EVALUATION OF CORDOVA ACCESSOR HOST FOR RAPID DEVELOPMENT OF IOT APPLICATIONS ON MOBILE EDGE DEVICES

by

Jesuloluwa S. Eyitayo, B. Eng

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
May 2020

Committee Members:

Anne Ngu, Chair

Vangelis Metsis

Guowei Yang

FAIR USE AND AUTHOR'S PERMISSION STATEMENT

Fair Use

Duplication Permission

DEDICATION

This project is dedicated to my parents who have made it possible for me to pursue my master's degree in computer science and to my siblings and close friends who supported me throughout my graduate studies.

Finally, I would like to dedicate this project to God for granting me wisdom, knowledge and understanding to complete my graduate studies with a thesis.

ACKNOWLEDGEMENTS

I would like to first thank my thesis advisor Dr. Anne Ngu for providing me the opportunity to work with her on this thesis, for her continuous guidance and endless support during my studies both as a professor and an advisor.

Thanks to Dr. Vangelis Metsis and Dr. Guowei Yang for their time and inputs into this thesis. Also, thanks to other faculty and staff members of the Department of Computer Science and the College of Engineering for their direct and indirect support and for creating an excellent learning and research environment through outstanding teaching, learning, and research facilities. I am especially grateful for the graduate instructional/research assistantship provided to me during my master's program by the Computer Science department.

I wish to appreciate my colleagues that also contributed their time and expertise to making this thesis possible.

I sincere appreciate my parents, Mr. Samuel Eyitayo and Mrs. Adejoke Eyitayo; and my siblings, Jesuloba, Jesulayomi and Jesulonimi; and also my close friends for their constant love and support throughout my years of study and through the process of writing this thesis. This accomplishment would not have been possible without them. Thank you all.

TABLE OF CONTENTS

Page

ACKNOWLEDGEMENTS v

LIST OF TABLES viii

LIST OF FIGURES ix

LIST OF ABBREVIATIONS x

ABSTRACT xi

CHAPTER

1. INTRODUCTION 1

2. RELATED IOT MIDDLEWARE SERVICE FRAMEWORKS 5

3. ACCESSOR DESIGN PATTERN AND ACCESSOR HOST 9

    3.1. Overview 9
    3.2. Accessor Design Pattern 10
    3.3. Accessor Hosts 13
        3.3.1. The Common Host 14
        3.3.2. The Browser Host 16
        3.3.3. Node Host 16
        3.3.4. CapeCode Host 16

4. CORDOVA ACCESSOR HOST 17

    4.1. Cordova Host 17
        4.1.1. Installation of the Apache Cordova 18
        4.1.2. Cordova Plugins 19
        4.1.3. Building a Cordova Plugin 20
    4.2. Architecture of Cordova Accessor Host 21

5. EXPERIMENT 23

5.1. Overview 23
5.2. The Setup of Cordova Accessor Host 23
5.3. Accessors of the Fall Detection Application 24
5.3.1. Data Collection Accessor 26
5.3.2. Data Prediction Accessor 26
5.3.3. Alert Accessor 26
5.3.4. Display Accessor 27
5.4. Accessors of the Heart Rate Monitoring Application 27
5.4.1. Data Collection Accessor 28
5.4.2. Data Processing Accessor 28
5.4.3. Alert Accessor 28
5.4.4. Display Accessor 29
5.5. Reusability of Accessors 29
5.5.1. Data Collection Accessor 30
5.5.2. Data Prediction Accessor 31
5.5.3. Display Accessor 33
5.5.4. Alert Accessor 33
5.6. Portability of Devices When Using Cordova Accessor Host 34
5.7. Evaluation of Battery Performance 37
5.7.1. Battery Performance Setup 37
5.7.2. Observations 38

6. CONCLUSION AND FUTURE WORK 40

APPENDIX SECTION 42

REFERENCES 49

LIST OF TABLES

Table **Page**

1. Display of the Smartwatches and their Operating System..... 34

2. Display of the Smartphones and their Operating System 34

3. Average Battery Usage for Native (Java) App 39

4. Average Battery Usage for Cordova Accessor Host App..... 39

LIST OF FIGURES

Figure **Page**

1. Cloud-based IoT Middleware [2] 10

2. Design Pattern of Accessors taken from [23] 12

3. Hello World Accessor 13

4. Apache Cordova Architecture 18

5. File Structure of the Cordova Plugin for an Android App 20

6. Cordova Accessor Host Architecture 22

7. SmartFall App User Interface 25

8. Main Accessors Used for the Fall Detection App 27

9. Main Accessors Used for the Heart Rate Monitor App 29

10. Swarmlet.js File Architecture 30

11. Data Collection Accessor Architecture 31

12. Data Prediction Accessor Architecture for the Fall Detection App 32

13. Data Processing Accessor Architecture for the Heart Rate Monitoring App 32

14. Alert Accessor Architecture 33

15. Fall detection App to Receive Sensor Data from MSBAND Smartwatch 35

16. Fall detection App to Receive Sensor Data from WEAROS Smartwatch 36

17. Graph Showing Smartphone Battery usage of Accessor-based App Versus Native App 39

LIST OF ABBREVIATIONS

Abbreviation	Description
IoT	Internet of Things
MQTT	Message Oriented Telemetry Transport
CoAP	Constrained Application Protocol
BLE	Bluetooth Low Energy
GSN	Global Sensor Network
REST	Representational State Transfer
iCyPhy	Industrial Cyber-Physical Systems
AAC	Asynchronous Atomic Callbacks
RPC	Remote Procedure Call
MSBAND	Microsoft Band
npm	Node Package Manager
CSS	Cascade Style Sheet
APIs	Application Programming Interfaces
App	Application
bpm	Beat per minute

ABSTRACT

The Internet of Things (IoT) middleware service provides the ability for human and computers to learn and interact from billions of things that include sensors, actuators, applications, and other Internet connected objects. The realization of an edge-based IoT service framework will enable seamless integration of the Cyber-world with new physical devices and will fundamentally change and empower human interaction with the world. While there are many cloud-based IoT service frameworks, many health-care related IoT applications such as real-time fall detection systems cannot utilize cloud-based framework due to latency, privacy, and security concerns.

In this thesis, we first present an open source plug and play IoT middleware called Cordova Accessor host and we discuss the accessor design pattern, Apache Cordova and the accessor hosts for bridging the heterogeneity among IoT devices and allowing for smarter interactions, sharing and portability. We discuss the Cordova Accessor host, an edge-based IoT middleware service, and a thorough analysis of the opportunities and challenges in the implementation of a fall detection application as components of accessors that embrace the heterogeneity of IoT devices and supports the composition and adaptability of IoT services. We demonstrate the reusability of accessors by building a heart rate monitoring IoT application. Finally, we shown that IoT services deployed on Android compatible devices using this framework consume about 35% less battery power than the same IoT services implemented in native Java language.

1. INTRODUCTION

The Internet of Things (IoT) paradigm is a domain that enables the interconnectedness among devices, anytime, anywhere on the planet [1]. IoT provides the ability for humans and computers to learn and interact with billions of things including services, actuators, sensors and objects connected to the Internet [2]. The IoT is known to be the next logical technological evolution since the Internet, providing extensive services in smart manufacturing, smart grids, security, healthcare, automotive engineering, education and consumer electronics. IoT provides the capability for us to build smart cities and smart homes that are safe and energy efficient where parking spaces, traffic congestion, urban noise, irrigation can be monitored and managed in real time with low latency effectively [2].

The opportunities and future prospects of IoT are numerous and exciting but it can be challenging to seamlessly integrate the physical thing with the cyber world. Practical Issues like the determination of device proximity, dramatic increase in network scale, disparate connectivity protocols and IoT programming models must be addressed. For example, Message Oriented Telemetry Transport (MQTT), Constrained Application Protocol (CoAP) and BLE (Bluetooth Low Energy) are popular connectivity protocols designed specifically for IoT devices. However, the plethora of IoT connectivity protocols and middleware are not facilitating the ease of connecting IoT devices and interpreting collected data from them. This is compounded by the fact that each IoT middleware advocates different programming abstraction and architecture for accessing and connecting to IoT devices. Another example is the Global Sensor Network (GSN) project [3], 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 Node-RED project at IBM [4], a node is proposed as the main abstraction. In the TerraSwarm project [5], an accessor design pattern implemented in JavaScript is proposed as the main abstraction. In the Google Fit project [6], no particular high-level abstraction is provided for encapsulating a new device type. The system is pre-programmed to support a fixed set of IoT devices, which can be accessed by Representational State Transfer (REST) APIs [7]. As a result of this, the addition of an IoT device which is not already supported requires the experience of Java Programming in order to extend the Google Fit's FitnessSensorService class. In addition, data are collected and stored solely in the cloud in Google Fit which might not be acceptable for privacy conscious consumers.

The current state-of-the-art support for IoT service development is application specific which is equivalent to the scenario where every IoT device requires a different web browser for connection to the Internet as echoed by Zachariah et al. in the paper entitled "The Internet of Things Has a Gateway Problem"[8]. Therefore, there is a demand for the urgency in launching an IoT middleware service like that of the Web frameworks such as Laravel, ASP.NET, Django and Express.js. These frameworks coupled with the launch of the mobile phone operating systems (Android and IOS) have brought disruptive applications such as Airbnb and Uber and have therefore transformed and improved how business is being conducted. As a result, we believe that the launch of an edge IoT service middleware that can be deployed within the constrained physical IoT devices will produce incredible and a wide range of IoT services that have impacts beyond our imagination and fundamentally change and empower human interaction with

the physical world.

The main focus of this thesis is to present and implement the open source Cordova Accessor host, an edge-based IoT service middleware, which is built on Apache Cordova tools —a cross platform design tool— that utilizes its plugins for integration with the accessor design pattern [5] for the rapid prototyping of IoT services on edge devices. While an experimental Cordova Accessor host has been developed and listed on TerraSwarm's accessor project website [9], but there has never been a development of a real-world IoT service/application using that experimental host. There is no empirical evaluation of the benefit of using Cordova Accessor host for IoT services development in terms of re-usability of accessors, reduction of programming and deployment barriers and conservation of battery power of the edge devices.

First, we will demonstrate in this paper how the Cordova Accessor host is used for the development and deployment of a real-world fall detection IoT application though the composition of accessors running on a commodity smartwatch that is paired with a smartphone. We will further demonstrate the reusability of accessors we have previously developed for fall detection application for composing a different IoT service that can perform real-time heart rate monitoring. Based on these demonstrations, we will show that the stream of accelerometer data used for fall detection application and the heart rate data used for the heart rate monitor can be collected from multiple smartwatch vendors with minimal programming by isolating device-specific communication codes from application codes.

The main contributions of this thesis are:

- Analysis of the advantage of an edge IoT service middleware based on Apache Cordova platform for the development of real-world IoT applications.
- Evaluating the effectiveness of Cordova Accessor host for rapid development of IoT services by composing two different IoT applications using accessors as the basic components and measures the reusability and code changes required.
- Demonstrating how accessor design pattern can facilitate sensor data collection across multi-vendors IoT devices (three different types of smartwatches from Google, Microsoft and Huawei) with minimal additional programming.
- Demonstrating the energy efficiency of accessor- based IoT services as compared with native implementation.

The remainder of this thesis is organized as follows:

First, we present the related IoT middleware service in Chapter 2, then the background work in accessors design pattern and accessor hosts will be discussed in Chapter 3. We then discuss architecture of an edge-based IoT service framework, Cordova Accessor host, and its capabilities in Chapter 4. In Chapter 5, we present our analysis on two IoT services developed and deployed on Cordova Accessor host and document the reusability of accessors and reduction in deployment barriers across heterogeneous IoT devices. In the same Chapter, we present the performance study in battery power consumption of IoT services implemented on Cordova Accessor host verses one on native Android environment. Finally, in Chapter 6 we present our conclusion and future work.

2. RELATED IOT MIDDLEWARE SERVICE FRAMEWORKS

We explored three classical IoT frameworks: Service-oriented, Cloud based, and Actor oriented in search of a framework that can allow seamless integration of heterogeneous IoT devices from multiple vendors to build real-time IoT applications with local data storage and analysis and without dictating a particular communication protocol. The availability of local storage is important to avoid the unpredictable latency from wireless transmission of data to the cloud or server for analytic. In addition, to ensure that the user's privacy is not violated, users should have the option to archive data generated from their personal IoT devices in a secure local storage medium of their own choice.

The service-oriented framework that we explored was Global Sensor Network (GSN) in [3]. GSN aims to provide a uniform platform for flexible integration, sharing and deployment of heterogeneous IoT devices. The central concept is the virtual sensor abstraction, which enables developers to declaratively specify XML-based deployment descriptors to describe how to connect to a physical or virtual sensor. This is similar to the concept of deployment descriptors used in the deployment of enterprise Java beans in J2EE server [10]. The architecture of GSN follows the same container architecture as in J2EE where each container can host multiple virtual sensors. The container provides functionalities/capabilities for lifecycle management of the sensors including persistency, security, notification, resource pooling and event processing. GSN servers can fulfill our local data storage requirement, however, GSN is a heavy weight system to run on an edge device like a smartwatch or smartphone. Till date, there is no working edge-based GSN framework. Another service-oriented platform for IoT is presented in [11]. The main architecture is similar to GSN. The key contribution of this service framework is its

scalability and robust scheduling that have shown to support more than 1000 services. However, it can only be deployed on high-end servers or cloud.

We examined various cloud based frameworks such as AWS IoT from Amazon [12], Watson IoT from IBM [13], ThingSpeak IoT [14] and Google IoT Cloud [15] (e.g. GoogleFit). These cloud-based frameworks usually provide the following four fundamental services:

1. Web-based administrative console for managing device connection
2. Cloud-based data storage
3. Cloud-based analytic services and
4. Advanced reporting or visualization

We also examined Google's GoogleFit [6] cloud service in details for IoT application development. GoogleFit provides a set of Application Programming Interfaces (APIs) for connecting third-party IoT devices to its cloud storage. For example, it provides APIs for subscribing to a particular fitness data type or a particular fitness source (e.g., Fitbit or Samsung Smartwatch) and APIs for querying of historical data or persistent recording of the sensor data from a particular source (e.g., a smartwatch). GoogleFit is not appropriate because the user is tied to storing his/her sensor data in GoogleFit's cloud storage, in the format dictated by GoogleFit and in the size limit enforced by GoogleFit. It is not possible to get access to the collected raw data and pre-process them for analysis and visualization purposes, which is a critical component for many IoT applications. Moreover, GoogleFit requires all collected data to be stored remotely in the Google cloud. GoogleFit is not suitable for IoT services that must be performed quickly in real-time on the edge devices.

The third framework that we investigated is the actor based IoT middleware from the TerraSwarm project [16]. The advantage of an actor-based framework is that it is light-weight and portable for capability and energy constrained IoT devices. The actor-based framework (accessor host) was first presented in the paper entitled "A Vision of Swarmlets" by Latronico, Lee, Lohstoh, Shaver, Wasicek, and Weber at University of California, Berkeley [5]. As stated in the accessor homepage[17]:

"Accessors are a technology for making the Internet of Things accessible to a broader community of citizens, inventors, and service providers through open interfaces, an open community of developers, and an open repository of technology. Developed by the TerraSwarm Research Center, accessors enable composing heterogeneous devices and services in the Internet of Things (IoT)".

An accessor is designed with the actor model of computation that embraces concurrency, atomicity and asynchrony. In other words, an accessor can be viewed as an actor that wraps a sensor, actuator, or a service and hide the different implementations from developers. An accessor host is a service or application running on the client platform that can provide execution environment for accessors. The client platform can be a server (e.g. a high-end desktop computer), a gateway (e.g. smartphone) or an edge device (e.g. a wearable device). In the context of TerraSwarm project, an accessor host is also known as a Swarmlet host.

In iCyPhy (Industrial Cyber-Physical Systems) project, a sequel to TerraSwarm project, a semantic accessor framework is proposed [18]. The framework is an attempt to combine Semantic Web technology with accessor to create a platform that can dynamically discover and instantiate context-relevant accessors for dynamic real-time

IoT service provisioning such as the connected cars applications. However, this semantic framework cannot be deployed on the edge.

In the realm of local data storage, GSN servers and accessor hosts are both able to store data locally. However, the final choice of using the actor-based framework with accessor and accessor host came from the fact that it gave us the flexibility to use IoT devices from multiple vendors without dictating using a specific standard and it is very light-weight. We describe the functionalities of accessors and accessor hosts in greater details in the following chapter.

3. ACCESSOR DESIGN PATTERN AND ACCESSOR HOST

3.1. Overview

An IoT middleware framework typically comprises of a three layers architecture (edge, gateway, cloud) [2]. The cloud architecture [19] includes the connection of mobile clients such as mobile phones to a powerful centralized cloud service that runs remotely. The cloud service provides easy management of information gotten from the clients as shown in Figure 1. It also provides frequent backup of all data, enforced privacy and physical security. The modern clients used in this cloud architecture are both capable and flexible to connect the cloud services to users as they now contain powerful processors and utilize a high-performance multicore technology. The gateway architecture further provides the management of connections among edge devices and the cloud [20]. The traditional IoT systems consist of edge devices that consists of sensors, actuators and a gateway connected either through wired or wireless Internet connection to the Cloud services. The gateways are highly powered hardware devices with advanced computing power and storage space which can collect and process data from the smart edge device before sending it to the cloud for more computationally intensive tasks [21]. With the aim of addressing the needs of IoT in terms of managing the huge amount of data and reducing processing latency, edge devices are now equipped with more sophisticated processors and smarter applications to take over certain roles from the cloud. This is referred to as edge computing which aims at pushing the computations performed from the cloud towards the edge, with the aim of avoiding bottlenecks and reducing latency. Developing an edge-based IoT middleware framework is essential for the success of edge computing.

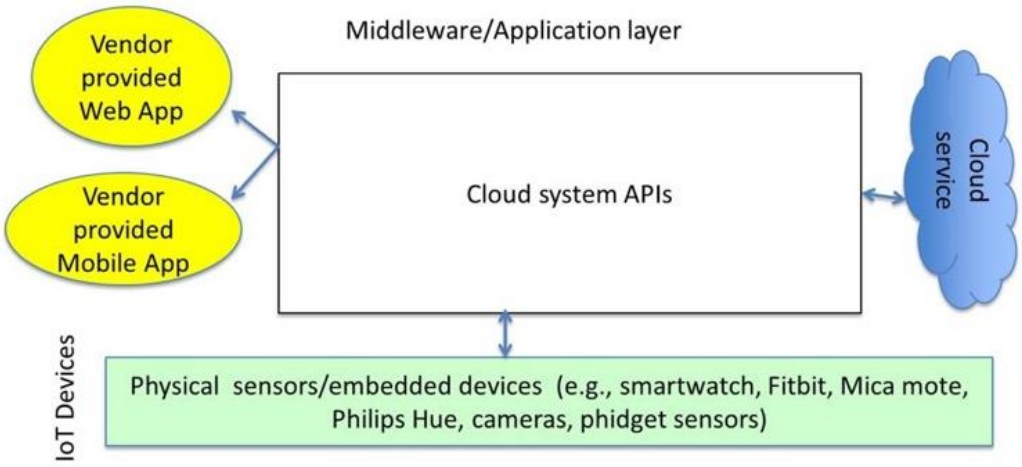


Figure 1. Cloud-based IoT Middleware [2]

The term middleware and service framework are used interchangeably in this thesis. Middleware usually refers to a software system designed to be the intermediary between physical IoT devices and IoT applications. TerraSwarm's IoT middleware focuses on the open, plug and play component architecture [22]. A unique feature of this middleware is a lightweight software system or host with standardized capabilities for running and deploying IoT services in any layer of an IoT three layers architectural system. The main concept proposed in such a framework for seamless interaction with an IoT device is accessor.

3.2. Accessor Design Pattern

Accessors are defined using a lightweight JavaScript programming model consisting of input and output events as well as a set of functions. Accessors provide the abstraction for smart "Things" across different hardware or software platforms to interact, bridging the heterogeneity among IoT systems and allowing for smarter interactions,

sharing and portability. The JavaScript programming model of accessors enable accessors to be ubiquitous and allow Things to communicate and share information in a message-oriented fashion. Figure 2 shows an accessor design pattern. The horizontal traversal of the design governs the interactions among assessors using ports while the vertical traversal governs the asynchronous interaction with the other physical or logical devices on the edge, on a local server or on the cloud. As shown in Figure 2, the lower box represents an IoT device or an external service on the cloud. This means that an accessor can send request to the lower box and receive a response from it. An Asynchronous Atomic Callbacks (AAC) pattern is used for the request. AAC is a non-blocking protocol and enables many concurrent pending requests to be active at once without having the overhead of managing threads. AAC invocation is atomic as contrasted to interrupt-driven threads or Remote Procedure Call (RPC). It does not use locks and thus cannot deadlock. The accessor design pattern isolates the device specific communication protocol from the IoT application and enables composition of a complex IoT service from multi-vendors IoT devices. Each accessor is defined by an interface with a number of input and output ports for managing and processing the data transfer between “Things”. Ports provide a common paradigm of communication independence from the low-level device communication protocols. These ports also connect accessors together to provide a complex service as well as enable the ease in the exchange, deletion or addition of capabilities to the service. The JavaScript programming model of accessors also enables accessors to essentially act like web pages on a browser, exchanging information with a variety of other services compliance with the vertical contract as shown in Figure 2.

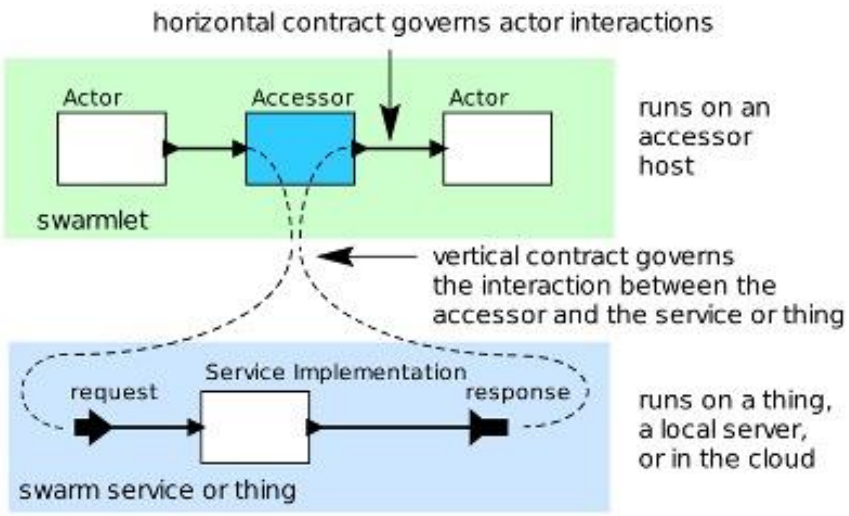


Figure 2. Design Pattern of Accessors taken from [23]

An accessor interface defines the modules that are required for the accessor host in order to execute the accessor. This means an accessor host can instantiate or execute an accessor by simply examining its interface. The setup function and the initialize function form the basic functions for all accessors. The code below and the block diagram in Figure 3 show the implementation of a Hello World accessor. This accessor accepts a username through the name input port and returns a greeting message as output on the output port of the accessor. The setup function defines the input and output port for the accessor while the Initialize function performs the required computation the accessor is designed for which in this case will be to concatenate Hello with the input username. Lastly, the initialize function sends the result as output for the next accessor. The snippet of this accessor is shown in the code below:

```
// Hello World Accessor
exports.setup = function () {
  this.input('name');
  this.output('greeting');
};

exports.initialize = function () {
  this.addInputHandler('name', function () {
     this.send('greeting', 'Hello World, ' + this.get('name'));
  });
};
```

Figure 3. Hello World Accessor

3.3. Accessor Hosts

An accessor host is a service running in the network or on a client platform that hosts applications built as a composition of accessors that stream data to each other. The host is like a browser for Things [17]. The TerraSwarm Research Center has provided several accessor hosts which are built on top of the Common host which contains the

basic functionalities that can be reused by other hosts. The following are accessor hosts provided by the TerraSwarm Research Center: Common host, Browser host, CapeCode host, Node host and Cordova host.

3.3.1. The Common Host

The common host is a platform-independent pure-JavaScript host that provides a constructor for instantiating an accessor. It includes the instantiateAccessor function which takes as an argument the qualified accessor class name. This class name is enough to instantiate the accessor. Another function getAccessorCode retrieves the accessor's source code when given the class name. The common host is considered the base host when defining other hosts. This means that a defined host inherits all the functions defined in the common host. Some of the common host functions inherited by other hosts are described below:

- **react():** This function reacts to the input provided to the accessor.
- **require():** This function loads the required library for the accessor to function correctly.
- **instantiateAccessor():** This function takes as an argument the fully qualified accessor class name and initialize it.
- **getAccessorCode():** This function retrieves the accessor's source code given the class name.

- **setParameter():** This function sets a parameter with parameter reference name and parameter value.
- **initialize():** This function initializes the accessor once by the host on startup.
- **fire():** This function performs a set action or output upon receiving the appropriate input signal.
- **setup():** This function provides the information to establish a preliminary accessor such as names of all input and output ports.
- **connect():** This function connects the input port of one accessor with the output port of the previous accessor in the pipeline.
- **latestOutput():** This function retrieves the latest output value produced in react().
- **send():** This function sends an output to another port,
- **setTimeout():** This function sets the specified function to execute at certain time
- **setInterval():** This function sets the specified function to execute after certain time interval, and repeat at interval,
- **wrapup():** This function releases used resources and terminate the accessor
- **provideInput(name, value):** This function provides an input value.

3.3.2. The Browser Host

The Browser host runs in web browsers and basically supports the inspection and execution of accessors within the browser environment. The Brower host is layered on top of the common host and loads common host's functions if required [24].

3.3.3. Node Host

The Node host is a host defined leveraging the Node.js engine with capability of the common host. This host requires the installation of the Node Package Manager (npm) [25]. The Node Package Manager is an open-source online repository which has node.js based projects published. The npm provides easy packaging of project packages that promotes sharing and reusability across the developer's community. The Node host is defined as an extension of the 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 [26].

3.3.4. CapeCode Host

The CapeCode host is an interactive graphical editor for creating, composing and executing accessors. It makes use of Java Nashorn script engine for executing accessors. The CapeCode host provides a block diagram editor for prototyping IoT applications and code generators for deploying those applications on other accessor hosts [27].

4. CORDOVA ACCESSOR HOST

4.1. Cordova Host

Cordova host is based on Apache Cordova's cross mobile program development platform that is used for building applications using HTML, CSS and JavaScript [28] in one code base and targeted to multiple platforms such as Android, IOS, Window and Browser with no additional programming. The JavaScript interface in Cordova interacts with native languages and APIs of physical or virtual IoT devices through a number of plugins. In essence, the plugin hides the various native code implementations behind a common JavaScript interface. Figure 4 shows the relationship between the mobile devices and Apache Cordova. Developed applications are executed within wrappers targeted to each platform and rely on standards-compliant API bindings to access each device's capabilities such as sensors, data, network status, etc as shown in the mobile device view in Figure 4. The web view provides the application with its entire user interface. Applications built with Cordova are implemented like a web page by making use of the default local file called index.html with references to the Cascade Style Sheet (CSS), JavaScript, Images, media files and other resources needed for the application to run. There exists a very important config.xml file that provides information about the app such as the name and author of the application, it also specifies parameters affecting how it works, such as whether it responds to orientation shifts.

Cordova is useful when a mobile developer wants to build an application across more than one platform without re-implementing it with each platform's language and tool set. It is also useful when a web developer wants to deploy a web application that is packaged for distribution in various store portals. Lastly, when there is a need for a

mobile developer to develop a plugin interface between native and WebView components or a need to mix native application components with a WebView that can access device-level APIs. One advantage of using Cordova is that it includes a lot of plugins that can help to quickly build your application.

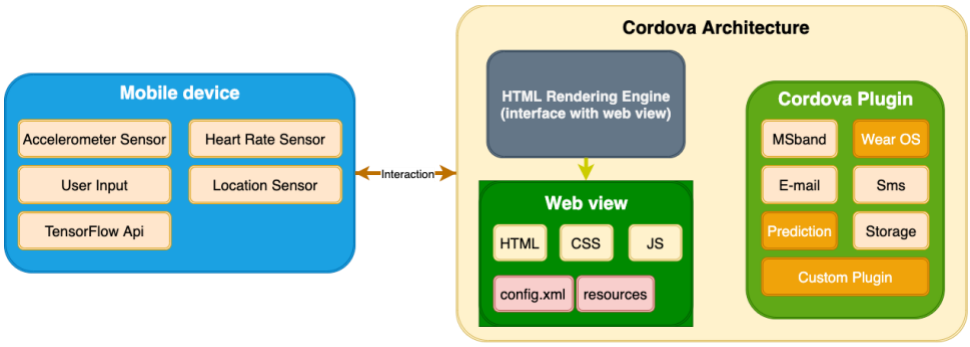


Figure 4. Apache Cordova Architecture

4.1.1. Installation of the Apache Cordova

In order to install the Apache Cordova, there is a requirement to install Node.js and npm on the computer client. Node.js is a JavaScript runtime engine built on Chrome's V8 JavaScript engine. There will be a need to install Android Studio for the Android application development and Xcode for IOS application development. To install Cordova, open the command line interface on Windows, Linux or OSX and type "*npm install -g Cordova*". This command installs the Cordova command line tool that allows you to run commands such as:

- "*Cordova create [arguments]*" to create a new Cordova project.
- "*Cordova platform add [mobile operating system]*" to add a new platform to your project. The operating systems include IOS, Android and browser.
- "*Cordova platform ls*" to check the current set of platforms in the Cordova project.
- "*Cordova platform add [mobile operating system]*" to remove the operating system platform in the Cordova project.
- "*Cordova build [mobile operating system]*" to compile and build the Cordova project.
- "*Cordova emulate android*" to run the Cordova project on an emulator.
- "*Cordova run android*" to run the Cordova project on a connected mobile device.

4.1.2. Cordova Plugins

Cordova Plugins are very important part of building applications for the Cordova ecosystem. They provide an interface for Cordova and native components to communicate with each other and bindings to standard device APIs. This enables you to invoke native code from JavaScript [28]. Apache Cordova project contains a set of basic plugins called the Core Plugins. These core plugins provide your application to access device capabilities such as battery, camera, contacts, Emails, SMS etc.

In addition to the core plugins, there are several custom plugins that provide additional features not necessarily available on all platforms. You can search for Cordova plugins using plugin search [29] or npm. Plugins is necessary to communicate between Cordova and custom native components.

4.1.3. Building a Cordova Plugin

In order to build a Cordova plugin, there is a need to develop the plugin as shown in Figure 4. The following steps are involved in building a Cordova plugin:

1. Create the following the files as showing in Figure 5. The *src/android* folder contains the source code that would be installed on the native platform (Java in this case). The *www* folder contains the plugin in JavaScript that serves as the interface to the native platform, it provides methods for accessing the native platform defined functionalities. The *package.json* contains the json configuration for the project such as the name, version, description, id and platform of the project while the *plugin.xml* file contains information about how the directories in the native platform where the plugin should be installed.

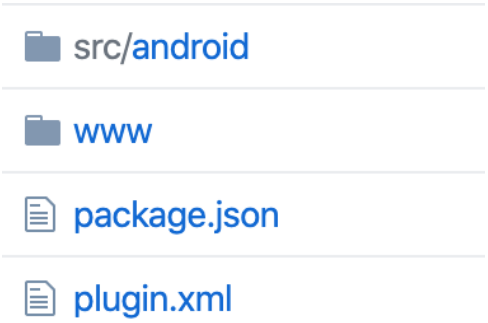


Figure 5. File Structure of the Cordova Plugin for an Android App

2. To install the plugin that was created in step 1, type "*Cordova plugin add Cordova-plugin-prediction*". This command setups the Cordova plugin for the Cordova project. In order to remove an already installed plugin, type "*Cordova plugin remove Cordova-plugin-prediction*".

Apache Cordova's cross platform capabilities and the lightweight script engine makes it attractive to be used as an edge IoT middleware. As we mentioned earlier, there is a Cordova host developed by the TerraSwarm Research Center at UC Berkeley, but it is still classified as an "Experimental Accessor Host". However, as mobile devices continue to phase out laptops and desktops for daily use, we need an accessor host that can run with less processing power, storage, and energy than the established CapeCode host. The experimental Cordova host is only tested using simple accessors. This does not provide the evidence that it can handle continuous large streaming data from IoT devices and serve as an IoT service framework.

To demonstrate the practically and advantage of Cordova host as an edge IoT middleware, we refactored two monolithic mobile Apps: 1) Fall Detection and 2) Heart Rate Monitoring into composition of accessors. We analyzed the reusability of accessors, the barrier of programming and deployment for consumers and the power consumption IoT services running on Cordova host.

4.2. Architecture of Cordova Accessor Host

Figure 6 shows the architecture of Cordova Accessor host. The capabilities of the Cordova Accessor host are found in the JS directory of Cordova/Webview. It is in the folders of Cordovahost and Commonhost in Figure 6. The accessors folder houses all the custom accessors contributed by the communities of developers. The Cordova plugins provide the association or binding between custom accessor's interfaces and their native implementation of the functionalities of those devices. For example, the data collection accessor can access the WEAROS APIs to obtain specific sensor data from a WEAROS

compatible smartwatch via WEAROS plugin and MSBAND SDK to obtain sensor data from a Microsoft watch. The plugins for each type of IoT device needs to be developed once and shared with the communities. The swarmlet.js is used to compose a pipeline of accessors to perform a specific task (a.k.a an IoT service).

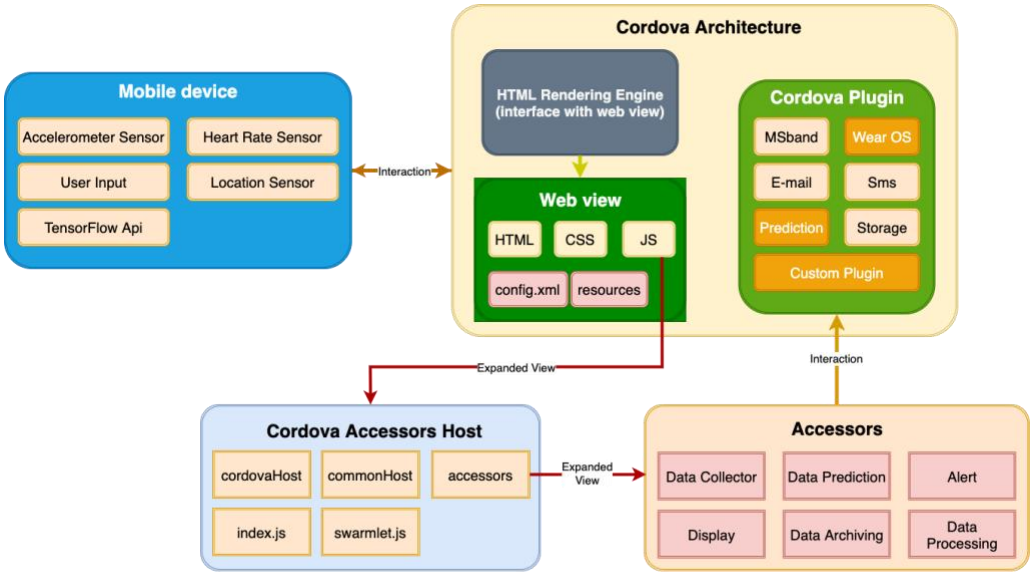


Figure 6. Cordova Accessor Host Architecture

5. EXPERIMENT

5.1. Overview

In this section, the practicality of Cordova Accessor host would be demonstrated by showing the advantages of using the Cordova host as an edge IoT middleware. We would be refactoring two monolithic mobile Application - The Fall Detection application and the Heart Rate Monitoring application - into composition of accessors. Furthermore, there would be a need to analyze the reusability, portability of accessors, the barrier of programming and deployment for consumers and the power consumption IoT services running on Cordova host.

5.2. The Setup of Cordova Accessor Host

Getting started with Cordova Accessor host requires some installations and setup which will require the following prerequisites similar to the Cordova host described in Section 4.1. In order to install the Cordova host, the following steps are required:

1. Install the Node package manager by typing “npm install -g Cordova” in the command line / terminal of windows, OSX or Linux.
2. To get started with our Fall Detection Application developed with Cordova Accessor Host, clone the public Repository at *https://github.com/jileyitayo/Cordova-accessor-host*.
3. Configure the downloaded project by setting up the config.xml and package.json files found in the root folder based on the specifics of the project.

The config.xml file consists of the widget id which includes the identifier for your project which can be edited by the developer, the name tag which includes the name the developer wishes to give the Application while the description explains briefly what the application is about and lastly, the author tag which includes the details of the developer building the application. These details would be loaded into the application once you build the application for the first time.

The package.json file is similar to the config.xml file but consists of the more information as follow: name, displayName, version, description, main, scripts, author, licence, Cordova plugins information, platforms and dependencies.

4. In the directory, change directory to the platforms/android.
5. In order to run the application, you need to open it with Android Studio, then Build and Run.

5.3. Accessors of the Fall Detection Application

The Fall detection is an application that senses the streaming accelerometer data from a commodity-based smartwatch device and applies a deep learning algorithm over the streaming data to detect falls. The smartwatch is paired with a smart phone that has the fall detection application installed in it. The application on the smartphone performs the necessary computations required for a fall prediction in real time with little or no latency. Figure 7 shows the main user interfaces of the Fall Detection App.

Figure 7. SmartFall App User Interface

The screen on the left shows the home screen UI for the application and the second screen shows the UI when a fall is detected. The home screen (leftmost screen in Figure 7) launches the App when the user presses the "ACTIVATE" button. The user must set up a profile and load the profile before the App can be activated.

When a fall is detected, the second screen of Figure 7 pops up on the smartphone, an audible sound is generated, and a timer of 30 seconds is initiated. The user is shown three buttons for interaction. The "NEED HELP" button will send a text message to the caregiver and also save and label the sensed data samples as true positives. The "FELL BUT OK" button will save the sensed data during that prediction interval as true positives without notifying the caregiver. The "I'M OKAY" button will save these data as false positives. If a fall is detected and the user does not interact with any of these three buttons, after the timer expires, the system assumes that the user might be hurt or unconscious and an alert message is generated and sent to the caregiver automatically. The third UI screen is for the one-time initialization of the user profile before the application can be launched. This UI includes setting up the contact details of the

caregiver. Note that minimal personal data is collected, and all those data are stored locally in the phone.

We refactored this Java implementation of Fall Detection application into various accessors as shown in Figure 8 such that each accessor performs a particular function or service and interact with other accessors in the pipeline via message passing. For Fall Detection application, we created the Data Collection Accessor, Data Prediction Accessor, Alert Accessor and the Display Accessor.

5.3.1. Data Collection Accessor

When the fall application is activated, the Data Collection Accessor is triggered to collect accelerometer sensor data from the smartwatch in a set interval/sampling period and sends the data as an output to the Data Prediction Accessor.

5.3.2. Data Prediction Accessor

The Data Prediction Accessor takes a sequence/stream of accelerometer sensor data as input and predict fall or not fall as the output which is passed to the Display Accessor. The data Prediction Accessor predicts fall by making use of the pre-trained deep learning RNN model.

5.3.3. Alert Accessor

The Alert Accessor receives the prediction status as input and send either an E-mail or SMS to a registered recipient (care giver) if a fall is detected.

5.3.4. Display Accessor

This accessor is responsible for displaying the accelerometer data to the user as a sign that communication between the watch and the phone is not faulty. It is used as a debugging tool.

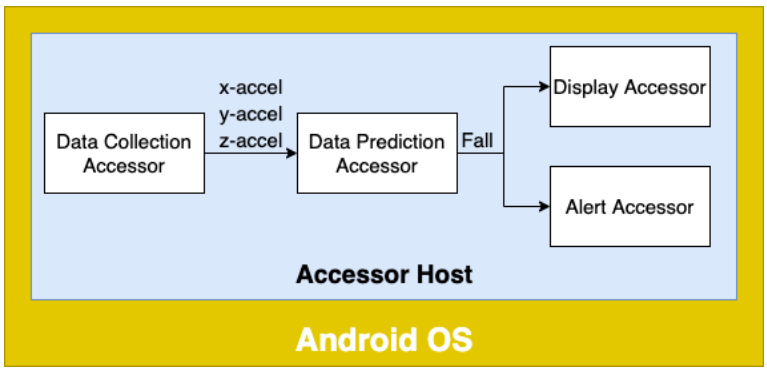


Figure 8. Main Accessors Used for the Fall Detection App

5.4. Accessors of the Heart Rate Monitoring Application

The Heart Rate Monitoring application utilizes the heart rate data collected from a smartwatch (IoT) device and a threshold algorithm to detect if there is an unusual high heartbeat per minute (bpm) given the current context of the user and alert the user. Similarly, the Heart Rate Monitoring application can be composed of different accessors. In fact, some of the accessors can be reused as it is from those already defined for the fall detection application as shown in Figure 9. The accessors for the Heart Rate Monitoring application are the Data Collection Accessor, Data Processing Accessor, Alert Accessor and the Display Accessor.

5.4.1. Data Collection Accessor

This application, similar to the fall detection application, commences with the data collection accessor which collects heart rate sensor data (bpm) from the smartwatch through the smartphone and sends the data as output to the Data Processing Accessor. The only change required in this accessor is the selection of heart rate rather than accelerometer sensor.

5.4.2. Data Processing Accessor

The Data Processing Accessor takes the heartbeat per minutes (bpm) data as input and gives an output of status of high bpm which is passed to the Display Accessor. The Data Processing Accessor performs a simple threshold algorithm to determine this high bpm based on its input. The threshold algorithm currently is set to a simple conditional statement, but it can be replaced by other more complex algorithm based on the activity level of a user. This is the only accessor that needs to be written from scratch for this application.

5.4.3. Alert Accessor

In the same way, the Alert Accessor receives the status as input and send either E-mail or SMS to a registered recipient if a high bpm is detected. This accessor is reused as it is from the Fall Detection.

5.4.4. Display Accessor

This accessor is responsible for displaying the heart rate information to the user as well as the status of monitoring. Since the type of data displayed is different from fall detection, the only change needed is to set the data to be displayed to a different type.

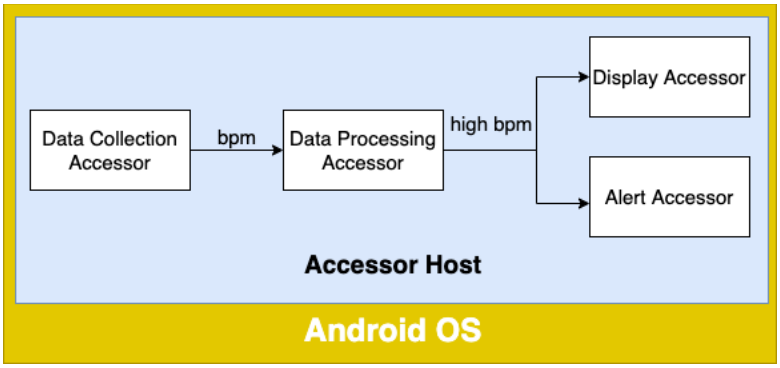


Figure 9. Main Accessors Used for the Heart Rate Monitor App

5.5. Reusability of Accessors

This sub-section discusses the reusability of accessors. Across the 2 applications, the Fall Detection application and the Heart Rate Monitoring application, the Data Collection Accessors and the Alert Accessors are reusable while the Data Prediction Accessor and Display Accessor are not reusable. The double bordered boxes represent the portion of the accessors that are reusable while the single bordered boxes represent the part of the accessors that are not reusable.

5.5.1. Data Collection Accessor

The Data Collection Accessor perform calls to the Cordova Plugin similar to a regular function call with the aim of retrieving the sensor data from the smartwatch. In order to achieve this, the supported device type and the type of sensor data to retrieve has to be specified in the data collection file. The Figure 9, 10 and the code snippet shown below are to be considered when as an overview during the discussion of the reusability of the Data Collection accessor. The Fall Detection application makes use of the accelerometer data as shown in the code snippet for the Data Collection Accessor and can be reused by simply changing a single line of code.

The highlighted lines of code on line 3 and 6 indicate the types of sensor the Data Collection accessor currently support and also shows how to make the accessor subscribe to sensor on the defined devices. In order to change the Data Collection Accessor to receive HEART_RATE sensor data for the Heart Rate Monitoring application, line 6 of the data collection accessor snippet code would need to be change to *var sensor_type = Sensors[0];*. This implementation makes it easy for the Data Collection Accessor to be reusable across the different IoT devices.

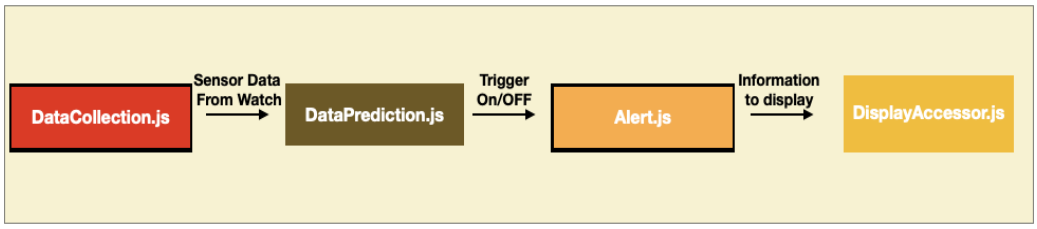


Figure 10. Swarmlet.js File Architecture

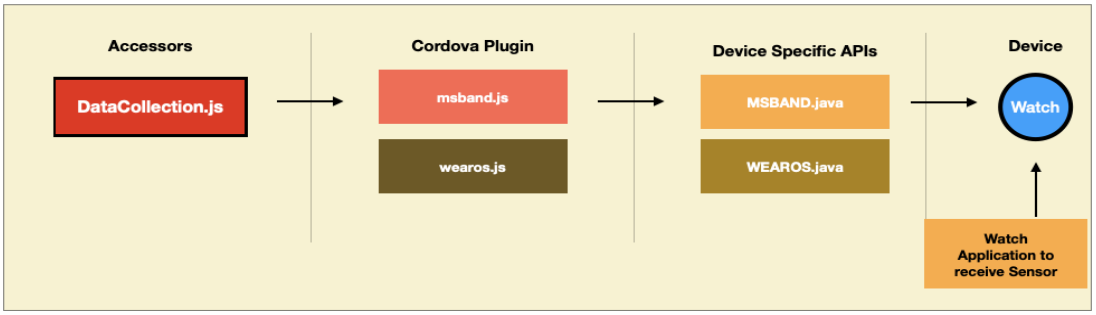


Figure 11. Data Collection Accessor Architecture

```
1 // SENSOR DATA and DEVICES
2 // variable declarations
3 var Sensors = ["HEART_RATE", "ACCELEROMETER"];
4 var Devices = ["WEAROS", "MSBAND"];

5 // set sensor to ACCELEROMETER
6 var sensor_type = Sensors[1];

7 // set device to WEAROS
8 var device_type = Devices[0];
...
...
```

5.5.2. Data Prediction Accessor

The Figure 12 shows the architecture for the Data Prediction Accessor. The Data Prediction Accessor is not reusable because it is an accessor specific to the Fall Detection

application. When building the Heart Rate Monitoring application, there would be a need to develop an application specific accessor to process the heart rate sensor data as shown in Figure 12.

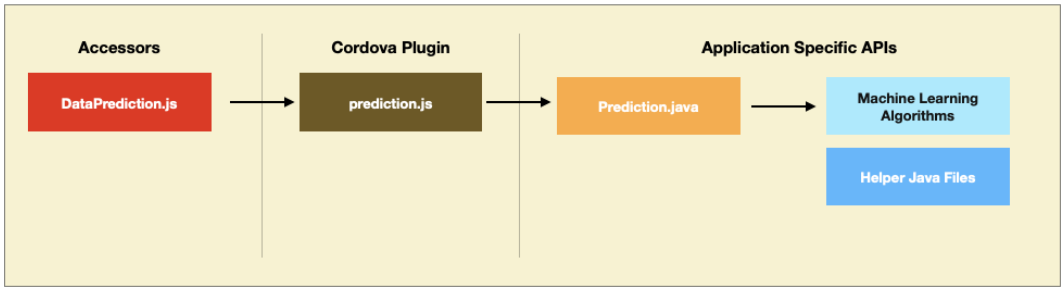


Figure 12. Data Prediction Accessor Architecture for the Fall Detection App

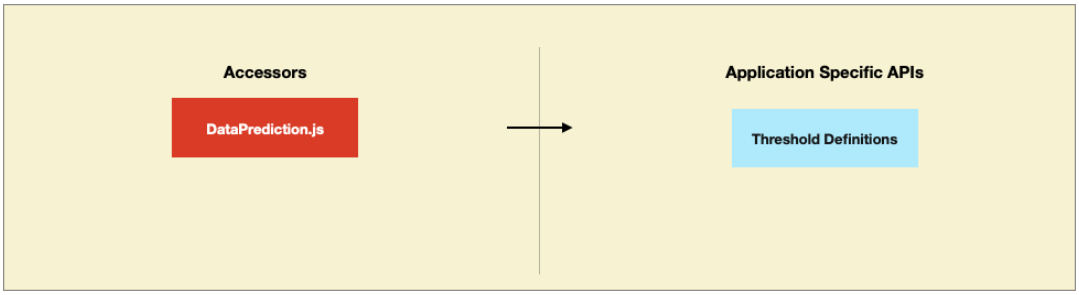


Figure 13. Data Processing Accessor Architecture for the Heart Rate Monitoring App

5.5.3. Display Accessor

The Display Accessor can be reusable since the different sensor data types can be displayed on the same display along with process results from predictions and processes. Some applications make use of array data type like in the case of the Fall Detection Application while some make use of string data type.

5.5.4. Alert Accessor

The Figure 13 shows the architecture for the Alert Accessor. This accessor is completely reusable because the application involves notifying a registered contact for the user of the application. The features of the alert accessor include sending SMS to registered contacts. These features can also be extended to other contact methods like phone calls, sending EMAIL. The Fall Detection application and the Heart Rate Monitoring application both use the same alert accessor without making any change its code.

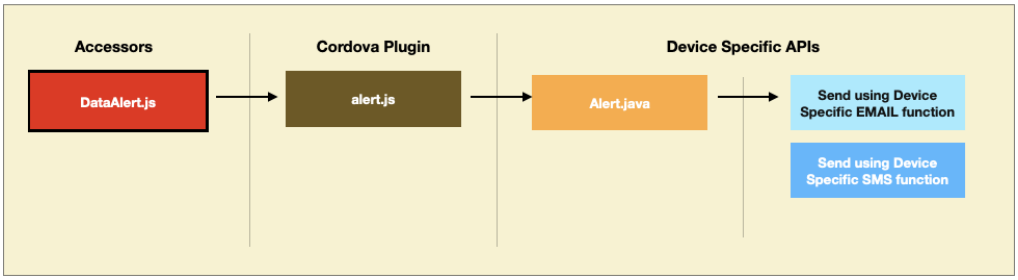


Figure 14. Alert Accessor Architecture

In conclusion, the total number of codes changed when reusing the reusable accessors is minimal. It is one line of code in the case of Data Collection Accessor.

5.6. Portability of Devices When Using Cordova Accessor Host

The portability of devices when using Cordova Accessor Host refers to the support Cordova Accessor host provide for different devices and how easy it is to port from one device to another. As previously discussed, the only accessor that communicates with the device is the Data Collection Accessor. Due to this, the Data Collection Accessor has been designed to support multiple devices and has provided an easy way to switch between devices. Figure 14 and 15 are considered when assessing the portability of devices when using Cordova Accessor host. Different Smartphones and Smartwatches were used and tested for portability. The following Smart Phones and Smartwatches shown in Table 1 and 2 were used in our experiments.

Table 1. Display of the Smartwatches and their Operating System

SmartWatch	**Operating System**
Huawei Watch 2	Android Wear OS (Version 2)
TicWatch Pro	Android Wear OS (Version 2)
Microsoft Band 2	Microsoft OS

Table 2. Display of the Smartphones and their Operating System

Smartphone	**Operating System**
Huawei Mate 9	Android OS (Version 8)
Google Nexus 5	Android OS (Version 8)

The smartwatches and smartphones were grouped into Operating systems. The MSBAND and WEAROS were used for the smartwatches while the ANDROID OS was used for the smartphones. The code below shows the source code for porting from MSBAND to WEAROS supported devices. The changes are highlighted in line 4 and 8.

```
1 // SENSOR DATA and DEVICES
2 // variable declarations
3 var Sensors = ["HEART_RATE", "ACCELEROMETER"];
4 var Devices = ["WEAROS", "MSBAND"];

5 // set sensor to ACCELEROMETER
6 var sensor_type = Sensors[1];

7 // set device to WEAROS
8 var device_type = Devices[0];
...
...
```

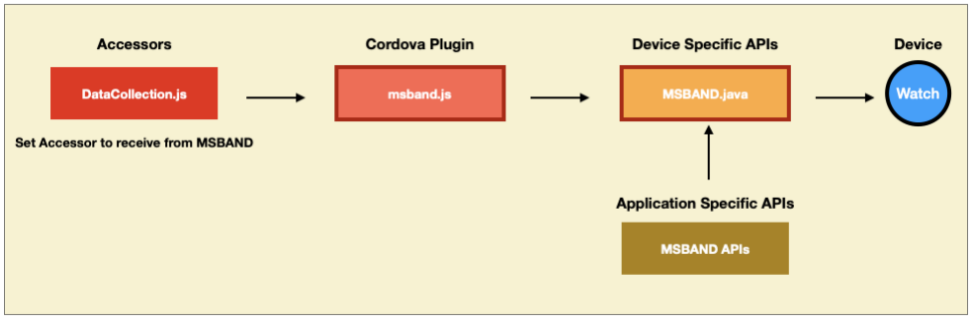


Figure 15. Fall detection App to Receive Sensor Data from MSBAND Smartwatch

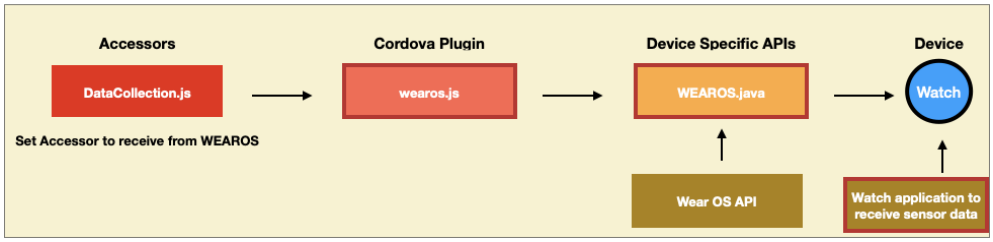


Figure 16. Fall detection App to Receive Sensor Data from WEAROS Smartwatch

There was a challenge noticed when porting to the WEAROS supported smartwatches. While the Microsoft Band 2 smartwatch did not require a separate application to be developed in order to retrieve sensor data, there was a need to develop an application for the WEAROS supported Smartwatches (Huawei Watch 2 and TicWatch Pro) so that sensor data can be retrieved from the them. This is because the MSBAND Cordova Plugin was created by Microsoft and it was designed to provide an SDK library to assess data from the smartwatch without the need to develop a separate application for the MSBAND smartwatch. This is not the case for the WEAROS supported smartwatches, we noticed that there the WEAROS made use of a separate method of extracting sensor data therefore we had to develop the WEAROS Cordova Plugin and its corresponding WEAROS application running on the smartwatches.

In order to change the type of sensor data received from the WEAROS smartwatch, the 3 highlighted lines of code that need to be modified in the WEAROS Smartwatch application as shown below in the code below.

...

heartrate = mSensorManager.getDefaultSensor(Sensor.TYPE_HEART_RATE);

boolean sensorHeartRateRegistered = mSensorManager.registerListener(this, heartrate,

SensorManager.SENSOR_DELAY_FASTEST);

...

5.7. Evaluation of Battery Performance

An edge-based IoT framework must be energy-efficient since it is deployed on power constrained IoT devices. In this section, we evaluate energy efficiency of Cordova Accessor host by comparing the battery power consumption of running an accessor-based Fall Detection App verses a native Java-based Fall Detection App. The two versions of Fall Detection App will be running on a TicWatch Pro smartwatch paired with Huawei Mate 9 smartphone running Android OS (version 8.0).

5.7.1. Battery Performance Setup

Both the phone and watch batteries are fully charged, and each application is made to run continuously for four hours for each test. The test is carried out by wearing the watch and activating the Fall Detection App on the phone while carrying out daily activities and recording the battery percentage at various intervals. Five tests will be run for each version of the Fall Detection App. The battery percentage of both the smartphone and the smartwatch will be recorded at every hour from the start of the experiment having 100% until the smartwatch gets to 0%. We report the average battery consumption percentage over the five tests for each application on the smartphone and the smartwatch.

5.7.2. Observations

We ran the first test for the accessor-based Fall Detection App deployed on the Cordova Accessor host and its corresponding test for the native Java Fall Detection App. We observed that the accessor-based version deployed on Cordova Accessor host performed a lot better than the native application. For example, we noticed that the native application had its smartphone's battery at 8% when the smartwatch battery became 0% while the accessor-based application had its smartphone's battery still at 35% when the smartwatch battery became 0%. We examined the codes in the native application and realized that it made use of some of the background services which might consume more battery as compared to the accessor-based application that did not make use of the background services. In order to compare them fairly, we removed the background service codes in the native application that might account for the differences.

After the modification, we performed the second of the five tests and noted that there was a little improvement with, but the difference is not that significance. The remaining three tests shows the similar trend. The average battery usage over the five tests at each one-hour interval are displayed in Table 3 and 4. We also plotted the line graphs showing the smartphone battery usage of accessor-based App versus native App in Figure 16.

The accessor-based App running on Cordova host used around 32.65% less battery power than the native App. The graph in Figure 16 gives a visual representation of the battery usage at each time interval. The battery usage of the smartwatch is the same because in both experiment the same service is run on the smartwatch to send the sensed data to the smartphone periodically.

Table 3. Average Battery Usage for Native (Java) App

Time (hr)	**0**	**1**	**2**	**3**	**4**
SmartPhone (%)	100	73.00	47.20	19.20	11.60
SmartWatch (%)	100	70.00	42.20	12.40	0.00

Table 4. Average Battery Usage for Cordova Accessor Host App

Time (hr)	**0**	**1**	**2**	**3**	**4**
SmartPhone (%)	100	84.75	68.75	50.25	44.25
SmartWatch (%)	100	72.00	42.25	12.00	0.00

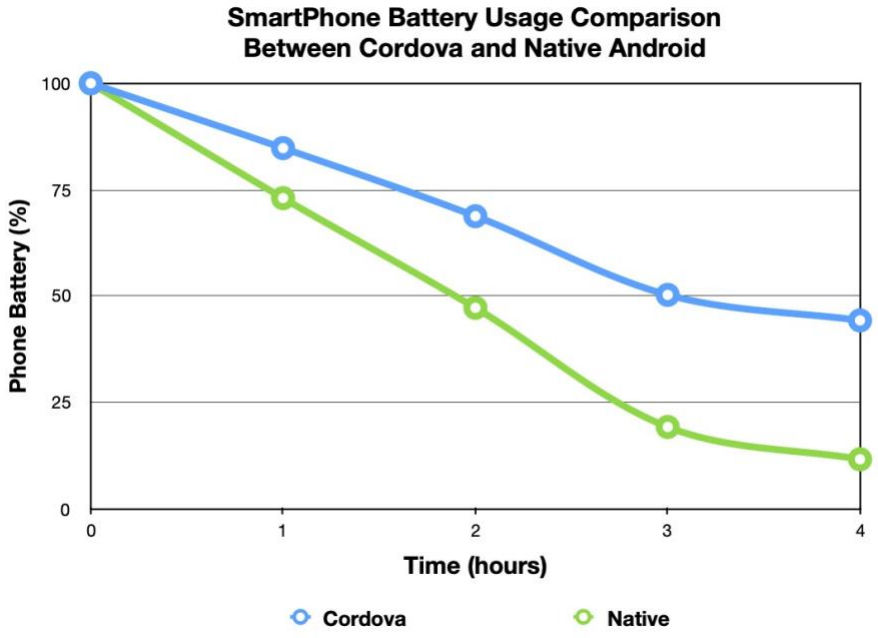


Figure 17. Graph Showing Smartphone Battery usage of Accessor-based App Versus Native App

6. CONCLUSION AND FUTURE WORK

We discussed the need for an edge-base IoT service framework and explored the Cordova Accessor host for that. We presented the architecture of Cordova Accessor host and analyzed the advantage of an edge IoT service middleware based on Apache Cordova platform for the development of real-world IoT applications with experimental evaluations of the effectiveness of Cordova Accessor host for rapid development of IoT services by composing two different IoT applications using accessors as the basic components and measures the reusability and code changes required.

We have also further demonstrated how accessor design pattern can facilitate sensor data collection across three different types of smartwatches from Google, Microsoft and Huawei with minimal additional programming. We also performed experiments to demonstrate the better battery utilization using IoT services on Cordova Accessor host verses native implementation.

In the Future, there will be a need to update accessors like the data collection accessor that supports the portability of heterogeneous IoT device such as Arduino, Raspberry Pi and other IoT devices other than smartwatches. There will also be a need to extend the alert accessor by adding more notification features like making phone calls. We have discussed the development of accessors and have made use of ANDROID operating system for our development, there will be a need to demonstrate the cross-platform feature of Cordova to deploy the accessor-based Fall Detection app and Heart Rate Monitoring app in IOS operating system for iPhones. Lastly, in terms of performance test, we only completed the study on battery consumption. There is a need to investigate other performance metrics. For example, there will be a need to study the

latency when collecting sensor data and undergo performance evaluation by comparing a similar framework (an edge-based framework) to the Cordova accessor host as well as comparing the memory utilization of using the Cordova accessor host when building apps versus using the native app.

APPENDIX SECTION

Appendix A: Setting up Apache Cordova

Step 1: Requirements

The following must be installed in order to start using Apache Cordova

1. Node.js
2. Apache Cordova – install by typing "*npm install -g Cordova*"
3. Install the following for Android
    a. Android Studio
    b. Java JDK
    c. Android SDK
    d. Android target (android-19, android-20)
    e. Gradle
4. For more requirements needed to be installed, type "*cordova requirements*"

Step 2: Create the Application

1. Create the required directory needed for your cordova app by typing "cordova create test com.example.testapplication TestApplication"
2. Add a platform to the Cordova application by typing "*cordova platform add <platform name>"*
3. View the current set of platforms by typing "*cordova platform ls*"

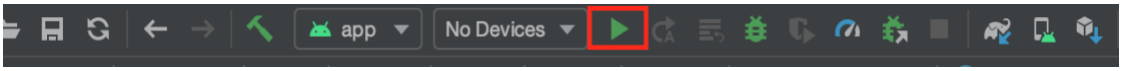
Step 3: Build the Application

1. To build the project for all platforms, type *"cordova build"*
2. To build for just a particular platform, for example: android.
   Type "*cordova build android*"

Step 4: Run the Application

There are different ways of running the application

1. Running with Android Studio
    a. Open the Android Cordova application using Android Studio
    b. Build the application and install missing android dependencies
    c. Run the application by clicking on the play Icon as shown in the Android Studio interface below



2. Running with Cordova command line
    a. Open command line for windows and Terminal for Mac or Linux
    b. Change directory to your project path
    c. From the command line, run "*cordova run <platform name>*". For example, to run an android application "*cordova run android*"

Appendix B: Working with Cordova Plugins

There are 2 ways to create cordova plugins

Using Plugman

1. To Install Plugman, type "*npm install -g plugman*"
2. Change directory to your project plugins found in *"web/hosts/cordova/plugins/"*.
3. To create a plugin, type "*plugman create --name huawei --plugin_id cordova-plugin-huawei -plugin_version 0.0.1*".
4. Change directory to the plugin just created.
5. Create package json by typing, "plugman createpackagejson <name of the plugman plugin created >". For example: "*plugman createpackagejson Huawei*"
6. To install the created plugin, type "*plugman install --platform android --project platforms/android --plugin <plugin_name>*".
7. To uninstall the installed plugin, type "*plugman uninstall --platform android --project platforms/android --plugin <plugin_name>*"

Using Cordova

1. Clone the sample cordova plugin repository at https://github.com/jileyitayo/cordova-plugin-wearos
2. Reconstruct this plugin to fit your desired plugin configuration
3. Change directory to your cordova application
4. Add the plugin to your application by typing, "*cordova plugin add <plugin_name>*".

Appendix C: Developing WearOS Application for a Phone Application

Building the WearOS application to receive accelerometer data

DataService.java

This file is found in "cordova-accessor - host/platforms/android/wearos/src/main/java/com/jse52/sensors_wearos/DataService.java "

MainActivity.java

This file is found in cordova-accessor- host/platforms/android/wearos/src/main/java/com/jse52/sensors_wearos/MainActivity.java"

activity_main.xml

This file is found in "cordova-accessor- host/platforms/android/wearos/src/main/res/layout/activity_main.xml"

wear.xml

This file is found in "cordova-accessor- host/platforms/android/wearos/src/main/res/values/wear.xml". This file should also be copied and pasted in the same location for the app module for the smartphone.

Integrating the WearOS application with Android SmartPhone Application

The following checks has to be considered when Integrating to Android SmartPhone Application.

- In the SmartPhone and Smartwatch manifest file, the manifest attribute package has to match as shown below

  package="com.jse52"
- The “applicationId” in the gradle has to match
- for both phone and watch, create a resource file in res/values/wear.xml that has the same item attribute “android_ wearos_capability”.

Setting Permissions when running Wearos application

When you run the wearos app for the first time, there will be a need to allow permission for the watch to receive the data from the watch. This can be done by going to Settings, Apps and permissions then the application name.

Appendix D: Getting Started with Cordova Accessor host with Fall Detection Application

Step 1: Requirements

The following must be installed in order to start using Cordova Accessor host

a. Node.js

b. Apache Cordova – install by typing "*npm install -g Cordova*"

c. Install the following for Android

- Android Studio
- Java JDK
- Android SDK
- Android target (android-19, android-20)
- Gradle

d. For more requirements needed to be installed, type "*cordova requirements*"

Step 2: Building the Fall Detection Application

a. Create a folder you want to save the Application

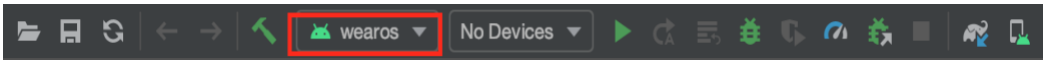
b. Open Terminal or Command line and Change directory to the created folder

c. Clone the Cordova Accessor host repository that contains the Fall detection application by typing "git clone https://github.com/jileyitayo/cordova-accessor-host"

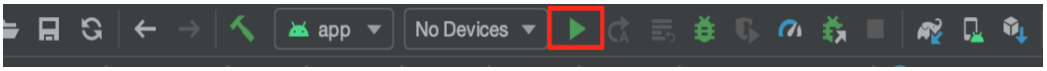
d. Open Android Studio and Open the existing folder found at "<created_folder>/*cordova-accessor-host/platforms/android*", then open the android folder.

e. Android studio will the build the application. This can take a while.

f. Once the gradle builds, select which device to run the application on. You can install on smartwatch (WearOS) or the smartphone(app)



g. To Run the application, click on the play button as shown below



REFERENCES

[1] R. Want, B. N. Schilit, and S. Jenson, "Enabling the Internet of Things," *Computer,* Periodical vol. 48, no. 1, pp. 28-35, 01/01/ 2015, doi: 10.1109/MC.2015.12.

[2] A. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, and Q. Z. Sheng, "IoT Middleware: A Survey on Issues and Enabling Technologies," *IEEE Internet of Things Journal, Internet of Things Journal, IEEE, IEEE Internet Things J.,* Periodical vol. 4, no. 1, pp. 1-20, 02/01/ 2017, doi: 10.1109/JIOT.2016.2615180.

[3] A. Karl Aberer, A. Manfred Hauswirth, and A. Ali Salehi, "A middleware for fast and flexible sensor network deployment," ed: VLDB Endowment, 2006, p. 1199.

[4] "NodeRed." http://nodered.org (accessed April 8th, 2020).

[5] E. Latronico, E. A. Lee, M. Lohstroh, C. Shaver, A. Wasicek, and M. Weber, "A Vision of Swarmlets," (in English), *IEEE Internet Computing,* Article vol. 19, no. 2, pp. 20-28, 03 / 01 / 2015, doi: 10.1109/MIC.2015.17.

[6] "Google Fit." https://developers.google.com/fit/ (accessed April 8th, 2020).

[7] "Architectural Styles and Design of Network-based Software Architectures." https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm (accessed 2000).

[8] A. Thomas Zachariah, A. Noah Klugman, A. Bradford Campbell, A. Joshua Adkins, A. Neal Jackson, and A. Prabal Dutta, "The Internet of Things Has a Gateway Problem," ed. New York, NY, USA: ACM, 2015, p. 27.

[9] "CordovaHost." https://wiki.eecs.berkeley.edu/accessors/Main/CordovaHost (accessed April 8th, 2020).

[10] "Entreprise JavaBeans." https://en.wikipedia.org/wiki/Enterprise_JavaBeans (accessed April 8th, 2020).

[11] A. HyunJae Lee, A. EunJin Jeong, A. Donghyun Kang, A. Jinmyeong Kim, and A. Soonhoi Ha, "A novel service-oriented platform for the internet of things," ed. New York, NY, USA: ACM, 2017, p. 1.

[12] "AWS IoT." https://aws.amazon.com/iot/ (accessed April 8th, 2020).

[13] "Watson IoT." https://www.ibm.com/internet-of-things/learn/what-is-iot/ (accessed April 8th, 2020).

[14] "ThingSpeak IoT." https://thingspeak.com (accessed April 8th, 2020).

[15] "Google Cloud IoT." https://cloud.google.com/solutions/iot (accessed April 8th, 2020).

[16] "The TerraSwarm Research Center." https://ptolemy.berkeley.edu/projects/terraswarm/ (accessed April 8th, 2020).

[17] "Accessors." https://www.terraswarm.org/accessors (accessed April 8th, 2020).

[18] M. Weber, R. Akella, and E. A. Lee, "Service Discovery for the Connected Car with Semantic Accessors," ed: IEEE, 2019, pp. 2417-2422.

[19] X. Lu, "An investigation on service-oriented architecture for constructing distributed web GIS application," 2005 / 01 / 01 / 2005, vol. I, pp. 191-197, doi: 10.1109/SCC.2005.27. [Online]. Available: http://libproxy.txstate.edu/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=edselc&AN=edselc.2-52.0-33745966720&site=eds-live&scope=site

[20] B.-Y. Ooi, Z.-W. Kong, W.-K. Lee, S.-Y. Liew, and S. Shirmohammadi, "A collaborative IoT-gateway architecture for reliable and cost effective measurements," *IEEE Instrumentation & Measurement Magazine, Instrumentation & Measurement Magazine, IEEE, IEEE Instrum. Meas. Mag.,* Periodical vol. 22, no. 6, pp. 11-17, 12/01/ 2019, doi: 10.1109/MIM.2019.8917898.

[21] B. D. Marah *et al.*, "Smartphone architecture for edge-centric iot analytics," (in English), *Sensors (Switzerland),* Article vol. 20, no. 3, 02 / 01 / 2020, doi: 10.3390/s20030892.

[22] C. Brooks *et al.*, "A Component Architecture for the Internet of Things," (in English), *Proceedings of the IEEE,* Article vol. 106, no. 9, pp. 1527-1542, 09 / 01 / 2018, doi: 10.1109/JPROC.2018.2812598.

[23] C. Brooks *et al.*, "A Component Architecture for the Internet of Things," vol. 106, ed, 2018, pp. 1527-1542.

[24] "BrowserHost." https://wiki.eecs.berkeley.edu/accessors/Main/BrowserHost (accessed April 8th, 2020).

[25] "Node Package Manager." https://www.npmjs.com (accessed April 8th, 2020).

[26] "NodeHost." https://wiki.eecs.berkeley.edu/accessors/Main/NodeHost (accessed April 8th, 2020).

[27] "CapeCode." http://capecode.org/ (accessed April 8th, 2020).

[28] "Apache Cordova." https://cordova.apache.org/docs/en/latest/guide/overview/index.html (accessed April 8th, 2020).

[29] "Cordova Plugin." https://cordova.apache.org/plugins/ (accessed April 8th, 2020).