Complex Activity Recognition

Haidar Chikh
Department of Computer Science, Electrical and Space Engineering
Lulea, Sweden
haidar.chikh@gmail.com

ABSTRACT

Internet of Things (IoT) vision visualizes a future where smart objects are ubiquities, connecting the physical and virtual world together. Smart objects generate a tremendous amount of data; this data represent the physical world state in numbers. The main challenges are interpreting and extracting useful information from this data. Measuring human activities by numbers is easy, but inferring what the activities are from numbers is challenging. Many researchers tackled this challenge, developing different algorithms to infer complex human activities. In this paper, we develop a system that uses state of art hardware/software to recognize human activities.

Author Keywords

Internet of things, Complex Activity Recognition, Cloud computing, Docker Containers

INTRODUCTION

Internet of Things (IoT) vision is to make the physical world reachable throw the virtual world. Over the course of 50 years the Internet grown from a handful of users in military and academia to more than a billion user today. This growth changed the Internet's purpose and paradigm. 1960 started as communication between two computers throw a network, TCP/IP introduced later and commercial use of the Internet Started the late 1980s. Later the World Wide Web become available in 1991 and in this stage we reaching information over the network was on a massive scale [3]. Later, mobile devices and PCs become ubiquitous and social media gave us means to reach people over the network. Today, cheap electronics, reliable software and high bandwidth networks enable us to reach the physical world throw the network; experts estimate that by 2020 we will have 50 billion connected devices.

Smart home falls into IoT vision. Different solutions were built by various companies to increase life quality of the users. These solutions enable us to sense and control the home state. [if we know what the user is doing we can make the home much smarter than today, we do that by spreading the smart objects at home]

No one stander regulate how to interface with a smart object, there is different wireless sensor networks (e.g. ZigBee, Z-Wave, etc.) and different API's. We end up like in 1990's, where we had a remote for every object,a TV remote, a sound system remote and a VCR remote. Today we have a smartphone or a tablet with a different application to control various objects. To solve this problem the industry come up with a universal remote control, you buy it from radio shake and hopefully it will be compatible with all of your appliances. OpenHAB : represent the same idea, an open Source software for integrating different home automation systems into one single solution, easily extensible for integrating new devices. Fetching the data from a smart home is the easy part, it introduces a bigger challenge, how to extract information from data.

Human activities tend to be complex in nature. Activities of Daily Living (ADLs)such as house work, office work, cooking, eating, shopping, exercising and grooming are highly complex, and humans tend to multi-task these with ease[5]. Algorithms and machine learning techniques were developed to enable researchers and the industry to infer users activities throw the steady state. Such as Complex Activity Recognition (CAR) algorithm: It's an algorithm based on Context Driven Activity Theory and Activity Signature [5].

Shipping code to a server and deploying a service usually consume many hours of developers time, making sure that what they developed and tested on their local machines will work on a given server. Taking care of all dependencies and conflicts between different application. Linux Container (LXC) released in the Linux kernel 2.6.24 attempts to solve this problem, where your application runs in its own isolated container which contains all of its dependencies. Docker containers use LXC control groups and namespace to give us a standard way to ship code in a box; this box will act the same regardless of the underlying Operating system. Building a software today's distributed world most likely will follow micro-services style, where the software is split into micro-services each has its well-deified task to do within the overall system. This style has its advantages and disadvantages, on the one hand, its simplifies building and debugging individual components. On the contrary, its introduces complexity in term handling messages between all the components to make the system run. The industry proposed many solutions to solve this, one

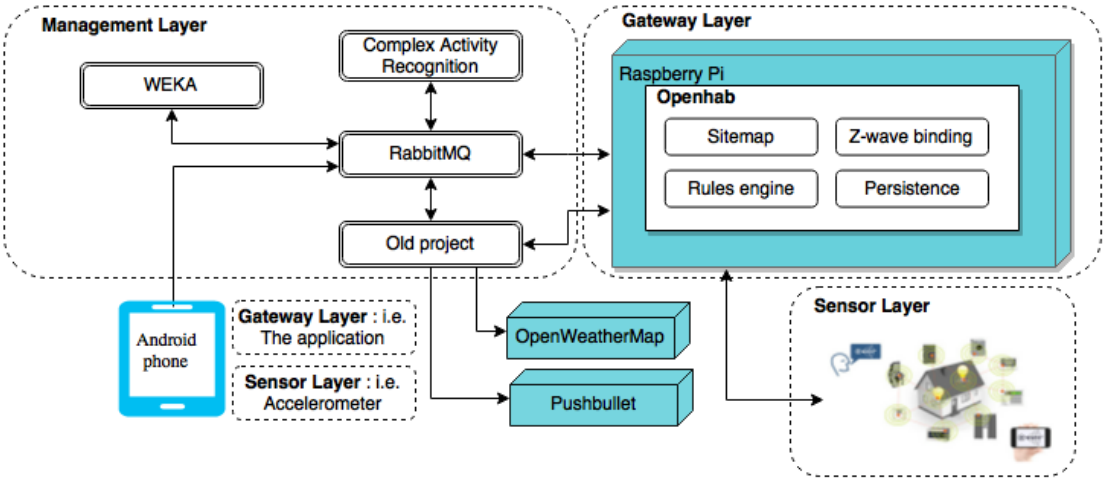

Figure 1. project architecture

of which is RabbitMQ. RabbitMQ is a message broker. In essence, it accepts messages from producers, and delivers them to consumers. In-between, it can route, buffer, and persist the messages according to rules you give it[1].

What is the potential of the state of art technology to cope with IoT demands?. In this paper, we developed a system to answer that question. Using OpenHAB, Docker containers, RabbitMQcac, Complex Activity Recognition (CAR) algorithm and a smartphone. This paper is organized into section as following: Method provides details of the software and hardware components used to develop the system. Results in this section we test the system, and conclusion was we discuss what we learned pros and cons of different technologies used.

Device	Capability
Android Phone	Accelerometer
Aeon Labs USB stick	Sink node
Smart switch	On/Off
Light-bulb socket	On/Off socket
Aeon Labs Z-Wave Multi-sensor	Motion, Humidity, Temperature, Illumination

Table 1. List of used sensors.

METHOD

The developed system is shown in Figure 1 consists of three layers. Sensor layer, Gateway layer and Management layer. The sensor layer gathers data from the environment and the gateway gives us means to push data over the network where the management layer process the data. The developed code architecture uses UNIX pipeline methodology, each component works as a pipeline with an input and output queue. This architecture gives us flexibility in experimentation. For example using Weka to infer an activity needs. collect the data from an accelerometer, originate the data in ARFF format, extract features and classify the data. We can with ease do the required computation in a single container or spread it to multiple containers running in various locations.

Sensor Layer

We used a mixture between warblers (i.e. Android Phone) and static sensors nodes (i.e. Z-Wave). Devices shown in Table 1 sense user's environment. Z-wave is a proprietary wireless communication protocol specifies hardware and software specification for smart home devices. Z-wave is a non-IP sensor network, in the project we used it as a flat architecture network, data transfer from/over sensor nodes (i.e. multi-sensor, smart switch) to a sink node (z-stick).

Gateway Layer

Two different component are used as a gateway. OpenHAB for home sensors and an android application for wearables, both applications push sensors data to a RabbitMQ broker.

Android Application

First We developed a simple android application shown in Figure 2. The application streams accelerometer data to a TCP server. First, The application was implemented using Node.js sending data using JSON objects over raw TCP sockets. After experimenting with the application we found out that it's inconvenient to work with raw TCP sockets and debugging Node.js was not helpful too so, we implemented a RabbitMQ client instead. The application has a toggle button to switch between training mode (stream data to train a classifier) and detection mode (unlabeled data is streamed). The application repository `https://github.com/haidarchikh/StreamAccelerometer` , where

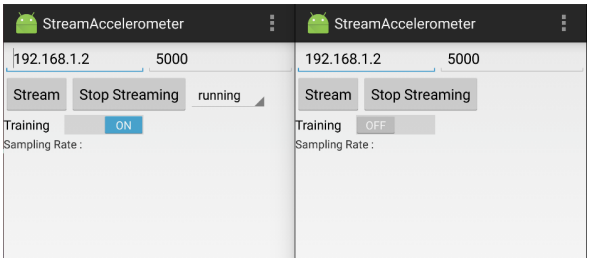

Figure 2. Android application screen shots.

you can import the application source code using android studio.

Management Layer

Docker Containers

The system is developed with loosely coupled component and fault tolerance in mind. This architecture made Docker containers even more useful. As shown in Figure 1 the system uses four containers.

1. RabbitMQ : Messaging broker, we use Advanced Messaging Queuing protocol (AMQP).

2. TCP server: The container runs a TCP server written using Node.js, getting data from the Android mobile phone, format the data to be used later by WEKA and pushing it to RabbitMQ. The container is deprecated, and we implemented a RabbitMQ client in the Android application.

3. WEKA: The container subscribes to accelerometer's exchange, gets raw accelerometer data, extract features, classify it and push the result back to event exchange.

4. Complex Activity Recognition : The container runs and implementation of Complex Activity Recognition Algorithm [5]. It subscribes to event exchange, gets event data and when it recognizes a Complex Activity, it prints the activity name.

5. Old project :I'm going to port pushbullet, OpenWeatherMap and OpenHabClient. // yet to port

Features Extraction

The Android application sends accelerometer data using JSON objects over AMQP protocol to a container where we extracts features from raw data. We extracts elven features, the features are used in previous work [2] proven their accuracy and save in computation. Description of each feature shown in [2]. Features 1-8 are standard statistical metrics. Feature 9 (Energy) has been shown to result in accurate detection of certain postures and activities. For instance, the energy of a subject's acceleration can discriminate low-intensity activities such as lying from moderate-intensity activities such as walking and high-intensity activities such as jogging [4] [2]. Feature 11 (Correlation) has been shown to improve the detection of activities involving movements of multiple body parts. It is helpful for differentiating among activities that involve translation in just one dimension [2].

NO.	Feature Description
1	Mean value for each axis (x, y, and z)
2	Average Mean over 3 axes
3	Standard Deviation value for each axis (x, y, and z)
4	Average Standard Deviation over 3 axes
5	Skewness value for each axis (x, y, and z)
6	Average Skewness over 3 axes
7	Kurtosis value for each axis (x, y, and z)
8	Average Kurtosis over 3 axes
9	Energy value for each axis (x, y, and z)
10	Average Energy over 3 axes
11	Correlations: x_y, x_z, x_total, y_z, y_total, z_total

Table 2. Features Description

Algorithm 1: Complex Activity Recognition

Input: $A_i\ C_i\ S_i$
Output: CA_k

checkCurrentSituation();
deleteDeadCA();
foreach $(CA_{list} \leftarrow CA_k)$ **do**
  if $(timecounter > T_{Lmax})$ **then**
    $delete\ CA_i$
  end
end
checkForNewCA(A_i, C_i);
foreach $(CA_{Situationlist} \leftarrow CA_k)$ **do**
  if $(A_i == A_s)$ **then**
    add $CA_{list} \leftarrow CA_i$
  end
end
passNewEvent($Event$);
foreach $(CA_{list} \leftarrow CA_k)$ **do**
  if $(A_i\ element in\ \gamma A_i)$ **then**
    add $A_i \rightarrow \gamma A_i$
  end
  if $(C_i\ element in\ \gamma C_i)$ **then**
    add $C_i \rightarrow \gamma C_i$
  end
end
checkIfAnyCAFulfilled();
foreach $CA_{list} \leftarrow CA_k$ **do**
  if $(A_E, C_E\ found for CA_i$ ***and*** $\rho A_i, \rho C_i$ *are completed* ***and*** $\omega_{CA_k} >= \omega^T_{CA_k})$ **then**
    $Found\ CA_k$
  end
  return CA_k
end

WEKA

We use WEKA 3.6.13 (stable) to interpret the accelerometer readings. First we trained a classier (J48) then used the resulting model to classify the data. We used a training set of 30 thousand sample, sampled at 110 Hz.

Complex Activity Recognition (CAR)

We implemented Complex Activity Recognition algorithm shown in Algorithm 30. The Algorithm is inspired from [5] with slight modifications. For testing purpose we defined two complex activities shown in Table **??** The implementation is done using Java and the code is available in the following repository https://github.com/haidarchikh/complex-AR/tree/master/ComplexActivityRecognition

System installation

The code is available on https://github.com/haidarchikh/complex-AR. You need to have git and gradle installed on your

CA_k ($\omega_{CA_k}^{A_i}$)	$\gamma A\, \omega_{CA_k}^{A_i}$	$\rho C \omega_{CA_k}^{C_i}$	Core γA and γC	A_S, C_S	A_E, C_E	T_L
CA_1:Brushing teeth	A_3:standing A_7:open bathroom cupboard A_8:take toothbrush A_9:take toothpaste A_{10}:brushing teeth hand move	C_8:at bathroom C_9:bathroom light on	A_{10} , C_8	A_3	A_{10}	3 min
CA_2:Preparing coffee	A_1:walking A_3:standing A_{15}:coffee mug A_{16}:coffee machine	C_{10}:at kitchen C_{11}:kitchen light on	A_3 , C_{15}	A_{15} ,C_{10}	A_{16}	1 min

Table 3. Complex Activity

```
$ git clone https://github.com/haidarchikh/complex−AR
$ cd complex−AR
$ gradle build
$ gradle eclipse
$ gradle copyJar
```

machine to build it. Using terminal, execute the following Unix shell commands: After the execution you will have five eclipse projects to import and two executable jars in /jar folder.

Attribute/Implementation	TCP	RabbitMQ(AMQP)
Sampling Rate	120Hz	120Hz
Allocated Memory	20 MB	22MB
Network Usage Upload	10KB/s	14KB/s
Network Usage Download	4KB/s	4KB/s

Table 4. Android application performance running on Oneplus One

RESULTS

WEKA

When we first trained the J48 decision tree classifier, The J48 used to time stamp as prancing value. The accuracy was 99 percent, but that works only on that particular data set. We removed the time stamp and precision went down to 94 percent, but the resulting model was useful to classify new data. After using the resulting classifier, we found out that classifying raw data is not easy to handle. When accelerometer data is sampled at 100 Hz, Each instance of raw data represents 10 ms and the classier is predicting according to a 10 ms data of human activity. Extracting features from the raw data and use the new data set to train a classier proved to be useful. The accuracy of the classifier went up to 99.1579 taking window size of 128 sample and sliding window size of 64. Extracting features from raw data has its positive and negative effects. A negative side effect of using the featured data that the classifier needs a minimum of 196 sample to be able to predict. If the data is sampled at 100 Hz that means we need two seconds of data to get a prediction. Positive effects are the higher accuracy and ease of handle the results because the prediction is a result of 2 seconds data and need no more proceeding to be used.

Android Application

We have tested the application using two phones, OnePlus One and Samsung Galaxy S2. Each phone samples the accelerometer data with a different sampling rate, related to the accelerometer sensor. 110 Hz for Oneplus one and 96 Hz for galaxy s2.Table 4 shows the performance of the application, the table also shows a comparison between sending JSON objects over raw TCP sockets or JSON objects over AMQP.

TCP server

Using Node.js for the server has its pros and cons. One of the major pros is Node's event loop that spares the effort to handle multi-threading. One of the cons we faced is node.js overhead and debugging issues comparing to Java for example. As mentioned before the final implementation uses AMQP instead of raw TCP sockets.

Complex Activity Recognition (CAR)

We did not test the algorithm in the real world. The testing is done using a mockup. The mockup uses a pool of activities, sending random activities from the pool while sending the tested activity. The algorithm was able to recognize the defined complex activity.

Docker Container

Using Docker container to deploy our system proved to be helpful. We used three containers to deploy our system, and we were able to have the same result on the developer machine and an EC2 instance.

CONCLUSION

The objective of this paper was to implement Complex Activity Recognition algorithm (CAR)[5] and use machine learning (i.e. Weka) to recognize complex activities. We developed an Android application, implemented Complex Activity Recognition and developed a Weka based machine learning classifier. The deployment was using Docker containers, and we tested the machine learning part using the developed Android

application and got a good accuracy. The Complex Activity Recognition algorithm was tested using a mockup, the mockup showed that the implementation works correctly.

REFERENCES

1. 2015. RabbitMQ - RabbitMQ tutorial - "Hello World!". (April 2015). https://www.rabbitmq.com/tutorials/tutorial-one-java.html
2. Ian Cleland, Basel Kikhia, Chris Nugent, Andrey Boytsov, Josef Hallberg, Kåre Synnes, Sally McClean, and Dewar Finlay. 2013. Optimal placement of accelerometers for the detection of everyday activities. *Sensors* 13, 7 (2013), 9183–9200.
3. Charith Perera, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. 2014. Context aware computing for the internet of things: A survey. *Communications Surveys & Tutorials, IEEE* 16, 1 (2014), 414–454.
4. Stephen J Preece, John Y Goulermas, Laurence PJ Kenney, Dave Howard, Kenneth Meijer, and Robin Crompton. 2009. Activity identification using body-mounted sensors-a review of classification techniques. *Physiological measurement* 30, 4 (2009), R1.
5. Saguna Saguna, Arkady Zaslavsky, and Dipanjan Chakraborty. 2013. Complex activity recognition using context-driven activity theory and activity signatures. *ACM Transactions on Computer-Human Interaction (TOCHI)* 20, 6 (2013), 32.