

Context-aware Dynamic Discovery and Configuration of 'Things' in Smart Environments

Charith Perera, Prem Prakash Jayaraman, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos

Abstract The Internet of Things (IoT) is a dynamic global information network consisting of Internet-connected objects, such as RFIDs, sensors, actuators, as well as other instruments and smart appliances that are becoming an integral component of the future Internet. Currently, such Internet-connected objects or 'things' outnumber both people and computers connected to the Internet and their population is expected to grow to 50 billion in the next 5 to 10 years. To be able to develop IoT applications, such 'things' must become dynamically integrated into emerging information networks supported by architecturally scalable and economically feasible Internet service delivery models, such as cloud computing. Achieving such integration through discovery and configuration of 'things' is a challenging task. Towards this end, we propose a Context-Aware Dynamic Discovery of Things (CADDOT) model. We have developed a tool *SmartLink*, that is capable of discovering sensors deployed in a particular location despite their heterogeneity. *SmartLink* helps to establish the direct communication between sensor hardware and cloud-based IoT middleware platforms. We address the challenge of heterogeneity using a plug in architecture. Our prototype tool is developed on an Android platform. Further, we employ the Global Sensor Network (GSN) as the IoT middleware for the proof of concept validation. The significance of the proposed solution is validated using a test-bed that comprises 52 Arduino-based Libelium sensors.

Key words: Internet of Things, Sensing as a service, Configuration tool, Context awareness, Dynamic discovery, Plug and play, Sensor devices, Middleware

Charith Perera, Prem Jayaraman, Arkady Zaslavsky, Dimitrios Georgakopoulos
CSIRO Computational Informatics, Canberra, ACT 2601, Australia, e-mail: firstname.lastname@csiro.au

Charith Perera, Peter Christen
Research School of Computer Science, The Australian National University, Canberra, ACT 0200, Australia, e-mail: firstname.lastname@anu.edu.au

1 Introduction

The Internet of Things (IoT) [4] first received attention in the late 20th century. The term was firstly coined by Kevin Ashton [3] in 1999. “*The Internet of Things allows people and things¹ to be connected Anytime, Anyplace, with Anything and Anyone, ideally using Any path/ network and Any service*” [18]. As highlighted in the above definition, connectivity among devices is a critical functionality that is required to fulfil the vision of IoT. The following statistics highlight the magnitude of the challenge we need to address. Due to the increasing popularity of mobile devices over the past decade, it is estimated that there are about 1.5 billion Internet-enabled PCs and over 1 billion Internet-enabled mobile devices today. The number of ‘things’ connected to the Internet exceeded the number of people on earth in 2008 [23]. By 2020, there will be 50 to 100 billion devices connected to the Internet [45]. Similarly, according to BCC Research, the global market for sensors was around \$56.3 billion in 2010. In 2011, it was around \$62.8 billion, and it is expected to increase to \$91.5 billion by 2016, at a compound annual growth rate (CAGR) of 7.8% [5].

The above statistics allow us to conclude that the growth rate of sensors being deployed around us is increasing over time and will keep its pace over the coming decade. Over the last few years, we have witnessed many IoT solutions making their way into the market [40]. The IoT market has already been fragmented, with many parties competing with a variety of different solutions. Broadly, these IoT solutions can be divided into two segments: sensor hardware-based solutions [27] and cloud-based software solutions [14, 17, 31]. Some products specifically address one segment, while others address both. In this chapter, we propose a Context-Aware Dynamic Discovery of Things (CADDOT) model in order to support the integration of ‘things’ into cloud-based IoT solutions via dynamic discovery and configuration by also addressing the challenge of heterogeneity. We reduce the complexity of the ‘things’ configuration process and make it more user friendly and easier to use. One major objective is to support non-technical users by allowing them to configure smart environments without technical assistance.

This chapter makes the following contributions. We propose a model, CADDOT, that can be used to configure sensors autonomously without human intervention in highly dynamic smart environments in the Internet of things paradigm. To support this model, we developed a tool called *SmartLink*. *SmartLink* is enriched with context-aware capabilities so it can detect sensors using different protocols such as TCP, UDP, Bluetooth and ZigBee. CADDOT is designed to deal with highly dynamic smart environments where sensors are appearing and disappearing at a high frequency. This chapter also presents the results of experimental evaluations performed using 52 sensors measuring different types of phenomenon and using different communication sequences.

¹ We use both terms, ‘objects’ and ‘things’ interchangeably to give the same meaning as they are frequently used in IoT related documentation. Some other terms used by the research community are ‘smart objects’, ‘devices’, ‘nodes’. Each ‘thing’ may have one or more sensors attached to it.

We explain how our model can be used to enrich the existing solutions proposed in the research field. The chapter is organized as follows. We present background information and motivation in Section 2. In Section 3, we discuss the functional requirements of an ideal IoT configuration process. We discuss related work in Section 4. The proposed CADDOT model is introduced in Section 5. The design decisions we made are justified and compared with alternative options in Section 6. Implementation details and evaluations are presented in Section 7 and Section 8 respectively. The lessons learnt are discussed in Section 9. Open challenges are presented in Section 10 and we conclude the chapter in Section 11 with indications for future work.

2 Background and Motivation

This section briefly highlights the background details of the challenge we address in this chapter. Firstly, we explain the challenges in the smart environment from the perspective of dynamic discovery and configuration of ‘things’. Secondly, we discuss the concept of sensing as a service and its impact on the IoT. At the end, we present the importance of the configuration of ‘things’ in the big data domain.

2.1 Smart Environment

A smart environment can be defined as *“a physical world that is richly and invisibly interwoven with sensors, actuators, displays, and computational elements, embedded seamlessly in the everyday objects of our lives, and connected through a continuous network”* [46]. Smart environments may be embedded with a variety of smart devices of different types including tags, sensors and controllers, and have different form factors ranging from nano to micro to macro sized. As also highlighted by Cook and Das [13], device communication using middleware and wireless communication is a significant part of forming a connected environment. Forming smart environments needs several activities to be performed, such as discovery (i.e. exploring and finding devices at a given location), identification (i.e. retrieving information about devices and recognizing them), connection establishment (i.e. initiating communication using a protocol that the device can understand), and configuration. Further, users may combine sensors and services to configure smart environments where actuators are automatically triggered based on conditions [25]. In smart home environments, Radio Frequency for Consumer Electronics (RF4CE) has been used to perform automated configuration of consumer devices [43]. However, such techniques cannot be used to configure low-level smart ‘things’.

2.2 Sensing as a service

The sensing-as-a-service model [37] provides sensing capabilities as a service similar to other models such as infrastructure-as-a-service (IaaS), platform-as-a-service (PaaS), and software-as-a-service (SaaS). Mobile devices are widely used to collect data from inbuilt or external sensors [42].

It envisions that sensor descriptions and capabilities are posted on the Internet so the interested consumer can get access to the corresponding sensors by paying a fee [37]. The sensing as a service model is expected to drive the IoT from the business point of view by creating a whole new set of opportunities and values. It has been predicted that individuals as well as, private and public organizations will deploy sensors to achieve their primary objectives [8, 37]. Additionally, they will share their sensors with others so a collectively value-added solution can be built around them. Such sensor deployments and data collection allows the creation of real-time solutions to address tough challenges in Smart Cities [29, 37]. In order to support sensor deployments, easy-to-use *'things'* discovery and configuration tools need to be developed. Such a set of tools will stimulate the growth of sensor deployments in the IoT. They will help the non-technical community to become involved in building smart environments efficiently and effectively.

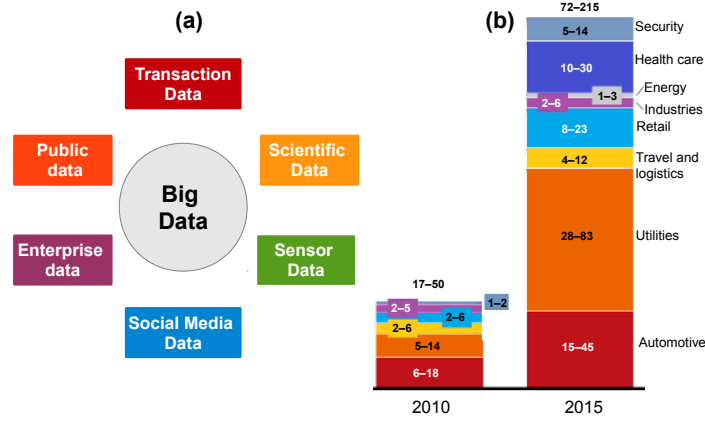


Fig. 1: (a) Big Data comprises six categories of data (b) Data generated from the IoT will grow exponentially as the number of connected nodes increases. Estimated numbers of connected nodes based on different sectors are presented in millions [28].

2.3 Big Data Challenge

Big Data [6] mainly comprises six categories of data, as illustrated in Figure 1(a) transaction data, scientific data, sensor data, social media data, enterprise data, and public data. The sensor data category is expected to be generated by the growing number of sensors deployed in different domains, as illustrated in Figure 1. The data streams coming from 'things' will challenge the traditional approaches to data management and contribute to the emerging paradigm of big data. Collecting sensor data on a massive scale, which creates big data, requires easy-to-use sensor discovery and configuration tools that help to integrate the 'things' into cloud-based IoT middleware platforms. Big data has been identified as a secondary phase of the IoT, where new sensors are cropping up and organizations are now starting to analyse data, that in some cases, they have been collecting for years.

This work is also motivated by our previous work which focused on utilising mobile phones and similar capacity devices to collect sensor data. In DAM4GSN [38], we proposed an application that can be used to collect data from sensors built into mobile phones. Later, we proposed MoSHub [33] that allows a variety of different external sensors to be connected to a mobile phone using an extensible plugin architecture. MoSHub also configures the cloud middleware accordingly. Later in MOSDEN [34], we developed a complete middleware for resource-constrained mobile devices. MOSDEN is capable of collecting data from both internal and external sensors. It can also apply SQL-based fusing on data streams in real time. As we mentioned earlier, in order to collect data from sensors, first we need to discover and configure the sensors in such a way that the cloud can communicate with them. In our previous efforts, discovery and configuration steps were performed manually. In this chapter, we propose an approach that can be used to discover and configure sensors autonomously.

3 Functional Requirements

The 'things' configuration process detects, identifies, and configures sensor hardware and cloud-based IoT platforms in such a way that software platforms can retrieve data from sensors when required. In this section, we identify the importance, major challenges, and factors that need to be considered during a configuration process. The process of sensor configuration in IoT is important for two main reasons. Firstly, it establishes the connectivity between sensor hardware and software systems which makes it possible to retrieve data from the deployed sensor. Secondly, it allows us to optimize the sensing and data communication by considering several factors as discussed below. Let us discuss the following research problem: *Why is sensor configuration challenging in the IoT environment?*. The major factors that make sensor configuration challenging are 1) the number of sensors, 2) heterogeneity, 3) scheduling, sampling rate, communication frequency, 4) data acquisition, 5) dynamicity, and 6) context [36].

Table 1: Heterogeneity in term of Wireless Communication Technology

	ZigBee	GPRS-GSM	WiFi	Bluetooth
Standard	802.15.4		802.11b	802.15.1
System Resources	4-32KB	16MB+	1MB+	250KM+
Battery life (days)	100-1000+	1-7	0.5-5	1-7
Network Size (nodes)	2 ⁶⁴	1	32	7
Bandwidth (KB/s)	20-250	64-128+	11000	720
Transmission Range (meters)	1-100+	1000	1-100	1-10+
Success Metrics	Reliability, power, cost	Reach, quality	flexibility, Speed	Convenience, cost

1) Number of Sensors: When the number of sensors that need to be configured is limited, we can use manual or semi-autonomous techniques. However, when the numbers grow rapidly towards millions and billions, as illustrated in Figure 1(b), such methods become extremely inefficient, expensive, labour-intensive, and in most situations impossible. Therefore, large numbers have made sensor configuration challenging. An ideal sensor configuration approach should be able to configure sensors autonomously as well as within a very short time period.

2) Heterogeneity: This factor can be interpreted in different perspectives. (1) Heterogeneity in terms of the communication technologies used by the sensors, as presented in Table 1. (2) Heterogeneity in terms of measurement capabilities, as presented in Figure 2 (e.g. temperature, humidity, motion, pressure). (3) The types of data (e.g. numerical (small in size), audio, video (large in size)) generated by the sensors are also heterogeneous. (4) The communication sequences and security mechanisms used by different sensors are also heterogeneous (e.g. exact messages/commands and the sequence that needs to be followed to successfully communicate with a given sensor). As illustrated in Figure 3, some sensors may need only a few command passes and others may require more. Further, the messages/commands understood by each sensor may also vary. These differences make the sensor configuration process challenging. An ideal sensor configuration approach that is designed for the IoT paradigm should be able to handle such heterogeneity. It should also be scalable and should provide support for new sensors as they come to the market.

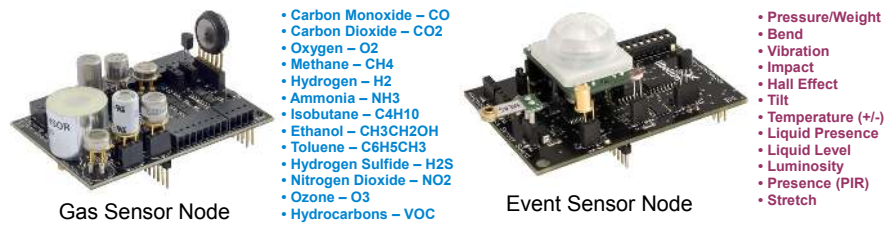


Fig. 2: Heterogeneity in term of sensing/measurement capabilities of sensor nodes

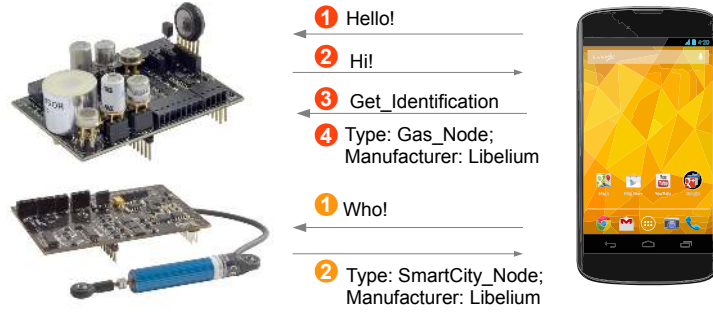


Fig. 3: Heterogeneity in term of communication and message/command passing sequences. Some sensors may need only a few message/command passes and others may require more. The messages/commands understood by each sensor may also vary.

3) Scheduling, Sampling Rate, and Network Communication: The sampling rate defines the frequency with which sensors need to generate data (i.e. sense the phenomenon) (e.g. sense temperature every 10 seconds). Deciding the ideal (e.g. balance between user requirement and energy consumption) sampling rate can be a very complex task and has a strong relationship with **6) Context** (see below). The schedule defines the timetable for sensing and data transmission (e.g. sense the temperature only between 8am and 5pm on weekdays). Network communication defines the frequency of data transmission (e.g. send data to the cloud-based IoT platform every 60 seconds). Designing efficient sampling and scheduling strategies and configuring the sensors accordingly is challenging. Specifically, standards need to be developed in order to define schedules that can be used across different types of sensor devices.

4) Data Acquisition: Such methods can be divided into two categories: based on responsibility and based on frequency [36]. There are two methods that can be used to acquire data from a sensor based on responsibility: push (e.g. the cloud requests data from a sensor and the sensor responds with data) and pull (e.g. the sensor pushes data to the cloud without continuous explicit cloud requests). Further, based on frequency, there are two data acquisition methods: instant (e.g. send data to the cloud when a predefined event occurs) and interval (e.g. send data to the cloud periodically). Pros, cons, and applicabilities of these different approaches are discussed in [36]. Using the appropriate data acquisition method based on context information is essential to ensure efficiency.

5) Dynamicity: This means the frequency of changing positions / appearing / disappearing of the sensors at a given location. IoT envisions that most of the objects we use in everyday lives will have sensors attached to them in the future. Ideally, we need to connect and configure these sensors to software platforms in order to analyse the data they generate and so understand the environment better. We have observed several domains and broadly identified different levels of dynamicity based

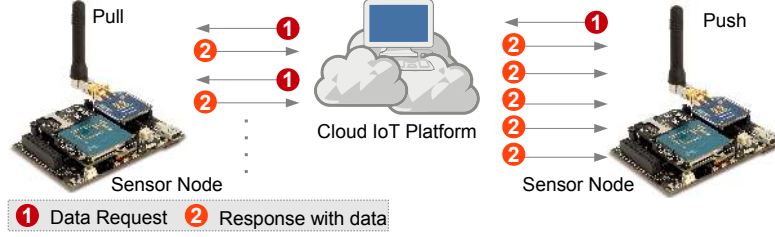


Fig. 4: Data can be retrieved from a sensor using both push (right side) and pull (left side) communication methods. Each method has its own advantages and disadvantages which make them suitable for different situations.

on mobility². Sensors that move/ appear/ disappear at a higher frequency (e.g. RFID and other low-level, low-quality, less reliable, cheap sensors that will be attached to consumables such as stationery, food packaging, etc.) can be classified as highly dynamic. Sensors embedded and fitted into permanent structures (such as buildings and air conditioning systems) can be classified as less dynamic. An ideal sensor configuration platform should be able to efficiently and continuously discover and re-configure sensors in order to cope with high dynamicity.

6) Context: Context information plays a critical role in sensor configuration in the IoT. The objective of collecting sensor data is to understand the environment better by fusing and reasoning them. In order to accomplish this task, sensor data needs to be collected in a timely and location-sensitive manner. Each sensor needs to be configured by considering context information. Let us consider a scenario related to smart agriculture to understand why context matters in sensor configuration. *Severe frosts and heat events can have a devastating effect on crops. Flowering time is critical for cereal crops and a frost event could damage the flowering mechanism of the plant. However, the ideal sampling rate could vary depending on both the season of the year and the time of day. For example, a higher sampling rate is necessary during the winter and the night. In contrast, lower sampling would be sufficient during summer and daytime. On the other hand, some reasoning approaches may require multiple sensor data readings. For example, a frost event can be detected by fusing air temperature, soil temperature, and humidity data. However, if the air temperature sensor stops sensing due to a malfunction, there is no value in sensing humidity, because frost events cannot be detected without temperature. In such circumstances, configuring the humidity sensor to sleep is ideal until the temperature sensor is replaced and starts sensing again. Such intelligent (re-)configuration can save energy by eliminating ineffectual sensing and network communication.*

² It is important to note that the same object can be classified at different levels depending on the context. Further, there is no clear definition to classify objects into different levels of dynamicity. However, our categorization allows us to understand the differences in dynamicity.

4 Related Work

In this section, we review some of the state-of-the-art solutions developed by the research community, as well as commercial business entities. Our review covers both mature and immature solutions proposed by start-up initiatives as well as large-scale projects. Our proposed CADDOT model as well as the *SmartLink* tool help to overcome some of the weaknesses in the existing solutions.

There are commercial solutions available in the market that have been developed by start-up IoT companies [40] and the research divisions of leading corporations. These solutions are either still under development or have completed only limited deployments in specialized environments (e.g. demos). We discuss some of the selected solutions based on their popularity. *Ninja Blocks* (ninjablocks.com), *Smart-Things* (smarthings.com), and *Twine* (supermechanical.com) are commercial products that aim at building smart environments [40]. They use their own standards and protocols (open or closed) to communicate between their own software systems and sensor hardware components. The hardware sensors they use in their solutions can only be discovered by their own software systems. In contrast, our pluggable architecture can accommodate virtually any sensor. Further, our proposed model can facilitate different domains (e.g. indoor, outdoor) using different communication protocols and sequences.

In addition, the CADDOT model can facilitate very high dynamicity and mobility. *HomeOS* [15] is a home automation operating system that simplifies the process of connecting devices together. Similar to our plugin architecture, *HomeOS* is based on applications and drivers which are expected to be distributed via an on-line store called *HomeStore* in the future. However, *HomeOS* does not perform additional configuration tasks (e.g. scheduling, sampling rate, communication frequency) depending on the user requirements and context information. Further, our objective is to develop a model that can accommodate a wider range of domains by providing multiple alternative mechanisms, as discussed in Section 6. Hu et al. [21] have proposed a sensor configuration mechanism that uses the information store in TEDS [22] and SensorML [7] specifications. Due to the unavailability and unpopularity of TEDS among sensor manufacturers, we simulate TEDS using standard communication message formats, as explained in Section 6.

Actinium [26] is a RESTful runtime container that provides Web-like scripting for low-end devices through a cloud. It encapsulates a given sensor device using a container that handles the communication between the sensor device and the software system by offering a set of standard interfaces for sensor configuration and life-cycle management. The Constrained Application Protocol (CoAP), a software protocol intended to be used in very simple electronics devices that allows them to communicate interactively over the Internet, has been used for communication. Pereira et al. [32] have also used CoAP and it provides a request/response interaction model between application end-points. It also supports built-in discovery of services and resources. However, for discovery to work, both the client (e.g. a sensor) and the server (e.g. the IoT platform) should support CoAP. However, most of the sensor manufacturers do not provide native support for such protocols. *Dynamix*

[9] is a plug-and-play context framework for Android. *Dynamix* automatically discovers, downloads, and installs the plugins needed for a given context sensing task. *Dynamix* is a stand-alone application and it tries to understand new environments using pluggable context discovery and reasoning mechanisms. Context discovery is the main functionality in *Dynamix*. In contrast, our solution is focused on dynamic discovery and configuration of 'things' in order to support a sensing as a service model in the IoT domain. We employ a pluggable architecture which is similar to the approach used in *Dynamix*, in order to increase the scalability and rapid extension development by third party developers. The Electronic Product Code (EPC) [16] is designed as a universal identifier that provides a unique identity for every physical object anywhere in the world. EPC is supported by the CADDOT model as one way of identifying a given sensor. Sensor integration using IPv6 in building automation systems is discussed in [24]. Cubo et al. [12] have used a Device Profile for Web Services³ (DPWS) to encapsulate both devices and services. DPWS defines a minimal set of implementation constraints to enable secure web service messaging, discovery, description, and eventing on resource-constrained devices. However, discovery is only possible if both ends (client and server) are DPWS-enabled.

5 Overview of the CADDOT Model

Previously, we identified several major factors that need to be considered when developing an ideal sensor configuration model for the IoT. This section presents a detailed explanation of our proposed solution: Context-aware Dynamic Discovery of Things (CADDOT). Figure 5 illustrates the main phases of the proposed model.

Phases in CADDOT model: The proposed model consists of eight phases: *detect*, *extract*, *identify*, *find*, *retrieve*, *register*, *reason*, and *configure*. Some of the tasks mentioned in the model are performed by the *SmartLink* tool and other tasks are performed by the cloud middleware. Some tasks are performed collectively by both *SmartLink* and the cloud.

1) Detect: Sensors are configured to actively seek open wireless access points (WiFi or Bluetooth) to which they can be connected without any authorization, because in this phase sensors do not have any authentication details. Sensors will receive the authentication details in phase **phase 8**). As a result, in this phase sensors are unable to connect to an available secured network. The mobile device that *SmartLink* is installed in becomes an open wireless access point (hotspot) so the sensors can connect to it. However, it is important to note that there are different application strategies that *SmartLink* can use to execute the CADDOT model, as discussed in Section 6.

2) Extract: In this phase, *SmartLink* extracts information from the sensor detected in the previous phase. Each sensor may be designed to respond to different message-passing sequences, as illustrated in Figure 3, depending on the sensor

³ <http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01>

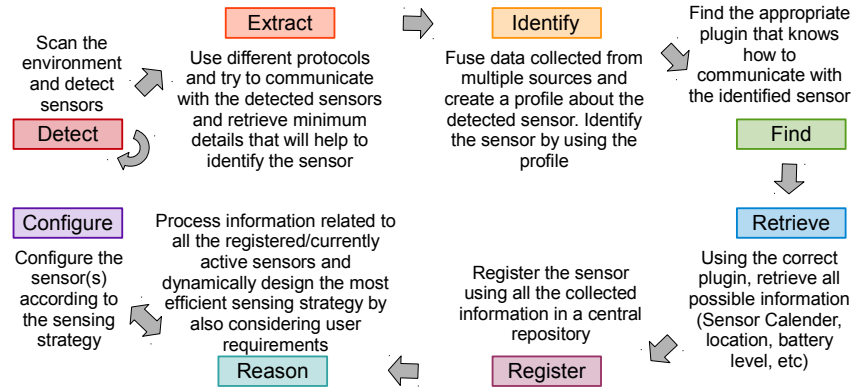


Fig. 5: Context-aware Dynamic Discovery of Things (CADDOT) model for configuration of things in the IoT paradigm consists of eight phases.

manufacturer and the sensor program developer. Even though the sensors and the *SmartLink* may use the same communication technology/ protocol (e.g. TCP, UDP, Bluetooth), the exact communication sequence can vary from one sensor to another. Therefore, it is hard to find the specific message-passing sequence that each sensor follows. To address this challenge, we propose that every sensor will respond to a common message during the communication initiation process. Alternatively, CADDOT can support multiple initiation messages (extraction mechanisms). However, such alternative approaches will increase the time taken to extract a minimum set of information from a given sensor due to multiple communication attempts that need to be carried out until a sensor successfully responds. For example, *SmartLink* broadcasts a message [WHO], as illustrated in (C1) in Figure 10, where the sensors are expected to respond by providing a minimum amount of information about themselves, such as a sensor’s unique identification number, model number / name, and manufacturer. This is similar to the TEDS mechanism discussed in [21]. It is important to note that we propose this [WHO] constraint only for minimum information extraction. Once the sensor is identified, subsequent communications and heterogeneity of message-passing sequences are handled by matching plugins.

3) Identify: *SmartLink* sends all the information extracted from the newly detected sensor to the cloud. Cloud-based IoT middleware queries its data stores using the extracted information and identifies the complete profile of the sensor. The descriptions of the sensors are modelled in an ontology⁴.

4) Find: Once the cloud identifies the sensor uniquely, this information is used to find a matching plugin (also called drivers) which knows how to communicate

⁴ This is an extended version of an SSN ontology (www.w3.org/2005/Incubator/ssn/ssnx/ssn). The detailed description of our extended ontology is out of the scope of this chapter.

with a compatible sensor at full capacity. The IoT middleware pushes the plugin to *SmartLink* where it is installed⁵.

5) Retrieve: Now, *SmartLink* knows how to communicate with the detected sensor at full capacity with the help of the newly downloaded plugin. Next, *SmartLink* retrieves the complete set of information that the sensor can provide (e.g. configuration details such as schedules, sampling rates, data structures /types generated by the sensor, etc.). Further, *SmartLink* may communicate with other available sources (e.g. databases, web services) to retrieve additional information related to the sensor.

6) Register: Once all the information about a given sensor has been collected, registration takes place in the cloud. The sensor descriptions are modelled according to the semantic sensor network ontology (SSNO) [12]. This allows semantic querying and reasoning at a later stage to perform operations such as sensor search [35]. Some of the performance evaluation related to the SSN ontology and semantic querying is presented in [39].

7) Reason: This phase plays a significant role in the sensor configuration process. It designs an efficient sensing strategy. Reasoning takes place in a distributed manner. The cloud IoT middleware retrieves data from a large number of sensors and identifies their availabilities and capabilities. Further, it considers context information in order to design an optimized strategy. Context-aware reasoning is performed by IoT middleware on the cloud. However, the technical details related to this reasoning process are out of the scope of this chapter. At the end of this phase, a comprehensive plan (i.e. sensing schedule) for each individual sensor is designed.

8) Configure: Sensors as well as cloud-based IoT software systems are configured based on the strategy designed in the previous phase. Schedules, communication frequency, and sampling rates that are custom-designed for each sensor are pushed into the individual sensors. The connections between sensors and the cloud-based IoT software system are established through direct wireless communication or through intermediate devices such as MOSDEN [34] so the cloud can retrieve data from sensors. The configuration details (e.g. IP address, port, authentication) required to accomplish the above task are also provided to the sensor.

6 Design Decisions and Applications

We made a number of design decisions during the development of the CADDOT model. These decisions address the challenges we highlighted in earlier sections.

Security Concerns and Application Strategies: There are different ways to employ our proposed model CADDOT as well as the tool *SmartLink* in real world deployments. Figure 6 illustrates two different application strategies. It is important to note that neither our model nor the software tool is limited to a specific device or platform. In this paper, we conduct the experimentations on an Android-based mobile phone, as detailed in Section 7. In strategy (a), a Raspberry Pi (raspberrypi.org)

⁵ In practice, the IoT middleware sends a request to the application store (e.g. Google Play). The application store pushes the plugin to the *SmartLink* autonomously via the Internet.

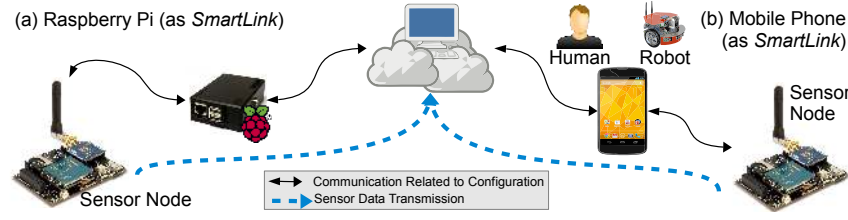


Fig. 6: Application strategies of CADDOT model and *SmartLink* tool. (a) usage of static *SmartLink* (b) usage of mobile *SmartLink*.

is acting as the *SmartLink* tool. This strategy is mostly suitable for smart home and office environments where WiFi is available. Raspberry Pi continuously performs the discovery and configuration process, as explained in Section 5. Finally, Raspberry Pi provides the authentication details to the sensor which is connected to the secure home/office WiFi network. The sensor is expected to send data to the processing server (local or on cloud) directly over the secured WiFi network. In this strategy, *SmartLink* is in static mode. Therefore, several *SmartLink* installed Raspberry Pi devices may be required to cover a building. However, this strategy can handle a high level of dynamicity.

The strategy (b) is more suitable for situations where WiFi is not available or less dynamic. Smart agriculture can be considered as an example. In this scenario, sensors are deployed over a large geographical area (e.g. Phenonet [11]). Mobile robots⁶ (tractors or similar vehicles) with a *SmartLink* tool attached to them can be used to discover and configure sensors. *SmartLink* can then help to establish the communication between sensors and sinks. The permanent sinks used in the agricultural fields are usually low-level sinks (such as Messhablium [27]). Such sinks cannot perform sensor discovery or configuration in comparison to *SmartLink*. Such sinks are designed to collect data from sensors and upload to the cloud via 3G.

Many more different strategies can be built by incorporating the different characteristics pointed out in the above two strategies. This shows the extensibility of our solution. For example, Raspberry Pi, which we suggested for use as a *SmartLink* in strategy a), can be replaced by corporate mobile phones. So, without bothering the owner, corporate mobile phones can silently perform the work of a *SmartLink*.

System Architecture: The CADDOT model consists of three main components: sensors, a mobile device (i.e. *SmartLink*), and the cloud middleware. All three components need to work collectively in order to perform sensor discovery and configuration successfully. Figure 7 illustrates the interactions between the three components. The phases we explained earlier relating to the CADDOT model in Figure 5 can be seen in Figure 7 as well. As we mentioned before, *SmartLink* is based on a plugin architecture. The core *SmartLink* application cannot directly communicate with a given sensor. A plugin needs to act as a mediator between the sensor and the *SmartLink* core application, as illustrated in Figure 7. The task of the mediator is

⁶ In small agricultural fields, farmers themselves can carry the *SmartLink* over the field.

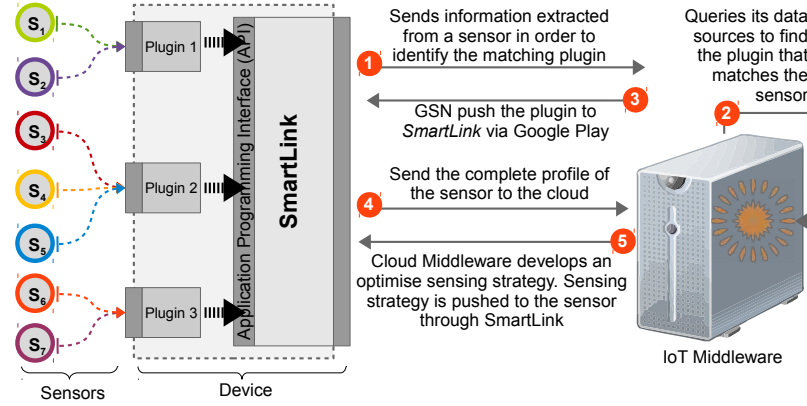


Fig. 7: System architecture of the CADDOT model which consists of three main components: sensors, *SmartLink* tool, and the cloud middleware. Interactions are numbered in order.

to translate the commands back and forth. This means that in order to configure a specific sensor, the *SmartLink* core application needs to employ a plugin that is compatible with both the *SmartLink* application itself and the given sensor. We discuss this matter in the programming perspective later in this section.

Sensor-level Program Design: One of the most important components in the CADDOT model is the sensor. Sensors can be programmed in different ways. In this chapter, we propose a program design that supports all the functional requirements identified in Section 3. The program we propose may not be the only way to support these requirements. Further, we do not intend to restrict developers to one single sensor-level program design. Instead, our objective is to demonstrate one successful way to program a sensor in such a way that it allows sensors to be re-configured at runtime (i.e. after deployment) depending on the requirements that arise later. Developers are encouraged to explore more efficient program designs. However, in order to allow *SmartLink* to communicate with a sensor which runs different program designs, developers need to develop a plugin that performs the command translations. We explain the translation process using both sensor-level program code as well as plugin code later in this section. First, we illustrate the simplest sensor-level program that can be designed to perform the task of sensing and transmitting data to the cloud in Figure 8. We refer to this program design as *SPD* (Simple Program Design) hereafter. The basic structure of a sensor-level program is explained in [27].

The main problem in this program design is that there is no way to configure (i.e. sampling rate, communication frequency, data acquisition method) the sensor after deployment other than by re-programming (e.g. Over the Air Programming). However, such re-programming approaches are complex, labour-intensive and time consuming. In Figure 9, we designed a sensor-level program that supports a comprehensive set of configuration functionalities. We refer to this design as *CPD* (Con-

```

1. Include Libraries    2. Definitions    3. Global variables declaration
void setup()
{
    4. Modules initialisation [Communication] } // This code only runs once
}
void loop()
{
    5. Connect to an access point
      [IP address, Port number]
    6. Sense the phenomenon
    7. Send information to the cloud
    8. Sleep [Communication Frequency] } // This code runs continuously,
                                        forming an infinite loop.
}

```

Fig. 8: A simple sensor-level program design (SPD) that sends and transmits data to the cloud. It does not support dynamic discovery and configuration.

figurable Program Design) hereafter. In order to standardize the communication, we also defined a number of command formats. However, these messaging formats do not need to be followed by the developers as long as they share common standardised command formats between their own sensor-level program and the corresponding plugin. Different command formats used to accomplish different tasks in our approach are illustrated in Figure 10. In comparison to *SPD*, *CPD* provides more configuration functionalities. With the help of the command formats illustrated in Figure 10, *SmartLink* can configure a given sensor at any time.

Each command comprises several different segments, as depicted in Figure 10. The first segment denotes whether the command is related to configuration or a data request. In our approach, [CON] denotes configuration and [DAR] denotes a data request. The CPD is designed to forward the command appropriately through IF-ELSE branches. The CPD accepts five different types of commands under the [CON] branch. Commands are classified based on the second segment. The following list summarises these commands. The first segment of every command contains only three letters which makes it easy to process. The commands can be sent using frames⁷ or plain strings.

- **C1:** This command has only one segment. This segment always contains three letters [WHO]. This command is sent by *SmartLink* to a sensor. To support CADDOT, every sensor should be able to handle command C1. Then the sensor needs to respond with message **M1**. This is the only constraint that the sensor-level program developers are required to adhere to.
- **M1:** This message is sent by the sensor to *SmartLink* in response to C1. M1 contains information that helps to identify the sensor in *key-value pair* format. The information contained in this message is sent to the cloud IoT platform, as explained in phase (4) in the CADDOT model illustrated in Figure 5. Detailed explanation of this message is out of the scope of this chapter.

⁷ http://www.libelium.com/uploads/2013/02/data_frame_guide.pdf

```

    Include Libraries      Definitions      Global variables declaration

void setup()
{
    Read the parameters[SR, CF] from a file In SD card
    Modules initialisation [Communication]
}
void loop()
{
    while(isConnected()){
        // This method tries to discover an open access point. Next, it also
        // attempts to establish a connection between the sensor and SmartLink.

        RequestType = ReadSegmentOne()
        // Listen to the communication channel (e.g. WiFi)

        IF (RequestType[segment 1] == DAR){

            IF (RequestType[segment 2] == PL){
                // Sense data and send back to the requester
            }ELSE IF (RequestType[segment 2] == PT){
                // Sense and send data back continuously but temporarily
                // until the request expires. Request contains the information
                // such as sampling rate, communication frequency, duration

            }ELSE IF (RequestType[segment 2] == PS){
                // Sense and send data back continuously according to the
                // schedule specified. contains the information such as
                // sampling rate, communication frequency, start time, end time
            }
        }ELSE IF (RequestType[segment 1] == CON){

            IF (RequestType[segment 2] == SMP){
                // Change the sampling rate as specified
            }ELSE IF (RequestType[segment 2] == DCF){
                // Change the data communication frequency as specified
            }ELSE IF (RequestType[segment 2] == SCH){
                // Download the schedule file from the given location
            }ELSE IF (RequestType[segment 2] == CPR){
                // Send the complete sensor profile information
            }ELSE IF (RequestType[segment 2] == NET){
                // Store network setting such as access point, authentication
                // key, IP address and port numbers
            }ELSE{
                // Send back an error message
            }
        }ELSE IF (RequestType[segment 1] == WHO){
            // Send back the identification details (e.g. SensorID)
        }ELSE{
            // Send back an error message
        }
    }
}

```

Fig. 9: A configurable sensor-level program design (CPD) that supports dynamic discovery and configuration after deployment at runtime.

- **C2:** This command consists of two segments. The first segment [DAR] denotes that this is a data request. The second segment [PL] denotes that the command is a pull request which the sensor is expected to respond to with sensors data once.

- **C3:** This command consists of five segments. The first segment [DAR] denotes that this is a data request. The second segment [PS] denotes that the sensor is expected to push data according to the information provided in the rest of the segments. The third segment specifies the sample rate and the fourth segment specifies the data communication frequency rate. The final segment specifies the duration for which the sensor needs to push data to the cloud.
- **C4:** This command consists of two segments. The first segment [DAR] denotes that this is a data request. The second segment [PS] denotes that the sensor is expected to perform sensing and data transmitting tasks according to a sensing schedule specified in the sensing schedule file. It is expected to push data to the cloud.
- **C5:** This command consists of three segments. The first segment [CON] denotes that this is a configuration command. The second segment [SMP] denotes that this command configures the sampling rate. The third segment holds the actual sampling rate value that the sensor needs to sense in the future.
- **C6:** This command consists of three segments. The first segment [CON] denotes that this is a configuration command. The second segment [DCF] denotes that this command configures the data communication frequency. The third segment holds the actual data communication frequency rate value that the sensor needs to transmit data to the cloud in the future.
- **C7:** This command consists of five segments. The first segment [CON] denotes that this is a configuration command. The second segment [SCH] denotes that this command configures the sensing schedule. The rest of the segments contain information that is essential (i.e. FTP server path, user name, password) to download a sensing schedule file from an FTP server, as depicted in Figure 10.
- **C8:** This command consists of seven segments. The first segment [CON] denotes that this is a configuration command. The second segment [NET] denotes that this command configures the network settings. The rest of the segments contain the information that is essential to connect to a secure network (i.e. access point name, authentication key, IP address, remote port) so the sensor can directly communicate with the cloud IoT platform.
- **C9:** This command stops the sensor completely and pushes it back to a state where the sensor listens for the next command.
- **C10:** This command consists of two segments. The first segment [CON] denotes that this is a configuration command. The second segment [CPR] denotes that the sensor is expected to reply with the complete sensor profile.

Scalable and Extensible Architecture: As we mentioned earlier, the reason for employing a plugin architecture is to support scalability and extensibility. Plugins that are compatible with *SmartLink* can be developed by anyone as long as they follow the basic design principles and techniques explained below. Such a plugin architecture allows us to engage with developer communities and support a variety of different sensors through community-based development. We expect to release our software as free and open source software in the future. We provide the main *SmartLink* application as well as the standard interfaces which developers can use to start to develop their own plugins to support different sensors. We provide sample

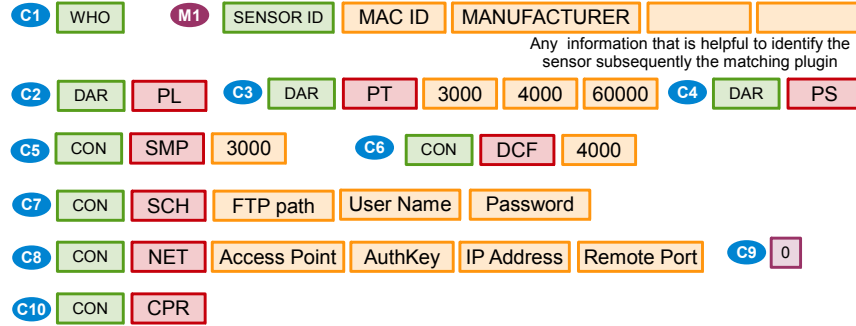


Fig. 10: Command formats used to perform sensor configuration.

plugin source code where developers only need to add their code according to the guidelines provided. The plugin architecture will enable more number of sensors to be supported by *SmartLink* over time. Applications stores (e.g. *Google Play*) built around the Android ecosystem provide an easy way to share and distribute plugins for *SmartLink*. The pluggable architecture dramatically reduces the sensor configuration time.

Let us explain how third party developers can develop plugins in such a way that their plugins are compatible with *SmartLink* so that *SmartLink* can use the plugins to configure sensors at runtime when necessary. In plugin development, there are three main components that need to be considered: (1) the plugin interface written in the Android Interface Definition Language (AIDL), (2) the plugin class written in Java, and (3) the plugin definition in the AndroidManifest file. Figure 11 shows the plugin interface written in AIDL. *IPlugin* is an interface defined in AIDL. Plugin developers should not make any changes in this file. Instead they can use this file to understand how the *SmartLink* plugin architecture works. *IPlugin* is similar to a Java interface. It defines all the methods that need to be implemented by all the plugin classes.

```
package au.csiro.smartlink;
import au.csiro.smartlink.beans.SensorProfile;
interface IPlugin {
    boolean setSamplingRate(int rate);
    boolean setCommunicationFrequency(int frequency);
    boolean setSchedule(in Map ftpSettings);
    boolean setNetworkSettings(in Map netSettings);
    SensorProfile getSensorProfile();
}
```

Fig. 11: *IPlugin* written in AIDL (Android Interface Definition Language) that governs the plugin structure. It defines the essential methods that need to be implemented in the plugin class.

Figure 12 presents the basic structure of a *SmartLink* plugin. Each plugin is defined as an Android service. *SmartLink* plugin developers need to implement five methods: *setSamplingRate(int rate)*, *setCommunicationFrequency(int frequency)*, *setSchedule(in Map ftpSettings)*, *setNetworkSettings(in Map netSettings)* and *getSensorProfile()*. The methods are briefly explained below.

- *setSamplingRate(int rate)*: This method needs to send a command specifying the required sampling rate. For example, in our approach, we defined such a command, C5, in Figure 10.
- *setCommunicationFrequency(int frequency)*: This method needs to send a command specifying the required communication frequency. For example, in our approach, we defined such a command as C6 in Figure 10.
- *setSchedule(in Map ftpSettings)*: This method needs to send a command specifying details (e.g. user-name, password, FTP path) that are required to connect to an FTP server and download the schedule. For example, in our approach, we defined such a command as, C7, in Figure 10.
- *setNetworkSettings(in Map netSettings)*: This method sends a command specifying the details that are required to connect to a secure network so that direct communication between the sensor and the cloud IoT platform can be established. For example, in our approach, we defined such a command, C8, in Figure 10.
- *getSensorProfile()*: This method sends a command to the sensor by asking for profile information. The sensor is expected to reply by providing information such as the data structure it produces, measurement units, and so on. Details of the sensor profiling are out of the scope of this chapter.

Figure 13 shows how the plugins need to be defined in the AndroidManifest so that the *SmartLink* application can automatically query and identify them. The Android plugin must have an intent filter which has action name *au.csiro.smartlink.intent.action.PICKPLUGIN*. Developers can provide any category name.

Support and Utilize Existing Solutions: Our model utilizes a few existing solutions. We employed Global Sensor Network [1] as the cloud IoT middleware. In CADDOT, GSN performs phases 3, 4, and 7. GSN is a widely used platform in the sensor data processing domain and is used in several European projects, including OpenIoT [31]. MOSDEN [34] is middleware that collects sensor data. MOSDEN is ideal for the application strategies we discussed in Section 6 (Figure 6) for use in conjunction with *SmartLink*. *SmartLink* only performs the configuration. Sensor data collection needs to be performed by either cloud IoT middleware or solutions like MOSDEN. The proposed CADDOT model as well as the *SmartLink* tool complement the other solutions proposed by us as well as other researchers. Together, these solutions enable smooth data flow from sensors to the cloud autonomously.

```

public class [Class] extends Service implements [Any Interface]{
    public int onStartCommand(Intent intent, int flags, int
startId) {...}
    public void onDestroy() {...}

    public IBinder onBind(Intent intent) {...}

    private final IFunction.Stub mulBinder = new IPlugin.Stub(){
        public boolean setSamplingRate(int rate) throws
RemoteException {...}

        public boolean setCommunicationFrequency(int frequency)
throws RemoteException {...}

        public boolean setSchedule(Map ftpSettings) throws
RemoteException {}

        public setNetworkSettings(Map netSettings) throws
RemoteException {...}

        public SensorProfile getSensorProfile() throws
RemoteException {}
    }
}

```

Fig. 12: *SmartLink* plugin is an Android service. This is the basic structure of a *SmartLink* plugin. The body of each method needs to be added by the developer based on the sensor-level program design.

```

<service
    android:name=[Plugin name]
    android:exported="true" >
    <intent-filter>
        <action android:name="au.csiro.smartlink.intent.action.PICK_PLUGIN"/>
        <category android:name="au.csiro.smartlink.intent.category.[PLUGIN_NAME]"/>
    </intent-filter>
</service>

```

Fig. 13: Code snippet of the plugin's *AndroidManifest* file.

7 Implementation and Experiment Testbed

We deployed the *SmartLink* application in a Google Nexus 4 mobile phone (Qualcomm Snapdragon S4 Pro CPU and 2 GB RAM), which runs the Android platform 4.2.2 (Jelly Bean). We deployed 52 sensors on the third floor of the CSIT building (#108) at the Australian National University. All sensors we employed in our experiment are manufactured by Libelium [27]. The sensors we used sense a wide variety of environmental phenomena, such as temperature, proximity & presence, stretch, humidity and so on [27]. *SmartLink* supports sensor discovery and configuration using both WiFi and Bluetooth. Other communication technologies such as ZigBee and RFID are supported through Libelium *Expansion Radio Boards* [27]. In order to simulate the heterogeneity of the sensors (in terms of communication sequence), we programmed each sensor to behave and respond differently. As a result, each sensor can only communicate with a plugin that supports the same communication sequence.

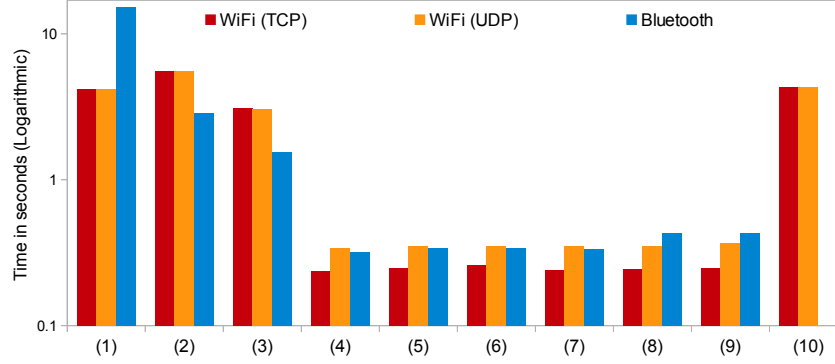


Fig. 14: Time taken (y-axis) to discover and configure a sensor step-by-step (x-axis). The experiments were conducted using three protocols: TCP, UDP, and Bluetooth.

8 Evaluation of the Prototype

In this section, we explain how we evaluate the proposed CADDOT model and *SmartLink* tool using prototype implementations. We identified ten steps performed in the dynamic discovery and sensor configuration process. We measured the average amount of time taken by each of these steps (average of 30 sensor configurations). Figure 14 illustrates the results and the following steps are considered: Time taken to (1) set up the sensor, (2) initiate connection between the sensor and *SmartLink*, (3) initiate communication between sensor and *SmartLink*, (4) extract sensor identification information, (5) retrieve the complete profile of the sensor, (6) configure the sampling rate, (7) configure the communication frequency, (8) configure the sensing schedule, (9) configure the network and authentication details (so the sensor can directly connect to the cloud), and (10) connect to the secure network using the provided authentication details.

Results: According to the results, the actual configuration tasks take less than one second. There is a slight variation in completion time in configuration step (4) - (9). This is due to storage access and differences in processing of configuration commands. Sensors take comparatively longer time to connect to a network as well as to discover and connect to *SmartLink*. Especially, Bluetooth takes much longer to scan for devices in a given environment before it discovers and connects to *SmartLink*. Configuration is slightly faster when using TCP in comparison to UDP and Bluetooth. This is mainly due to reliability. However, the time differences are negligible. FTP is used to retrieve a scheduling file from a file server. This can take 15-25 seconds depending on the network availability, traffic, and file size. If a sensor cannot access a server via the Internet, a file can be transferred from *SmartLink* to the sensor as typical commands. Sensors generate the scheduling file using the data it receives from *SmartLink*. When using WiFi, a sensor may take up to 4.5 seconds to connect to a secure network (e.g. WPA2). In contrast, sensors can connect to

SmartLink's open access point in less than four seconds. Despite the protocol we use, sensors take 5 to 15 seconds to boot and setup themselves. The setup stage consists of activities such as reading default configuration from files, and switching necessary modules and components (communication modules, real-time clock, SD card, sensor boards and so on).

9 Discussion and Lessons Learned

In what follows, we discuss major lessons we learned along with limitations. According to our results, it is evident that a single sensor can be configured in less than 12 seconds (i.e. assuming sensors are already booted, which takes an additional 5 to 15 seconds depending on the communication protocol). This is a significant improvement over a manual labour intensive sensor configuration approach. Additionally, *SmartLink* can engage with number of sensor configuration processes at a given time in parallel. The proposed CPD has not made any negative impact towards the sensing functionality though it supports advance configuration capabilities. The IF-ELSE structure used in CPD makes sure that each request gets to the destination with minimum execution of lines (e.g. 'PL' request passes through only two IF conditions). Such execution reduced the impact on sensing tasks while configuration tasks are also supported efficiently. Even though a detailed discussion on data acquisition methods is out of scope, it is important to note that pull, temporary push, and schedule based push add a significant amount of flexibility where each of the techniques is suitable to be used in different circumstances [36]. The cloud server has the authority to decide which method to be used based on the context information. This increases the efficiency and application scenario where the sensors can be used in sustainable (i.e. in term of energy) manner. Once the initial discovery and configuration of smart things are done, further configuration can be done in more user friendly manner by using techniques such as augmented reality [19].

10 Open Challenges

In this section, we briefly introduce some of the major open research challenges in the domain that are closely related to this work. We identify four main challenges that provide different research directions.

Sensing strategy optimization: We briefly highlighted the importance of optimizing sensing schedules based on context information in Section 3. Sensing strategy development encapsulates a broad set of actions such as deciding the sensing schedule, sampling rate, and network communication frequency for each sensor. Such a development process needs to consider two main factors: user requirements and availability of sensors. In IoT, there is no single point of control or authority. As a result, different parties are involved in sensor deployments. Such disorganized

and uncoordinated deployments can lead to redundant sensor deployment. In order to use the sensor hardware in an optimized manner, sensing strategies need to be developed by considering factors such as sensor capabilities, sensor redundancies (e.g. availability of multiple sensors that are capable of providing similar data), and energy availability. Energy conservation is a key in sustainable IoT infrastructure because the resources constrained nature of the sensors. We provided such an example in Section 3 related to the agricultural domain. We believe that sensing as a service is a major business model that could drive IoT in the future. In such circumstances, collecting data from all the available sensors has no value. Instead, sensor data should be collected and processed only in response to consumer demand [37].

Context discovery: This is an important task where discovered information will be used during a reasoning process (e.g. sensing strategy development). "*Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves*" [2]. Further discussion on context information and its importance for the IoT is surveyed in [36]. Context-based reasoning can be used to improve the efficiency of the CADDOT model where a matching plugin can be discovered faster, especially in situations where a perfect match cannot be found. For example, the location of a given sensor⁸, sensors nearby, details of the sensors configured recently, historic data related to sensor availability in a given location, etc. can be fused and reasoned using probabilistic techniques in order to find a matching plugin in an efficient manner. After integrating sensors into cloud-based IoT, the next phase is collecting data from the sensors. Annotating context information to retrieve sensor data plays a significant role in querying and reasoning them in later stages. Especially, in the sensing as a service model, sensor data consumers may demand such annotation so that they can feed data easily into their own data processing applications for further reasoning and visualization tasks. Some context information can be easily discovered at sensor-level (e.g. battery level, location) and others can be discovered at the cloud-level by fusing multiple raw data items (e.g. activity detection). Such context annotated data help to perform more accurate fusing and reasoning at the cloud level [30].

Utilization of heterogeneous computational devices: Even though the IoT envisions billions of 'things' to be connected to the Internet, it is not possible and practical to connect all of them to the Internet directly. This is mainly due to resource constraints (e.g. network communication capabilities and energy limitations). Connecting directly to the Internet is expensive in terms of computation, bandwidth use, and hardware costs. Enabling persistent Internet access is challenging and also has a negative impact on miniaturization and energy consumption of the sensors. Due to such difficulties, IoT solutions need to utilize different types of devices with different resource limitations and capabilities. In Figure 15, we broadly categorise these devices into six categories (also called levels or layers). Devices on the right side

⁸ Location can be represented in many ways: GPS coordinate (e.g. -35.280325, 149.113166), name of a building (e.g. CSIT building at ANU), name of a city (e.g. Canberra), part of a building (e.g. living room), floor of a building (e.g. 2nd floor), specific part of a room (e.g. kitchen-top).

may use low-energy short distance wireless communication protocols to transmit the collected sensor data to the devices on the left. Devices on the left can use long distance communication protocols to transmit the data to the cloud for further processing. However, the more devices we use in smart environments, the more difficult it becomes to detect faults where an entire system could fail [44]. Providing a unified middleware support across heterogeneity of devices with wider range of capabilities is an open challenge [10, 20].



Fig. 15: Categorization of IoT devices based on their computational capabilities. The devices belonging to each category have different capabilities in terms of processing, memory, and communication. They are also different in price, with devices becoming more expensive towards the left. The computational capabilities also increase towards the left.

Security and privacy: In this work, we considered some degree of security as briefly discussed in Section 6. However, research on security in the IoT is largely unexplored. Security and privacy need to be provided at both sensor-level and cloud-level. It is critical to develop a security model to protect the sensor configuration process, considering questions such as (1) *when to allow reconfiguration of a sensor*, (2) *who has the authority to configure a sensor at a given time*, (3) *how to change ownership of a sensor*, (4) *how to detect sensors with harmful programs installed on them that may cause security threats to a network*. Security and privacy concerns related to the IoT are presented in [41]. Additionally, security challenges unique to the sensing as a service model are discussed in [37].

11 Conclusions and Outlook

In this chapter, we addressed the challenge of integrating sensors into cloud-based IoT platforms through context-aware dynamic discovery and configuration. Traditionally, integration of 'things' to software solutions is considered a labour-intensive, expensive and time-consuming task that needs to be carried out by technical experts. Such challenges hinders the non-technical users from adopting IoT to build smart environments. To address this problem, we presented the CADDOT model, an approach that automates the sensor discovery and configuration process in smart environments efficiently and effortlessly by handling key challenges such as a higher number of sensors available, heterogeneity, on-demand sensing schedules, sampling rate, data acquisition methods, and dynamicity. It also encourages non-

technical users to adopt IoT solutions with ease by promoting automatic discovery and configuration IoT devices.

In this work, we supported and evaluated different types of communication technologies (i.e. WiFi and Bluetooth), application strategies, and sensor-level program designs, each of which has their own strengths and weaknesses. We validate the CADDOT model by deploying it in an office environment. As CADDOT required minimum user involvement and technical expertise, it significantly reduces the time and cost involved in sensor discovery and configuration. In the future, we expect to address the open challenges discussed in Section 10. In addition, we expect to integrate our solution with other existing solutions such as MOSDEN [34] and OpenIoT [31]. The functionality provided by CADDOT can improve these solutions in a major way.

Acknowledgements Authors acknowledge support from SSN TCP, CSIRO, Australia and ICT Project, which is co-funded by the European Commission under seventh framework program, contract number FP7-ICT-2011-7-287305-OpenIoT. The Author(s) also acknowledge help and contributions from The Australian National University.

References

1. K. Aberer, M. Hauswirth, and A. Salehi. Infrastructure for data processing in large-scale interconnected sensor networks. In *International Conference on Mobile Data Management*, pages 198–205, May 2007.
2. G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles. Towards a better understanding of context and context-awareness. In *Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, HUC ’99, pages 304–307, London, UK, 1999. Springer-Verlag.
3. K. Ashton. That ‘internet of things’ thing in the real world, things matter more than ideas. *RFID Journal*, June 2009. <http://www.rfidjournal.com/article/print/4986> [Accessed on: 2012-07-30].
4. L. Atzori, A. Iera, and G. Morabito. The internet of things: A survey. *Comput. Netw.*, 54(15):2787–2805, Oct. 2010.
5. BCC Research. Sensors: Technologies and global markets. Market forecasting, BCC Research, March 2011. <http://www.bccresearch.com/report/sensors-technologies-markets-ias006d.html> [Accessed on: 2012-01-05].
6. C. Bizer, P. Boncz, M. L. Brodie, and O. Erling. The meaningful use of big data: four perspectives – four challenges. *SIGMOD Rec.*, 40(4):56–60, Jan. 2012.
7. M. Botts and A. Robin. Opengis sensor model language (sensorml) implementation specification. Technical report, Open Geospatial Consortium Inc, 2007. https://portal.opengeospatial.org/modules/admin/license_agreement.php?suppressHeaders=0&access_license_id=3&target=http://portal.opengeospatial.org/files/%3fartifact_id=12606 [Accessed on: 2011-12-15].
8. A. B. Brush, E. Filippov, D. Huang, J. Jung, R. Mahajan, F. Martinez, K. Mazhar, A. Phanishayee, A. Samuel, J. Scott, and R. P. Singh. Lab of things: a platform for conducting studies with connected devices in multiple homes. In *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication*, UbiComp ’13 Adjunct, pages 35–38, New York, NY, USA, 2013. ACM.
9. D. Carlson and A. Schrader. Dynamix: An open plug-and-play context framework for android. In *Internet of Things (IOT), 2012 3rd International Conference on the*, pages 151–158, 2012.

10. M. Chaqfeh and N. Mohamed. Challenges in middleware solutions for the internet of things. In *Collaboration Technologies and Systems (CTS), 2012 International Conference on*, pages 21–26, 2012.
11. Commonwealth Scientific and Industrial Research Organisation (CSIRO), Australia. Phenonet: Distributed sensor network for phenomics supported by high resolution plant phenomics centre, csiro ict centre, and csiro sensor and sensor networks tcp., 2011. <http://phenonet.com> [Accessed on: 2012-04-20].
12. M. Compton, P. Barnaghi, L. Bermudez, R. Garca-Castro, O. Corcho, S. Cox, J. Graybeal, M. Hauswirth, C. Henson, A. Herzog, V. Huang, K. Janowicz, W. D. Kelsey, D. L. Phuoc, L. Lefort, M. Leggieri, H. Neuhaus, A. Nikolov, K. Page, A. Passant, A. Sheth, and K. Taylor. The ssn ontology of the w3c semantic sensor network incubator group. *Web Semantics: Science, Services and Agents on the World Wide Web*, 17(0):25 – 32, 2012.
13. D. Cook and S. Das. *Smart Environments: Technology, Protocols and Applications (Wiley Series on Parallel and Distributed Computing)*. Wiley-Interscience, 2004.
14. Cosm. Cosm platform, 2007. <https://cosm.com/> [Accessed on: 2012-08-05].
15. C. Dixon, R. Mahajan, S. Agarwal, A. Brush, B. Lee, S. Saroiu, and V. Bahl. An operating system for the home. In *Symposium on Networked Systems Design and Implementation (NSDI), USENIX*, April 2012.
16. EPCglobal. Epc tag data standard version 1.5. Standard specification, EPCglobal, August 2010. http://www.gs1.org/gsmp/kc/epcglobal/tds/tds_1_5-standard-20100818.pdf [Accessed on: 2011-08-16].
17. GSN Team. Global sensor networks project, 2011. <http://sourceforge.net/apps/trac/gsn/> [Accessed on: 2011-12-16].
18. P. Guillemin and P. Friess. Internet of things strategic research roadmap. Technical report, The Cluster of European Research Projects, September 2009. http://www.internet-of-things-research.eu/pdf/IoT_Cluster_Strategic_Research_Agenda_2009.pdf.
19. V. Heun, S. Kasahara, and P. Maes. Smarter objects: using ar technology to program physical objects and their interactions. In *CHI '13 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '13, pages 961–966, New York, NY, USA, 2013. ACM.
20. Y. Hong. A resource-oriented middleware framework for heterogeneous internet of things. In *Cloud and Service Computing (CSC), 2012 International Conference on*, pages 12–16, 2012.
21. P. Hu, J. Indulska, and R. Robinson. An autonomic context management system for pervasive computing. In *Pervasive Computing and Communications, 2008. PerCom 2008. Sixth Annual IEEE International Conference on*, pages 213 –223, march 2008.
22. IEEE Instrumentation and Measurement Society. Ieee standard for a smart transducer interface for sensors and actuators wireless communication protocols and transducer electronic data sheet (teds) formats. *IEEE Std 1451.5-2007*, pages C1 –236, 5 2007.
23. International Data Corporation (IDC) Corporate USA. Worldwide smart connected device shipments, March 2012. <http://www.idc.com/getdoc.jsp?containerId=prUS23398412> [Accessed on: 2012-08-01].
24. M. Jung, C. Reinisch, and W. Kastner. Integrating building automation systems and ipv6 in the internet of things. In *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on*, pages 683–688, 2012.
25. J. Kiljander, J. Takalo-Mattila, M. Etelapera, J.-P. Soininen, and K. Keinanen. Enabling end-users to configure smart environments. In *Applications and the Internet (SAINT), 2011 IEEE/IPSJ 11th International Symposium on*, pages 303–308, 2011.
26. M. Kovatsch, M. Lanter, and S. Duquenooy. Actinium: A restful runtime container for scriptable internet of things applications. In *Internet of Things (IOT), 2012 3rd International Conference on the*, pages 135–142, 2012.
27. Libelium Comunicaciones Distribuidas. libelium, 2006. <http://www.libelium.com/> [Accessed on: 2012-011-28].
28. J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers. Big data: The next frontier for innovation, competition, and productivity. Technical report, McKinsey Global Institute, May 2011. http://www.mckinsey.com/Insights/MGI/Research/Technology_and_Innovation/Big_data.The_next_frontier_for_innovation [Accessed on: 2012-06-08].

29. M. Naphade, G. Banavar, C. Harrison, J. Paraszczak, and R. Morris. Smarter cities and their innovation challenges. *Computer*, 44(6):32–39, 2011.
30. Y. Oh, J. Han, and W. Woo. A context management architecture for large-scale smart environments. *Communications Magazine, IEEE*, 48(3):118–126, 2010.
31. OpenIoT Consortium. Open source solution for the internet of things into the cloud, January 2012. <http://www.openiot.eu> [Accessed on: 2012-04-08].
32. P. Pereira, J. Eliasson, R. Kyusakov, J. Delsing, A. Raayatinezhad, and M. Johansson. Enabling cloud connectivity for mobile internet of things applications. In *Service Oriented System Engineering (SOSE), 2013 IEEE 7th International Symposium on*, pages 518–526, 2013.
33. C. Perera, P. Jayaraman, A. Zaslavsky, P. Christen, and D. Georgakopoulos. Dynamic configuration of sensors using mobile sensor hub in internet of things paradigm. In *IEEE 8th International Conference on Intelligent Sensors, Sensor Networks, and Information Processing (ISSNIP)*, pages 473–478, Melbourne, Australia, April 2013.
34. C. Perera, P. P. Jayaraman, A. Zaslavsky, P. Christen, and D. Georgakopoulos. Mosden: An internet of things middleware for resource constrained mobile devices. In *Proceedings of the 47th Hawaii International Conference on System Sciences (HICSS)*, Hawaii, USA, January 2014.
35. C. Perera, A. Zaslavsky, P. Christen, M. Compton, and D. Georgakopoulos. Context-aware sensor search, selection and ranking model for internet of things middleware. In *IEEE 14th International Conference on Mobile Data Management (MDM)*, Milan, Italy, June 2013.
36. C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos. Context aware computing for the internet of things: A survey. *Communications Surveys Tutorials, IEEE*, 2013. (in press).
37. C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos. Sensing as a service model for smart cities supported by internet of things. *Transactions on Emerging Telecommunications Technologies (ETT)*, 2014. (in press).
38. C. Perera, A. Zaslavsky, P. Christen, A. Salehi, and D. Georgakopoulos. Capturing sensor data from mobile phones using global sensor network middleware. In *IEEE 23rd International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC)*, pages 24–29, Sydney, Australia, September 2012.
39. C. Perera, A. Zaslavsky, C. H. Liu, M. Compton, P. Christen, and D. Georgakopoulos. Sensor search techniques for sensing as a service architecture for the internet of things. *IEEE Sensors Journal*, 2014. (in press).
40. Postscapes.com. A showcase of the year's best Internet of Things projects, 2012. <http://postscapes.com/awards/winners> [Accessed on: 2013-01-10].
41. R. Roman, P. Najera, and J. Lopez. Securing the internet of things. *Computer*, 44(9):51–58, 2011.
42. X. Sheng, J. Tang, X. Xiao, and G. Xue. Sensing as a service: Challenges, solutions and future directions. *Sensors Journal, IEEE*, 13(10):3733–3741, 2013.
43. T. Shon and Y. Park. Implementation of rf4ce-based wireless auto configuration architecture for ubiquitous smart home. In *Complex, Intelligent and Software Intensive Systems (CISIS), 2010 International Conference on*, pages 779–783, 2010.
44. J.-Y. Son, J.-H. Lee, J.-Y. Kim, J.-H. Park, and Y.-H. Lee. Rafd: Resource-aware fault diagnosis system for home environment with smart devices. *Consumer Electronics, IEEE Transactions on*, 58(4):1185–1193, 2012.
45. H. Sundmaeker, P. Guillemin, P. Friess, and S. Woelffle. Vision and challenges for realising the internet of things. Technical report, European Commission Information Society and Media, March 2010. http://www.internet-of-things-research.eu/pdf/IoT_Clusterbook_March_2010.pdf [Accessed on: 2011-10-10].
46. M. Weiser, R. Gold, and J. S. Brown. The origins of ubiquitous computing research at parc in the late 1980s. *IBM SYSTEMS JOURNAL*, 38(4):693–696, 1999.