list and invokes the callback functions successively, so the web components which subscribe it could receive the message and trigger the corresponding handlers.

3) APPLICATION BUILD APPROACH

The application build service is encapsulated in the server module transparent for users. We define an application template as a collection of essential files to assist application creation. The resource manager on the server side maintains application templates and corresponding package scripts for different platforms. Users send requests via GUI controls in the le operation area. When a package request is submitted, the create package handler in the server-side receives parameters of the platform option, and the app container is copied from the user's directory to the corresponding application template. Meanwhile, a native plugin list maintained by the



(a)Task Completion Time
(b)Task Correctness Persentage
(c)Incomplete User Number
(d)Users' Score

FIGURE 5. Evaluation results.

app container recording applied native plugins in the project is read, and the corresponding resources of native plugins are copied to the native plugin library maintained by the template. Configuration le and dependencies of the app container is imported into the application template. The corresponding package script is executed to generate the apk or ipa that responds to the client.

IV. IMPLEMENTATION

We implement our proposed environment based on an implementation of the OSGi framework, Eclipse Equinox. Web components are implemented based on DOJO mobile toolkit. Native plugins and application creation approaches are implemented based on Apache Cordova. The environment supports running on different desktop OS, such as Windows, Mac OS, and Linux. Besides, it could be deployed on remote servers and accessed by local browsers. In this paper, we demonstrate the functionalities on a Lenovo laptop. Figure 2 shows the visual editor page. In the central position of the page is the app container. This part is the area where users assemble applications' page elements and layout. It supports two display mode, graphics mode, and source code mode. On the right of the page is the palette tab bar. On the left of the page is the le operation area.

V. DEMONSTRATION

In this section, we demonstrate the process of customizing web components and native plugins and create an application using them. We create Bluetooth relative web components and create a Bluetooth application through the following procedures. The main requirements of it are controlling Bluetooth switch, counting step numbers on smartphones, and discovering nearby smart devices such as smart shoes.

Customize Web Components and Native Plugins: This step is transparent for ordinary users in normal circumstance. We demonstrate this process brie y to show how to import the extended web component libraries and the native plugins. First, we create a new web component library, namely an OSGi bundle with a service activator and essential configuration les, in the root directory of EasyApp. Second, we add a web component to the library, including its meta-data, resources and implementation le. At last, we complete the

GUI in the visual editor. The component palette information is recorded in a configuration le in JSON. In this case, we add a library tab with a component item to the palette definition le. We create a new JSON object to de ne a new tab and item to show in the palette.

To implement the native function, we need a Bluetooth native plugin. We create a new directory to save native plugin's resource les. We implement Bluetooth functionalities in Java and encapsulate JavaScript interface for each of them. We place JavaScript les and Java les in separated folders and three configure les in the directory. Finally, we compress the directory as a zip le and upload it to the environment.

Component Assemble and Application Build As is shown in figure 4(a) and 4(b), create a new project in the visual editor and drag the new web component to the app container modifying size and position, then configure its native plugin in the palette. Finally, click android package button to get the installer and send it to an Android phone to test as is shown in figure 4(c) to 4(e).

VI. EVALUATION

The goal of this evaluation is to demonstrate the effectivity for ordinary users of our environment compared to other proposals. Since the special requirements of ordinary users, we did not add performance the objective of the cross-platform applications. Therefore, we use the development speed and users' experience score as the major objectives.

To evaluate the usability of EasyApp, we have an experiment compared with two tools, MIT App Inventor and EasyApp. We organized 22 users without programming expertise from Beijing University of Posts and Telecommunications to watch a video tutorial and complete three tasks about developing similar applications in two environments, then fill a questionnaire to score their experience for each tool. The result of accuracy and efficiency is shown in figure 5(a) and 5(b). For four tasks, participants all complete faster in EasyApp than in MIT App Inventor. In the end, participants completed 64 tasks correctly in EasyApp while 60 tasks correctly in MIT App Inventor. We computed the average score and the result is shown in figure 5(c). After the development process, every participant filled a questionnaire with three categories questions about the overall, GUI and operation of the development experience which include five possible answers: 1 (strongly disagree), 2 (disagree), 3 (neutral), 4 (agree), and 5 (strongly agree). Figure 5(d) shows the average scores of each category. The overall and the operation have an obvious superiority over MIT AppInventor and the GUI score obtains the same score.

From the result, we can see that EasyApp is a friendly enough for end-users and perform better in some fields compared to other tools.

VII. CONCLUSION

This paper proposes a WYSIWYG cross-platform mobile applications development environment for ordinary users. We describe the architecture and implementation in detail. It provides users an easy-operating visual editor, abundant web component libraries and native plugins, and an automatic application creation approach. Summarizing the above, our proposed environment has learned from traditional development environments and improved the functionalities to be friendlier to ordinary users.

Future work mainly includes three aspects. First is aiming at the functionalities of the environment, which is expanding the web component libraries to meet various requirements. JavaScript has been growing mature with the progress of web techniques, and a significant number of excellent frameworks have been released. We are expecting to import more frameworks and libraries to enrich the diversity of the web component libraries to give end-user more options. The second is to increase the cross-platform applications' execution efficiency and support more mobile platforms. We are expecting to adapt to mobile cloud application development. The third is to make the creation process more automatic via planning and learning technologies.

REFERENCES

[1] (2019). *Android Developers*. Accessed: Oct. 24, 2019. [Online]. Available: https://developer.android.com

[2] (2019). *AppArchitect*. Accessed: Oct. 24, 2019. [Online]. Available: http://www.apparchitect.com/

[3] (2019). *DCloud*. Accessed: Oct. 24, 2019. [Online]. Available: http://www.dcloud.io/

[4] (2019). *Market Share for Mobile, Browsers, Operating Systems and Search Engines | NetMarketShare*. Accessed: Oct. 24, 2019. [Online]. Available: http://www.netmarketshare.com/

[5] Openajax.org. (2019). *OpenAjax Alliance*. Accessed: Oct. 24, 2019. [Online]. Available: http://www.openajax.org/

[6] H. Artail, K. Fawaz, and A. Ghandour, "A proxy-based architecture for dynamic discovery and invocation of Web services from mobile devices," *IEEE Trans. Serv. Comput.*, vol. 5, no. 1, pp. 99–115, Jan. 2012.

[7] G. Cabri, L. Leonardi, M. Mamei, and F. Zambonelli, "Location-dependent services for mobile users," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 33, no. 6, pp. 667–681, Nov. 2003.

[8] R. J. Ennals and M. N. Garofalakis, "MashMaker: Mashups for the masses," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2007, pp. 1116–1118.

[9] R. Francese, M. Risi, G. Tortora, and G. Scanniello, "Supporting the development of multi-platform mobile applications," in *Proc. 15th IEEE Int. Symp. Web Syst. Evol. (WSE)*, Sep. 2013, pp. 87–90.

[10] R. Francese, M. Risi, G. Tortora, and M. Tucci, "Visual mobile computing for mobile end–users," *IEEE Trans. Mobile Comput.*, vol. 15, no. 4, pp. 1033–1046, Apr. 2016.

[11] L. He, G. Meng, Y. Gu, C. Liu, J. Sun, T. Zhu, Y. Liu, and K. G. Shin, "Battery–aware mobile data service," *IEEE Trans. Mobile Comput.*, vol. 16, no. 6, pp. 1544–1558, Jun. 2017.

[12] N. Laga, E. Bertin, R. Glitho, and N. Crespi, "Widgets and composition mechanism for service creation by ordinary users," *IEEE Commun. Mag.*, vol. 50, no. 3, pp. 52–60, Mar. 2012.

[13] A. H. Ngu, M. P. Carlson, Q. Z. Sheng, and H.-Y. Paik, "Semantic–based mashup of composite applications," *IEEE Trans. Serv. Comput.*, vol. 3, no. 1, pp. 2–15, Jan. 2010.

[14] S. C. Pokress and J. J. D. Veiga, "MIT app inventor: Enabling personal mobile computing," 2013, *arXiv:1310.2830*. [Online]. Available: https://arxiv.org/abs/1310.2830

[15] Z. Tang, S. Guo, P. Li, T. Miyazhaki, H. Jin, and X. Liao, "Energy-efficient transmission scheduling in mobile phones using machine learning and participatory sensing," *IEEE Trans. Veh. Technol.*, vol. 64, no. 7, pp. 3167–3176, Jul. 2015.

[16] R. N. Taylor, N. Medvidovic, K. M. Anderson, E. J. Whitehead, J. E. Robbins, K. A. Nies, P. Oreizy, and D. L. Dubrow, "A component-and message-based architectural style for GUI software," *IEEE Trans. Softw. Eng.*, vol. 22, no. 6, pp. 390–406, Jun. 1996.

[17] P. Wang, F. Ye, and X. Chen, "A smart home gateway platform for data collection and awareness," *IEEE Commun. Mag.*, vol. 56, no. 9, pp. 87–93, Sep. 2018.

[18] Z. Zhai, B. Cheng, Z. Wang, X. Liu, M. Liu, and J. Chen, "Design and implementation: The end user development ecosystem for cross-platform mobile applications," in *Proc. 25th Int. Conf. Companion World Wide Web (WWW)*, 2016, pp. 143–144.

[19] Y. Maezawa, K. Nishiura, H. Washizaki, and S. Honiden, "Validating ajax applications using a delay-based mutation technique," in *Proc. 29th ACM/IEEE Int. Conf. Automated Softw. Eng. (ASE)*, 2014.

[20] D. Liu, L. Khoukhi, and A. Hafid, "Prediction–based mobile data offloading in mobile cloud computing," *IEEE Trans. Wireless Commun.*, vol. 17, no. 7, pp. 4660–4673, Jul. 2018.

[21] B. Zhou, A. V. Dastjerdi, R. N. Calheiros, S. N. Srirama, and R. Buyya, "MCloud: A context–aware offloading framework for heterogeneous mobile cloud," *IEEE Trans. Serv. Comput.*, vol. 10, no. 5, pp. 797–810, Sep. 2017.

[22] R. T. Tiburski, C. R. Moratelli, S. F. Johann, M. V. Neves, E. D. Matos, L. A. Amaral, and F. Hessel, "Lightweight security architecture based on embedded virtualization and trust mechanisms for IoT edge devices," *IEEE Commun. Mag.*, vol. 57, no. 2, pp. 67–73, Feb. 2019.

[23] B. Cao, L. Zhang, Y. Li, D. Feng, and W. Cao, "Intelligent offloading in multi–access edge computing: A state-of-the-art review and framework," *IEEE Commun. Mag.*, vol. 57, no. 3, pp. 56–62, Mar. 2019.

[24] W. Gaaloul, S. Bhiri, and M. Rouached, "Event–based design and runtime verification of composite service transactional behavior," *IEEE Trans. Serv. Comput.*, vol. 3, no. 1, pp. 32–45, Jan. 2010.

[25] M. Fiedler, K. Wac, R. Bults, and P. Arlos, "Estimating performance of mobile services from comparative output–input analysis of end-to-end throughput," *IEEE Trans. Mobile Comput.*, vol. 12, no. 9, pp. 1761–1773, Sep. 2013.

[26] W. S. El-Kassas, B. A. Abdullah, A. H. Yousef, and A. M. Wahba, "Enhanced code conversion approach for the integrated cross–platform mobile development (ICPMD)," *IEEE Trans. Softw. Eng.*, vol. 42, no. 11, pp. 1036–1053, Nov. 2016.

[27] S. Guo, J. Liu, Y. Yang, B. Xiao, and Z. Li, "Energy–efficient dynamic computation offloading and cooperative task scheduling in mobile cloud computing," *IEEE Trans. Mobile Comput.*, vol. 18, no. 2, pp. 319–333, Feb. 2019.

[28] A. Mehrabi, M. Siekkinen, and A. Yla-Jaaski, "Edge computing assisted adaptive mobile video streaming," *IEEE Trans. Mobile Comput.*, vol. 18, no. 4, pp. 787–800, Apr. 2019.

[29] K. W. Choi, D. T. Wiriaatmadja, and E. Hossain, "Discovering mobile applications in cellular device-to-device communications: Hash function and Bloom filter–based approach," *IEEE Trans. Mobile Comput.*, vol. 15, no. 2, pp. 336–349, Feb. 2016.

[30] C.-Y. Shen, D.-N. Yang, and M.-S. Chen, "Collaborative and distributed search system with mobile devices," *IEEE Trans. Mobile Comput.*, vol. 11, no. 10, pp. 1478–1493, Oct. 2012.

[31] J. Ruan, Y. Wang, F. T. S. Chan, X. Hu, M. Zhao, F. Zhu, B. Shi, Y. Shi, and F. Lin, "A life cycle framework of green IoT–based agriculture and its finance, operation, and management issues," *IEEE Commun. Mag.*, vol. 57, no. 3, pp. 90–96, Mar. 2019.

[32] B. Cheng, S. Zhao, J. Qian, Z. Zhai, and J. Chen, "Lightweight service mashup middleware with REST style architecture for IoT applications," *IEEE Trans. Netw. Serv. Manage.*, vol. 15, no. 3, pp. 1063–1075, Sep. 2018.

[33] Y. Zhang, J.-L. Chen, and B. Cheng, "Integrating events into SOA for IoT services," *IEEE Commun. Mag.*, vol. 55, no. 9, pp. 180–186, Sep. 2017.

[34] Y. Cheng, S. Zhao, B. Cheng, S. Hou, X. Zhang, and J. Chen, "A distributed event–centric collaborative workflows development system for IoT application," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2017.

[35] S. Zhao, L. Yu, and B. Cheng, "An event–driven service provisioning mechanism for IoT (Internet of Things) system interaction," *IEEE Access*, vol. 4, pp. 5038–5051, 2016.

[36] B. Cheng, D. Zhu, S. Zhao, and J. Chen, "Situation–aware IoT service coordination using the event–driven SOA paradigm," *IEEE Trans. Netw. Serv. Manage.*, vol. 13, no. 2, pp. 349–361, Jun. 2016.

ZHAONING WANG is currently pursuing the Ph.D. degree in computer science and technology with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. His current research interests include service composition, end-user programming, and mobile service.

BO CHENG received the Ph.D. degree in computer science from the University of Electronics Science and Technology of China, in 2006. He is currently a Professor with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. His research interests include the Internet of Things, the mobile Internet, and services computing.

JUNLIANG CHEN is currently a Professor with the Beijing University of Posts and Telecommunications. His research interest includes service creation technology. He was elected as a member of the Chinese Academy of Science, in 1991, and the Chinese Academy of Engineering, in 1994.