Dynamic Configuration of Sensors Using Mobile Sensor Hub in Internet of Things Paradigm

Charith Perera [*#1], Prem Jayaraman [*2], Arkady Zaslavsky [*3], Peter Christen [#4], Dimitrios Georgakopoulos [*5]

[#]*Research School of Computer Science, The Australian National University, Canberra, ACT 0200, Australia*
[4]peter.christen@anu.edu.au

[*]*CSIRO ICT Center, Canberra, ACT 2601, Australia*
[1]charith.perera@csiro.au, [2]prem.jayaraman@csiro.au, [3]arkady.zaslavsky@csiro.au,
[5]dimitrios.georgakopoulos@csiro.au

Abstract—**Internet of Things (IoT) envisions billions of sensors to be connected to the Internet. By deploying intelligent low-level computational devices such as mobile phones in-between sensors and cloud servers, we can reduce data communication with the use of intelligent processing such as fusing and filtering sensor data, which saves significant amount of energy. This is also ideal for real world sensor deployments where connecting sensors directly to a computer or to the Internet is not practical. Most of the leading IoT middleware solutions require manual and labour intensive tasks to be completed in order to connect a mobile phone to them. In this paper we present a mobile application called *Mobile Sensor Hub* (*MoSHub*). It allows variety of different sensors to be connected to a mobile phone and send the data to the cloud intelligently reducing network communication. Specifically, we explore techniques that allow MoSHub to be connected to cloud based IoT middleware solutions autonomously. For our experiments, we employed Global Sensor Network (GSN) middleware to implement and evaluate our approach. Such automated configuration reduces significant amount of manual labour that need to be performed by technical experts otherwise. We also evaluated different methods that can be used to automate the configuration process.**

I. Introduction

As we are moving towards the Internet of Things (IoT), the number of sensors deployed around the world is growing at a rapid pace. Market research has shown a significant growth of sensor deployments over the past decade and has predicted a significant increment of the growth rate in the future. Due to advances in sensor technology, sensors are getting more powerful, cheaper and smaller in size, which has stimulated large scale deployments. Ultimately, these sensors will generate *big data* [1]. As shown in Figure 2, communication of data from a sensor to a cloud application (or middleware) costs significant amount of energy in comparison to local data processing. Minimizing such communication using intelligent filtering and fusing techniques will save enormous amount of cost in IoT paradigm due to the magnitude.

Typical structure of a sensor network is presented in Figure 1. It comprises the most common components in a sensor network. As we have shown, with the orange coloured arrows, data flows from right to left. Data is generated by the low-end sensor nodes and high-end sensor nodes. Then, data is collected by mobile and static sink nodes. The sink nodes send the data to low-end computational devices. These devices

perform a certain amount of processing on the sensor data. Then, the data is sent to high-end computational devices to be processed further. Finally, data reaches the cloud where it will be shared, stored, and processed significantly.
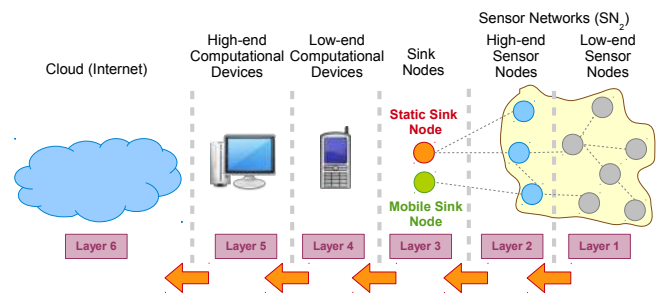

Fig. 1. Layered structure of a sensor network: These layers are identified based on the capabilities posed by the devices. In IoT, this layered architecture may have additional number of sub layers as it is expected to comprises large variety of sensing capabilities.
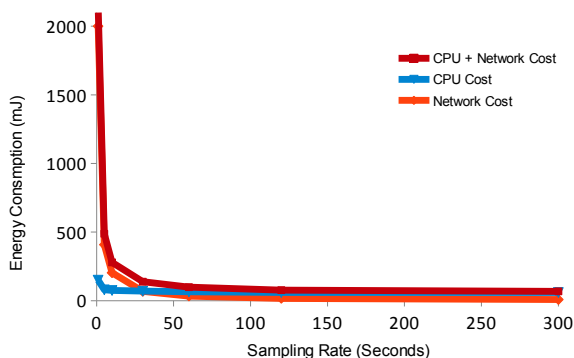
Based on the capabilities of the devices involved in a sensor network, we have identified six layers. Information can be processed in any layer. Capability means the processing, memory, communication, and energy capacity. Capabilities increase from layer one to layer six. Based on our identification of layers, it is evident that an ideal system should understand the capability differences, and perform data management accordingly. For example, processing in the first three layers could reduce data communication. However, devices in the first three layers do not have a sufficient amount of energy and processing power to do comprehensive data processing. Mobile phones have computational capabilities so it can fuse and filter data, which will help to reduce communication cost.

The rest of this paper is organized as follows: Section II describes the background and related work. The motivations and our contribution of this paper is presented in Section III. In Section IV, we explain our proposed solution including MoSHub application in detail. Implementation and results of the evaluations are presented in Section V and VI respectively. Section VII summarizes the lessons learned from this research.

II. Background and Related Work

This work is based on two of our previous research efforts. In [2], we proposed a model called DAM4GSN that

captures data using sensors built into the mobile phones. In this paper, we extend the support towards connecting external sensor. Further, we improved GSN middleware in such a way that it can dynamically generate custom wrappers[1] for each MoSHub at run time instead of using a generic wrapper. In [3], we proposed ASCM4GSN architecture that automates the process of developing wrappers. We developed a XML based specification called Sensor Device Definition (SDD) that is capable of generating GSN wrappers. Focus of paper [3] was to generate wrapper for sensors that uses manufacture released APIs. However, we utilized this technique in this work to generate customized wrappers for each MoSHub.

There are several other commercial solutions available: TWINE (supermechanical.com), Ninja Blocks (ninjablocks.com), and Smart Things (smartthings.com). All these solutions focus on event detection using If-THEN rules in smart environments. However, none of them support complex query processing capabilities similar to GSN. Their, automated configuration works only with limited number of devices they supports. Further, our plugin architecture allows to add more capabilities to MoSHub via existing app stores.

The Global Sensor Network (GSN) [4], the IoT middleware we employed in this work, is a platform aimed at providing flexible middleware to address the challenges of sensor data integration and distributed query processing. It is a generic data stream processing engine. GSN has gone beyond the traditional sensor network research efforts such as routing, data aggregation, and energy optimisation. The design of GSN is based on four basic principles: simplicity, adaptivity, scalability, and light-weight implementation. GSN middleware simplifies the procedure of connecting heterogeneous sensor devices to applications. Specifically, GSN provides the capability to integrate, discover, combine, query, and filter sensor data through a declarative XML-based language and enables zero-programming deployment and management. Further, we are engaged in extending GSN middleware towards OpenIoT [5] by adding more capabilities. The above reasons lead us to choose GSN for our experiments. Our findings do not depend on any specific middleware and remain open to be used in any solution that needs mobile devices to be used as sensor hubs.

Fig. 2. This graph shows how the energy consumption of Mobile Sensor Hub varies with the sampling rate. Only the network communication between server and MoSHub is considered.

[1]A program code (e.g. Java) that directly communicates with a sensor and retrieves data. It is a Java class that adheres to a specification.

III. MOTIVATIONS AND OUR CONTRIBUTION

As depicted in Figure 1, sensor data passes through different layers. In order to save network communication cost, we should be able to filter sensor data. Low computational devices such as mobile phones can be used to achieve above task. However, we need to understand what kind of operations can be done in mobile phones and what kind of operation need to be performed in server computers depending on their complexity and CPU cost. State-of-the-art mobile phones have many capabilities that make them ideal to be used in sensor data management in IoT domain. Mobile phones have built in wireless communication capabilities such as bluetooth, and WiFi. Further, these capabilities can be extended by connecting ZibBee modules via microUSB ports. Therefore, mobile phones can ideally be used as sensor data collecting devices. Technologies such as 3G and 4G allows transferring collected data to the cloud from place where WiFi networks are not available (e.g. environmental monitoring and agricultural domain). Latest mobile phones have up to 1.7 GHz Dual or Quad Krait CPU, 2 GB RAM and 8 GB internal storage. Therefore, mobile phones can ideally be used as sensor data processing devices as well.

The above-mentioned capabilities show that mobile phones can be used as hubs. Mobile phones can offer the functionality of collect, process, and communicate sensor data to the cloud for further processing. The research challenge is *how to connect mobile phones into an IoT middleware solution autonomously*. At a given time, variety of different sensors may connected to a mobile phone locally. However, IoT middleware does not know about details of those sensors (what type of data or how much data to be expected from each mobile phone). In such a situation, *how an IoT middleware can be configured autonomously so it can accept data streams send by mobile phones or similar devices*. This is the research question we addressed in this paper.

(a) Sensors built-in to the mobile phone

(b) External sensors that communicate with mobile phones

Fig. 3. Sensors in Smart Environments

The vision of the IoT is heavily energised by statistics and predictions. We present some statistics to justify our focus on the IoT and mobile computing and to show the magnitude of the challenges. It is estimated that there about 1.5 billion Internet-enabled PCs and over 1 billion Internet-enabled mobile phones today. These two categories will be joined with Internet-enabled devices (smart objects [6])) in the future. By 2020, there will be 50 to 100 billion devices connected to the Internet [7].

In this paper, we propose a lightweight mobile application, called *Mobile Sensor Hub* (MoSHub), which allows to connect and retrieve sensor data using wireless communication techniques such as WiFi and bluetooth from external sensors easily. We employed a plug-in architecture called Android Interface Definition Language (AIDL)[2] provided in Android platform to facilitate plug and play configuration between external sensors and MoSHub application.

We propose a model that can configure the communication between mobile phone-based MoSHub and server-based GSN middleware. This automated configuration reduces significant amount of manual labour need to be performed by technical experts otherwise. We extended and applied our previously proposed ASCM4GSN [3] approach to generate programming code at runtime which enable dynamic configuration.

Finally, we compare two possible approaches that can be used to automate the configuration in many perspective as presented in Section V in order to explore the most suitable approach to be used in MoSHub. We also carried out preliminary evaluations on scalability of the plug-in architecture.

IV. OUR APPROACH

In this section, we discuss our proposed solution in detail. First, we provide a high-level overview of our approach. Next, we explain how both client side and server side autonomous configuration works. Then, we describe the MoSHub application. Throughout our discussion, we highlight possible alternative approaches and justifications on our choice. In Section V, we evaluate and justify what approach is more appropriate based on experimentation results.

A. High-level Overview of the System

The high level communication between MoSHub and GSN server is depicted in Figure 4. We can explain how automated configuration works in order of activities as follows. MoSHub is an application that need to be installed in an Android mobile phone, which is intended collect data from both the internal and external sensors. Even though we use the term mobile phone, in actual architecture what we need is some device that has the capabilities, similar to a mobile phone, such as WiFi, bluetooth, CPU, memory. In the future, we expect there would be devices, powered by Android, specifically design for IoT paradigm. Such environment will add more value to our research and open up more opportunities. Different types of sensors can be connected to MoSHub via different wireless
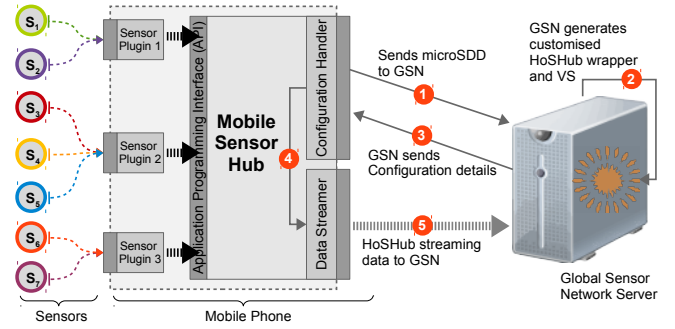
Fig. 4. Main steps of the automated dynamic configuration process that connects a MoSHub to a GSN server. Numbers show the order of execution.

technologies. Then, MoSHub generates a micro sensor device definition (μSDD) file based on sensors connected to it. μSDD is different from GSN virtual sensor definition (VS) and it is somewhat similar to SDD [3]. Figure 5 shows a μSDD definition file snippet.

```
<micro-sensor-device-definition name="Samsung_Galaxy_S_I9000">
.............
    <addressing>
        <predicate key="geographical">CSIRO ICT Center, Canberra</predicate>
        <predicate key="Device">Samsung Galaxy S I9000</predicate>
        <predicate key="LATITUDE">-35.275291</predicate>
        <predicate key="LONGITUDE">149.120585</predicate>
    </addressing>
    <data-structure>
        <data-field field-name="battery_voltage" type="double"
                description="Battery voltage of the sensor device"/>
        <data-field field-name="temperature_s1" type="double"
                description="Measures temperature"/>
        <data-field field-name="temperature_s2" type="double"
                description="Measures temperature"/>
        <data-field field-name="humidity" type="double"
                description="Measures humidity"/>
        <data-field field-name="pressure" type="double"
                description="Measures pressure"/>
    </data-structure>
.............
</micro-sensor-device-definition>
```

Fig. 5. Sample μSDD snippet which contains information about data produce by all the sensors connected to a MoSHub and context such a location.

The reason for generating a μSDD without directly generating a SDD is two fold. First, though both definitions share some amount of similarities, they should be able to extended independently from the each other depending on the requirements arises in the future. Second reason is the network communication. We want keep the packet size to the minimum, which will save energy that it take to generate the file as well as in network communication. Further, keeping only the minimum amount of information that is specific to each situation makes it easier and faster to process.

MoSHub sends the μSDD to the GSN server. GSN server then process the μSDD and generates a GSN wrapper class file that is specific to each individual MoSHub. GSN automatically compile the newly generated class file and add it to the wrapper repository. However, before generating a new class file GSN search for an existing matching class. In such case, GSN will use that class instead of generating a new one. The process of generating a wrapper based on a given SDD specification is described in our previous work [3]. In that perspective, μSDD acts same as the SDD.

Figure 6 depicts the structure of a GSN wrapper class. The content of the class would be generated based on the information provided in μSDD. Explanations are provided in

[2]. There are five methods in a typical GSN wrapper class. Method (1) runs only once and method (2) to (4) may run occasionally. In contrast, method (5) will run every time when a new data stream receives.

```
public class MoSHub001Wrapper extends AbstractWrapper {
  public boolean initialize ( ) {    ❶
    1. Analyse the micro sensor device definition and understand
       what types of data is going to be received
    2. Create data structures
  }
  public void run ( ) {    ❺
    while ( isActive( ) ) {
      1. Wait for the client to send Sensor data (Listen to a given port)
      2. Map sensor data to data strictures (Sensor Data, Data Structures)
      ….........................
      StreamElement streamElement = new StreamElement (....);
      postStreamElement( streamElement )
    }
  }
  public  DataField[] getOutputFormat ( ) { .... }    ❷
  public String getWrapperName( ) {.... }    ❸
  public void finalize ( ) {....}    ❹
}
```

Fig. 6.   The Structure of a Typical MoSHub Wrapper

We tested another approach that can be used to achieve above functionality. Without creating customize wrapper for each MoSHub, we developed a generic MoSHub wrapper that can retrieve any amount of sensor data. This generic MoSHub wrapper can configure its internal data structures depending on how many data items are sent by each MoHub. However, during our performance evaluation, it was found that using generic wrapper is inefficient compared to generating customised wrapper for each sensor. Details are presented in Section V and VII. Another ongoing study, we are conducting focusing on adding context discovery functionality to GSN middleware, also showed that generating customized wrapper for each MoSHub approach is better in term of extensibility.

After generating MoSHub wrapper, GSN generates a virtual sensor definition (VSD) using the information provided in the μSDD. GSN VSD is explained in details in [4]. Even though a virtual sensor definition can combine data coming from multiple wrappers, in default automated configuration process, GSN creates a dedicated virtual sensor definition for each MoSHub wrapper. Figure 7 presents a sample MoSHub virtual sensor definition. When GSN generates a VSD file, it triggers the virtual sensor creation processes. This process triggers the specified wrapper to be created. The wrapper that correspond to each stream source is defined under the address element in the VSD file. This process sends a Wrapper Connection Request (WCR) to the wrapper repository in the GSN server. WCR is an object, which contains a wrapper name and its initialisation parameters as defined in the virtual sensor definition. Whenever a WCR is generated at the virtual sensor loader, it will be sent to the wrapper repository. Then, steps are followed as depicted in Figure 8. A detailed description of this process is presented in [2].

Once the wrapper instance is ready to receive data, GSN sends configuration detail to the configuration handler of the MoSHub application. This information contains the port number where MoSHub needs to send data. At this point, automated configuration process completes. Finally, MoSHub starts streaming data to the GSN. When a new sensor connects

```xml
<virtual-sensor name="MobileSensorHub001" priority="10">
    <processing-class>
        <class-name>gsn.vsensor.BridgeVirtualSensor</class-name>
        <output-structure>
            <field name="battery_voltage" type="double" />
            <field name="temperature_s1" type="double" />
            <field name="temperature_s2" type="double" />
            <field name="humidity" type="double" />
            <field name="pressure" type="double" />
        </output-structure>
    </processing-class>
    <description>This Mobile Sensor Hub captures sensor reading from five
                external sensors </description>
    <life-cycle pool-size="10" />
    <addressing>
        <predicate key="geographical">CSIRO ICT Center, Canberra</predicate>
        <predicate key="LATITUDE">-35.275291</predicate>
        <predicate key="LONGITUDE">149.120585</predicate>
    </addressing>
    <storage history-size="5m" />
    <streams>
        <stream name="input1">
            <source alias="source1" sampling-rate="1" storage-size="1">
                <address wrapper="MobileSensorHub001"></address>
                <query>SELECT battery_voltage,temperature_s1,
                    temperature_s2, humidity, pressure, timed FROM wrapper
                </query>
            </source>
            <query>SELECT battery_voltage,temperature_s1,
                temperature_s2, humidity, pressure, timed FROM source1
            </query>
        </stream>
    </streams>
</virtual-sensor>
```

Fig. 7.   Virtual sensor definition (VSD) generated by GSN middleware during the automated configuration process using μSDD which sends by MoSHub.
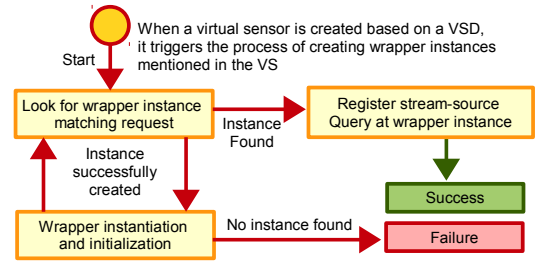

Fig. 8.   Wrapper life cycle of the GSN middleware

to MoSHub or existing sensor disconnects from MoSHub, the automated configuration process need to be executed again.

B. Mobile Sensor Hub (MoSHub)

MoSHub is a mobile application that collects, combines, processes, and sends sensor data to a GSN server. Communication between external sensors and MoSHub is conducted through independent software layer called *plug-ins*. MoSHub provides a specification that defines how developers should develop plug-ins that will be able to communicate with MoSHub application. Due to space limitation, we do not describe those specifications in this paper. In brief, the specification guides the developers on how to name their plug-ins, packages, and provides an interface (including list of methods need to be implemented, common data structures and so on) as an aidl file. The operations that can be conducted by a given plug-in is limited only by developers capability and Android platform. As long as plug-ins are adhered to the provided specification, they will be able to communicate with MoSHub application.

In order to generate μSDD file, MoSHub communicates with every active plug-in that collects data from a external or internal sensor. Each plug-in should at least provide the category/name of the sensor they are communicating with (e.g. temperature_s1) and type of data that connected sensor generates (e.g. int, double, string). Once MoSHub gathers

minimum amount of information from all the plug-ins, it generates the μSDD file. It may also include available context information such as location.

V. IMPLEMENTATION

We conducted all evaluations and experiments using a Samsung Galaxy S GT-I9000 mobile phone, which runs Android platform 2.3.6. GSN middleware was installed on a laptop with Intel Core i5 CPU and 4GB RAM. Network communications are conducted through CSIRO ICT centre WiFi network. Figure 9 shows a web interface of GSN middleware and Figure 10 shows main user screens of the MoSHub application.

Fig. 9. This is a sample web based user interface of the GSN middleware that shows a MoSHubs is connected to it.

Fig. 10. These are main user interfaces of the MoSHup application. (a) shows list of sensors that are connected to MoSHup though different plug-ins. (b) shows configuration screen of the MoSHup application.

VI. EVALUATION

In this section, we present results of several experiments we conducted in order to evaluate the performance and suitability of the approaches we proposed. Figure 11 graph shows a comparison of two approaches we explained earlier, in Section IV, in four different perspectives. Four comparisons are conducted using different measurement units: processing time (in milliseconds), memory (KB), lines of code (number of lines), and automated configuration time (in milliseconds). A MoSHub with eight sensors connected to it used for evaluations. In order to combine all perspectives into one graphs, we converted all of them to percentages. Static predefined single wrapper (SPSW) approach is kept as 100%. Dynamically generated customized wrappers (DGCW) approach is graphed in compared to SPSW. Therefore, this graph show how much DGCW approach is efficient or inefficient compared to SPSW as a percentage. For example, DGCW takes 18% less processing time

than SPSW. Figure 12 shows how storage requirement of the GSN middleware varies when number of MoSHubs connected to GSN increases in two different approaches. Figure 13 shows how much time it takes to generate a wrapper based on micro sensor device definitions (μSDD) when complexity increases. Interpretation of these results are presented in Section VII.
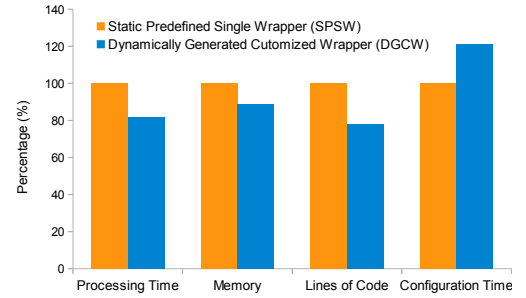

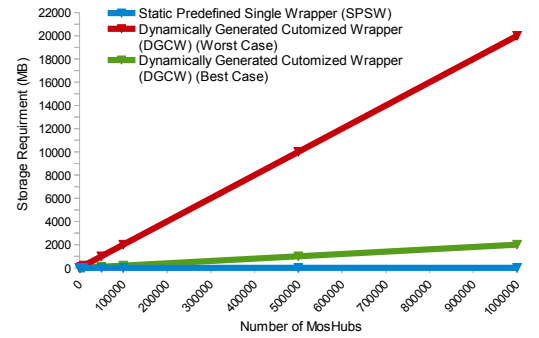Fig. 11. Comparison of SPSW and DGCW Approaches

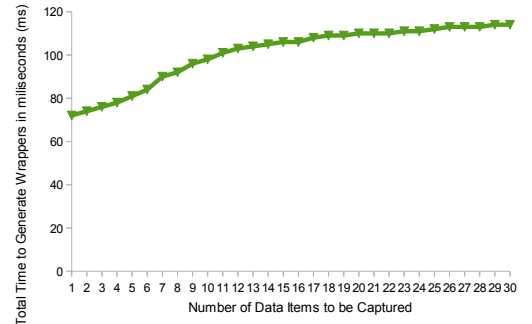

Fig. 12. Storage Requirments of SPSW and DGCW (Estimated)

Fig. 13. Performance Measurement of GSN Wrapper Generation

VII. LESSONS LEARNED AND FUTURE WORK

Our ultimate objective is to develop a mobile application that can be used to collect data from external sensors, intelligently process, and communicate them to servers over the Internet. In order to achieve above objective, both client (i.e. mobile application) and server (i.e. IoT middleware) should be able to configure themselves autonomously so they can work together efficiently. In this work, we explored approaches that can be used to automate the configuration process. Further, we employed plug-in architecture to increase the support towards external sensors. Lessons we leant can be listed as follows:

- As depicted in Figure 11, DGCW can reduce data stream processing time up to 18%. Therefore, generating customized wrapper for each MoSHub based on the sensors (e.g. number of sensors and type of sensors) connected to them, is more

efficient than using a generic wrapper that dynamically changes its data structures at runtime. In SPSW, all the exact details about data structure need to be understand via XML descriptions and then dynamically initialize during runtime (i.e. during wrapper initialization phase).

- When consider memory requirements, 11% can be saved by following DGCW approach. Further, wrapper in DGCW approach uses up to 22% less lines of code compared to wrapper in SPSW depending on the class complexity (This is true until the number of data items need to be captured stays below 25). In contrast, DGCW approach take more time to configure each MoSHub to GSN middleware. However, each MoSHub will need to configure itself with GSN only when number of sensors connected to it changes. This will not happen regularly. Therefore, DGCW approach is more efficient when all four factor considered together.

- In DGCW approach, GSN needs to generate a customised wrapper code for each MoSHub. A typical wrapper is around 15-25KB in size. In SPSW approach, GSN needs only one wrapper. If we consider storage requirement factor in isolate manner, it seems SPSW is more efficient as depicted in Figure 12. However, due to advances in computer hardware, storage is much cheaper than processing. For example, we can store one million different wrappers in a 20GB storage space. Therefore, when we take runtime efficiency into account, higher storage requirement of DGCW can be neglected. Further, GSN loads only one wrapper for each MoSHub to the memory. Therefore, no additional memory will be used in DGCW approach.

- In SPSW approach, there is no requirement to generate wrapper code at runtime. So there is no delay in configuration process. In contrast, DGCW approach needs to generate a wrapper code every time when MoSHub needs to be configured with GSN. (Note: This is only required when a MoSHub connects to GSN with specific sensor configuration for the first time. If a MoSHub connects to a GSN instance with same configuration for the second time, GSN will automatically select the previously generated wrapper code without creating a new wrapper). DGCW approach takes 70ms-120ms to create a wrapper code based on the complexity as depicted in Figure 13.

- As DGCW approach generates a customized wrapper for each MoSHub, it creates significant amount of opportunities. For example, DGCW approach allows adding context discovery functionality to GSN in the future, which is difficult to accomplish using SPSW approach.

- According to Figure 14, plug-in architecture seems promising in term of memory requirements. A plug-in library that comprises 15 different plug-ins needs only 40-60KB storage space. Therefore, plug-in architecture is scalable and suitable to be used in MoSHub. Based on our preliminary investigation, storage requirement for plug-ins is linear. However, there is an initial storage requirement of 20KB for meta-data and configuration information required by Android application model. Therefore, it is ideal to combine multiple plugins into libraries to minimize meta-data overhead.
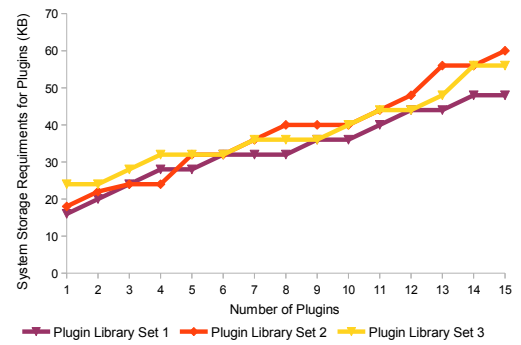

Fig. 14. This graph shows the amount of memory (system storage of the mobile phone) required by plug-ins when number of plug-ins increases. The storage is measured in Kilobytes (KB). We developed three sets of plug-in libraries where each contains 15 plug-ins. Each library comprises of different plug-ins that are capable of retrieving data from different wasp sensors from libelium (www.libelium.com) using WiFi and bluetooth.

- Our preliminary investigation showed us that the issue of re-configuration MoSHub, due to local sensor connectivity changes, can be minimized by accepting null value over some period. The sensors, which disconnect from MoSHub due to technical failures will establish its connection back within limited time so we can avoid triggering costly re-configuration process.

In our future work, we will evaluate plug-in architecture in a comprehensive manner in order to identify the scalability of our approach in term of energy and memory consumption at runtime. We will extend our evaluation platform towards different computational devices such as tablets with different hardware specifications. Further, we will explore techniques of using different protocols such as bluetooth, WiFi, and ZigBee to detect sensors and then select appropriate plug-ins autonomously which will help to retrieve data from detected sensors using cloud repositories such as Android market.

REFERENCES

[1] A. Zaslavsky, C. Perera, and D. Georgakopoulos, "Sensing as a service and big data," in *International Conference on Advances in Cloud Computing (ACC-2012)*, Bangalore, India, July 2012.

[2] C. Perera, A. Zaslavsky, P. Christen, A. Salehi, and D. Georgakopoulos, "Capturing sensor data from mobile phones using global sensor network middleware," in *IEEE International Workshop on Internet-of-Things Communications and Networking 2012 (PIMRC 2012-Workshop-IoT-CN12)*, Sydney, Australia, September 2012.

[3] ——, "Connecting mobile things to global sensor network middleware using system-generated wrappers," in *International ACM Workshop on Data Engineering for Wireless and Mobile Access 2012 (ACM SIGMOD/PODS 2012-Workshop-MobiDE)*, Scottsdale, Arizona, USA, May 2012.

[4] K. Aberer, M. Hauswirth, and A. Salehi, "Infrastructure for data processing in large-scale interconnected sensor networks," in *International Conference on Mobile Data Management*, May 2007, pp. 198–205. [Online]. Available: http://dx.doi.org/10.1109/MDM.2007.36

[5] OpenIoT Consortium, "Open source solution for the internet of things into the cloud," January 2012, http://www.openiot.eu [Accessed on: 2012-04-08].

[6] G. Kortuem, F. Kawsar, D. Fitton, and V. Sundramoorthy, "Smart objects as building blocks for the internet of things," *Internet Computing, IEEE*, vol. 14, no. 1, pp. 44 –51, jan.-feb. 2010. [Online]. Available: http://dx.doi.org/10.1109/MIC.2009.143

[7] H. Sundmaeker, P. Guillemin, P. Friess, and S. Woelffle, "Vision and challenges for realising the internet of things," European Commission Information Society and Media, Tech. Rep., March 2010, http://www.internet-of-things-research.eu/pdf/IoT_Clusterbook_March_2010.pdf [Accessed on: 2011-10-10].