

IMPLEMENTATION AND EVALUATION OF AN ANDROID
ACCESSOR-BASED IOT MIDDLEWARE

by

Vimal Moorthy Krishnamoorthy, B.E

A thesis submitted to the Graduate Council of
Texas State University in partial fulfillment
of the requirements for the degree of
Master of Science
with a Major in Computer Science
August 2017

Committee Members:

Anne Ngu, Chair

Vangelis Metsis

Mina Guirguis

COPYRIGHT

by

Vimal Moorthy Krishnamoorthy

2017

FAIR USE AND AUTHOR'S PERMISSION STATEMENT

Fair Use

This work is protected by the Copyright Laws of the United States (Public Law 94-553, section 107). Consistent with fair use as defined in the Copyright Laws, brief quotations from this material are allowed with proper acknowledgment. Use of this material for financial gain without the author's express written permission is not allowed.

Duplication Permission

As the copyright holder of this work I, Vimal Moorthy Krishnamoorthy, authorize duplication of this work, in whole or in part, for educational or scholarly purposes only.

DEDICATION

...this research is dedicated to Mother (Uma Moorthy) and my Thesis Advisor (Dr. Anne Ngu), my resource of encouragement

ACKNOWLEDGEMENTS

I am thankful to Dr. Anne Ngu for her practical advice and criticism. She sets high standard for students in classrooms as well as in research practice. I am extremely grateful to her for holding such a research standard, from proposing a project to enforcing strict validation for the research project. I would have not been able to finish this project without her advice.

I would like to thank my thesis committee. Dr. Vangelis Metsis and Dr. Mina Guirguis who have been great professors and have helped me review my thesis.

Most importantly, none of this would have been possible without the love, support, and encouragement of my parents who live thousand miles away.

TABLE OF CONTENTS

	Page
ACKNOWLEDGEMENTS	v
LIST OF FIGURES	viii
LIST OF ABBREVIATION.....	ix
ABSTRACT.....	x
 CHAPTER	
1. INTRODUCTION	1
2. ACCESSOR AND ACTOR-BASED IoT MIDDLEWARE	5
3. ANDROID ACCESSOR HOST	15
3.1 Need for Accessor Host in Android	15
3.2 Apache Cordova Host and its Architecture	15
3.3 Building a Cordova Host for Android platform	17
3.4 Reasons for Choosing J2V8 accessor Host over Cordova Host for Android Application.....	19
4. APPROACH	25
4.1 Overview	25
4.2 Setting up an Android Application Project in Android Studio and Integrating with J2V8.....	26
4.3 Migrating Android application with J2V8 to an accessor host	29
4.3.1 J2V8 Android Accessor Host Architecture	29
4.3.2 Writing an accessor in J2V8 Android Accessor Host.....	31
5. EXPERIMENT	33
5.1 Implementation of a Native Android Application in Java for Fall Detection	33
5.2 Implementation of the Fall Detection application as composition of accessor ..	36

5.3 Evaluating performance of J2V8 Android Accessor Host (Our Approach).....	38
5.4 Heterogeneity of J2V8 Android accessor host	41
6. RELATED WORK	43
7. CONCLUSION AND FUTURE WORK	45
APPENDIX.....	46
REFERENCES	51

LIST OF FIGURES

Figure	Page
1. Actor based IoT middleware.....	2
2. Design Pattern of accessors	6
3. Apache Cordova Architecture.....	16
4. Apache Cordova plugin architecture	21
5. Android Application File Structure	23
6. Project file Structure	27
7. Android accessor Host using J2V8 Architecture	30
8. Native Android Application in java for Fall Detection	34
9. Comparing Microsoft Band with Moto 360.....	41

LIST OF ABBREVIATIONS

Abbreviation	Description
IOT	Internet of Things
JVM	Java Virtual Machine
AUT	Application Under Test
J2V8	Java with V8 scripting engine
AH	Accessor Host
JNI	Java Native Interface
JS	JavaScript

ABSTRACT

This thesis proposes an IoT middleware for Android using V8 script engine. This middleware supports the accessor abstraction for IoT and composes IoT applications based on various accessors implemented in JavaScript. Accessor is a software component having both input and output events. An accessor reacts to input events and produces streaming output events. Accessor follows the “Java write once and run everywhere” paradigm. It can thus be reused in various IoT applications as long as the application can be deployed in an accessor-compatible IoT middleware, which is also known as the accessor host. We evaluated the effectiveness of our Android IoT middleware based on battery consumption and memory utilization using a Fall-detection IoT application. We also demonstrated the reusability of accessors for heterogeneous IoT devices.

1. INTRODUCTION

The Internet of Things (IoT) is a domain that represents the next most exciting technological revolution since the Internet [6], [7]. IoT will bring endless opportunities and impact in every corner of our planet. With IoT, we can build smart cities where parking spaces, urban noise, traffic congestion, street lighting, irrigation, and waste can be monitored in real time and managed more effectively. We can build smart homes that are safe and energy-efficient. We can build smart environments that automatically monitor air and water pollution and enable early detection of earthquakes, forest fires and many other devastating disasters. IoT can also transform digital manufacturing Industry, making it leaner and smarter.

While IoT offers exciting potential and numerous opportunities, it remains challenging to seamlessly integrate the physical and cyber worlds. Many IoT programming models and connectivity protocols are being developed and the number is increasing each day. For example, in the Global Sensor Network (GSN) [1] project, the concept of virtual sensor, which is specified in XML and implemented with a corresponding wrapper, is provided as the main abstraction for developing and connecting a new IoT device. In the TerraSwarm project [2], an accessor design pattern implemented in JavaScript is proposed as the main abstraction. An accessor is similar to an actor [8] which is used in the actor based IoT middleware.

Actor based IoT middleware focuses on the open, plug and play architecture. An actor-based IoT middleware architecture is first presented in TerraSwarm [8], a joint research project between universities, government and private companies in USA. The

architecture of the actor-based IoT middleware consists of three different layers. They are the sensory layer, cloud layer and the mobile layer. Figure 1.1 depicts this architecture. The sensory layer includes all the physical sensors, embedded devices or actuators, the Mobile layer provides a bridge between the physical sensors and services available on Web or mobile computation engines. The Cloud layer provides a cloud service where both the other layers interact.

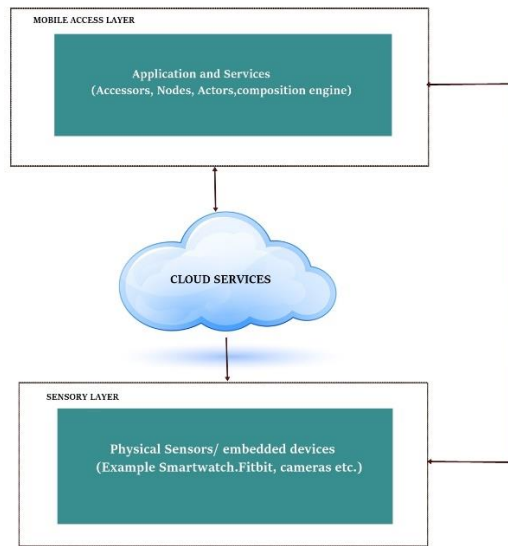


Fig 1.1 Actor based IoT middleware

In this thesis, we implement an Android accessor-based IoT middleware based on V8 script engine. Accessors are defined in JavaScript with input and output events and a set of functions. They react to input events and response with respective outputs using the initialize and the fire functions. These functions and all the input and output thus specify the behaviors of accessors.

An accessor is instantiated and run by an accessor host. An accessor host is a service which can be run in the network or on a client platform that hosts applications. The accessor host uses the accessor as a proxy to get device's specific data. The accessor host adopts the actor-based IoT middleware architecture. In a nutshell, an accessor host for an IoT application can be compared with what a browser is to the Internet.

The key advantages of an accessor host are the openness in supporting a new IoT device type and the support of a light-weight programming model. Users can extend the computational units of the accessor-based IoT middleware by developing a pluggable accessor. This means an accessor host embraces the heterogeneity of IoT devices without dictating a particular device interface standard.

This thesis focuses on developing and evaluating a J2V8 Android accessor Host. J2V8 Android accessor Host is also known as the accessor based IoT middleware for Android using J2V8. We will be using the term J2V8 Android Accessor Host from now in this thesis. There are a few existing accessor hosts developed by the TerraSwarm group but none of them explicitly runs on the Android platform. In April, this year, Cordova Host was released by the TerraSwarm group. This indeed is an accessor host for Android. However, Cordova Host is optimized for accessing Web APIs on Android platform. This host works great when an IoT application is created using solely Web technologies (Javascript, html, css). Cordova Host adds a layer of inefficiency when it comes to interaction with IoT application that requires access to hybrid technologies of Web and non Web-based. We are proposing an Android accessor host using V8 scripting

engine. This host has the advantage that it can interact well with both Web and non-Web based technologies.

The main contributions of this thesis are:

- Developing an accessor-based IoT middleware using J2V8 script engine for the Android platform which is also known as J2V8 Android accessor Host.
- Evaluating the effectiveness of the proposed J2V8 Android accessor Host in terms of battery power consumption and memory usage for a Fall-detection IoT application.
- Demonstrating that this J2V8 Android accessor Host can be used to compose IoT applications for heterogeneous physical or virtual sensors without additional programming.

2. ACCESSOR AND ACTOR- BASED IoT MIDDLEWARE

Existing architectures for IoT middleware fall into three categories from our observation. The first category is known as service-based solution, generally adopts the Service-Oriented Architecture (SOA) [9] and allows developers to add or deploy a diverse range of IoT devices as services. Such architecture provides users with simple tools such as Web applications to view the raw data that the IoT devices are collecting, but usually provides limited functionalities to users when it comes to composition with other applications or interpreting the collected data. The second category, which is known as cloud-based solution, limits the users on the type and the number of IoT devices that they can deploy, but enables users to connect, collect and interpret the collected data with ease since possible use cases can be determined and programmed in a-priori. The third category is the actor-based framework that emphasizes on the open, plug and play IoT architecture. A variety of IoT devices can be exposed as reusable actors and a graphical tool can be provided for the composition of IoT applications.

The advantage of actor-based IoT middleware is that the middleware can be embedded in all the layers (sensory layer, mobile access layer and the cloud layer). This allows the device to device communication which provides better privacy and security support in IoT applications. This style of architecture also allows better latency and scalability since IoT devices can perform computation where it is most beneficial. The basic unit of IoT application running in an actor-based IoT middleware is an accessor. Accessors provides a high-level abstraction for both physical and virtual IoT devices.

As a programming model, an accessor is a software component that contains a set of input/output ports, parameters and some functions. An accessor interface also specifies

what modules are required of an accessor host in order to execute the accessor. This means an accessor host can tell whether it can instantiate or execute an accessor by simply examining its interface. The functions in an accessor are triggered when an accessor receives data on its input ports. In this way, an accessor is like an actor that wraps a device or service in an actor interface.

Figure 2.1 shows the design pattern of an accessor. It supports the Asynchronous Atomic Callbacks (AAC) communication protocol which means that the responses need not be synchronous with the request. The figure below shows an accessor in a Swarmlet host and invokes a remote REST service [24].

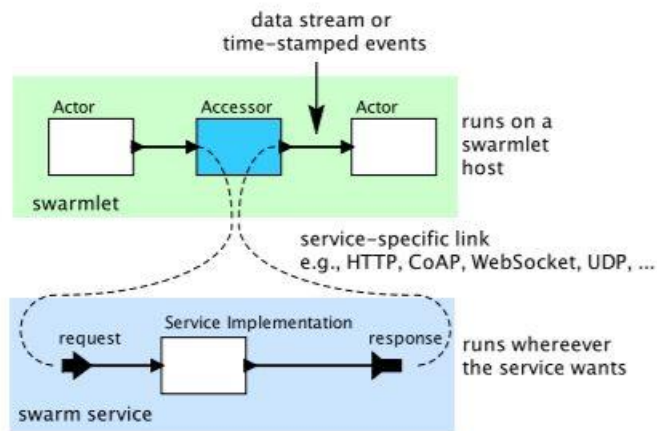


Fig 2.1 Design Pattern of accessors [24]

An accessor is implemented in a JavaScript file. The ECMAScript standard is chosen because it is supported in several environments (Browser, node.js and other JavaScript engines). The input port, output port, and parameters define the interface of an accessor. The functions define the behavior/implementation of an accessor. The

functions are triggered with the arrival of input events. The fire function and initialize function are the basic functions of all assessors. Other functions include the alert (message), which pops up a dialog with a specific message. The require (module) function loads the required library module for interacting with vendor specific IoT devices. Other functions are get function, send function, setTimeout function and setInterval function.

Below is an example implementation of an assessor that conforms to the assessor specification standard. This assessor accepts the numeric values from the accelerometer sensor of the Microsoft Band smartwatch via the assessors dataRate input port and sends the result to the assessor output port which is called accelerometerOutput. The display function makes use of the output function in Java and is used for debugging purpose. This assessor has two of the four parts, an interface definition in the setup function, and a specification of the function to be performed when the assessor is fired. In this case, the function is specified in the body of the fire function, which will be invoked by the assessor host whenever a new input is received. The send function which is invoked inside the fire function sends the input to the host where the assessor is called. The snippet of this assessor is shown below:

```
var setup = function () {  
    //display function defined inside java layer will display the String in  
    console  
    display ("MSBandAssessor setup with J2V8 Assessor");  
  
    //input port to receive accelerometer value from smartwatch.  
    dataRate is a
```

function in java layer(Microsoft Band API) to collect the sensor values

```
this.input("dataRate", 'type:number');

//output port for Accelerometer Sensor Values provided by the
function dataRate
this.output("accelerometerOutput", 'type:number');

};

var initialize = function() {
    display("MSBandAccessor initialized");
};

var fire = function() {
    display("MSBandAccessor fired");
// sending the input to the host where the accessor is called.
    send( "accelerometerOutput",this.get("dataRate"));
};
```

Fig 2.2 data collection accessor snippet

The addInputHandler function can be used to specify that a specific function needs to be performed when a particular new input named 'dataRate' is received. The addInputHandler function handles specific input and the function reacts only when the specified input is provided. In the following example, the addInputHandler function is placed inside the initialize function which handles the input named "dataRate".

```

var setup = function () {

    //display function defined inside java layer will display the String in console
    display ("MSBandAccessor setup with J2V8 Accessor");

    //input port to receive accelerometer value from the smartwatch.
    this.input("dataRate", 'type:number');

    //output port for Accelerometer Sensor Values provided by the function
    dataRate
    this.output("accelerometerOutput", 'type:number');

};

var initialize = function() {

    //display function defined inside java layer will output the String to the console
    display("MSBandAccessor initialized");

    var self=this;

    this.addInputHandler("dataRate", function () {

        // sending the input to the host where the accessor is called.
        self.send("accelerometerOutput",this.get("dataRate"));

    });
};

var fire = function() {
display("MSBandAccessor fired")
}

```

Fig 2.3 data collection accessor snippet with input handler

The value of the variable 'this' is captured in a variable called self. When a function is invoked, self-acts as an object on which the function is invoked.

The accessor host knows when and how to invoke the Setup function and the Initialize function in an accessor, which has functions input, output etc. defined. But the input handler function may or may not be invoked on the accessor. Any callback function should be written in such a way that it should execute, irrespective of the object the host invokes it on. In the above example, the callback function will be the function passed to addInputHandler.

TerraSwarm Research Center provides several accessor hosts. All existing accessor hosts are built on top of the Common Host. This host has basic host functionalities defined and those functions can be reused as it is by other hosts.

The Common Host provides a constructor for instantiating an accessor, this is named as instantiateAccessor() and takes as an argument the fully qualified accessor class name. Class name is enough to instantiate the accessor using instantiateAccessor function defined in common Host. The function getAccessorCode() will retrieve the accessor's source code given the class name.

The object which is returned by the accessorInstantiate() function is an instance of the accessor. A host inherits all the functions defined in the Common Host. These functions can be invoked using the instance as the handle. Some of the common host functions inherited by other hosts are described below:

- `react()`, react to the input provided to the accessor.
- `setParameter(name, value)`: set a parameter with a parameter reference name and a parameter value.
- `initialize()`: initialize the accessor.
- `provideInput(name, value)`: provide an input value.
- `latestOutput(name)`: retrieve an output value produced in `react()`.
- `wrapup()`: release used resources and terminate the accessor.

We briefly describe all the available working and experimental accessor hosts under TerraSwarm project in the following paragraphs.

Browser Host

Browser Host runs in web browsers. It basically supports executing accessors within the web browser. The Browser Host is layered on top of the common Host and loads Common Host's functions if required [10]. The web page of the Browser Host gives a list of working accessors [18].

To create an instance of an accessor, where you can provide input and parameter values in entry boxes and cause the accessor to react, include the following script tag in the head section of the web page:

```
<script src="/accessors/hosts/browser/browser.js"></script>
```

To include a specific accessor in a web page, include an HTML tag with the class name of the "accessor". In the example below, the HTML tag is `<div>` and the class is

named as “accessor”. The src will be the path of the accessor to be included. Here is an example of the MSBandAccessor resides inside the band folder therefore the path will be “band/MSBandAccessor”. For example:

```
<div class=”accessor” src=”band/MSBandAccessor” id=”MSBAND”></div>
```

The id attribute can have whatever value you like, but the value must be unique on the web page.

You can also create a directory of accessors by including in your document an element with class “accessorDirectory”[10]. For example:

```
<div class=”accessorDirectory”></div>
```

When the web page DOM content is loaded. The generate function is invoked to get access to each of the accessor.

Cape Code Host

Cape Code Host is a graphical-based host for creating, composing and executing accessors. Cape Code can run IoT application that contains accessors as well as actors. Cape Code uses Java Nashorn engine for executing accessors [11].

Cape Code Host provides the most comprehensive support for the accessor Specification . It is built from three components:

- The Nashorn JavaScript engine. Nashorn is the native JavaScript in Java 8 and above. It runs on the Java virtual machine and provides full access to Java, thereby offering a rich programming environment that combines a scripted,

gradually-typed language (JavaScript) with a strongly-typed object-oriented language (Java). Since Java has rich networking and I/O libraries, it is relatively easy to build interfaces to devices using Nashorn.

- Ptolemy II [25] provides a mechanism for importing accessors from a library and connecting them to each other and to actors. The default library of actors is described in [19]. Ptolemy II has a graphical block-diagram editor called Vergil for composing accessors and building an IoT application.
- The CapeCode Host makes use of Vertx to implement several of the optional modules that an accessor may require, including modules providing HTTP, Web socket, and publish-and-subscribe services

Figure 2.1 shows the graphical block-editor for a simple TestComposite Accessor for Cape Code Host [23].

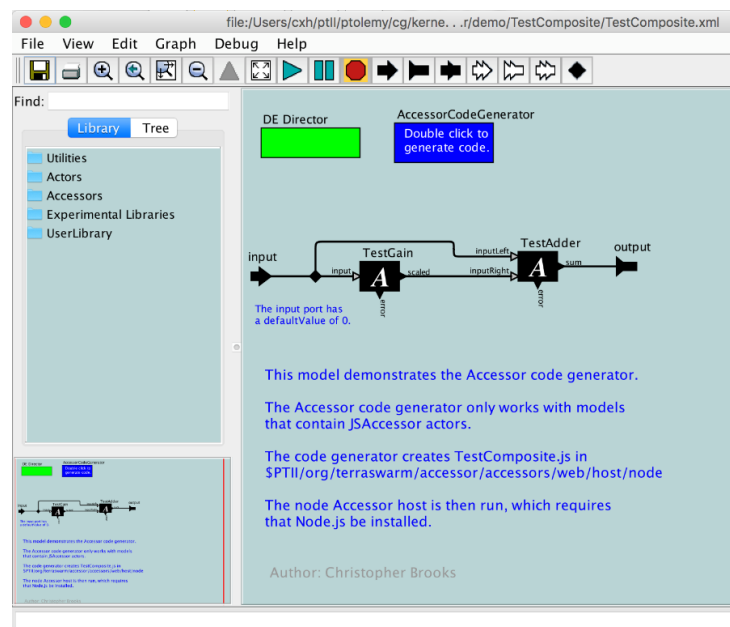


Fig 2.2 TestComposite Accessor for Cape Code [23]

The TestComposite accessor is an accessor which instantiates two simple accessors called TestGain and TestAdder in Cape Code. The Graphical Block in Figure 2.2 shows the two accessor connected with input and output port.

Node Host

Node Host is basically a Node.js engine developed on top of the Common Host. This host works well when working with npm (Node Package Manager).

npm is an online repository where all the open-source node.js projects are published. Node Package Manager provides an easy way for the JavaScript developer to share and reuse the codes. It also provides an easy way to update the code placed in the online repository.

Node Host extends Common Host and it is a pure JavaScript Swarmlet host. Industrial Cyber – Physical Systems Center (iCyPhy) explains in detail about Node Host and available accessors in Node Host [20].

Cordova Host

Cordova is an application development platform that is used for building mobile Apps using HTML, CSS and JavaScript [12]. Cordova Host is designed to work on Android operating system. Cordova Host is a wrapper around Node.js with the additional support of web application development. Cordova Host is explained in more detail in next chapter.

3. ANDROID ACCESSOR HOST

We will first summarize the need for an Android accessor host. Then we will discuss the process used in developing Apache Cordova Accessor Host. We will then justify why J2V8 is a better option for developing an Android accessor host.

3.1 Need for Accessor Host in Android

As discussed earlier, accessor is instantiated and run by an accessor host. An accessor host is a service running on the network or on a client platform that hosts IoT applications. Accessor host instantiates accessors and interacts with the physical devices. An accessor host manages the lifecycle of the accessors. Borrowing the analogy from an electrical outlet, an accessor host is like a socket where different accessors can be plugged in. This functionality of reusing accessors by plugging them into a common socket makes developing IoT application using the accessor paradigm faster, less complex and more reliable. This is the key reason for developing an Android accessor host. In this thesis, we will refer to accessor host in an Android platform as Android accessor host.

Cordova Host is an existing IoT middleware for Android platform. Apache Cordova is a platform for creating Web applications that can be deployed as Android applications. In other words, Apache Cordova is used to create an Android application using web technologies. We discuss the architecture of Apache Cordova in the following sections.

3.2 Apache Cordova Host and its Architecture

Apache Cordova is an open-source mobile application development framework. It allows you to use standard web technologies such as HTML5, CSS3, and JavaScript for

cross-platform development, avoiding each mobile platform's native development language.

We can think of Apache Cordova as a container for connecting Web application with native mobile functionalities. Web applications cannot use native mobile functionalities by default. This is where Apache Cordova helps. It offers a bridge for connecting between Web applications and mobile devices.

In Apache Cordova, applications execute within wrappers targeted to each platform, and rely on standards-compliant API bindings to access each device's sensors, data, and network status. Figure 3.2.1 shows the architecture of Apache Cordova.

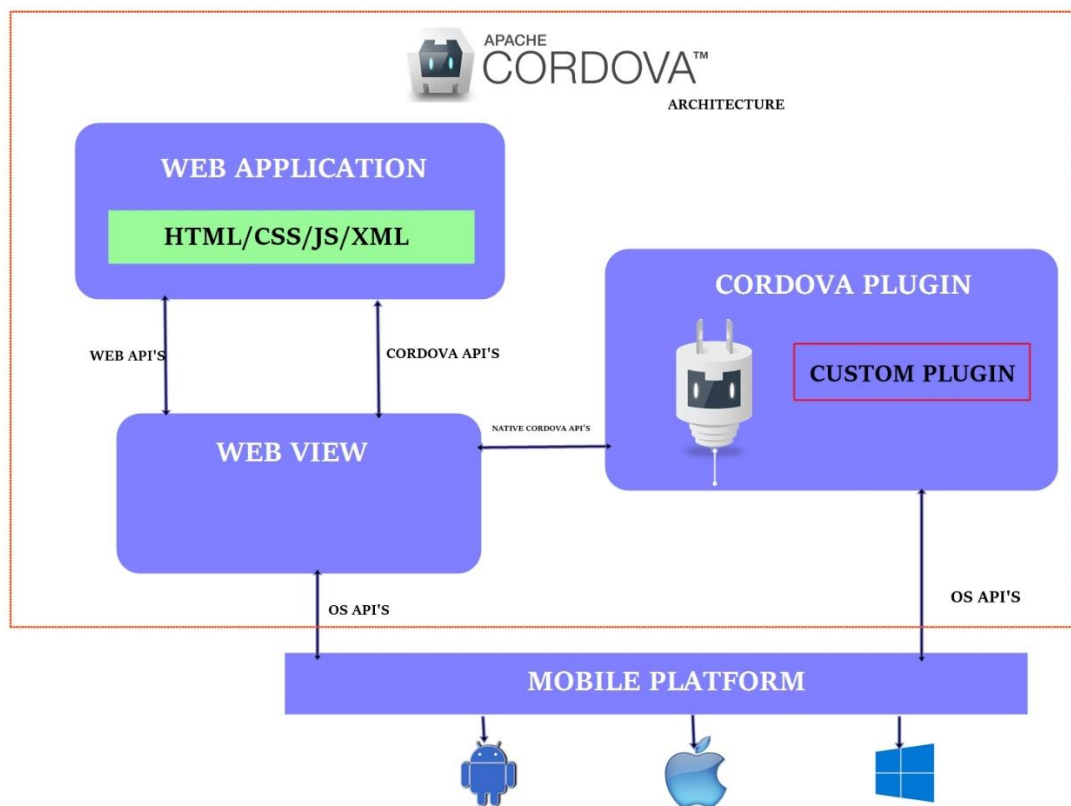


Fig 3.2.1 Apache Cordova Architecture

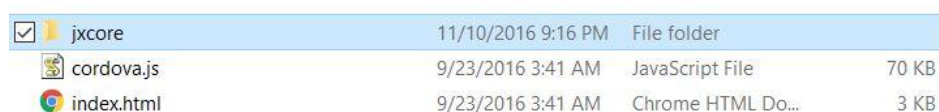
To install Apache Cordova, Node.js and NPM should be installed in the system. Node.js is the JavaScript engine needed for the Cordova application development. Android SDK should be installed on the machine for the Android platform. If the platform is IOS then XCode should be installed.

3.3 Building a Cordova Host for Android platform

To build the Android accessor host with Apache Cordova, there is a need to use a JXCore plugin for Cordova. The main benefits of building an Android accessor host using Cordova are:








- Cordova comes with a host of plugins that can be used without modification for an accessor host.
- With the JXCore plugin, Node host can be directly usable along with all the modules built for it. There is already a running Node host available from the TerraSwarm project. The only additional setup is installation of:
 - Android SDK
 - Node.js

Once all the required software are installed, the folder structure for the Cordova Host will be as follows for Window operating system:



<input checked="" type="checkbox"/>	jxcore	11/10/2016 9:16 PM	File folder	
	cordova.js	9/23/2016 3:41 AM	JavaScript File	70 KB
	index.html	9/23/2016 3:41 AM	Chrome HTML Do...	3 KB

Figure 3.3.1 Cordova Host Folder structure in Windows part-1

	node_modules	11/10/2016 9:16 PM	File folder	
<input checked="" type="checkbox"/>	 app.js	9/23/2016 3:41 AM	JavaScript File	5 KB
	 commonHost.js	9/23/2016 3:41 AM	JavaScript File	72 KB
	 events.js	9/23/2016 3:41 AM	JavaScript File	9 KB
	 nodeHost.js	9/23/2016 3:41 AM	JavaScript File	8 KB
	 TestAccessor.js	9/23/2016 3:41 AM	JavaScript File	4 KB
	 util.js	9/23/2016 3:41 AM	JavaScript File	17 KB

3.3.2 Cordova Host Folder structure in Windows part-2 (inside JXCore)

The main purpose of JXCore packaging is in making the deployment easier. In building the Cordova Host, JXCore is used to interact with the host (which is Node.js). The host then instantiates all the accessors which reside inside the JXCore folder as mentioned in Fig 3.3.1 and Fig 3.3.2. JXCore is defined in the main HTML file of the Cordova Application. Here index.html is the main HTML file. To Install the JXCore plugin, the following steps are needed:

1. Go to the root of Cordova application folder you created as part of the installation steps for Cordova
2. Run

Jxc install

If you have problems, you will see jxc install fails message

3. To verify JXCore plugin is correctly installed, run

`cordova plugin ls`

The output of the above command should show one of the plugins installed as:

```
io.jxc.core.node 0.1.1 "JXcore Mobile"
```

4. To test installation, run the following commands:

```
jxc install - - sample express_pref; cordova run android
```

Here the TestAccessor.js is the accessor called from the app.js which is the main entry point of the host. The TestAccessor.js is a simple accessor written as per the accessor specifications which will be discussed in later sections.

3.4 Reasons for Choosing J2V8 accessor Host over Cordova Host for Android Application

Apache Cordova is designed for cross platforms web application development. We can use the standard web technologies to create a web page such as HTML5, CSS3, and JavaScript etc. and then embed that as a Web view. However, when it comes to collecting sensor data in smart watches and accessing sensors in Android platform, we need to interface with non-web based native codes. Our IoT application uses an Android phone, a Microsoft Band 2 and a moto360 smartwatches. The Microsoft Band SDK is provided to access all the sensors in Microsoft Band 2. Each and every sensor is accessed using the provided SDK which is in Java. However, to use Java in Apache Cordova requires the development of a new Cordova plugin. The Java program accessing the sensors should be written in the form of a plugin. Developing a plugin in Cordova is a non-trivial task. The following paragraph will discuss how to custom develop Cordova plugin and why it is a complex task. In this thesis, we refer Apache Cordova and Cordova interchangeably.

Cordova plugins are an integral part of the Cordova ecosystem. The plugin provides an interface for Cordova and native components to communicate with each other and bindings to standard device's APIs. The plugin facilitates the invocation of native codes from JavaScript.

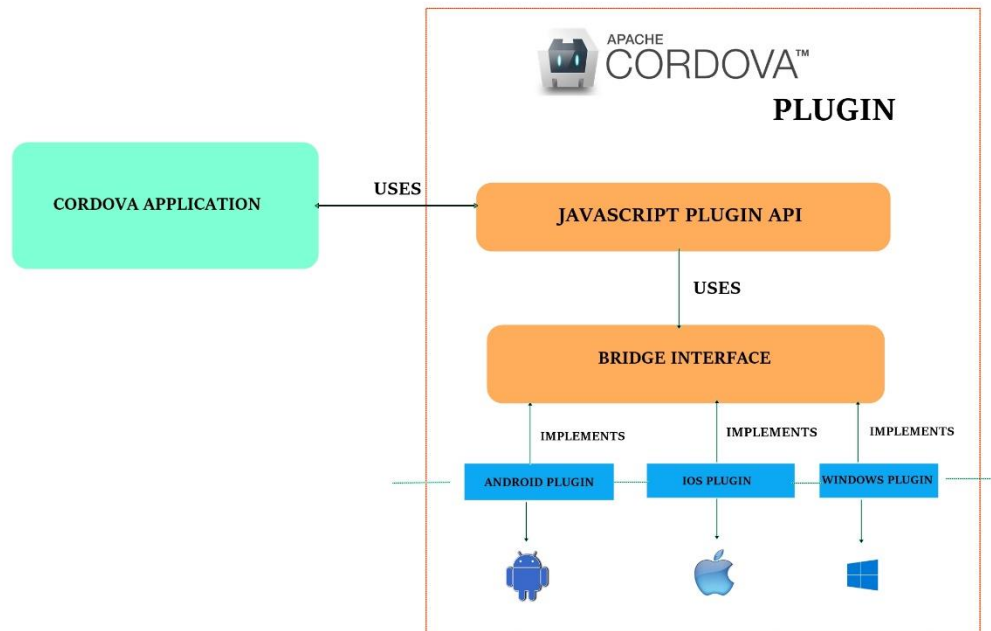
Apache Cordova project maintains a set of plugins called the core plugins [12]. These core plugins provide applications the ability to access device capabilities such as battery power management, camera, contacts, etc. The Figure 3.4.1 below shows a set of core plugins for Apache Cordova.

	amazon- fireos	android	blackberry10	Firefox OS	ios	Ubuntu	wp7 (Windows Phone 7)	wp8 (Windows Phone 8)	win8 (Windows 8)	tizen
cordova	✓ Mac, Windows, Linux	✓ Mac, Windows, Linux	✓ Mac, Windows	✓ Mac, Windows, Linux	✓ Mac	✓ Ubuntu	✓ Windows	✓ Windows	✓	✗
Embedded WebView	✓ (see details)	✓ (see details)	✗	✗	✓ (see details)	✓	✗	✗	✗	✗
Plug-in Interface	✓ (see details)	✓ (see details)	✓ (see details)	✗	✓ (see details)	✓	✓ (see details)	✓ (see details)	✓	✗
Platform APIs										
Accelerometer	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Camera	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Capture	✓	✓	✓	✗	✓	✓	✓	✓	✗	✗
Compass	✓	✓	✓	✗	✓ (3GS+)	✓	✓	✓	✓	✓
Connection	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
Contacts	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗
Device	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Events	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
File	✓	✓	✓	✗	✓	✓	✓	✓	✓	✗
Geolocation	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Globalization	✓	✓	✗	✗	✓	✓	✓	✓	✗	✗
InAppBrowser	✓	✓	✓	✗	✓	✓	✓	✓	uses iframe	✗
Media	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
Notification	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
Splashscreen	✓	✓	✓	✗	✓	✓	✓	✓	✓	✗
Storage	✓	✓	✓	✗	✓	✓	✓ localStorage & indexedDB	✓ localStorage & indexedDB	✓ localStorage & indexedDB	✓

3.4.1 Cordova Core Plugin [12]

In addition to the core plugins, there are several third-party plugins which provide additional bindings to features not necessarily available on all platforms. We can search for Cordova plugins using npm.

The APIs for accessing the sensors of smartwatches should be written in the form of a plugin if we use Cordova's host. Plugins comprise a single JavaScript interface along with corresponding native code libraries for each supported platform. In essence, this hides the various native code implementations behind a common JavaScript interface.



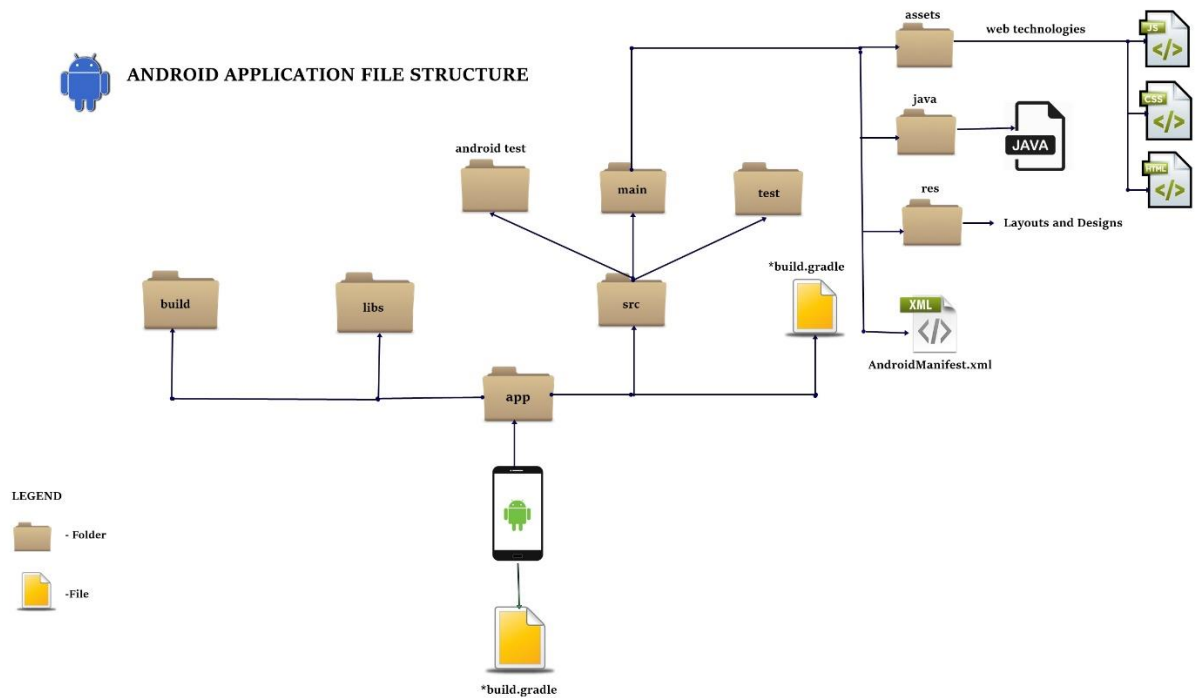
3.4.2 Apache Cordova plugin architecture

Apache Cordova uses Plugman to validate the plugin. So creating a plugin for each and every accessors and including it in the application made the process of accessing the sensors from IoT devices complicated and heavy.

We will illustrate with an example of creating an Android application in Cordova to show the difficulty of creating a plugin. This will explain why the Cordova accessor host for Android is not a best option. A typical Android application file structure is listed in figure 3.4.3. We will be using Android studio IDE to develop Android applications. The App file is the main file of the application which interacts with the Android mobile device.

The App is sub divided into build, libs, src and build gradle files. These are the main files of the application which are auto generated by Android Studio. Here the build folder is generated by Android Studio IDE. The libs folder is initially empty. All the library used in the app will be stored in the libs folder.

The build.gradle file is a place where we will mention all the library files which we are going to use in the app. Src folder contains only one important folder named as main. It is the most important folder which contains three sub folders and a file. The AndroidManifest.xml file provides essential information about the application to the Android operation system. The information's such as permission or activity.



3.4.3 Android Application File Structure

The assets folder contains all the web files such as HTML, CSS and JavaScript etc. The Java folder contains all the Java files related to the application. Basically, an Android application's initial entry point will be the Activity file. Activity file tells the application what it must load initially. It implements how the buttons must work when receiving a click event and how the layout of the page should be for the Application. The Activity file basically controls the UI of the application and its functionalities. This Activity file is written in JAVA. It can be a single file or multiple files or packages supporting the main activity file.

Cordova is a platform which provides a bridge between Android and the Web APIs. It wraps the Android application around Web APIs. Cordova is thus a wrapper for Android application that needs to be run as a Web application.

By using Cordova to develop an Android application gives an advantage to those who want to basically work with web technologies and convert the web output to Android application.

But when it comes to interacting with native codes such as Java programs to control the application, many libraries are involved which makes it hard to understand and also makes it heavy in terms of size. Cordova Host uses Node.js, JXCore, JavaScript and html to create an accessor. Cordova Host will be the best option to use when it comes to creating an accessor that only requires access to web-based technologies. In this thesis, we are developing an Android application to interact with the smartwatch and getting all the sensors information from the smartwatch to perform certain tasks. This requires interaction with non-web based technologies. For the MS Band smartwatch that we use, this requires access to vendors supplied Java-based SDK. Cordova Host requires development of plugins for interaction with non-Web-based technologies.

Therefore, we need to develop a new J2V8 Android accessor Host. J2V8 is a set of Java bindings for Google's popular JavaScript engine, V8. It was developed to bring highly efficient JavaScript to Android. As we have already discussed earlier that accessors are written in JavaScript. J2V8 will be one of the best options to develop an efficient accessor. Also, J2V8 is purely java based V8 Scripting engine. This means we can write JavaScript code inside Java program. In the next chapter, we will discuss the J2V8 and the J2V8 Android accessor Host in depth.

4. APPROACH

4.1 Overview

In this chapter, we will discuss the implementation of our J2V8 accessor Host for Android. We will discuss why J2V8 is a highly efficient JavaScript runtime for Java. We will also discuss the requirements and procedures to develop a J2V8 Android accessor Host.

Nashorn and Rhino are widely used technologies to run a JavaScript in Java. But both Nashorn and Rhino are not available in Android operating system. In Android, the most common way to interact with JavaScript is by using a Web view. A Web view helps Android Application to interact with Web APIs. It works well when JavaScript is not the main focus which means all the functionalities are handled by Java and only a small portion of codes are in JavaScript.

As we discussed earlier, accessors are written in pure JavaScript. To develop accessor host for Android we need a mechanism to run JavaScript inside Android environment which is the same as the Java environment. J2V8 uses Google's V8 Scripting engine to execute JavaScript in Java.

J2V8 is a high performance JavaScript runtime engine written in C++. J2V8 focuses on performance and tight integration with V8. It takes a primitive first approach. Primitive first approach means all the values are accessed as primitives if possible. This forces a more static type system between the JavaScript and Java code, but it also improves the performance since intermediate objects are not created.

Our approach is to integrate J2V8 with Android and create an Android accessor host using J2V8 as depicted in Figure 4.1.1



Fig 4.1.1 Approach

The first step in creating an accessor host using J2V8 is to create an Android App. We use Android Studio IDE to create an Android App.

4.2 Setting up an Android Application Project in Android Studio and Integrating with J2V8

Android Studio IDE is based on IntelliJ IDEA from Jet Brains and is being offered by Google for free. Android Studio uses a gradle build system to build Android applications. Android Studio uses modules to manage and organize codes. Each module has its own gradle build. This means each module can provide its own dependencies in its own gradle build. It is easy to work with modules when the application is huge.

In Android Studio, to create a new project, perform Select File → New Project. In the New Project screen, enter the application name and company domain. For us, the name of the project will be “J2V8AccessorHost” and name of the company domain will

be edu.txstate.j2v8.J2V8AccessorHost”. Select the project location where we want to save our project. This location will be our working directory. Once it is done, click next. The next step will prompt us to specify the target Android device. Keep the default values for the target Android device and click next. Now we have to add an Activity to the App. In our project, it will be the MainActivity. Once this is done, click next. Keep the default values for customizing the activity screen and click Finish [13]. Android official developer page explains in-depth about developing a simple Android application using the Android studio. The Android application will be referred to as App from now on in this chapter. The Android App structure of our project is shown in figure 4.2.1.

After we are done with setting up an Android Application project in Android Studio, we are ready to integrate J2V8 with the project.

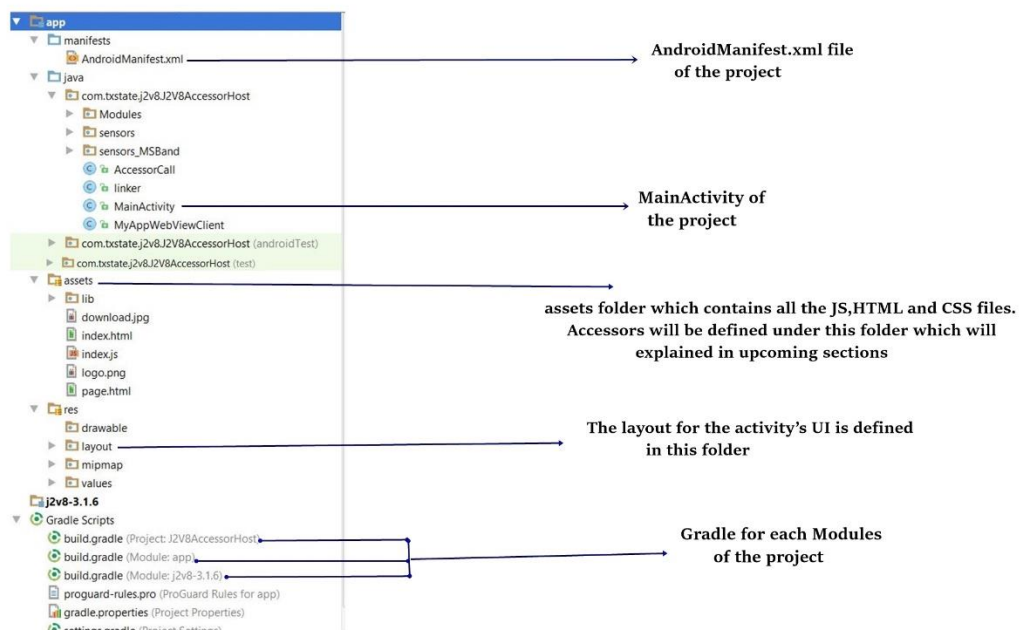


Fig 4.2.1 Project file Structure

In Android, we will integrate J2V8 as an AAR file. This is called Shipping J2V8 as an AAR. AAR is a binary distribution of an Android Library Project. An AAR file can contain both a jar as well as resources. Download the latest version of J2V8 AAR file from eclipsesource [<http://eclipsesource.com/blogs/getting-started-with-j2v8/>] . In our case we use the J2V8 version 3.1.6.

To use J2V8 in our Android Application, simply create a new module and import the J2V8 AAR file. To do so in Android studio:

- Select File→New→New Module→Import .JAR/.AAR and import the downloaded J2V8.AAR file. Once this is done, a module will be created in the App structure. In simple words, we can say a folder will be created. In our case we named the module as j2v8-3.1.6. A module with the same name will be created in the app structure as shown in Figure 4.2.1.
- Once the module is created we can find build.gradle (Module: j2v8-3.1.6) under Gradle Scripts. This is created automatically.
- Then in our project build.gradle (the one under ‘app’) add the following statement in the dependencies section.

```
dependencies {
```

```
    compile project(':j2v8-3.1.6')
```

Module Name,

In our case j2v8-3.1.6

- After adding the above statement in the dependencies, synchronizes the gradle. This compiles the module and make it available for our app. Note that, to synchronize gradle in Android studio, Select File->Synchronize.

Setting up an Android Application project in Android Studio and integrating with J2V8 is now complete. Now we will look into steps to convert an Android application integrated with J2V8 into an Android accessor host.

4.3 Migrating Android application with J2V8 to an accessor host

Android accessor host using J2V8 is also referred to as the J2V8 Android Accessor Host in this thesis. As discussed earlier, accessor is a software component that contains set of input/output ports, parameters and some action functions. These are written purely in JavaScript. The purpose of adding J2V8 library in Android Application is to be able to execute JavaScript in Android environment. J2V8 facilitates accessors interacting with native Java programs. We discuss the architecture of our J2V8 Android Accessor Host in the following section.

4.3.1 J2V8 Android accessor Host Architecture

As we already discussed earlier, the structure of an Android application basically consists of a MainActivity java file which is the entry point of the Android application. It tells the application about the layout to be loaded and UI functionalities. The Main Activity is a java program written for the Java layer. Sometimes the MainActivity.java is supported by other Java files or even another Activity files.

J2V8 executes the JavaScript inside MainActivity java. Accessors are written in JavaScript and they are defined in the assets folder where all the web technologies related

to Android application is specified. This includes files such as HTML, CSS and JavaScript. All the accessors in J2V8 Accessor Host are defined in the assets folder which is also known as the Web layers of the application.

This Web layer needs to be supported by two important JavaScript APIs to make it an accessor host. They are RequireJS API and CommonJS API. RequireJS and CommonJS together forms the functionalities of what is provided in a Common Host. We cannot just inherit Common Host because Common Host requires node.js and node.js is not available in V8 scripting engine. RequireJS and CommonJS provide some high-level scripting modules which help in developing accessors and the accessor host. Figure 4.3.1.1 shows the Architecture of J2V8 accessor Host

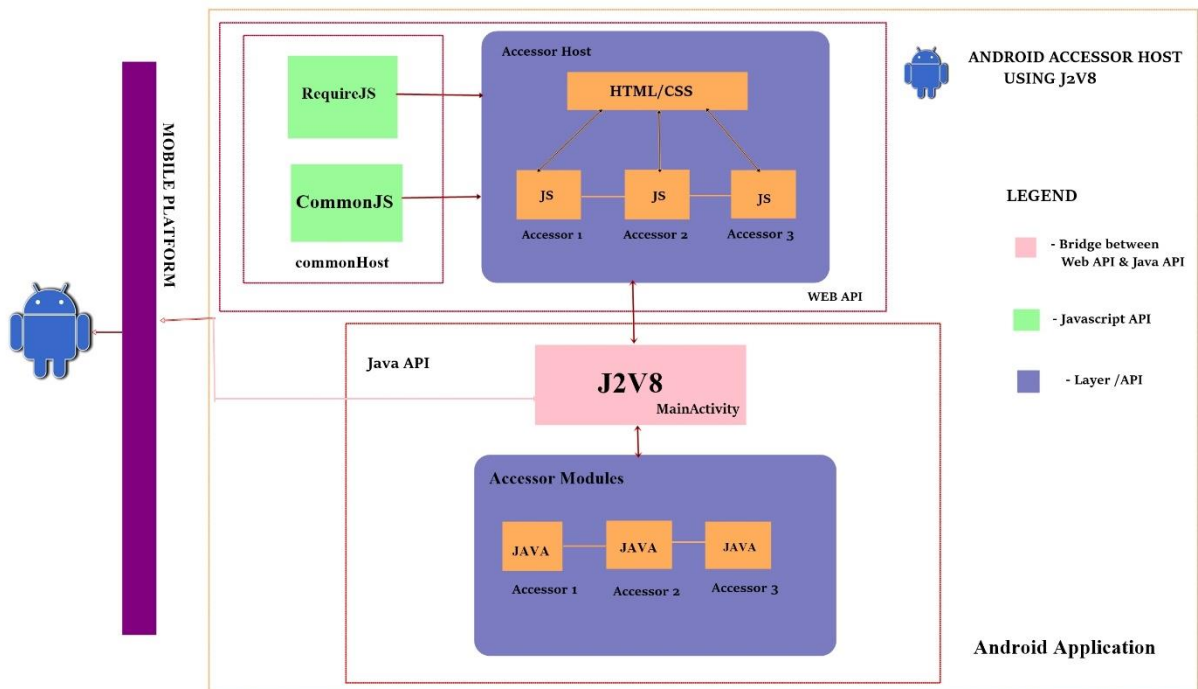


Fig 4.3.1.1 Android accessor Host using J2V8 Architecture

Below we will illustrate how to develop an accessor that can be executed by the J2V8 Android Accessor Host.

4.3.2 Writing an accessor in J2V8 Android Accessor Host

An accessors follows some standard specification. A simple data Collection accessor is mentioned in Fig 2.2 and Fig 2.3. This accessor is instantiated by the J2V8 Accessor Host and is executed by the J2V8 engine residing in the Java layer. Below is the J2V8 JavaScript Executer script which demonstrates how an accessor is instantiated.

```
V8 runtime = V8.createV8Runtime();  
WebViewJavaScriptInterface w=new  
WebViewJavaScriptInterface(this);  
V8Object v8Con=new V8Object(runtime);  
runtime.add("w",v8Con);  
String script=ReadAccessor();  
runtime.executeScript(script);  
String  
result=runtime.executeStringFunction("datacollectionAccessor",null);  
System.out.println(result);
```

Snippet 4.3.3 (J2V8 Script Executor)

The ReadAccessor() function reads the complete accessor from the Web layer. This is executed from the MainActivity.java. Some of the Common Host functionalities

such as `setInterval` and `addInputHandler` are defined in J2V8 Accessor Host with the help of `RequireJS` and `CommonJS`. All Common Host functionalities are implemented in J2V8 Accessor Host via the `requireJS` and `CommonJS` API's. (NEED to show the `require` and `common host JS` here).

5. EXPERIMENT

In this section, we evaluate the J2V8 Android Accessor Host by implementing two versions of an IoT Fall Detection application. The first one is a native Android application in Java, and the second one is an implementation of the same application as a composition of accessors. The Fall detection application is designed for monitoring the well-being of an elderly person in real-time at anytime and anywhere by wearing a smartwatch paired with a smartphone. The application works by collecting accelerometer data from a smartwatch and processing those data in a paired smartphone, it is possible to reliability detect (93.8% accuracy) whether a person has encountered a fall using this IoT application.

We will present an analysis of the performance of the two versions of Fall Detection application in terms of memory usage, battery consumption and reusability.

5.1 Implementation of a Native Android Application in Java for Fall Detection

In this application, we use a Microsoft Band smartwatch and an Android smartphone to detect fall. This application is written purely in Java. The architecture of a native Android application in Java for Fall Detection is shown below in figure 5.1.1

We used Android Studio to develop this application. The Android platform version used is 7.0. The Microsoft Band smartwatch is used to collect sensor data's and render it on the Android App running on the smartphone. As shown in Figure 5.1.1, the whole application is written in Java and interacting with the Microsoft MS Band SDK. The Microsoft MS Band SDK is a jar file which has to be included in the application

library. The Fall Detection application relies on WEKA's machine learning algorithm to predict the fall.

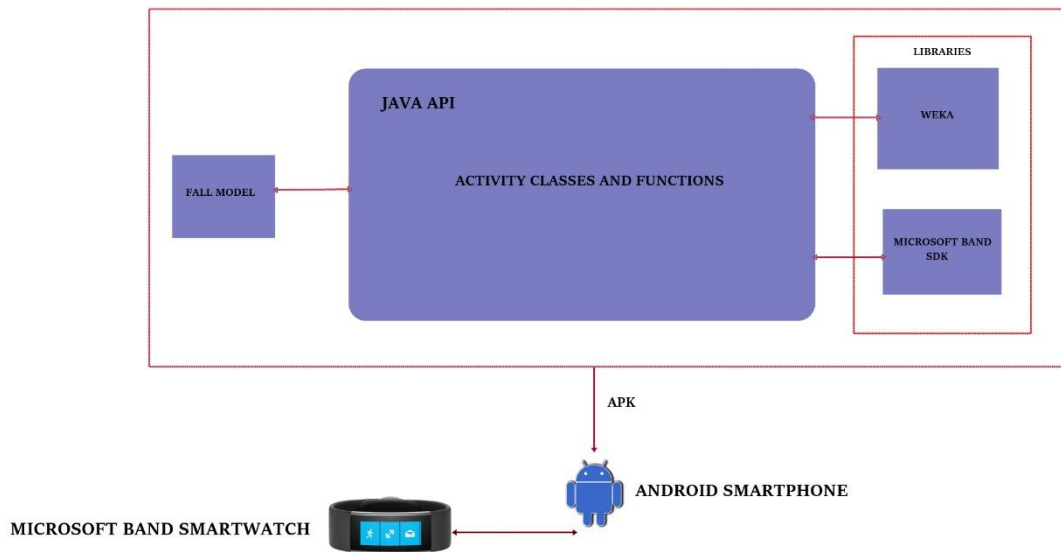


Fig 5.1.1 Native Android Application in java for Fall Detection

WEKA (Waikato Environment for Knowledge Analysis) is a suite of machine learning software written in Java, developed at the University of Waikato, New Zealand. It is a free software licensed under the GNU General Public License [14]. WEKA contains visualizations tools and various machine learning algorithms, which help in data analysis and predictive modeling [15]. The fall detection application uses Support Vector Machines (SVM) to train the fall detection model.

As discussed earlier, MainActivity.java is the entry point of this application. The MainActivity.java loads the layout and other dependent classes to be imported into the application. Figure 5.1.2 represents the folder structure of the native Android application in Java for Fall Detection.

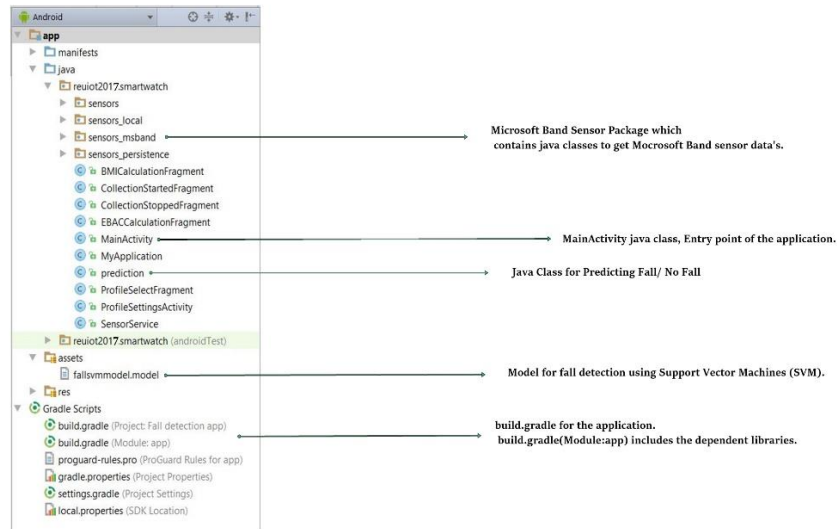


Fig 5.1.2 Native Android Application in java for Fall Detection

The Native Android application in Java for Fall Detection has different packages such as sensors, sensors_local, sensors_msband and sensors_persistence under the main package reuiot2015.smartwatch. These packages contain Java classes which are responsible for connecting to the smartwatch, access the sensor data from the smartwatch and retrieve the data value for each sensor. The main data collected from the smartwatch for the Fall Detection application are the accelerometer sensor data.

The main java class responsible for detecting FALL / NO FALL is the prediction class. The prediction class reads in a trained model named 'fallsvmmodel.model' and for

the data values collected from the smartwatch, it checks against the trained model to predict FALL / NO FALL.

For every 0.25 seconds approximately, the accelerometer sensor value is captured, and the prediction for FALL /NO FALL is done and stored along with the sensor data value in the CSV file. This is called internal prediction. The final prediction is detected in a sliding window of 3 seconds. In that 3 seconds, if FALL are predicted in a sequence between 2 and 5, the final prediction is a FALL, otherwise, it is a NOT Fall.

5.2 Implementation of the Fall Detection application as composition of accessor

Section 4.3 describes how to implement a J2V8 Android Accessor Host. As we have already discussed, accessors are written in JavaScript. To implement an accessor-based fall detection version, a dataCollection, and Fall Detection accessors are developed and placed in the Web layer under the assets folder. Figure 5.2.1 represents accessor-based fall detection application.

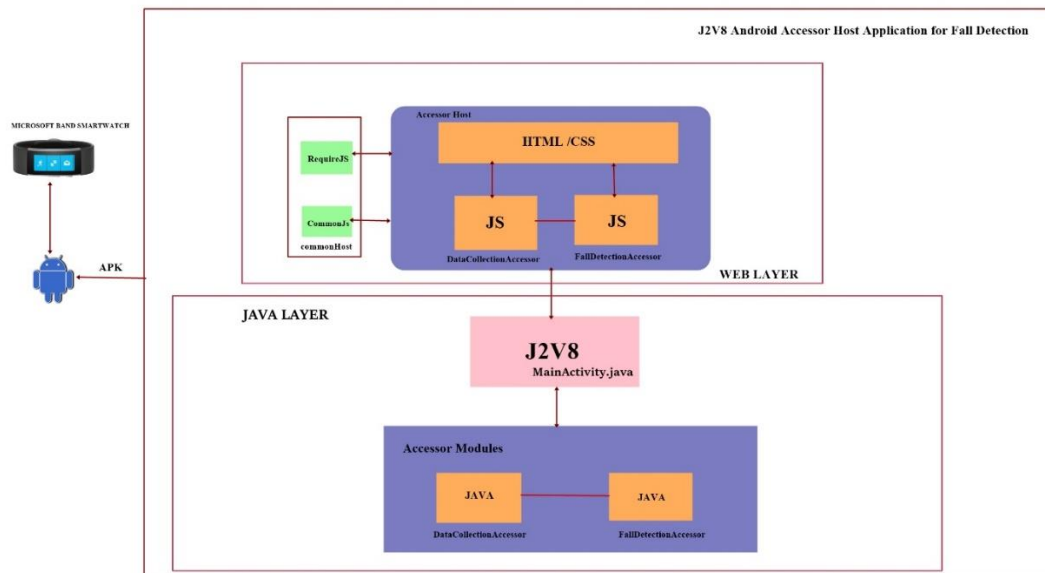


Fig 5.2.1 Architecture for accessor-based Fall Detection Application

The Fall Detection Application is composed of two different accessors which are the `DataCollectionAccessor` and the `FallDetectionAccessor`. These accessors are written in JavaScript hence having a JS extension. As shown in figure 5.2.1, both accessors are placed inside the Web layer. Both accessors have associated dependent modules. The associated module is basically a Java class that interacts with Microsoft SDK for the MS Band smartwatch. The name of the accessor's associated module should be the same as that of the accessor which calls it. In our case, the accessors are '`DataCollectionAccessor.js`' and '`FallDetectionAccessor.js`' therefore the name of the modules are '`DataCollectionAccessor.java`' and '`FallDetectionAccessor.java`'.

The '`DataCollectionAccessor`' collects the accelerometer sensor data from the smartwatch. These data's are then stored in the form of CSV file in the Android phone. The '`FallDetectionAccessor`' predicts the FALL or NO FALL. It fetches the accelerometer data from the CSV file and uses them to perform the prediction. Both the accessors are instantiated by the J2V8 Android Accessor Host.

Figure 4.2.1 shows the folder structure of the J2V8 Android Accessor Host. This folder structure will help us understand the placement of files in J2V8 Android accessor Host for fall detection.

The "`DataCollectionAccessor.js`" and "`FallDetectionAccessor.js`" should be placed under the assets folder and the associated modules, that is "`DataCollectionAccessor.java`" and "`FallDetectionAccessor.java`" should be placed in the Modules folder under `com.txstate.j2v8.J2V8AccessorHost` package shown in Figure 4.2.1. In addition, the `MainActivity.java` must contain the J2V8 script executor which

executes accessors from a Java program. This is the whole setup of J2V8 Android accessor Host for fall detection.

5.3 Evaluating performance of J2V8 Android Accessor Host (Our Approach)

We are going to evaluate the performance of our J2V8 Accessor Host by comparing an accessor-based IoT application verses a Java-based IoT application for fall detection. We will compare two important efficiency aspects. One is Power or battery consumption and another is memory usage.

We will start with the battery consumption test. We are going to run the two versions of Fall Detection Application in Nexus 5s Android phone. Nexus 5s Phone is used to test the power consumption aspects for both versions of Fall Detection Android application.

Before comparing, the phone is fully charged to 100% of battery. We install both versions of Fall Detection Application on the phone. First, the native Android application in java for Fall Detection is made to run in the phone continuously for 15 hours. The battery status is captured every hour. A graph is generated for time versus battery percentage.

Then phone is then fully charged again and the accessor-based Fall Detection is made to run in the phone for 15 hours. Again, for every hour, the battery percentage is captured and a time versus battery percentage graph is generated.

The above experiment is repeated 10 times and an average value of the time vs battery percentage graph is created. The following graph represents average battery

consumption of Native Android application in java for Fall Detection and J2V8 Android accessor Host application for Fall Detection.

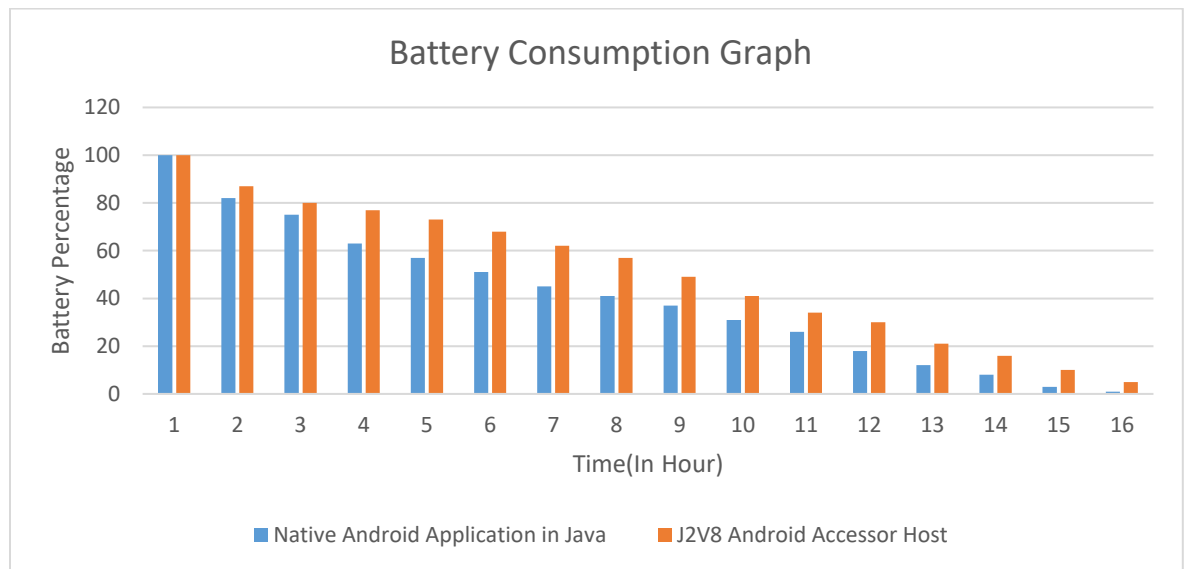


Fig 5.3.1 Battery Consumption Graph

Next, we test the memory usage of the both versions of Fall Detection application. First the native Android application in Java for Fall Detection is made to run and for every hour and the memory usage is noted in MB. This is done for about 7 hours. The experiment is repeated for about 10 times and the average memory usage is noted. The same experiment is performed by running the J2V8 Android accessor Host for Fall Detection and the average memory usage is noted.

The following graph represents the average memory usage of native Android application in Java for Fall Detection and the J2V8 Android accessor Host application for Fall Detection.

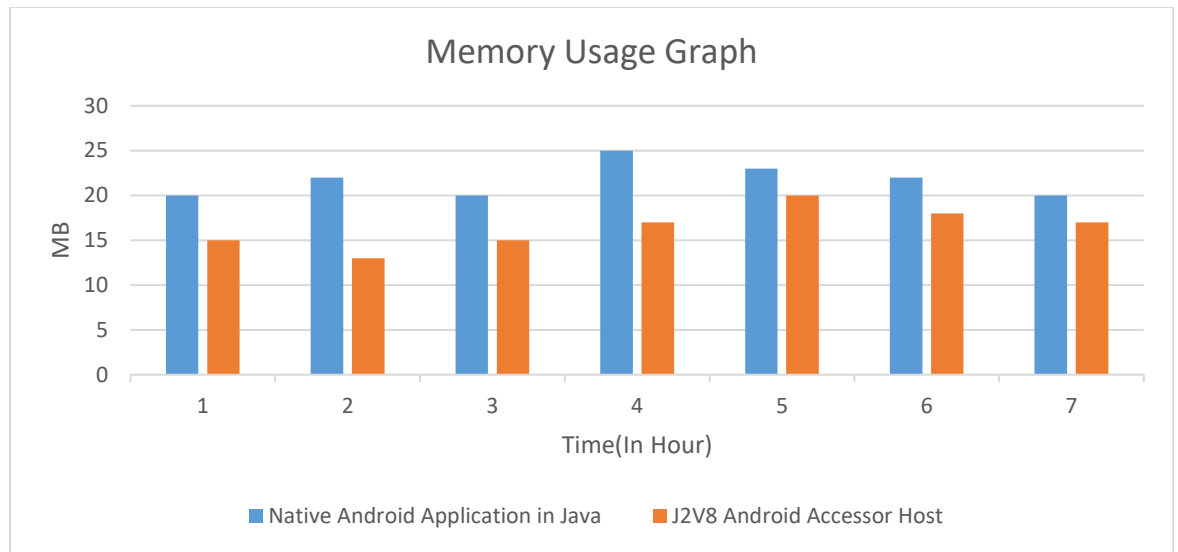
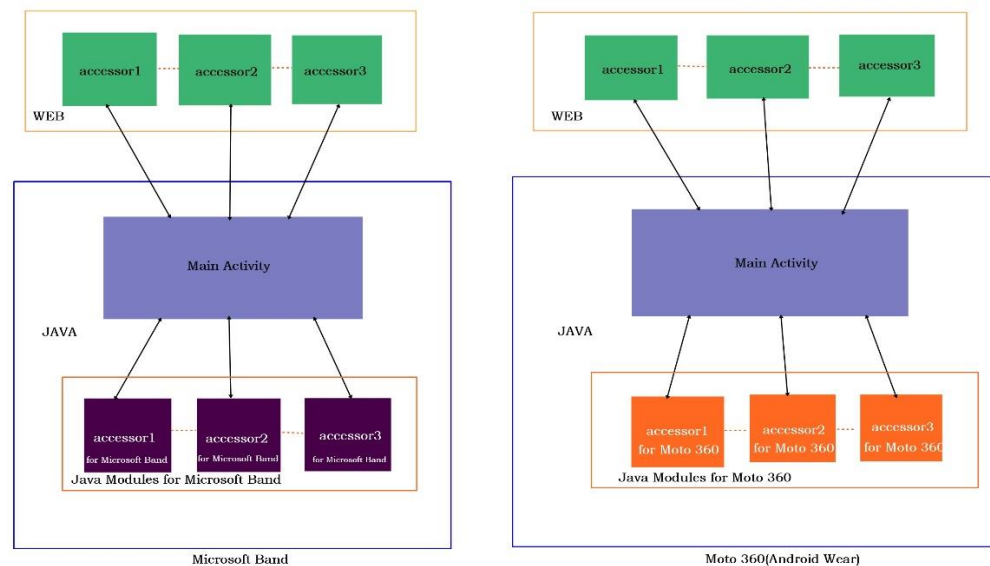


Fig 5.3.2 Memory usage Graph

Figure 5.3.1 and Figure 5.3.2 shows that J2V8 Android accessor Host is more efficient than Android's Java virtual machine in terms of both battery consumption and memory usage.

5.4 Heterogeneity of J2V8 Android accessor Host

The advantage of able to reuse accessors for different IoT devices using J2V8 Android accessor Host is also demonstrated with an experiment by using Moto360 smartwatch instead of Microsoft Band smartwatch. The DataCollectionAccessor which is used to collect sensor data's from Microsoft Band smartwatch can be reused as it is for collecting sensor data's from Moto360 with no additional programming. The data collection accessor knows which library module is required upon the detection of the type of smartwatch and the accessor host automatically loads that library for collecting data. This experiment also demonstrates the idea of write once and run everywhere concept of accessor. The diagram below shows where all the changes are required if the IoT device is changes to that of Moto360.



5.4.1 Comparing Microsoft Band with Moto 360

In the above diagram, the only change needed if Microsoft Band is replaced with Moto360 (Android Wear) is in the Java Module Part. The Java Module for Moto360 will

differ from that of Microsoft Band as the way the sensors are accessed in both the smart watched are different. Every other part for the J2V8 Android accessor host will remain same including the accessors.

6. RELATED WORK

There are many approaches in the literature for executing JavaScript in Java program. We will discuss why they cannot be used as the Android accessor host.

Nashorn [21] is a JavaScript runtime executor in Java. Nashorn JavaScript engine is part of Java SE 8 which extends Java's capabilities by running dynamic JavaScript code natively on the JVM. Nashorn is not available for Android hence it cannot be used to build Android accessor host.

Rhino [22], which is also a JavaScript runtime executor written fully in Java and managed by Mozilla Foundation. Before J2V8, Rhino was widely used. . Rhino compiles JavaScript into Java Classes. But compiled Rhino is not available for Android. Only non-optimized Rhino is available for Android thus making running JavaScript on Android inefficient.

Tabris [16] is another framework that uses J2V8 scripting engine. Tabris lets programmers develop Android and IOS applications from a code base written entirely in JavaScript. We can use existing JavaScript libraries and native extensions to extend the core functionalities of Tabris. Tabris can execute JavaScript within Java classes.

However, most of today's mobile Apps are developed using native codes and hybrid technologies such as Phone Gap, Cordova, and NoFlo etc. If we just want to combine native codes with web technologies using J2V8, Tabris is a good choice because you can achieve great performance with a great look & feel.

To use Tabris on a mobile device, the device has to download an app as mentioned earlier which runs the Tabris Application. Tabris is still a developing

framework and in the future maybe Tabris will be a good framework for developing an accessor host for Android. At present, it is not transparent and to run Tabris as a standalone application in Android it has to be wrapped around Cordova which adds another layer of indirection and complexity.

NoFlo [17] is a Flow-Based Programming environment for JavaScript. In flow-based programs, the logic of your software is defined as a graph. The nodes of the graph are instances of NoFlo components, and the edges define the connections between them. The NoFlo Components is comparable to accessors or actors.

NoFlo components react to incoming messages, or information packets. When a component receives packets in its input ports it performs a predefined operation, and sends its result out as a packet to its output ports. There is no shared state, and the only way to communicate between components is by sending packets.

NoFlo components are implemented in simple JavaScript or Coffee Script objects that define the input and output ports, and register various event listeners on them. When executed, NoFlo creates a live graph, or a network of a graph definition, instantiates the components used in the graph, and connects them together. NoFlo is still an experimental system. NoFlo for Android is still under development.

In this thesis, we choose J2V8 over all the other scripting engines described above since J2V8 is fastest as it compiles all JavaScript to machine code directly and it is available on Android platform.

7. CONCLUSION AND FUTURE WORK

We have presented a new Android IoT middleware called J2V8 accessor Host. This middleware implements the standard specification of an accessor host by leveraging J2V8's script engine and implementing the common functions specified in Common Host. The J2V8 script engine is one of the best scripting engines for executing JavaScript in Java. Our middleware gives support to both the Java technologies and Web technologies and thus making it easier for the user to develop an IoT application that utilities hybrid technologies.

As compared to Cordova Host, an existing Android accessor host, our J2V8 Host is the first Android accessor host that works well with both the Java and the Web technology.

In the future, we plan to add node.js in J2V8 Accessor Host as the present J2V8 Android Accessor Host does not support Node.js. The latest version of J2V8 script engine which is still in beta test stage has the capability to execute Node.js modules. With Node.js included, the implementation of J2V8 Android accessor Host can be enhanced. For example, the API such as requireJS and CommonJS which are used to create a common Host can be dropped and we would be able to inherit functionalities of the Common Host directly with node.js.

Finally, we hope to use the same procedure used for developing our Android accessor host for developing an IOS accessor host.

APPENDIX

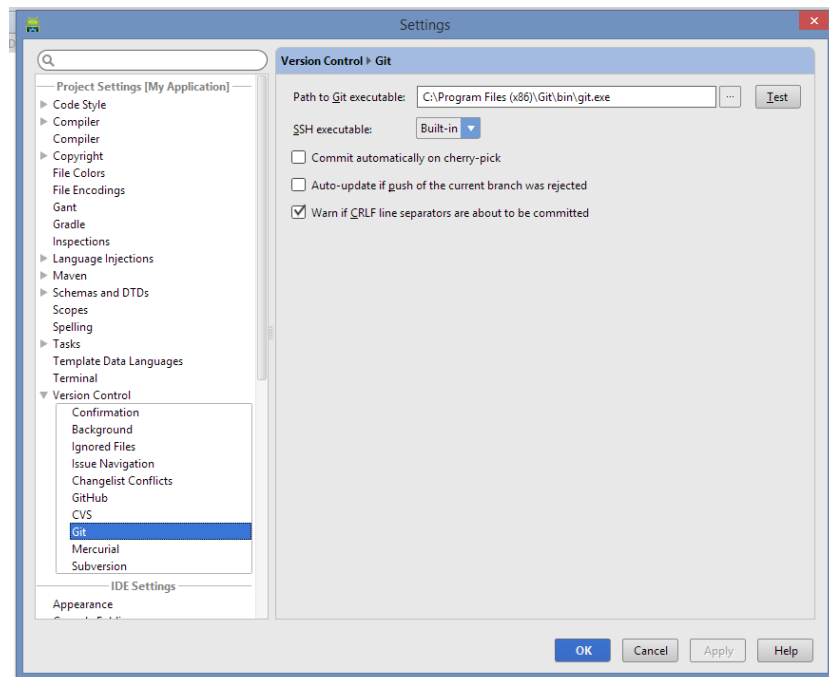
Perform the following steps to setup a J2V8 Android Accessor Host Application.

Step 1: Install Git for Windows

It can be downloaded for free from <https://git-scm.com/downloads>. Most settings available during the installation process should be compatible with Android studio. Just choose the settings you deem the most appropriate.

Step 2: Link Git executable to Android Studio

Open Android Studio and go to Settings. In the Setting dialog open the page Version Control / Git. Here defines the path to the Git executable you have just installed.



Step 3: Get the Path to your Repository from GitHub

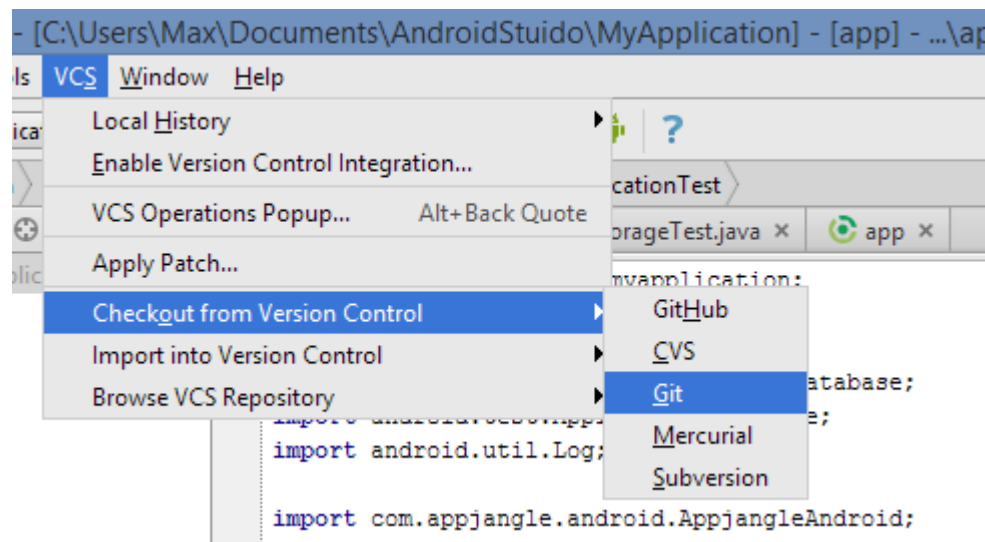
HTTPS Git Repository link for the project:

<https://git.txstate.edu/vkm4/J2V8AccessorHost.git>

Use the above Link as a Git repository Link for the upcoming steps.

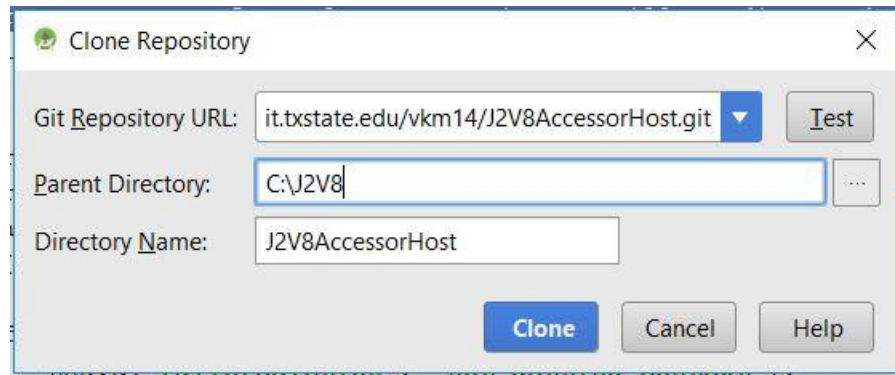
Step 4: Import the Git project to Android Studio

Go to Android Studio and go to Menu / VCS / Checkout from Version Control / Git

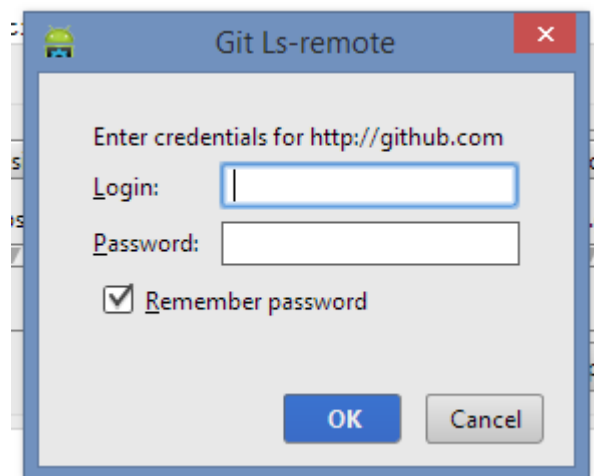


Paste the HTTPS path you obtained from GitHub in the previous step. The Parent Directory is the location where the Project will be placed in your local system. You can provide any location as per your convenience. In our case we have placed in 'C:/J2V8'. This directory location will act as a workspace for your project.

Select an appropriate Directory Name. We have provided 'J2V8AccessotHost' as a directory name. After filling all the required fields click [Clone].



The next step will show a popup window which asks for your GitHub username and password. Enter the credential and click [OK]



Now the project should be imported to Android Studio and you should be able to commit and push future changes back to GitHub if you're provided with Access.

The Project comes with three default Accessors. They are as follows:

- 1- Data Collection Accessor
- 2- Fall Detection Accessor

- 3- Alert Accessor
- 4- TestAccessor
- 5- TestCompositeAccessor

Test J2V8AccessorHost

To test whether J2V8AccessorHost is working properly or not, Import the project in your Android Studio. Once done in the MainActivity.java file which is under the package com.txstate.j2v8.J2V8AccessorHost find the function “ReadRawJS” and connect TestAccessor as follows.

```
input = assetManager.open("lib/TestAccessor.js");
```

Now run the Application in an emulator or Phone. If the J2V8AccessorHost is imported properly with desired android version, then the TestAccessor will be executed successfully and the output will be display in the phone/emulator screen.

The minimum requirement to run these Accessors are:

- 1- Android Development IDE (Android Studio Recommended with version 6.0.0 and above)
- 2- Microsoft MS Band Smartwatch version 1.3.20307.2.
- 3- Android Smartphone. (Android Version 6 and above)

How to Connect Smartwatch with Android Phone over Bluetooth

The Smartwatch works together with your phone, so you'll need to connect them.

1. Turn on your watch.
2. On your phone, Open the Bluetooth setting and search for smartwatch name.
3. When you see your watch's name, tap it.
4. You'll see a code on your phone and watch.
 - **If the codes are the same:** On your phone, tap **Pair**. Pairing can take a few minutes, so be patient.
 - **If the codes are different:** Restart your watch and try again.
5. To finish set up, follow the onscreen instructions. Make sure the box for notifications to Smartwatch is checked.

Note: When your phone and watch are paired, you'll see "Connected" on your phone Bluetooth device display.

REFERENCES

- [1] K. Aberer, M. Hauswirth, and A. Salehi, "A middleware for fast and flexible sensor network deployment," in Proceedings of the 32Nd International Conference on Very Large Data Bases, ser. VLDB '06. VLDB Endowment, 2006, pp. 1199–1202. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1182635.1164243>
- [2] E. Latronico, E. Lee, M. Lohstroh, C. Shaver, A. Wasicek, and M. Weber, "A vision of swarmlets," Internet Computing, IEEE, vol. 19, no. 2, pp. 20–28, Mar 2015.
- [3] LEE, E. A. ET AL. Actor-oriented design of embedded hardware and software systems. Journal of Circuits, Systems, and Computers 12, 3 (2003).
- [4] 8. C. Hewitt. Viewing control structures as patterns of passing messages. Journal of Artificial Intelligence, 8(3):323–363, 1977.
- [5] 20. A. Sangiovanni-Vincentelli and G. Martin. Platform-based design and software design methodology for embedded systems. IEEE Design & Test of Computers, 18(6):23–33, 2001.
- [6] D. Raggett, "The Web of Things: Challenges and Opportunities," IEEE Computer, vol. 48, no. 5, pp. 26–32, May 2015.
- [7] R. Want, B. N. Schilit, and S. Jenson, "Enabling the Internet of Things," IEEE Computer, vol. 48, no. 1, pp. 28–35, 2015.
- [8] The TerraSwarm Research Center. Retrieved 2017, <https://terraswarm.org>.
- [9] 1. Xiaolin Lu "An investigation on service-oriented architecture for constructing distributed Web GIS application" ,Conf. Rec. 2005 IEEE Int. Conf. Services Computing, vol. 3 pp. 191-197.

[10] iCyPhy . Retrieved 2017, <https://www.icyphy.org/accessors/doc/jsdoc/module-browser.html>

[11] Industrial Cyber-Physical Systems Center (iCyPhy) . Retrieved 2017, <https://www.icyphy.org/accessors/wiki/Main/CapeCodeHost>

[12] Apache Cordova . Retrieved 2017, <https://cordova.apache.org/docs/en/latest/guide/support/index.html>

[13] Android Studio. Retrieved 2017, <https://developer.android.com/training/basics/firstapp/creating-project.html>

[14]Wikipedia . Retrieved 2017, [https://en.wikipedia.org/wiki/Weka_\(machine_learning\)](https://en.wikipedia.org/wiki/Weka_(machine_learning))

[15] Ian H. Witten; Eibe Frank; Mark A. Hall (2011). "[Data Mining: Practical machine learning tools and techniques](#), 3rd Edition". Morgan Kaufmann, San Francisco.
Retrieved 2011-01-19.

[16] Tabris.js . Retrieved May 8th,2017, <https://tabrisjs.com/documentation/latest/>

[17] NoFlo . Retrieved May 8th ,2017, <https://noflojs.org/documentation/>

[18] Browser Host._ Retrieved May 8th, 2017, <https://www.icyphy.org/accessors/doc/jsdoc/module-browser.html>

[19] Accessors,. Retrieved May 10th ,2017, <https://www.icyphy.org/accessors>

[20] iCyPhy. Retrieved May 10th , 2017, <https://www.icyphy.org/accessors/wiki/Main/NodeHost>

[21]Nashorn Wiki. Retrieved May 12th 2017,
[https://en.wikipedia.org/wiki/Nashorn_\(JavaScript_engine\)](https://en.wikipedia.org/wiki/Nashorn_(JavaScript_engine))

[22] Rhino Wiki. Retrieved May 12th, 2017 ,
[https://en.wikipedia.org/wiki/Rhino_\(JavaScript_engine\)](https://en.wikipedia.org/wiki/Rhino_(JavaScript_engine))

[23] Cape Code Setup. Retrieved May 12th ,2017,
[https://www.icyphy.org/accessors/wiki/Main/CapeCodeHost\]](https://www.icyphy.org/accessors/wiki/Main/CapeCodeHost)

[24] A vision of Swarmlets, Revision 2, Oct 25,204, Elizabeth Latronico, Edward A. Lee,
Marten Lohstroh, Chris Shaver, Armin Wasicek, Matthew Weber

[25]Ptolemy. Retrieved May 22, 2017, <http://ptolemy.eecs.berkeley.edu/ptolemyII/>