An Adaptive Middleware for Opportunistic Mobile Sensing

Rafael Oliveira Vasconcelos, Luis Talavera, Igor Vasconcelos, Marcos Roriz, Markus Endler
Department of Informatics
Pontifical Catholic University of Rio de Janeiro (PUC-Rio)
Rio de Janeiro, Brazil
{rvasconcelos,lrios,ivasconcelos,mroriz,endler}@inf.puc-rio.br

Berto de Tácio Pereira Gomes, Francisco José da Silva e Silva
PPGEE, and DEINF
Federal University of Maranhão
São Luiz, Brazil
{bertodetacio}@gmail.com, {fssilva}@deinf.ufma.br

Abstract—The current ubiquity of smartphone with mobile Internet and several short-range wireless interfaces (NFC, Bluetooth, Bluetooth Smart) and the fact that these devices are carried almost anytime and anywhere by users, enables potentially new pervasive sensing applications where smartphones can act as universal hubs for interaction with sensors (or sensor networks) that have only short-range wireless connectivity. Thus, in next years we can expect an increasing number of long-term and large-scale deployments for various crowd-sourced monitoring applications, such as environment monitoring, domestic utility meter reading, urban monitoring, etc. In this paper, we present the implementation and initial performance results with our mobile-cloud middleware that enables such opportunistic mobile sensing. One of the singular features of our middleware is the capability to discover, dynamically download and install sensor-specific transcoding modules on the mobile phone according to the encountered sensor type and make.

Index Terms—Internet of Things, Middleware, Mobile objects, Mobile Sensing, Dynamic Adaptation.

I. INTRODUCTION

With the continuous improvement and maturation of wireless sensor networks, we can expect an increasing number of long-term and large-scale deployments for various crowd-sourced monitoring applications, such as environment monitoring, domestic utility meter reading, urban monitoring, etc. For example, several European cities are installing air quality monitoring systems on the streets to satisfy EU regulations.

The current ubiquity of smartphones with mobile Internet and several short-range wireless interfaces (NFC, Bluetooth, Bluetooth Smart) and the fact that smartphones are used almost anytime and anywhere, enables potentially new pervasive sensing applications where smartphones can act as universal hubs for interaction with simple sensors (and sensor networks) that have only short-range wireless connectivity. For instance, a smartphone user can retrieve various information (temperature, humidity, CO concentration, etc.) from sensor nodes around him/her, both for making this information available to him/her, or for relaying it through the mobile Internet (3G/4G, or Wi-Fi) to a public environmental monitoring service or agency, so that any citizen can learn about the current situation in each part of the city.

In the following, we present a hypothetic mobile collaborative ambient sensing application, where our proposed middleware could be used.

In a region with high density of smartphone users, such as a metropolitan area, we can imagine the need for collaborative air quality monitoring. Common citizens may obtain tax incentives to install and deploy affordable Wireless Air Monitoring Stations (WAMS) in their yards, along neighborhood driveways, parks or other public spaces. All such WAMS would have CO, NO2 SO2 Lead sensors, etc., a short range and low-power wireless interface, be weather-resistant and run on solar energy. In this context, the collaborative monitoring app would be a crowd-sourced one where pedestrians passing close to some WAMS would donate their smartphone's Internet connectivity and energy to upload the current collected sensor data from the nearby WAMS to a city-wide monitoring service in the cloud, where all this information would be presented on a map both on-line and in consolidated statistics, for access by any citizens. Actually, through this monitoring application air quality indeed would not be monitored uniformly in the entire city region, but only in those parts of the city where it is most relevant, i.e., the places with much intense pedestrian and bike traffic. Moreover, some citizens may even decide for an air monitoring “on the go”, carrying a smaller and lighter version of the WAMS on their bike baskets or their knapsacks, so as to measure the air pollution on their daily paths through the city. And through their smartphone, this data would be collected and uploaded to a publicly accessible air quality monitoring service executing in the cloud.

Using smartphones (tablets or phablets) as universal hubs allows the otherwise sparsely deployed sensors (with their short-range wireless technologies), to become accessible through the smartphone's mobile Internet connection. However, because such devices are not stationary but move, this wide-range sensor accessibility will be temporary and intermittent, and sensor data must be gathered opportunistically and automatically (without user intervention), as proposed by other works [1], [2].
Such opportunistic data collection through smartphones also has the benefit of avoiding stationary, centralized and technology-specific sink nodes for the wireless sensors (networks), and pushing communication and data processing closer to the sensors and the edge networks. Although sensor data transmission in this case shall be less predictable and may have higher latency there are many promising environment monitoring applications which are delay-tolerant, since they do not depend on the latest sensor data, but instead on medium-term statistics of the sensor readings. For example, analysis of environmental and urban monitoring is rarely urgent and meter readings for billing can be delayed for some days.

In this paper, we present the design and prototype implementation of a mobile-cloud middleware that supports such opportunistic mobile sensing. On the cloud side, it consists of a software infrastructure based on OMG DDS (Data Distribution Service) [3] for high-performance, reliable and scalable communications, both among cloud nodes, as well as with mobile end user devices (smartphones/tablets) over the mobile Internet. On the mobile device side, it consists of the Mobile Hub (M-Hub), a middleware component that has embedded support for several wireless short-range (WPAN) technologies and realizes the opportunistic discovery, identification and connection with nearby wireless sensors, as well as the transcoding, preprocessing and transmission of sensor data to the cloud, while optimizing the smartphone’s Internet connection.

One big problem when designing a general-purpose middleware for opportunistic mobile sensing is the large set of WPAN technologies/standards and the heterogeneity of sensor types, models and manufacturers, making it almost impossible to implement a universal Mobile Hub. Moreover, such middleware must be extensible and dynamically adaptive to incorporate sensor and WPAN-specific software modules on the fly, whenever it discovers a new sensor device. And for enabling opportunistic sensing this software module download and update must be a fast, network failure tolerant, and the least interfering with the remaining processing on the smartphone.

Aiming to address the extensibility problem and support timely dynamic adaptation, our Mobile Hub has a modular software architecture with WPAN technology specific plug-ins, and the ability for fast dynamic download and deployment of sensor-specific transcoding modules whenever the M-Hub discovers a new sort of sensor in its vicinity. Our current implementation of the M-Hub is for the Android platform and supports both Classic Bluetooth and Bluetooth LE (BLE or Smart) WPANs. We also present initial performance results of experiments using our middleware in a scenario of opportunistic connection to a new sensor type. In particular, we show that if the sensor communicates through BLE, then the entire cycle from sensor discovery until activation of the sensor-specific module in the M-Hub can be sufficiently fast so as to enable opportunistic mobile sensing.

The following are the main contributions of this paper:

1. The concept of Mobile Hub as main component of a comprehensible middleware for opportunistic mobile sensing, and discussion of its main functions and limitations;
2. Modular architecture of the Mobile Hub which allows extensibility in respect to WPANs and dynamic deployment of sensors-specific modules;
3. Prototype implementation of the Mobile Hub on Android with WPAN plug-ins for Classic Bluetooth and Bluetooth LE, and the implementation of dynamic and on-demand deployment of new sensor modules.
4. Assessment of the performance of our M-Hub implementation (using BLE) for opportunistic connection and probing of data from a specific sensor device.

The remainder of this paper is structured as follows: In Section II we give an overview of the communication infrastructure of chosen and the Mobile Hub (M-Hub). In Section III we present the general architecture and main components of the M-Hub. Later, in Section IV, we explain how the M-Hub is able to dynamically download and deploy new sensor modules for specific sensors. We present in Section V the preliminary performance experiments and results. In Section VI related work is discussed. Finally, Section VII contains some concluding remarks and future work.

II. OVERVIEW OF THE MIDDLEWARE

The proposed mobile-cloud software infrastructure is essentially composed of two parts: the M-Hub and The Scalable Data Distribution Layer (SDDL). The M-Hub is a general-purpose component of our middleware that runs on mobile personal devices (e.g., smartphone, tablets), which is responsible for discovering and opportunistically connecting to sensor devices which are accessible only through short-range WPAN technologies such as Bluetooth, NFC, ANT+, etc. In order to enable transmission of sensor data to cloud services, the M-Hub uses SDDL, our scalable middleware for mobile communications, processing and data storage in clouds or clusters. In the following, we give an overview of SDDL and M-Hub.

A. SDDL

The Scalable Data Distribution Layer (SDDL) [4] is a communication middleware that connects mobile nodes (smart phones or tablets) with any Internet connection (3G/4G or Wi-Fi) to stationary server nodes executing in the cloud or cluster (a.k.a. SDDL Core), SDDL employs two communication protocols: the Data Distribution Service (DDS) for Real Time Publish/Subscribe Protocol for the wired communication within the SDDL Core, and the Mobile Reliable UDP (MR-UDP) for the inbound and outbound message exchange between the SDDL core and the mobile nodes, including M-Hubs. The Data Distribution Service for Real-Time Systems (DDS) [3] is an Object Management Group (OMG) [5] standard that specifies a fully decentralized publish/subscribe communication infrastructure aimed at high performance and real-time distribution of critical information in distributed...
systems. This specification was designed with the intention of obtaining a high scalability, portability and interoperability.

The MR-UDP, on the other hand, is Reliable-UDP enhanced by mechanisms for tolerating intermittent connectivity, dynamic IP address changes of the Mobile nodes and reaching these nodes behind firewalls/NATs. MR-UDP is used to connect the mobile nodes with a special type of SDDL Core node called Gateway (GW), of which any number can be deployed in the SDDL Core. Each Gateway maintains one independent MR-UDP connection with each mobile node, and is responsible for translating application messages from MR-UDP to the DDS Topic-centric asynchronous communication primitives used within SDDL Core, and vice-versa. All mobile nodes rely on SDDL’s ClientLib, a library used to establish and manage a MR-UDP connection with one or more SDDL Gateways. This library hides MR-UDP protocol details and retransmission issues from the application layer, and also supports an automated, and application-transparent handovers of mobile node between SDDL Gateways. The SDDL Core includes several other specialized services in charge of load balancing, data persistency, data stream processing and group-cast communication, whose explanation can be found in papers [4], [6].

B. Mobile Hub

Since the Mobile Hub plays a crucial role as the intermediary between the sensors and the cloud services, it must have full-fledged processing power, sufficient memory, and network interfaces both to Mobile Internet (2G/3G/4G and Wi-Fi) and to some short-range, low-power WPAN communication technology. We further require the M-Hub to be aware of its current position, e.g. through its GPS sensor or network-based positioning, so that it can geo-tag the probed sensor data. The M-Hub has been designed as a general-purpose middleware component. The goal is that it supports several types and models of sensors and supports several WPAN technologies for discovering and accessing them. Regardless of the heterogeneity and specificity of sensors and WPANs, the functionality of the M-Hub can be classified into the following general tasks:

1. Discovery, identification and monitoring of nearby sensors: periodically, the M-Hub will scan for nearby sensor devices announcing their IDs and capabilities. This information about reachable sensor devices will be kept in the M-Hub database and eventually forwarded to some service of the SDDL Core executing in the cloud.

2. Connecting with the sensors: depending on the WPAN protocol used, the M-Hub may first need to establish a communication link with the sensor, over which it will issue synchronous or asynchronous requests to periodically receive the sensor data.

3. Sensor Data Transcoding: Data packets received from the sensors may have different formats and encodings. Thus, the M-Hub must transcode and serialize them, before transmitting them over the mobile Internet connection. Of these two tasks, the data transcoding is highly dependent on the type and the brand of the sensor device.

4. Caching of probed sensor data: in order to optimize transmission to the cloud over the mobile Internet, the M-Hub may group several sensor data items obtained from several nearby sensors into a single "bulk message" for transmission to the SDDL core. And to do this, it must cache the most recent data items probed from the sensors.

5. Configuring and Probing sensors: depending on the type of sensor, the M-Hub may eventually or periodically send commands, parameter settings, or request-reply data queries over the WPAN to the connected sensors.

6. Pre-processing of sensor data: Sometimes, there is need for some data pre-processing (e.g. normalization, transformation, or comparison with other sensor data) right after sensor probing and prior to serialization and transmission. This data pre-processing should also be done by the M-Hub, so to avoid the waste of the mobile Internet bandwidth for transmitting raw sensor data, and doing this processing close to the “edge of the network”. Unlike, the Transcoding, however, this pre-processing is solely dependent on the sensor type, rather than specific for the WPAN and the sensor device technology.

7. Dynamic deployment and life-cycle management of sensor-specific modules: Since the set of possible sensors to be served by the M-Hub is very large and will always be changing, as new technologies rise and others disappear, it becomes necessary that sensor-specific modules be dynamically installed at the M-Hub on demand, i.e. at the first time it encounters a new sensor device, rather than being pre-installed in the middleware. Moreover, because of the mobile device’s limited memory the M-Hub will have to periodically uninstall some of the sensor modules that are not being used for some time, or that are unlikely to be used in near future. This dynamic management of sensor modules shall be made as fast, efficient and unobtrusive (for the smartphone/tablet user) as possible.

8. Access Authorization and Accounting: this is necessary because the mobile user with the M-Hub might have to register himself and get access authorization from the owner - or operator - of the ambient sensors, who in turn need some guarantee that “its” sensor data will be correctly relayed to the cloud service. On the other hand, a summary of all sensor data transferred by the M-Hub may be necessary for giving to the mobile user some kind of payment (or reputation assessment).

Our current M-Hub prototype, that was implemented on the Android platform supporting Classic Bluetooth and Bluetooth Light Energy (BLE), and so far implements only a subset of the above tasks, namely 1, 2, 3, 4, 6 and 7, where task 3 was implemented only for a specific sensor device, the SensorTag\(^1\), that has 6 embedded sensors.

\(^1\) Texas Instruments CC2541 Sensor Tag - 
http://www.ti.com/lit/ml/swru324b/swru324b.pdf
III. STRUCTURE AND FUNCTIONALITY OF THE M-HUB

As already mentioned, the Mobile Hub (M-Hub) is a general purpose component coordinator executed on a conventional personal mobile device that extends SDDL (see Section II) for access to smart objects with sensors or actuators. For managing uniformly the discovery, connection and data transfer from/to nearby sensors/actuators using different short-range wireless technologies, we designed the Short-range Sensing, Presence & Actuation (S2PA) service. S2PA has an extensible plug-in architecture which allows it to control several short-range WPAN technologies simultaneously from he M-Hub.

A. Short-Range Sensor, Presence and Actuation API

The S2PA was designed to handle any short-range communication with sensor devices, and defines an interface that can be directly mapped to the capabilities of the supported short-range wireless communication technologies (WPANs). To this end, it defines some basic methods and interfaces that all these technologies should implement: 1) Discovery of, and connection to sensor devices, 2) Discovery of services provided by each peripheral/sensor device, 3) Read and write of service attributes (e.g., sensor values, and actuator commands) and 4) Notifications about disconnection of sensor devices. For this, S2PA defines the Technology Interface, shown in Fig 2. The Technology interface includes an ID, defined at programming time, to uniquely identify each technology (e.g. BLE, ANT+, Classic Bluetooth, etc.), and a set of required methods that are sufficient for handling a variety of short-range protocols. For example, methods readSensorValue(), and writeSensorValue(), request a read or write of a sensor, respectively, and serviceName represents the sensor name (e.g., "Temperature", "Humidity"). All relevant information regarding device’s discovery, connectivity, and sensor data obtained from the specific WPAN technology is captured through the TechnologyListener, which is implemented by the S2PA service, and is either cached or directly forwarded to the SDDL Core.

As its first realization, we have implemented S2PA for Bluetooth 4.0 (a.k.a. Bluetooth Smart/Low Energy - BLE) and for Classic Bluetooth. In fact, BLE is emerging as a very promising technology, because it is power efficient, enables fast discovery of peripheral devices and supports approximately 2500 simultaneous connections. However, the most important reason for choosing BLE is the fact that it is now made available on a growing number of Android, iOS, Blackberry smart phone models. Moreover, BLE is being embedded into a growing array of peripheral devices, gadgets, beacons, small Sensor Tags, etc. Despite its higher pairing/connection time, Classic Bluetooth (2.0 and 3.0) is also very important since it is supported by the majority of current peripheral devices (e.g. health and fitness devices, such as the Zephyr BioHarness 3).

B. M-Hubs’s Main Components

The M-Hub is multi-threaded and consists of the following five local services and two managers, all executing in background and in parallel with user apps, as illustrated in Figure 3.

The LocationService is responsible for sampling the M-Hub's current position and attaching it to whatever message is sent to the Gateway (GW), which can be either a static, manually entered geo-point, or the latest geo-coordinate obtained from the smart phone's embedded GPS sensor. The S2PAService implements the TechnologyListener and interacts with all nearby sensor devices that use the supported WPAN technologies. This service is responsible for discovering, monitoring and registration of nearby sensor devices, by periodically doing scans for each supported WPAN. Depending on the kind of interaction (and the WPAN technology capabilities), a communication link may be established with some sensor device, over which the M-Hub will interact in a request-reply mode. Data packets and messages from/to such peripheral devices may have different formats and encodings, so it will also transcode sensor data and commands from the specific device-specific data format to serialized Java objects, for transmission to the GW, and vice versa.

Internet messages are received from - and sent to - the GW by the ConnectionService, which runs the ClientLib for communication with the SDDL Core and, in order to optimize communication over the Internet link, the M-Hub may group several pieces of sensor data or commands assembled by the S2PA Service into a single "bulk message" for transmission. It is also important to mention that some messages (e.g. sensor device connection/disconnection) have a high delivery priority so that they will be relayed directly to the SDDL core, instead of being buffered for further bulk Internet transmission. The periodicity and duration of all of these three services’ actions, is influenced by the device's current energy level (LOW, MEDIUM, HIGH). This will be set by the Energy Manager,
which from time to time samples the device's battery level and checks if it is connected to a power source. Finally, as already mentioned, data packets from sensor devices may have different formats and encodings, raising the necessity of *sensor transcoding modules* (a.k.a. *sensor modules*) that encapsulate the specific transcoding function of the different types of devices. However, as it is unfeasible to have transcoding modules for all possible sensor devices included into the M-Hub’s code, it is necessary to deploy them on-demand. To address this problem, we make use of a **Mobile Client AdaptationService** that is in charge of requesting transcoding modules for new sensor device types from the Adaptation Manager, a server executing at the SDDL Core, storing them locally, and deploying these modules in the S2PA service.

### IV. Dynamic Download and Deployment of Sensor Modules

In order to enable the dynamic download and deployment of sensor modules at the M-Hubs, we have designed and implemented a software layer that supports dynamic software adaptation. Our adaptive approach provides a simple development model, which is based on components, that hide most of the complexity of dynamism management, transparently loading, unloading and updating the M-Hub’s sensor modules. Moreover, our approach supports distributed and transactional software adaptations among M-Hubs by using adaptation plans (i.e., sequences of commands that are executed at the M-Hubs), which are described in [7] [8]. However, considering the M-Hub’s functionalities presented in this paper, we focus only on some local adaptations performed at a single M-Hub. In our design (Fig. 3), we assume that all M-Hubs are executing on top of the same platform (e.g., Android) and version, and that the sensor modules to be deployed execute properly in the M-Hub. In other words, we assume that only correct sensor modules are downloaded and executed in the M-Hub.

Our approach for dynamic software adaptation requires the following two components to execute in the M-Hubs: **Sensor Module Repository** and **Mobile Client Adaptation Service**. The Sensor Module Repository stores sensor modules previously downloaded from the Adaptation Manager (with a global sensor module repository) so to avoid the need of resending sensor modules in case of a M-Hub reboot, for instance. In this way, the M-Hub is able to restore its latest deployed sensor modules. Both local and global repositories are implemented using SQLite [9]. The Mobile Client Adaptation Service is the mechanism that executes the adaptation at the M-Hub (e.g., deployment of a new functionality) and informs the Adaptation Manager whether this operation could be successfully executed or not.

![Fig. 2. Main two interfaces of the S2PA](image)

**Fig. 2. Main two interfaces of the S2PA**

**Fig. 3. Mobile Application Architecture**

In order to reduce latency, battery and network utilization, we apply the concept of local and global repositories. When the M-Hub discovers a new sensor and needs to load a new sensor module for such sensor, as illustrated in Fig. 3, our adaptive layer retrieves the specific module from its local repository and then loads it. In case of a “repository miss”, (similar to cache miss), the local repository requests the sensor module through the Connection Service and the ClientLib from the global sensor module repository, which verifies whether it has the sensor module. If the global repository holds the requested sensor module, it is sent to the M-Hub through the SDDL Gateway to be stored in M-Hub’s Sensor Module Repository and instantiated in the M-Hub. Both repositories are persistent storages. In case of a network disconnection while the Mobile Client Adaptation Service is requesting a sensor module from the global repository, the adaptive layer informs that it did not find the requested sensor module, after a specific timeout.

Two key artifacts are manipulated by the Mobile Client Adaptation Manager during dynamic software adaptation at the M-Hubs: the sensor module itself and a wrapper (i.e., a kind of Java proxy or container). The sensor module is the software artefact that implements the sensor’s data transcoding functions at the M-Hubs and that may be newly installed or updated for a
newer version. When we update a sensor module (A.jar), we have to exchange all previous instances of this sensor module A. For such, we need a mechanism that is able to transparently update the sensor module instances while keeping their references for the application [10] (i.e., M-Hub) since the sensor module’s reference change after it is updated to a newer version. Thus, the wrapper instance, which manages the dynamism related to the process of update the sensor module instance, keep the same even after the update process, avoiding the burden of propagating the new sensor module references to all M-Hub’s components (e.g., S2PA Service and Location Service).

The wrapper’s interface, shown in Fig. 4, has methods to start the dynamic module update process (updateComponentInstance), to commit an update process (commitUpdateComponentInstance), to rollback the update process in case of error (rollbackComponentInstance), and additional two methods to set the current sensor module’s instance (setComponentInstance) and to set the new sensor module’s instance (setFutureComponentInstance), which is a tentative instance that may be committed in the future as the current instance whether no errors occurs during the update process. During the update process, the wrapper may have two sensor module instances; one is the current (and outdated) instance that has to be updated and the other is the new (and updated) instance that will replace the old instance. Thus, the setFutureComponentInstance is applied to set such new sensor module instance into the wrapper. After having the two instances, the updateComponentInstance method may be called to replace the current instance for the new instance, as shown in Fig. 5. The rollbackComponentInstance and commitUpdateComponentInstance methods are applied for transactional adaptations, which are not the focus of this paper.

In order to be managed by the wrapper, the sensor module must implement a simple interface, shown in Fig. 6. With the IComponent interface, the wrapper is able to initialize a sensor module and to transfer the state (i.e., any variables and values that has to be passed to copy the state from the old to the new instance) from one sensor module to another, as illustrated in Fig. 7. It is important to note that the IComponent interface has a generic State type, which enables to handle any sort of state.

Some adaptive capabilities are implemented using Java reflection in order to enable us to dynamically add new JARs (Java Archives) into the application’s classloader and to instantiate sensor modules. Most of our implementation uses software engineering designs (e.g., interfaces and abstract classes) and programming techniques (e.g., Java generic types) in order to reduce the use of computational reflection, since it introduces a considerable overhead at runtime [11], [12]. More details about the adaptive layer are found in [7] [8].

![Fig. 4. IWrapper interface](image)

```java
public interface IWrapper {
    public void setComponentInstance(TComponent instance);
    public void setFutureComponentInstance(TComponent instance);
    public void rollbackComponentInstance();
    public void commitUpdateComponentInstance();
    public void updateComponentInstance(boolean transactionalAdaptation);
}
```

![Fig. 5. Code that demonstrates the update process in a non-transactional manner](image)

```java
componentInstance = this.newComponentInstance(componentName);
componentWrapper.setFutureComponentInstance(componentInstance);
componentWrapper.updateComponentInstance(false);
```

![Fig. 6. IComponent interface](image)

```java
public interface IComponent {
    public boolean initialize(Parameters state);
    public boolean loadState(State state);
    public State getState();
}
```

![Fig. 7. Code that transfers the state from the old the new instance in an update process](image)

```java
this.futureComponentInstance.loadState(
    (this.componentInstance, getState());
```

From a global perspective, we assume that extensions and modifications to the set of sensors used in a crowd-sensing application using the M-Hub are to be initiated by the provider or owner of the new/updated sensor devices, as follows: he/she will submit the sensor device specs to the developer/operator of the crowd-sensing application, who will then develop the transcoding module and its wrapper, and produce corresponding metadata about the new sensor (e.g. in Transducer Markup Language) with information about sensor make and type, the owner’s network domain name, data types and sizes, ordering and arrangement, calibration information, units of measurement, information about uncertainty, and physical phenomena relating to the data, etc. The sensor module and the metadata will be stored in the global sensor module repository of the Adaptation Manager, and digitally signed by the application developer. From this point on, updates of sensor modules at the M-Hub will be done automatically, and on demand: whenever an M-Hub discovers and connects to the new sensor device, it will obtain its type, name, the owner’s domain name, etc., and query the Adaptation Manager’s repository for the required transcoding module. If available, it will download the module, verify the digital signature, store the module into the local Sensor Module Repository and install it in the S2PA service. If the module is not available, or the signature does not match, then the M-Hub simply ignores the discovered sensor device.

V. INITIAL PERFORMANCE EXPERIMENTS AND RESULTS

In this section, we present preliminary results of experiments where we measured the total time spent by the M-Hub for discovering, connecting and receiving data from sensors using Classic Bluetooth and BLE technologies. For
sensors with BLE technology, we present the aforementioned total time in several scenarios; where the Sensor Modules are already installed versus when they have to be first downloaded to M-Hub’s local module repository; when the download happens with WI-FI connection, versus 3G Internet connectivity. For all experiments, we used a Moto X smartphone executing Android 4.4.4, and the S2PA Service was configured to perform a WPAN scan every 3 seconds with duration of 2 seconds. We also show some results of memory, energy and CPU consumption by the M-Hub.

A. M-Hub Evaluation Using Bluetooth Classic Sensors

For the evaluation of the M-Hub with Bluetooth Classic technology, we used the Zephyr BioHarness-3 sensor. The sensor used in this experiment was already paired with the mobile device before the connection attempt. The evaluation parameters considered were: Connection Time (CoT); Time to Receive the first value after the connection was established (TR); and Total Time to Receive the first data (TTR), which corresponds to CoT + TR. All values were measured in seconds. The time for discovering the services implemented by the sensor device was not measured, because for some sensors, such as the Zephyr BioHarness-3, there is no service discovery step. The M-Hub must already be aware of the services that this sensor implements and just needs to periodically send a signal message for keep receiving the sensor data. However, other sensors such as the Zephyr HxM BT do not operate in the same way. In this case, the sensor simply starts sending all available data as soon as the connection is established (also without requiring a service discovery procedure). We ran each experiment 12 times and calculated the mean value and the standard deviation. The obtained results are presented in Table I.

<table>
<thead>
<tr>
<th>First Connection</th>
<th>CoT (s)</th>
<th>TR (s)</th>
<th>TTR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean value</td>
<td>1.39415</td>
<td>0.21756</td>
<td>1.61171</td>
</tr>
<tr>
<td>Std. Deviation</td>
<td>0.29737</td>
<td>0.31839</td>
<td></td>
</tr>
</tbody>
</table>

B. M-Hub connecting to BLE Sensors with installed Sensor Module

For the evaluation of the M-Hub accessing sensors through BLE technology we used four TI SensorTags. The evaluation parameters considered were: Connection Time (CoT); Service Discovery Time (SDT); Time to Receive the first value (TR); and Total Time to Receive the first data (TTR), measured in seconds. We collected these parameters for the first connection of the M-Hub with the sensor devices, and for follow-up reconnections, both with the corresponding Sensor Module already installed. The TTR corresponds to the sum of CoT, SDT and TR, which corresponds to the total time required for receiving the first valid read from a sensor since the SensorTag was discovered. We ran each experiment 12 times and calculated the mean value and the standard deviation. It is also important to remark that the behavior of BLE in Android is synchronous, so the operations like connections to each of the sensor devices is done sequentially. The obtained results are presented in Table II.

<table>
<thead>
<tr>
<th>First Connection</th>
<th>CoT (s)</th>
<th>SDT (s)</th>
<th>TR (s)</th>
<th>TTR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean value</td>
<td>0.21192</td>
<td>0.132</td>
<td>0.64958</td>
<td>9.9935</td>
</tr>
<tr>
<td>Std. Deviation</td>
<td>0.1578</td>
<td>0.14381</td>
<td>0.1511</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reconnections</th>
<th>CoT (s)</th>
<th>SDT (s)</th>
<th>TR (s)</th>
<th>TTR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean value</td>
<td>0.2325</td>
<td>0.1552</td>
<td>0.64952</td>
<td>1.0372</td>
</tr>
<tr>
<td>Std. Deviation</td>
<td>0.10007</td>
<td>0.01609</td>
<td>0.1509</td>
<td></td>
</tr>
</tbody>
</table>

The services discovery for BLE technology is the slowest operation, since it must find all the services’ characteristics and descriptors that the sensor device possesses. Thus, this delay is proportional to the number of services, which in the case of the SensorTag is 6, as it has 6 sensors. However, this services discovery time decreases very much with follow-up reconnections, as can be seen from Table II.

C. M-Hub connecting to BLE Sensors requiring download of Sensor Module

We now repeated the experiment with the SensorTags, but this time the Sensor Module was not installed in the M-Hub, but had to be downloaded from the Global Module Repository through a server executing in SDDL Core in the cloud. First, we measured the time it takes for the M-Hub (with Wi-Fi connectivity), to get the module from the server in two cases: when Global Repository only contained this single required sensor module, and when it stored a 1,000 different sensor modules. The obtained results are presented in the first part of Table III.

In a second experiment, we repeated the same measurements with the M-Hub connected only through 3G mobile Internet. The goal was to see how the TTR would be influenced by the type of mobile Internet connectivity and discover if the Sensor Module download and deployment under 3G connectivity would still render feasible the opportunistic mobile sensing. Again, we ran each experiment 12 times and calculated the mean value and the standard deviation. The download and install time required by the M-Hub with 3G are presented in the second part of Table III.

<table>
<thead>
<tr>
<th>Module’s Download</th>
<th>Wi-Fi</th>
<th>1 Module (s)</th>
<th>1000 Modules (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean value</td>
<td>1.9556</td>
<td>2.1254</td>
<td></td>
</tr>
<tr>
<td>Std. Deviation</td>
<td>0.55918</td>
<td>0.00795</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Module’s Download</th>
<th>3G</th>
<th>1 Module (s)</th>
<th>1000 Modules (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean value</td>
<td>5.1168</td>
<td>6.2298</td>
<td></td>
</tr>
<tr>
<td>Std. Deviation</td>
<td>2.1149</td>
<td>1.14991</td>
<td></td>
</tr>
</tbody>
</table>

As a matter of comparison, the ping delay of the 3G and Wi-Fi connections used was around 71ms and 7ms, respectively.
We also measured the time to install the sensor module in S2PA when it has already been downloaded and is in the M-Hub’s local Sensor Module Repository. And in the case that the required sensor module of the discovered sensor is neither in the global nor in the local Module Repository, then the M-Hub has some timeout to know that it will not be able to get data from this sensor. These times are shown in Table IV.

TABLE IV. Performance for the Load of the Stored Modules

<table>
<thead>
<tr>
<th>SensorTags</th>
<th>RAM(MB)</th>
<th>CPU(%)</th>
<th>PC(mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>22</td>
<td>0.4</td>
<td>52</td>
</tr>
<tr>
<td>3</td>
<td>26</td>
<td>3.2</td>
<td>326</td>
</tr>
<tr>
<td>6</td>
<td>26</td>
<td>5.6</td>
<td>339</td>
</tr>
</tbody>
</table>

D. Energy and Memory Overhead

To measure memory, CPU and energy usage by the M-Hub, we ran three experiments that lasted one hour each. The S2PA was configured to make WPAN scans every 60 seconds with duration of 2 seconds, and the Connection Service to send the bulk messages every two minutes. For the first experiment, we started the M-Hub without any SensorTag in its vicinity. In the second, we included three tags, and in the third experiment added another three tags. Table V summarizes the memory, CPU and energy usage, where RAM is the average memory consumed in Megabytes, CPU gives the percentage of CPU usage, and PC, is the average power consumption, in mW (milliWatts).

TABLE V. Resources Consumption of the M-Hub

<table>
<thead>
<tr>
<th># SensorTags</th>
<th>RAM(MB)</th>
<th>CPU(%)</th>
<th>PC(mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>22</td>
<td>0.4</td>
<td>52</td>
</tr>
<tr>
<td>3</td>
<td>26</td>
<td>3.2</td>
<td>326</td>
</tr>
<tr>
<td>6</td>
<td>26</td>
<td>5.6</td>
<td>339</td>
</tr>
</tbody>
</table>

E. Analysis of the Results

Adding the measured times of experiments B and C, we obtain the Total Time to Receive the first data (TTR) for the First Connection and the Reconnection, with Wi-Fi and 3G connectivity, which are shown in Table VI.

TABLE VI. Performance for Download of a Module with First Connection and with Reconnection

<table>
<thead>
<tr>
<th>Mean value of TTR with download (in s)</th>
<th>Wi-Fi connection</th>
<th>3G connection</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Connection</td>
<td>11.9491</td>
<td>15.1084</td>
</tr>
<tr>
<td>Reconnection</td>
<td>2.9928</td>
<td>6.154</td>
</tr>
</tbody>
</table>

VI. RELATED WORK

In this section, we compare our middleware with other work on mobile sensing, and more especially, with adaptive middleware proposals. The work in [22] proposes an architecture that leverages the increasingly ubiquitous presence of Bluetooth Low Energy to connect smart devices and sensors to the Internet through smartphones. The paper explains how gateways should be configured by using the meta information that each IoT device must provide, and address other important problems as user incentivization and security. However, different from our approach, they do not focus on support for several WPANs, and deal with the problem of dynamically installing specific software modules for unknown sensor devices. To the best of our knowledge, the work by Perera et al [13] is the closest work of ours. The authors propose a mobile application - Mobile Sensor Hub (MoSHub) - that allows a mobile phone to connect to a variety of different sensors and therefore collects, combines, processes, and sends sensor data to a server. A software architecture was developed to dynamically interconnect sensors to a mobile application (on smartphones) by generating a wrapper class based on a XML (eXtensible Markup Language) called Sensor Device Definition (SDD), which describes the sensor’s capabilities. Thus, whenever the MoSHub needs to connect to a sensor, it sends its SDD to the server. It then dynamically creates a wrapper class with its corresponding API, stores it in a repository, and then sends the wrapper to MoSHub. If the server receives requests from other MoSHub, but with the same capabilities it will directly send the already existing wrapper found in the repository. After generating MoSHub’s wrapper, a virtual sensor definition is created. A virtual sensor is an abstraction that hides implementation details sensor data access, similar to our sensor modules. While MoSHub approach is tailored for generating wrappers class for sensor devices, our approach is more general and intends to support general-purpose dynamic adaptation on mobile devices through software components, such as M-Hub’s sensor modules. The authors do not present performance results of their software for different wireless networks, and also omit any discussion about scalability, tolerating faults, distributed adaptation and connection problems, for instance.

Several related works uses the Android platform, primary due to its open and extensive nature. Zheng et al. [14] proposes BraceForce, a middleware platform that incorporates event and
model-driven concepts to provide efficient and simpler access abstractions to sensing devices in mobile applications. Using a similar approach to ours, they require sensor devices to implement a standard Java interface (sensor driver) to be manipulated by the middleware. This API contains essential commands (e.g., open, close), configurations (e.g., start, restart), and communication primitives for sending and receiving key-value data. However, BraceForce does not address or provides operations to retrieve or restore a sensor state. Although BraceForce can discover sensor devices, it can only do this for known devices, i.e., using pre-developed sensors drivers. Thus, contrary to our approach, it is not capable to adapt (load/unload sensor modules) for access of previously unknown sensor devices.

Lee and Chung [15] propose SlimWare, a middleware that combines reflective components concepts with event-based communication to produce a lightweight middleware platform for adaptive mobility sensing. SlimWare’s primary feature is the ability to dynamic load and unload components based on the user context, e.g., when a user is on 3G connectivity the middleware can unload the application location component. Their main motivation is with minimizing resource usage on different scenarios by loading and unloading components, while our work is concerned with sensing, discovering and loading components on demand. The paper suggest a series of parameters, such as user location, interface usage (reference counting), and memory status, to be considered in the dynamic changing process; however they do not specify nor discuss the process algorithm to do so. Further, the authors neither implemented nor evaluated their approach.

Yet another related work is the one by Dar et al [16]. They present an architectural model addressing flexible and adaptive composition of services in Very Large Scale (VLS) IoT systems by exploiting the concepts of service orchestration (i.e., centralized approach) and choreography (i.e., decentralized approach). While the authors follow a service orchestration/choreography model, which seems to be more adequate for web applications, we chose to follow the service-oriented component approach. Although the authors address VLS IoT systems, they are targeting components re-configuration rather than component discovering. In addition, there is no information about how the architecture achieves scalability, and how the adaptation engineer defines the service composition. The work is on early stage, and as with the previous work, neither implementation nor evaluation is presented.

iPOJO (injected Plain Old Java Object) [10] is a service-oriented component framework that aims to simplify the development of dynamic service applications. The iPOJO framework is implemented on top of the OSGI (Open Service Gateway initiative) [17] service platform, which is a framework to deploy services in a centralized and non-distributed environment. The basic idea of iPOJO is using containers to inject handlers with the aim of managing non-functional behavior, such as dynamism and service discovery. The most remarkable differences among iPOJO and our approach is that it completely adheres to the Service-Oriented Computing (SOC) and there is no notion of distributed adaptation that involves many mobile devices. On the other hand, by being inspired by SOC, we can create an adaptation plan logic to decide which components (or services) have to be deployed on each mobile device. Thereby, we can enable the development of autonomic managers in charge of managing the whole system. Although such differences, we share ideas such as the concept of containers (or wrappers) and handlers, and the separation among the business logic and the non-functional properties such as dynamism.

In addition, there are other research projects that address the adaptation problem, such as [18]–[21] that had some indirect similarity with our solution.

![Fig. 6 – Parts of the M-Hub screen with the Gateway (Internet) connection and energy settings, four discovered SensorTags and the sensor values obtained from the first sensor device.](image)

VII. CONCLUSION AND NEXT STEPS

In this paper, we presented a mobile-cloud middleware for opportunistic mobile sensing using conventional smart phones. These devices execute the Mobile Hub, which can support several WPAN technologies and has the capacity to dynamically download and install sensor-specific transcoding modules according to the discovered sensor type and make. Our prototype implementation (see Figure 6) of the M-Hub is based on Android KitKat and has support for Classic Bluetooth and Bluetooth Low Energy (BLE).

Using this prototype implementation and BLE SensorTag devices (with 6 sensors) we did performance experiments that measured the time interval from sensor discovery until the reception of the first sensor data, in the case where download of the sensor module from a cloud server was required. These experiments were done for both Wi-Fi and 3G Internet connections.

The results obtained from these initial experiments are quite encouraging and show that even with the download through a 3G connection the total time to receive the sensor data is, in average, 15 seconds, and that moreover, approx. 10
seconds of this delay is due to BLE’s services discovery process, which takes 1.7 seconds per sensor.

In particular, the obtained results show that for peripheral devices and sensors with BLE (with a wireless coverage of 50-100 meters) a smartphone (with the M-Hub) of a pedestrian walking at 5 km/h along an environment sensor would have sufficient time to download the sensor module and collect data from this sensor. This suggests the feasibility of opportunistic mobile sensing environment as described in this paper, using our middleware.

So far, our M-Hub prototype requires the smartphone’s CPU and Bluetooth interface to be active all time, and only performs very simple energy-aware control over the WPAN scanning, the SDDL communication and probing of the geo-location. In future, we will implement some more sophisticated energy management that will take into account also the responsiveness requirements of the crowd-sensing application.

Many other interesting issues remain to be researched, prototyped and tested. As next steps, we plan to do more measurements and realistic experiments (e.g., walk-by BLE sensor access). Regarding M-Hub development, we will implement the management of sensor modules in the M-Hub’s local Repository, the parallelization of the BLE service discovery and the module download functions, and implement a WPAN technology plug-in for ANT+.

ACKNOWLEDGMENT

This work is partially supported by CNPq (National Council for Scientific and Technological Development) and FAPEMA.

REFERENCES


