Charith Perera, Arkady Zaslavsky, Peter Christen, Ali Salehi, Dimitrios Georgakopoulos, Connecting Mobile Things to Global Sensor Network Middleware using System-generated Wrappers, *Proceedings of the 11th ACM International Workshop on Data Engineering for Wireless and Mobile Access* (ACM SIGMOD/PODS-Workshop-MobiDE), Scottsdale, Arizona, USA, May, 2012, Pages 23-30 (8) More: www.charithperera.net

Connecting Mobile Things to Global Sensor Network Middleware using System-generated Wrappers

Charith Perera*+ Arkady Zaslavsky+ Peter Christen* Ali Salehi+
Dimitrios Georgakopoulos+

*Research School of Computer Science, The Australian National University, Canberra, ACT 0200, Australia
+CSIRO ICT Center, Canberra, ACT 0200, Australia

ABSTRACT

Internet of Things (IoT) will create a cyberphysical world where all the things around us are connected to the Internet, sense and produce "big data" that has to be stored, processed and communicated with minimum human intervention. With the ever increasing emergence of new sensors, interfaces and mobile devices, the grand challenge is to keep up with this race in developing software drivers and wrappers for IoT things. In this paper, we examine the approaches that automate the process of developing middleware drivers/wrappers for the IoT things. We propose ASCM4GSN architecture to address this challenge efficiently and effectively. We demonstrate the proposed approach using Global Sensor Network (GSN) middleware which exemplifies a cluster of data streaming engines. The ASCM4GSN architecture significantly speeds up the wrapper development and sensor configuration process as demonstrated for Android mobile phone based sensors as well as for Sun SPOT sensors.

Keywords

Internet of Things, Sensor Networks, Global Sensor Network Middleware, Mobile Sensors

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Sensor networks*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems; D.2.11 [**Software Engineering**]: Software Architectures; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Current awareness systems*

1. INTRODUCTION

The term Internet of Things (IoT) was firstly coined by Kevin Ashton [3] in a presentation in 1998. Further expanding this idea, the European Union has defined the above vision as "*The IoT allows people and things to be connected Anytime, Anyplace, with Anything and Anyone, ideally using Any network and Any service* [12]". It is expected that 50 to 100 billion devices will be connected to the Internet by 2020. According to the BCC Research [4], global market for sensors was around $56.3 billion in 2010. In 2011, it was around $62.8 billion. Global market for sensors is expected to increase up to $91.5 billion by 2016, at a compound annual growth rate of 7.8%. The connection and configuration of these sensor devices are not feasible to be done manually. Automation is essential to achieve the vision of IoT. This is the challenge we have addressed.

There is an increasing trend of developing middleware solutions in order to connect sensors and actuators to the Internet. These middleware solutions support fast and simple deployment of sensor networks. GSN [1], Sgroi et al. [24], Hourglass [25], HiFi [9], IrisNet [10], and EdgeServers [22] are some of the major middleware solutions. These systems share a common objective with minor differences in features and functionality. The GSN solution provides advanced and sophisticated functionality. Therefore, we decided to use GSN as the sensor network middleware to exemplify our proposed solution.

One of the major drawback in these existing middleware solutions is connectivity and configurability. Sensors come with APIs that provide software interfaces to retrieve sensor data to the middleware solutions or applications. Different middleware solutions use different mechanisms to retrieve data from sensors. The solutions are referred using different terms such as *wrappers*, *gateways*, *handlers*, *proxies*, *mediators*, etc. For example, GSN has a concept called *Wrapper* [1]. Each sensor should have a supported wrapper to be attached to the GSN server, in order to communicate with the sensor hardware. These wrappers need to be developed manually. This is an overwhelming task. There are many sensor devices and smart objects [16] that come to the market regularly. Therefore, developing wrappers or similar solutions manually is not a scalable and feasible approach. Thus, we investigate the methods that can significantly automate this process.

The rest of the paper is organised as follows. Section 2 presents an overview on sensor networks. Section 3 describes Global Sensor Network (GSN) middleware in brief. GSN wrapper is briefly discussed in Section 4. The life cycle of the wrapper is presented in Section 5. Section 6 explains the problem that we have addressed in detail. Our proposed so-

MobiDE '12, May 20, 2012 Scottsdale, Arizona, USA

lution is presented in detail in Section 7. The Sensor Device Definition (SDD) files are explained in Section 8. Section 9 presents the experiment of connecting Android mobile devices to GSN. Finally, Section 10 presents the related work and is followed by a conclusion.

2. SENSOR NETWORKS

Sensor networks are the major enabler of the IoT. A *sensor network* [2] comprises one or more sensor nodes which communicate between each other using wired and wireless means. Each sensor node has the capability to sense, communicate and process data. In sensor networks, sensors can be homogeneous or heterogeneous. These sensors are deployed in densely manner around the phenomenon which we want to sense [2]. These sensor nodes are typically low-cost and small in size that enable large deployments.

Today, increasing number of mobile sensors becoming available in the market as well as in sensor network deployments. Mobile sensors are capable of sensing while moving from one location to another. They add more efficiency and accuracy to the sensor networks, because a single mobile sensor can sense a larger area than a static sensor. Sensors built-in to the mobile phones can behave as mobile sensors and make a significant contribution in community sensing. For an experiment, we connected a mobile phone to the GSN to collect data via built-in sensors using manual wrapper development approach. We were able to identify the difficulties and challenges related to manual wrapper development approach as presented in Section 9. We proposed our ASCM4GSN approach to mitigate these difficulties.

Before IoT, sensor networks have been used in domains such as health, military and home [2]. Today, sensor networks and IoT share substantial amount of similarities. However, most of the sensor network researches have been focused on low level design and deployment issues. For example, fault tolerance, scalability, production cost, hardware constraints, sensor network topology, environment, transmission media, unpredictability, heterogeneity, energy efficient counting, localisation algorithms and power consumption are some of the leading research areas in the sensor networks [2, 5, 14, 15]. High level research areas such as data fusion and processing, have gained significant attention only in recent years.

Sensor network deployment has been considered as a difficult task in early days due to the heterogeneity of sensors. Developers and engineers had to deal with low level programming tasks in order to connect these sensor nodes together and get the sensor data into applications. A number of frameworks and middleware solutions have been developed in order to make this process easier. Global Sensor Network (GSN) [23] is such a middleware solution that enables zero-configuration deployment. It is used widely in over ten EU/Swiss funded research projects [11]. Figure 1 summarises our discussion and shows how the components we discussed above fit in real world. The next section introduces the Global Sensor Network (GSN) middleware.

3. GLOBAL SENSOR NETWORK

The Global Sensor Network (GSN) [1, 23] is a platform aimed at providing flexible middleware to address the challenges of sensor data acquisition, 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 lightweight implementation. GSN simplifies the process 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. The above reasons lead us to choose GSN as our sensor network middleware over other alternative solutions.

The GSN adopts container based architecture. A detailed explanation is provided in [1]. *Virtual Sensor* is the key element in the GSN. A virtual sensor can be any kind of data producer, for example, a real sensor, a wireless camera, a desktop computer, a mobile phone, or any combination of virtual sensors. Typically, a virtual sensor can have multiple input data streams but have only one output data stream.

A *Wrapper* is a piece of code that does the data acquisition from a specific type of sensor device. The GSN is capable of retrieving data from various data sources. Wrappers transform the raw data into the GSN standard data model that can be queried and manipulated later. All the wrapper classes need to extend the *AbstractWrapper* class. Typically, third party libraries are initialised in the wrapper constructor. Each sensor needs to have a specific wrapper that can be used to retrieve raw sensor data. In order to connect a Mica2 [7] sensor, for example, the GSN should have a corresponding wrapper that can talk to Mica2 sensor and retrieve data from it. Currently, the GSN provides wrappers for all TinyOS [27] based sensors, RFID sensors, web cams, actuators, etc. Likewise, in order to connect an Android phone's built-in sensors to the GSN, it has to have a wrapper that can retrieve raw sensor data from Android phones. We discuss GSN wrappers in general and wrapper's life cycle in details in the next sections.

4. GSN WRAPPER

In this section, we discuss GSN wrappers. As we explained earlier, each and every sensor that needs to be connected to GSN should have a corresponding wrapper. The Figure 2 depicts a basic code structure for a GSN wrapper.

```
public class EmptyWrapper extends AbstractWrapper {
  public boolean initialize ( ) { 1
  }
  public void run ( ) { 5
    while ( isActive( ) ) {
    }
  }
  public  DataField[] getOutputFormat ( ) { .... } 2
  public String getWrapperName( ) {.... } 3
  public void finalize ( ) {....} 4
}
```

Figure 2: GSN Wrapper

All the wrappers need to extend the Java class *gsn.wrapper.AbstractWrapper*. Therefore, all the wrappers are subclasses of *AbstractWrapper*. There are four methods that need to be implemented by the subclasses. Those methods are numbered 1-4 in the Figure 2. The methods are 1. *boolean initialise()*, 2. *void finalise()*, 3. *String getWrapperName()*, and 4. *DataField[] getOutputFormat()*.

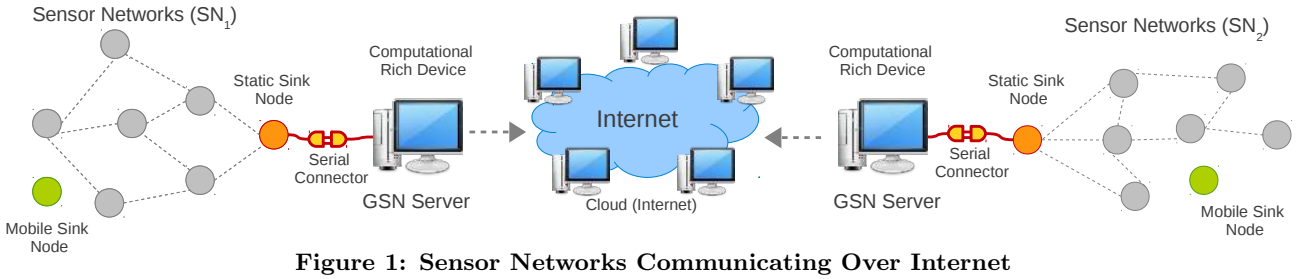

Figure 1: Sensor Networks Communicating Over Internet

A new thread is created for each wrapper in GSN. After creating the wrapper object, *initialise()* method is called as shown as (1) in Figure 2. All the communication using third party libraries should happen within this method. For example, a camera wrapper may talk to a third party API in order to talk to the camera and retrieve the camera images. In an *AndroidWrapper*, *initialise()* method creates a socket and waits until the client mobile phone sends the data packets.

The method *finalise()* is called at the end of the wrapper's life cycle. This method can be used to close all the connections that established with the outer world by the wrapper. Concretely, all the resources acquired during the *initialise()* method should be released here. For example, in the Android wrapper, all the client communication resources such as sockets and ports are released in this method.

The method *getWrapperName()* returns the name of the wrapper. The method *getOutputFormat()* returns a *DataField* object that provides a description of the data structure produced by the wrapper. The *run()* method is responsible for retrieving sensor data from sensors and transform them into GSN data model. All the sensor specific API calls need to be done inside this method. This is the most complex section of the class. The content of this method is hard to generate automatically due to third party library dependencies. The auto generation of GSN wrappers is discussed in Section 7.

5. GSN WRAPPER'S LIFE CYCLE

The life cycle of a wrapper begins with the initiation of Virtual Sensor Definition (VSD) [11] file. When a user defines a VSD file, it triggers the virtual sensor creation process. This process triggers the specified wrapper to be generated.

```
<virtual-sensor name="AndroidHandler80" priority="10">
    .............
    <streams>
        <stream name="input1">
            <source alias="source1" sampling-rate="1" storage-size="1">
                <address wrapper="SunSPOT">
                </address>
                    ..............
            </source>
                ..............
        </stream>
    </streams>
</virtual-sensor>
```

Figure 3: Virtual Sensor

The wrapper that corresponds to each stream source (i.e. sensor) is defined under the address element in the VSD file. For example, the VSD file segment depicted in Figure 3 triggers the SunSPOT [19] wrapper to be instantiated. Virtual sensor creation process sends a Wrapper Connection Request (WCR) to the wrapper repository [11] in the GSN server. A Wrapper Connection Request is an object which contains a wrapper name and its initialisation parameters as defined in the *Virtual Sensor*. Sequentially, the following steps are followed:

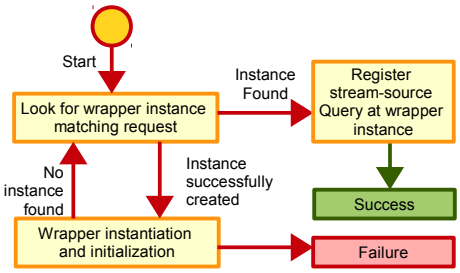

Figure 4: Wrapper Life Cycle

First, wrapper repository looks for a wrapper instance that matches to the WCR. If found, then the stream-source query will be registered with the wrapper and returns true.

If there isn't any wrapper object that matches WCR in the repository, the wrapper repository generates a new appropriate wrapper object. Then, the newly created object would be added to the wrapper repository. Finally, the stream-source query will be registered with the wrapper and returns true.

If there isn't any wrapper object that matches WCR in the repository and wrapper repository does not have an appropriate wrapper class to be instantiate, then returns false. The virtual sensor loader fails to load a virtual sensor if at least one of the stream sources required by an input stream fails. For example, if user defines a virtual sensor as depicted in the Figure 3 and if the GSN wrapper repository does not have a SunSPOT wrapper, then the virtual sensor would fail. Figure 4 summaries the life cycle of the GSN wrappers. We proposed to extend this process in our solution. The details are explained in Section 7.

6. THE CHALLENGE OF IMPROVING EFFICIENCY IN CONNECTING THINGS

We introduced the problem in brief in earlier sections. Let's discuss it in details. Almost all the sensors come with third-party libraries or API released by the sensor manufacturer. If we want to retrieve sensor readings, we need to access the sensor hardware through these provided third party libraries. This stays true when we want to develop sensor networks using sensor network middleware solutions. For example, sensor network middleware solution such as

GSN provides features such as window based continuous sensor data querying. In order to accomplish this task, GSN should retrieve sensor data from the sensor devices and organise them according to GSN specific data model. Most of the sensor network middleware solutions are good at providing high-level features such as querying which deal with internal data structure. However, the biggest problem is getting the sensor data from the sensor hardware devices into middleware solutions. This challenge has been addressed by different sensor network middleware solutions using different mechanisms. For example, GSN uses *Wrappers* to accomplish this task. The Figure 5 shows the current GSN Data Acquisition Architecture.

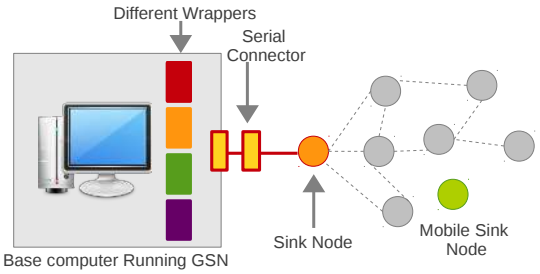

Figure 5: Current GSN Data Acquisition Architecture

However, we identify two drawbacks in current GSN data acquisition architecture. First problem is that these wrappers need to be developed manually by the programmers. For example, if we want to connect a SunSPOT sensor into the GSN, developers have to develop a wrapper that is specific for SunSPOT. This wrapper will use the libraries provided by the SunSPOT manufacturer to talk to the SunSPOT sensor and retrieve sensor data. When the sensor devices get updated, GSN developers have to update these wrappers as well. That means developers have to keep on updating these wrappers. This process decreases the scalability and it also requires more effort and cost.

The second problem is lack of code sharing. For example, one project may develop a wrapper for SunSPOT sensors. However, there is no mechanism to distribute this wrapper among other projects. The same wrapper may be developed by different project groups. It reduces the effectiveness and efficiency due to repetition. We have addressed these two issues, first by automating wrapper generation and second by developing a cloud repository. We discuss the details of our proposed solution in Section 7.

Let us discuss some of the possible approaches that we can follow to solve the problem presented above. By evaluating several sensor devices and IoT middleware systems, it was understood that the method (steps) of connecting a device to an IoT middleware system is significantly similar. The most commons steps can be explained as bellow.

- ***Acquire Manufacturers' APIs***: As we mentioned earlier, each sensor device comes with an API. In some instances, APIs are common across all the sensors devices produce by the manufacturer. In other cases, APIs are strictly related to specific type of sensors. However, middleware system should use these APIs to communicate with the sensor devices. Therefore, the first step is to identify the path to the APIs.
- ***Acquire System Configuration Details***: Most of the IoT middleware solutions need system configuration details such as IP addresses, ports and protocols to communicate with the sensor devices. Sometimes, it is possible to identify these details automatically and otherwise users may need to enter them manually. Therefore, identifying the machine specific and platform specific information is also a critical step.
- ***Initiate the Data Structure***: Every middleware maintains its own data structure. Once the sensor devices are connected to the middleware, the data sensed by the sensors need to be stored in these data structures. Therefore, identify the required data structure and allocate them is a major steps.
- ***Initiate the Communication between IoT Middleware and Sensor Device***: The communication between a sensor device and a IoT middleware solution usually starts by initiating the communication. The exact process could be varied among different sensor platforms. This initiation does not exchange any sensor data, but it opens and establishes the necessary ports and paths to proceed. This step also allocates the required resources and makes them ready for the data communication. This step further ensures that both sensor and middleware is ready to communicate between each other.
- ***Data Communication***: This is the most important step. Based on the communication path established in the previous step, the sensor and the middleware will communicate either in push or pull method. As a result, middleware will receive sensor data periodically. The IoT middleware can store the sensor data on the data structure which prepared in an earlier step.
- ***Close the Communication and Release the Resources***: All the connections established between the sensor device and the middleware system needs to be closed. Furthermore, the relate resources need to be released, so they are available for used by other operations.

We encapsulated these steps into five segments in the Sensor Device Definition (SDD) files as discussed in Section 8. After identifying these commonalities, we investigate the methods of simplifying the process of connecting the sensors to IoT middleware systems.

Every IoT middleware has its own way of communicating with sensor devices. Mostly, there are dedicated handlers to accomplish this task. For example, in GSN, the handlers are called *wrappers* as discussed in Section 4. In other approaches, these handlers are called *gateways*, *proxies*, *mediators*, etc. Furthermore, different technologies can also be employed to develop these handlers such as web services, RESTful APIs, native code, etc. From the previous work conducted by different researchers, it has been identified that native code gives better performance in term of scalability and efficiency compared to other technologies such as web services [21]. Therefore, we decided to use native code to develop the handler, in our case GSN's *wrappers*.

The process of automated wrapper generation is depicted in Figure 6. Once we define a SDD file for a specific sensor as explained in Section 8, it can be used to develop wrappers for different IoT middleware systems. As we mentioned earlier, every IoT middleware has its own component similar to

wrappers. We can combine the wrapper template of an IoT middleware and a SDD file to generate a Middleware specific wrapper. Wrapper template explains the basic structure of a wrapper such as functions, methods, data structures, etc.

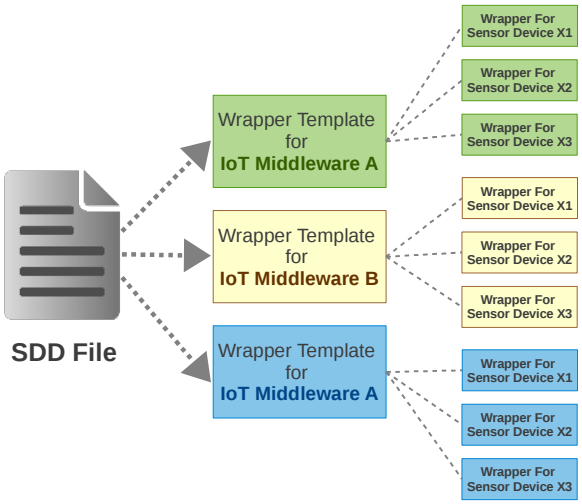

Figure 6: The Wrapper Generation Process

7. ASCM4GSN ARCHITECTURE

The two main problems in the current approach are addressed as follows. We propose Automated Sensor Configuration Model For Global Sensor Network (ASCM4GSN) architecture to address these issues. We introduce a *Automated Wrapper Generation Layer*. As the name implies, this layer automate the process of wrapper generation. The process can be explained as follows.

As we discussed in Section 5, the wrapper generation always begins with a Virtual Sensor Definition (VSD). When a VSD mentions a name of a wrapper that GSN currently does not have in the wrapper repository, first, GSN searches the Sensor Device Definition Local Repository (SDDLR) to look for a matching Sensor Device Definition (SDD) file. We discuss SDD file in detail in the next Section. For now, SDD file can be explained as a specification file that contains all the information that is required to generate a wrapper class.

If Sensor Device Definition Local Repository (SDDLR) does not have a matching SDD file, GSN will automatically connect to the Sensor Device Definition Cloud Repository (SDDCR) and search for a matching wrapper definition file. If there is a matching SDD file, the GSN will fetch the SDD file and feed it to the *Automated Wrapper Generation Layer*. This layer generates the wrapper class, compiles it, and pushes it to the wrapper repository. After that, the wrapper life cycle proceeds as explained in Section 5. Figure 7 shows our proposed architecture.

If there isn't any SDD file in the Sensor Device Definition Cloud Repository (SDDCR) for a specific sensor, then the developers may need to develop a SDD file based on the SDD specification. However, the developers can upload their SDD file to the cloud so other users do not need to develop it again. This approach saves time and cost. In the future, there will be increasing number of sensors available in the market. Our community based cloud approach would be ideal to deal with wider adaptation of IoT and sensor network deployments.

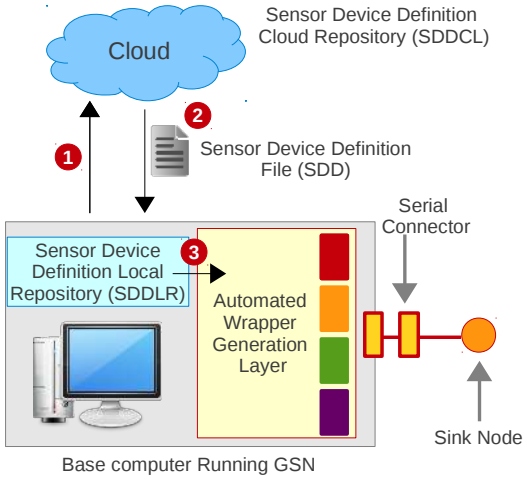

Figure 7: Proposed ASCM4GSN Architecture

There are many reasons to base our approach on SDD files rather than wrapper classes. We could also use direct wrapper classes instead of using SDD files. The reason for adding extra step to our approach can be explained as follows.

If we use a wrapper class, then it would be a platform specific such as Java, C#, etc. We keep SDD files platform independent or cross platform. Some sensor devices may need to be configured using machine specific details such as port numbers, IP address, etc. Editing a SDD file is a much cleaner and easier way compared to editing a class file. In future, software tools can be developed to make the editing significantly easier. In that case, designing tools to edit platform independent file is much easier than editing different class files written in different programming languages.

At a later stage it may be required to combine SDD files with semantic technologies in order to make the automation more sophisticate. As we mentioned earlier, wrapper class also need to use third-party libraries which are developed by sensor device manufacturers (e.g. camera). If we based our approach on wrapper classes, then we may need to distribute the libraries with the class file as well. This could create licensing and legal issues. In SDD files, we only need provide the link to the libraries so IoT middleware can download the libraries from the sensor device manufacturers.

The second problem we identified has been addressed by connecting GSN to the cloud. We developed a Sensor Device Definition Cloud Repository (SDDCR), which acts as a global repository for SDD files, so software developers and hardware manufacturers can upload SDD files to the cloud. This means if someone develops a SDD file for SunSPOT once, any GSN instance deployed around the world would be able connect SunSPOT sensors automatically using that SDD file.

8. ASCM4GSN SENSOR DEVICE DEFINITION

Sensor Device Definition (SDD) is an XML file which comprises number of elements. Describing each and every sections and elements of SDD is beyond the scope of this paper. Here, we describe how SDD works using a real world example. The left side of Figure 8 shows the SDD file which cor-

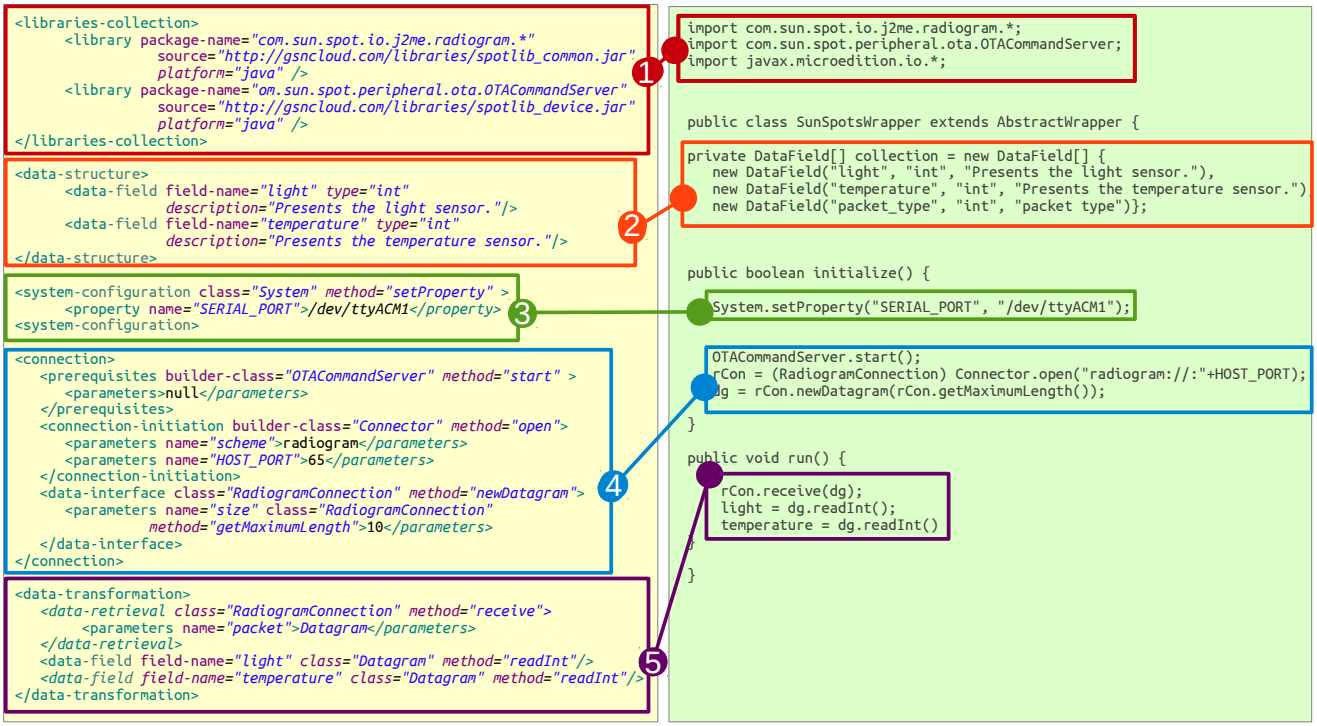

Figure 8: Comparison of SDD File and System-generated Wrapper

responds to SunSPOT sensor device. The right side of the figure shows the SunSPOT wrapper generated by the *Automated Wrapper Generation Layer* based on the SDD file on the left. Please note that both files are used for demonstration purposes and only contain major sections and elements.

Even though we do not intend to explain each and every element in the two files, a high level explanation can be made as follows. In Section 6, we identified six major steps that are common across all the sensor platforms. After analysing the steps, we encapsulate them into five segments in the SDD file. The segments are marked 1 to 5 in Figure 8. The segments are *libraries collection, data structure, system configuration, connection,* and *data transformation.*

The segment (1) contains all the libraries that need to be downloaded for GSN in order to compile the wrapper. It consists of package names and the sources where the files can be downloaded. This section can also be extended to add installation files. For example, if a specific sensor needs to install drivers on the GSN server, this section can provide the source link to download and install the driver automatically. It also consists of platform information. As we are intended to make these SDD files platform independent, the parameters can specify which libraries are related to which platform (e.g. Java, .Net, C, etc.).

The next segment (2) includes the information about the data structures. It can be used to provide all the information required by the GSN in order to create GSN data model. The sensor data will be stored in the data structure created in this segment. The segment (3) is dedicated to store system level configurations. There are configuration settings that need to be configured. The properties and values need be changed depending on the operating system. For example, serial ports are named as /dev/ttyACM in Linux and as COM in Windows.

The connection segment (4) comprises the information related four steps: initiate connection with the sensor devices, initiate data retrieval mechanisms, retrieve sensor data, and close connection. This segment would be a lengthy section. According to our preliminary investigation, most sensors do have these four steps. Final segment (5) is for the data transformation. The retrieved data packets need to be examined and extract the values from them. These values need to be stored in the data structure defined in the segment (2).

If we consider the length and the complexity of the two files, it is true that SDD file is more lengthy and complex. However, we can easily develop a graphical user interface to produce SDD file very easily which will reduce and hide the complexity in major way.

9. ANDROID TO GSN CONNECTIVITY EXAMPLE

We evaluated the process of connecting sensors to an IoT middleware called GSN. *AndroidWrapper* was developed in order to retrieve sensor data from Android mobile phones. It was realised that each and every sensor should have a wrapper talking to a GSN server in order to collect data. Developing such wrappers is a time consuming and tedious job. It could take one or more days for a developer to develop a single wrapper for a specific sensor including the time that would take to familiarise with the specific sensor platform. Our ASCM4GSN approach drastically reduces the development time as the developer does not need to familiarise with the sensor platform in order to generate a wrapper using our approach. Figure 9 shows the client application we developed and installed on Android mobile phones. It generates

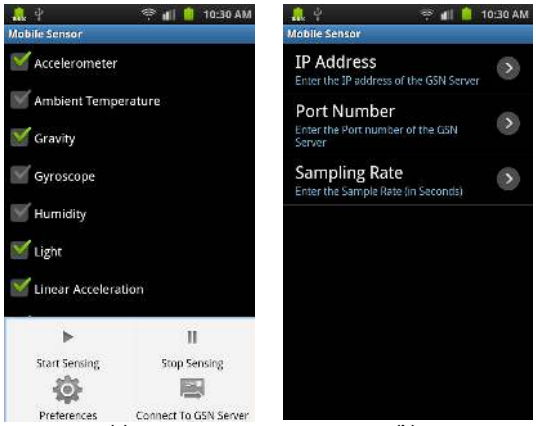

Figure 9: Sensor Data Generator Application

and sends sensor data packets to the *AndroidWrapper* in GSN.

10. RELATED WORK

We recognise two initiatives related to our work; SensorML [6] and IEEE 1451 standards [26]. SensorML provides functionalities such as describing sensors and sensor systems for inventory management and sensor discovery, describing sensor geolocation of observed values, describing processes and observations, describing performance characteristics, describing processing and analysis of the sensor observations, etc. IEEE 1451 standards are dedicated to standardise the data communication among sensors, actuators, and related devices. In our solution, we are dealing with sensor installation and configuration challenge where both above initiatives do not explicitly address this challenge.

Californium (Cf) CoAP framework [17] has proposed a *thin server* architecture to solve the problem of connecting heterogeneous devices from different manufacturers with diverse functionalities to the Internet of things. Cf [17] puts a *thin server* in front of the device to work as a proxy. Thin server only provides a low-level API to the elementary functionality of a device. The client applications or IoT middleware system can communicate with the device via the thin server using a RESTful API. All the functionalities are encoded as REST resources.

However in this type of approach, the communication between the thin server and the device needs be developed by the developers. This decrease the efficiency in connecting thing to the IoT. Furthermore, using a thin server could also raise issues in term of energy efficiency and communication overheads, specially due to the magnitude of IoT. This approach will perform very well, if the device manufacturer provides a RESTful API integrated to the device's software package itself. Further, this approach would be ideal for device manufacturers to be followed as a step forward in standardising the communication between devices and applications.

Web Services Gateways [21] is an approach based on Model Driven Architecture (MDA) and Device Profile for Web Services (DPWS). The focus is on connecting industrial devices, where the devices have a lifetime of often more than 40 years, to the client applications in IoT paradigm. They have developed gateways comprises with web services that provide interfaces to access the devices.The web service are generated automatically using predefined models.

In [21], performance evaluation results has shown that the approach is not scalable. The papers [21, 13] admit that the web services approach would perform less compared to native code approach. That is why we proposed a native code generation approach over other mechanisms. The native code approach increases the node complexity and the web service approach limits the scalability. There is a trade of between the two.

InterX [20] is a smart phone-based service interoperability gateway for heterogeneous smart objects. It employs a mediator gateway that can transform one protocol to another. For example, InterX enables the communication between Bluetooth based smart object and UPnP based smart object via a gateway in runtime.

In contrast, our approach is based on development time mediator. We use XML as the mediator to produce the native code that can establish the communication between IoT middleware systems and smart devices. Runtime mediators give less performance due to communication overheads. Another difference is that InterX is focused on well recognised appliances such as digital camera, tv, video player, etc. In our approach, we focused on low level sensors such a SunSPOT, Arduio, and we also offer extensibility to facilitate more low level sensors.

Hydra [8] is an IoT middleware that allows developers to incorporate heterogeneous physical devices into their applications. The interaction between devices and the middleware are enabled though web services. Hydra is based on a semantic Model Driven Architecture for easy programming. This is similar to the Web Services Gateways [21] approach we presented earlier. Even though the performance evaluation of the Hydra middleware is not available in device connection perspective, the paper [21] has raised the similar issues related to employing web services as the gateways for smart devices.

uMiddle [18] is a bridging framework that enables seamless device interaction over diverse middleware platforms. This approach is similar to the InterX [20]. uMiddle transforms one protocol to another in runtime. This middleware is focused on the interoperability between popular protocols such as Bluetooth, UPnP, etc. In contrast, we are more concerned about connecting low level sensors to IoT middleware solutions. uMiddle have identified three essential requirements of an interoperability middleware platform; transport-level bridging, service-level bridging, and device-level bridging. We have also considered these requirements in our approach.

11. CONCLUSION AND FUTURE WORK

In this article, we propose and discuss the ASCM4GSN architecture and develop a specification that can be used to automate the sensor data acquisition and configuration process related to IoT middleware. We conducted our implementation and experiments based on a popular IoT middleware platform called GSN. Sensor devices come with APIs that need to be used in order to retrieve sensor data from the sensor devices. For this, middleware has to be aware of sensor devices. Typically, sensor drivers/wrappers need to be developed manually by a programmer using the third-party libraries. This requires more time, cost and effort. We have demonstrated that automating the process of develop-

ing sensor drivers/wrappers will improve efficiency and productivity. We introduced Sensor Device Description (SDD) specification to facilitate the automation. We have proposed an SDD sharing mechanism using a cloud repository to reduce the repetitive work. Currently, SDD files can be used to generate wrappers for GSN. However, we intend to keep SDD file as a generic sensor device definition mechanism that would be independent from IoT middleware solutions.

Our future research work aims at efficient and effective automation of connecting things to IoT middleware as well as incorporating generated extended functionality.We will combine context capturing and semantic data technologies with procesing of sensor data inside the wrapper itself.

12. REFERENCES

[1] K. Aberer, M. Hauswirth, and A. Salehi. Infrastructure for data processing in large-scale interconnected sensor networks. In *Mobile Data Management, 2007 International Conference on*, pages 198–205.

[2] I. F. Akyildiz, S. Weilian, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *Communications Magazine, IEEE*, 40(8):102–114, 2002.

[3] K. Ashton. That 'internet of things' thing in the real world, things matter more than ideas, June 2009.

[4] BCC Research. Sensors: Technologies and global markets. Technical report, BCC Research, 2011.

[5] A. Bharathidasan and V. A. S. Ponduru. Sensor networks: An overview. Technical report, University of California, Davis,, 2003.

[6] M. Botts and A. Robin. Opengis sensor model language (sensorml) implementation specification. Technical report, Open Geospatial Consortium Inc, 2007.

[7] Crossbow Technology Inc. Crossbow-manuals getting started guide. Technical report, Crossbow Technology, September 2005.

[8] M. Eisenhauer, P. Rosengren, and P. Antolin. A development platform for integrating wireless devices and sensors into ambient intelligence systems. In *Sensor, Mesh and Ad Hoc Communications and Networks Workshops, 2009. SECON Workshops '09. 6th Annual IEEE Communications Society Conference on*, pages 1–3.

[9] M. J. Franklin, S. R. Jeffery, S. Krishnamurthy, and F. Reiss. Design considerations for high fan-in systems: The hifi approach. In *Conference on Innovative Data Systems Research*, pages 290–304, 2005.

[10] P. Gibbons, B. Karp, Y. Ke, S. Nath, and S. Seshan. Irisnet: an architecture for a worldwide sensor web. *Pervasive Computing, IEEE*, 2(4):22 – 33, oct.-dec. 2003.

[11] GSN Team. Global sensors networks. Technical report, Ecole Polytechnique Federale de Lausanne (EPFL), 2009.

[12] P. Guillemin and P. Friess. Internet of things strategic research roadmap. Technical report, The Cluster of European Research Projects, 2009.

[13] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio. Interacting with the soa-based internet of things: Discovery, query, selection, and on-demand provisioning of web services. *Services Computing, IEEE Transactions on*, 3(3):223 –235, july-sept. 2010.

[14] R. Gummadi, O. Gnawali, and R. Govindan. Macro-programming wireless sensor networks using kairos. Technical report, University of Southern California, 2005.

[15] S. Iyengar, N. Parameshwaran, V. Phoha, N. Balakrishnan, and C. Okoye. *Fundamentals of Sensor Network Programming: Applications and Technology*. Wiley-Blackwell, 2010.

[16] G. Kortuem, F. Kawsar, D. Fitton, and V. Sundramoorthy. Smart objects as building blocks for the internet of things. *Internet Computing, IEEE*, 14(1):44–51, 2010.

[17] M. Kovatsch, S. Mayer, and B. Ostermaier. Moving application logic from the firmware to the cloud: Towards the thin server architecture for the internet of things. In *Proceedings of the 6th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS 2012), Palermo, Italy*, July 2012.

[18] J. Nakazawa, H. Tokuda, W. Edwards, and U. Ramachandran. A bridging framework for universal interoperability in pervasive systems. In *Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on*, page 3, 2006.

[19] Oracle Corporation. Sun spot world: Welcome to the internet of things, 2012. http://www.sunspotworld.com/ [Accessed on: 2012-04-10].

[20] H. Park, B. Kim, Y. Ko, and D. Lee. Interx: A service interoperability gateway for heterogeneous smart objects. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2011 IEEE International Conference on*, pages 233 –238, march 2011.

[21] T. Riedel, N. Fantana, A. Genaid, D. Yordanov, H. Schmidtke, and M. Beigl. Using web service gateways and code generation for sustainable iot system development. In *Internet of Things (IOT), 2010*, pages 1 –8, 29 2010-dec. 1 2010.

[22] S. Rooney, D. Bauer, and P. Scotton. Techniques for integrating sensors into the enterprise network. *Network and Service Management, IEEE Transactions on*, 3(1):43 –52, jan. 2006.

[23] A. Salehi. *Design and implementation of an efficient data stream processing system.* PhD thesis, Ecole Polytechnique Federale de Lausanne (EPFL), 2010.

[24] M. Sgroi, A. Wolisz, A. Sangiovanni-vincentelli, and J. M. Rabaey. A service-based universal application interface for ad-hoc wireless sensor networks. In *in Ambient Intrelligence*. Springer, 2005.

[25] J. Shneidman, P. Pietzuch, J. Ledlie, M. Roussopoulos, M. Seltzer, and M. Welsh. Hourglass: An infrastructure for connecting sensor networks and applications. Technical report, 2004.

[26] The National Institute of Standards and Technology. Introduction to ieee p1451, 2011. http://www.nist.gov/el/isd/ieee/1451intro.cfm [Accessed on: 2012.03.01].

[27] TinyOS Alliance. Tinyos, July 2010. http://www.tinyos.net/ [Accessed: 2011-12-18].