Wearable OS

Project Overview

Fiore Basile <fiore.basile@gmail.com>

v.0.3 October 21, 2013

What’s Wearable OS?

Wearable OS aims to make it extremely easy to build wearable interactive applications connected to the internet eco-system and the physical environment.

Why?

- Today:
  - wearable tech and applications hard because of hardware and software fragmentation
  - many skills are required to build a complete app (embedded, mobile, pc, server side)

Why?

- Tomorrow:
  - Even more hardware fragmentation
  - More demanding use cases from customers
  - Lots of different protocols to deal with
  - Nobody knows where we are heading: smartwatches, augmented reality, internet coffee machines...

Micro-controllers

- Lilypad, Flora, Xadow all use AVR ATMega today but
  - Other architectures will be used in the future
- Even on Arduino, all sensors have different dedicated libraries
- Mixing-and-matching solutions from different vendors is often required

Mobile

- At least iOS + Android
- No standard way of communicating with micro-controllers
- No standard way of communicating with cloud servers
- No standard way of communicating with other Internet connected devices

Connectivity

- Wifi
- Zigbee
- Bluetooth Low Energy
- NFC
- more to come?

Internet of Things

- Many platforms for IoT (Xively, COSM, AllJoyn,...)
- No standard interface for wearable sensor data
- Many interoperability approaches
- Future fragmentation is also very probable

How to solve this?

- a set of ready to use library containing the most common bits of functionality in wearable apps
- an hardware abstraction layer supporting most wearable hardware platforms
- a network independent protocol
- a graphical IDE allowing to define application features in a visual fashion
- code generators of functional code skeletons for microcontrollers, mobile devices and backend servers

Wearable OS Functional Architecture

Application Layer

Services Layer

Bus Layer

Systems Layer

Hardware Abstraction Layer

Platform Drivers Layer

Code Generators

Functional architecture

- An **application layer**, where most application logic is found, in the form of sketches, configuration files and data sets. Apps written for the wearable OS will make use of the API provided by the application layer and the underlying layers when a depper control is necessary.
- A **services layer**, where the developer can access most high level functionality necessary for wearable apps, such as interaction, communication, environment and profile management.
- A **bus layer**, where publish-subscribe messaging, sensor fusion and orchestration is performed.
- A **systems layer**, where the necessary core services are implemented, for example providing access to displays, data sources, input / output channels, network services and external systems such as Smartphones, PCs, domotics, vehicles, etc.
- **Hardware abstraction layer**, where appropriate generalizations of hardware components, such as microcontrollers, sensors, actuators, and other building blocks are defined.
- **Platform drivers layer**, where a concrete implementation of abstract devices is provided. This layer is where support is added for the different hardware platform, using for example AVR code and Arduino Libraries, or other specific drivers.

Core Concepts

- A **Node** is an elementary component of a Wearable app, representing a device, a sensor, an actuator, or an external entity.
- A **Message** is a basic unit of information exchanged by nodes via the **Hub**. Each **message** could represent a state, a bit of sensor data, an action to be executed, and so on. Wearable OS relies on messages for distributing all data within the application.
- The **Hub** is the entity which is in charge of distributing **Messages**. It does so by allowing Nodes to publish and subscribe to Topics.
- A **Topic** can be seen as a many-to-many channel, and addresses a specific information delivery need of the application.
- **Profiles** are way in which **Nodes** describe their information needs as well as their dependencies. All **profiles** are stored in a hierarchical data structure, accessible by all nodes and called **Registry**.
- The **Manifest** is a special profile also residing in the **Registry**, which defines the **Nodes** and **Profiles** composing an application.
- The Wearable OS allows to define different Manifests for different applications, possibly running on the same hardware or distributed among different hardware components.

Software architecture

Core API

Sensing API

Things API

Interaction API

Message Bus

Registry

Hardware Abstraction

Networking

Drivers

- **Core API**: defines the core classes such as Node, Message, Topic, Profile, Registy, Hub, Manifest.

- **Sensing API**: defines the "senses" available to the application programmer, for example motion, vision, gesture, gaze, touch, etc

- **Interaction API**: provides high level recognition capabilities for gesture, object motion, touch and sensor data intepretation

- **Message Bus API**: provides the API for topic management and publish-subscribe message passing among different network channels

- **Registry API**: allows to manipulate a shared state for the wearable app, tracking configured nodes and managing configuration data

- **Networking API**: provides an abstract vision to the actual network channels used by the different nodes composing the wearable app

- **Things API**: allows the interaction with Internet of Things devices in a unified fashion, abstracting different protocols including HTTP REST, Alljoyn, OSC, Midi, CAN etc.

- **Hardware Abstraction API**: allows to interact with actual hardware in an abstract fashion, i.e. without actually knowing what specific hardware is implementing a function

- **Platform Drivers:** include concrete implementations of HAL devices depending on the different platforms.

Use case example

Physical Activity Tracking

- Wearable application running on open hardware allowing the user to track its physical activity and calories consumed through the day
- create a track history for every day during a given period
- check progress via a mobile or web app

Without Wearable OS

- the developer should choose an hardware platform, i.e. a microcontroller, a set of sensors to be used to detect the physical activitiy, a way to store activity data during the day, a communication channel to upload such data to a custom backend code written for the cloud and a mobile platform to develop the tracking app.
- start to write from scratch software for each of these components, for collecting, interpreting, storing and visualizing all data. Furthermore, the developer, very often not an hardware engineer, must make compulsory choices on the hardware used, networking protocol, and so on because quite low level code must also be written to make the hardware work properly

With Wearable OS

- define his app in terms of the Wearable OS core concepts:
  - A data collection node
  - A motion sensing node
  - A data storage node
  - A data visualization node

Define profiles

- The "data collection node" would have the following profile, acquiring accelerometer data in the form of time series:

```
- Input = accelerometer.sensor
- Output = accelerometer.acceleration,accelerometer.timestamp
- Format = json
- Target = topic.accelerometer_timeseries
```

Define profiles

- The "motion sensing node" would have the following profiles, converting accelerometer data into calories and summing up calories at an hourly rate:

```
- Input = topic.accelerometer_timeseries
- Module = accelerometer_to_calories
- Output = motion.calories, sum.hourly
- Format = json
- Target = topic.calories_timeseries
```

Define profiles

- The "data storage node" would then be like the following, storing the data into a data set within the registry

```
- Input = "topic.calories_timeseries"
- Output = "registry.calories.${today}"
- Format = json
```

Define profiles

- The "data visualization node" would then just read data from the registry and post it to the visualization server as a graph

```
- Input = "registry.calories"
- Output = graph.timeseries
- Format = http.post
- Target = "http://wearable.host/calories"
```

Features

- all the asynchronous communication, networking, sensor calibration, data parsing, storage and visualization are all abstracted by the profiles and manifest.
- Communication among nodes is realized through simple pattern matching of topics, distributed to the hub only when it is available.
- In this conceptual framework the link between nodes doesn't need a continuous data link, but communication happens when such a link becomes available.

Selecting hardware

From the Manifest, through a GUI we can select the matching hardware, creating an **hardware recipe**

```
- Node = "Xadow Accelerometer"
- Provides = "accelerometer.sensor"
- Depends = "Xadow Mainboard", "SPI", "MCU", "Memory"

- Node = "Xadow Mainboard"
- Provides = "SPI","MCU", "Memory"


- Node = "iPhone"
- Provides = "hub","topic"

- Node = "WearableOS Registry Cloud Server"
- Depends = "hub"
- Provides = "registry"

- Node = "WearableOS Visualization App"
- Depends = "registry"
- Provides = "graph"
```

Generate code

the combination of the **Manifest** and the **Hardware recipe** could easily be used to generate one or more application **skeletons** containing the pre-canned api calls used to build the data flow and processing for the actual application

- The Arduino .pde firmata code to be run on the Xadow Mainboard

- The iOS app code to acquire data from the Xadow Mainboard and pipeline it to the Wearable Hub Server

- The backend code to transform registry data into a browsable graph