Institute of Architecture of Application Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Master Thesis Nr. 0838-013

Designing a Context-Aware Discovery Service for IoT Devices

Muhammad Arsalan Khan

Course of Study:	Information Technology (INFOTECH)
Examiner:	Prof. Dr. Dr. h. c. Frank Leymann
Supervisors:	Dr. rer. nat. Uwe Breitenbücher M.Sc. Kalman Kepes
Commenced:	4. August 2016
Completed:	25. January 2017
CR-Classification:	C.0, C.3, D.2.11, H.3.4

Abstract

Internet of Things (IoT) is in a continuous expansion phase, from millions of devices to tens of billions in upcoming years, which will have major impacts on infrastructure, business models, and industry standards throughout the entire IT ecosystem. It is expected that several diverse devices to invade by 2020. Depending on different application domains, IoT applications require devices, sensors, middlewares, networks and other enabling technologies to be integrated e.g., the high-level central control of IoT applications can be deployed on the cloud while others are running close to the "edge", forming a unified, scalable and feasible system. One of the important integration aspects in IoT ecosystem is discovering devices and sensors based on a particular context regardless of their heterogeneity.

In this thesis, we propose Context-Aware Discovery Service for Internet of Things (CADsIoT) that deals with devices and sensors installed in IoT environments, streamlining the process of registration, management and dynamic discovery of devices based on contextual information. CADsIoT allows device and sensor registration, attaching them with particular context and leverage subscription features to enable dynamic discovery based on the attached context. Additionally, real-time notifications are triggered when new devices are discovered. For the validation of our concept, we discuss the requirements and a descriptive motivation scenario, which is followed by a discussion of the prototypical implementation. The prototype consists of *CADsIoT Core*, a Representational State Transfer (REST) based middleware and *Navigator*, an Android mobile application as a client.

Contents

List of Figures

List of Tables

Listings

1 Introduction

Internet has already made a significant impact in education, information technology and communication, business, government and social fields. The future aim of the internet is to connect physical and virtual objects by providing an infrastructure for immediate and easy access to information. The vision IoT is the next big step in the internet where the physical world, consisting of real devices and the virtual world of information interacts together. This interaction contributes to a common layer that enables the collection, analysis, storage and management of data and functions, providing high-level knowledge from aggregated and/or derived data.

Internet of Things, a novel paradigm was first introduced by Kevin Ashton [PI11] and is defined as: "all things are connected to the internet via sensing devices e.g. Radio Frequency Identification Device (RFID) to achieve intelligent identification and management". In IoT, "*Things*" [1] refers to any object. This includes everything from cell phones, coffee makers, washing machines, refrigerators, wearable devices and anything else you can think of which can eventually become part of the internet. IoT operates on the basis of Machine-to-Machine (M2M) interaction where no human intervention is required. At the same time, IoT involves non-connected entities which are empowered with RFID or barcodes, sensed through a device e.g., smartphones and eventually become part of the internet.

The speedy advancement of IoT and pervasive computing are contributing significantly in daily life which produces massive amount context related data, representing a continuous change of states taking place in the surroundings. The generated contextual data can further utilize to infer valuable insights e.g., weather and traffic patterns. However, context-aware services require a reliable computing paradigm to store, analyze, compute contextual data on large scale. Therefore, it is significantly important to discuss a paradigm which can absorb such massive amount of data generated by IoT, enabling connection of things in a wider sense. Hence, the notion of Cloud Computing (CC) is considered. It is not only a concept describing how we live but also, how we work every day. In order to provide more granularity of sensors and devices closer to the *Fog* or *Edge*

[1] The term Things and IoT devices are interchangeable in this research and represent physical devices, capable of generating data which are connected to some middleware.

discussed in Chapter 2, IoT will extend cloud, big data and social networks which will have more applications and scenarios that will accelerate entirely new business models and revenue opportunities.

1.1 Problem Definition and Motivation

Internet of Things is in a continuous expansion phase, from millions of devices to tens of billions in upcoming years which will have major impacts on infrastructure, business models, and industry standards throughout the entire IT ecosystem. It is expected that several diverse devices to invade by 2020 [AKA14]. Gartner predicts by 2020, the estimated growth i.e., 21 billion of devices connected to the internet, will exceed the number of people exist on earth [Gartner16]. The vision *"IoT"* is the future and this vision derives the new rule: *whatever can be connected will be connected*.

Several services and solutions are proposed to deal with context based discovery of growing number of heterogeneous devices, manufactured by various vendors, designed with particular specifications, protocols and architectures. However, current approaches entail discovery of devices and sensors manually which makes the life of the non-technical user more complicated. For instance, agriculture scientists often need to measure crops performance cultivated in the fields. They are interested to know what functions are available in a particular location in terms of Global Positioning System (GPS) coordinates, which sensor provides interesting information such as environment temperature, humidity level and so on.

With few hundreds of devices, discovering devices manually with the existing approaches works up to some extent. However, with the evolution of IoT era embedded with contextual information where the number of devices connected to the internet will be greater than the people, the proposed approaches discussed later, lacks in providing appropriate solutions to such non-technical users. Hence, a distinction between the user needs and the current available solutions.

1.2 Objective

In this thesis, we aimed to come up with a service that allows automatic discovery of various IoT devices based on a particular context[2]. The proposed concept will mainly

[2]The thesis focuses mainly on location feature rather than complete context computing paradigm. Therefore, the terms "context" and "location" are interchangeable.

focus on the discovery of available devices and sensors automatically, primarily based on location used as contextual data. However, additional information about devices and sensors are also considered which includes unique identifiers, manufacturer and current status. It should be noted that the proposed solution does not address all the requirements and issues of IoT and context-aware computing platform.

The proposed system can be used in various disciplines. However, a consideration of particular use case is done as discussed in Chapter 4. Figure 1.1 depicts a high-level scenario in which a user is passing by a particular building and interested to know available sensors on a mobile device that are installed inside the building. It is important to mentioned that the building is further classified into floors and rooms. Upon reaching within the proximity of the installed devices and sensors, the user is notified on his smartphone about the discovery of available devices. In addition, whenever the user passes from the same building, an automatic notification is generated on the mobile device which identifies the device and its details.

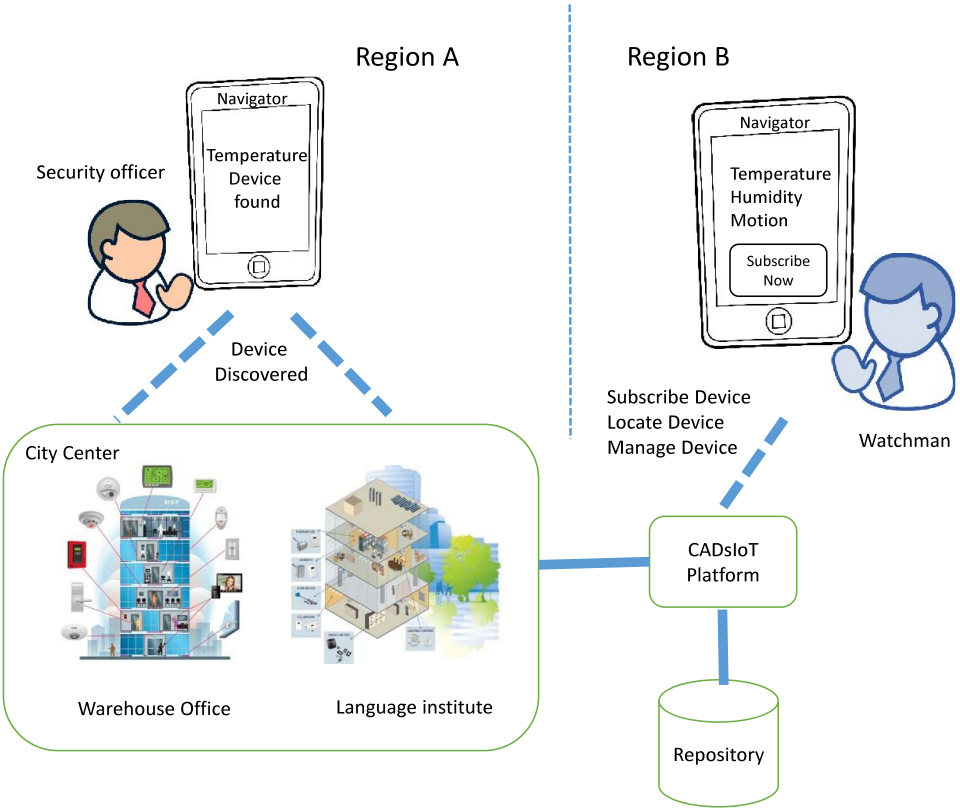


Figure 1.1: Concept Overview

The main research objective is outlined as follows:

> *To design and develop a context-aware IoT Discovery Service by analyzing state-of-art standards, protocols and technologies*
>
> Additionally, the following constitutes the entire objective.
>
> – Designing of device registry, capable of maintaining the device and sensor related information.
>
> – Consideration of motivation use case where the proposed concept can be applied.

1.3 Structure

The thesis is structured to provide a smooth flow of information, keeping state-of-art, proposed concepts and implementation of our solution in sync. The first three chapters give an understanding about research objective, problem statement, concepts, terminologies, existing solutions and their comparisons. Chapter four discusses about the requirements and the motivational use case of the proposed solution. Design aspects and proposed architectural details, followed by RESTful Web Services are outlined in chapter five. Lastly, chapter six explains the implementation and validation details of the proposed solution. Figure 1.2 summarizes the overview about each chapter.

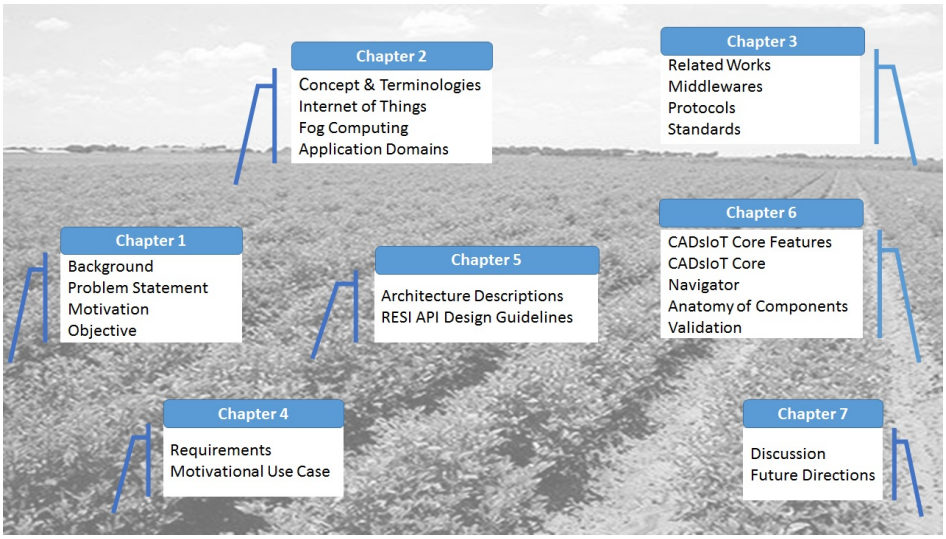


Figure 1.2: Research Structure

2 Foundation and Concepts

This chapter explains Internet of Things and Fog computing concepts in detail, followed by high level architectures and characterization of Fog Computing. Additionally, possible application scenarios utilizing Internet of Things and Fog Computing paradigm are also highlighted.

2.1 Internet of Things: Interconnecting Everything

Internet of Things, an evolving technology offers promising solutions which connect physical objects to the digital world. It is interconnected by a network of self-configuring nodes (things) in a dynamic and global infrastructure, enabling ubiquitous and pervasive computing scenarios through its most disruptive and bundled technologies. Before we go further, it is important to know, what these two terms means— Internet and Things.

The term "*Internet*" identifies the network related vision and "*Things*" are generic physical objects that become part of the internet. As discussed in Chapter 1, *Things* can be physical or virtual objects which can be sensed and connected with IoT networks. IoT was initially discussed by Kevin Aston in 1999, a founder of the original MIT Auto-ID center [SGP10]. Due to its frequent use, the whole term i.e., IoT became the one-liner word similar to Big Data, Cloud Computing. Semantically, IoT illustrates a *"world-wide network of interconnected objects uniquely addressable, based on standard communication protocols"* [AG-EPoSS08; AIA10]. Later on, researchers relates IoT with more sensors, actuators, GPS and mobile devices. However, a widely accepted definition of IoT in today's world can be viewed as:

> "*a dynamic global network infrastructure with self-configuring capabilities based on standard and interoperable communication protocols where physical and virtual 'Things' have identities, physical attributes, and virtual personalities, and use intelligent interfaces, and are seamlessly integrated into the information network.*" [KD08]

2.2 IoT Five Layered Architecture

Manufacturers are in the continuous development process to connect things with the internet since 1990 [JCIoT13]. At the same time, consumers are using connected devices e.g., thermostats, energy meters, lighting control systems, remote monitoring and irrigation systems typically using TCP/IP protocol stack. Due to the increase in the number of IoT devices, it is difficult to store and maintain a large amount of data generated [KSR12]. Therefore, a sophisticated architecture for IoT is required [AKA14]. Hence, IoT architecture is divided into five layers as depicted in figure 2.1.

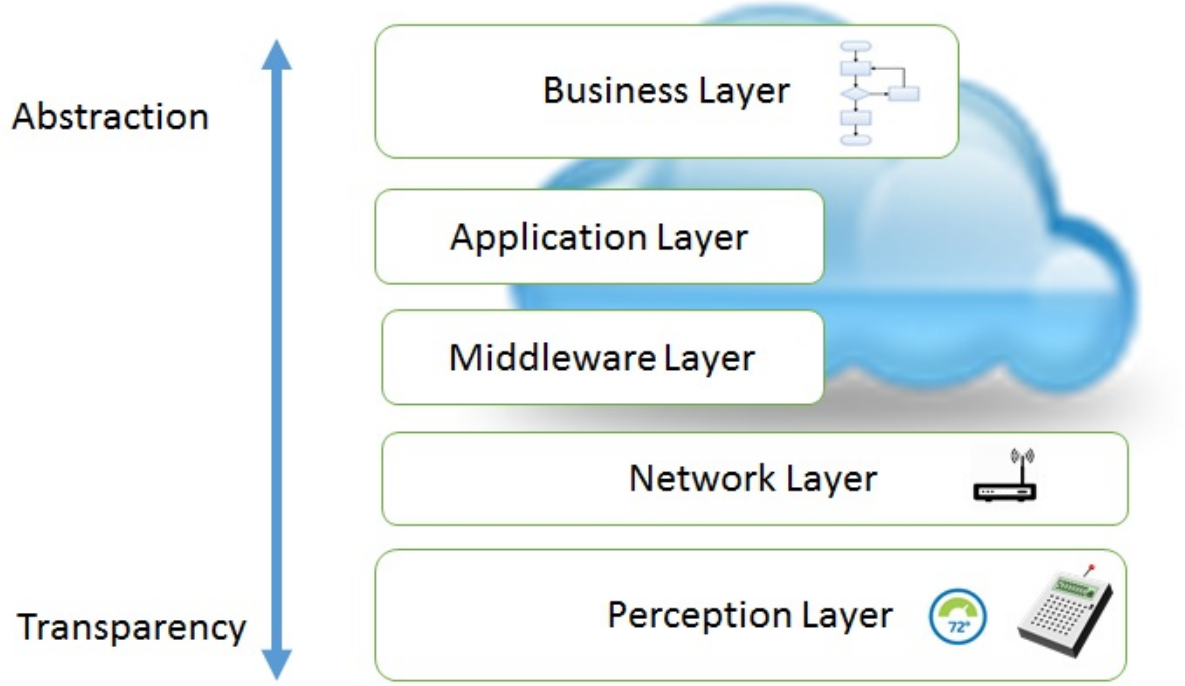


Figure 2.1: Internet of Things Layered Architecture [AKA14]

Perception Layer

The lowest layer in IoT architecture, also called Things or Device layer consists of physical objects and sensor devices such as RFID tags, barcode labels, GPS, cameras. The goal of this layer is to identify things/objects and collect data by the sensor devices. Typical examples of data include temperature, location, orientation, motion, vibration, acceleration, humidity, weather conditions etc. Finally, the collected information is transferred to the higher level called Network Layer.

Network Layer

This layer also called Transmission or Gateway layer. It behaves quite similar to Network and Transport layer in Open Systems Interconnection (OSI) model, as it is responsible for transferring information from the sensor devices to the internet using communication technologies such as 3G, 4G, UMTS, Wifi, Bluetooth, Infrared, ZigBee, etc. Gateways can also be included on this layer with one end connected to the sensor network and other to the internet. Thus, Network layer sends the information to middleware layer.

Middleware Layer

This layer is responsible for processing data from the Network layer where service management, storing data in appropriate data storage system tasks are performed. This layer is also responsible for automatic decision making based on the results. The generated output is then passed to the Application layer.

Application Layer

Application layer receives the processed information from the Middleware layer and performs final presentation of the information. It also provides global management of the application. According to the user requirements and the purpose of devices on the Perception layer, this layer delivers the data in various forms, enabling services for application domains e.g., smart home, smart cities, smart health.

Business Layer

Business layer is more about predicting and defining business strategies based on the services provided by Application layer. These services are modeled into meaningful services, turning information into knowledge and wisdom, through efficient means of usage. The output of this layer typically includes business models, graphs, flowcharts. Hence, Business layer provides numerous opportunities to service providers.

2.3 Fog Computing Platform: An Extension

Cloud Computing, an extension of distributed, parallel and grid computing, allow users to access on-demand services whenever required. This concept has evolved dramatically in recent years and according to National Institute of Standards and Technology (NIST), CC is defined as:

> "cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) that can be rapidly

> provisioned and released with minimal management effort or service provider interaction."[CC-NIST11]

Cloud computing offers a flexible pay-as-you-go model, helping IoT in providing virtual infrastructure for utility computing integrating applications, monitoring and storage devices, analytics tools, visualization platforms and client delivery, enabling users to access applications on-demand anytime and anywhere. The traditional cloud approach—process all data on cloud give rise to latency, bandwidth, mobility and geo-distribution issues which are essential requirement for IoT. Notably, the cloud platform serves as an abstraction layer between the low-end devices and the application services. This is because that the cloud platform does not leverage various IoT protocols and mainly communicates over Internet Protocol (IP). It is suitable to analyze and process data generated by sensors close to the vicinity where devices are located. Thus, a notion of Fog or Edge Computing is used.

What is Fog Computing

Fog Computing extends traditional Cloud paradigm close to the ground where "*Things*" are located. For instance, they are also called fog nodes, which can be deployed in a vehicle, in the forest, top of an electric pole etc. Any device can be fog node if it compasses computing, storage and networking capabilities. The interplay between Cloud and Fog enables new application services especially in the area of data analytics.

2.3.1 Characterization of Fog Computing

It is already understood that Fog Computing is an extension of Cloud Computing paradigm as it offers a virtualized platform. It possesses the main building blocks which are already present in the traditional model namely compute, storage and network services for IoT devices. Being located at the edge of the network is the key differentiator of Fog Computing. This distinguishes and gives rise to a number of notable characteristics [BFR12]. Some of them are listed below.

- **Edge location, positioning, and low latency** matters when data need to be analyzed in milliseconds closer to the edge. For instance, if power lines of a manufacturing plant shutdowns, the circumstances can be devastating. Fog supports applications which require rich services with low latency requirements e.g., video streaming, weather forecasts, gaming.
- **Geographical distribution** of the IoT devices varies on the type of deployment. As Fog services demand widely distributed deployments, it can be beneficial to provide streaming services in highly harsh environments e.g., railways, roadways, moving vehicles.

- **Mobility support** must be provided to Fog applications, enabling seamless communication between other mobile devices e.g., using LISP protocol[1], a technique that separates hosts identity from a location identity [BFR12].
- **Interoperation and federation** services must be available for Fog components to enable robust and seamless integration, allowing access to services across various domains.

2.3.2 Fog Computing Architecture

Due to a large amount of data generated which might be needed on real-time basis, a rich interplay between the edge (ground) and the core (cloud) of the network can be observed in several use cases. In order to enable context awareness and low latency, Fog nodes provide localization, while global centralization is provided by the (traditional) cloud. In the context of analytics and Big data, several applications require both Fog localization and Cloud centralization. A Fog computing architecture performs edge analytics on any object from the network center to the edge, since fog or edge analytics may perform analytics at devices closer to the edge of the network. Figure 2.2 depicts the Fog architecture as proposed by Cisco [JM16].

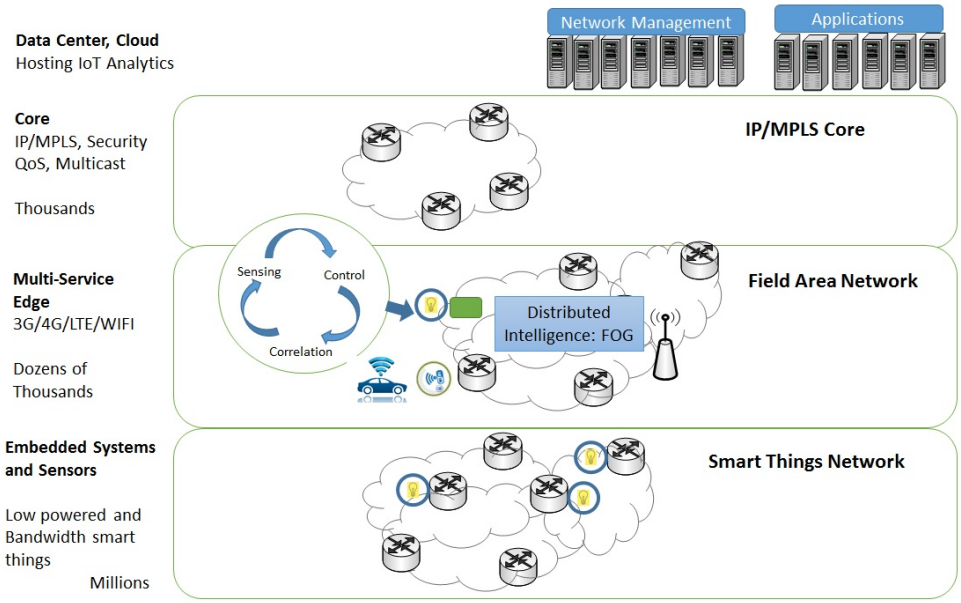


Figure 2.2: Fog Computing and Internet of Things Architecture Proposed By Cisco

[1] http://www.lispmob.org

Typically, M2M interaction is done on the first level of the Fog architecture. It collects, process data and allow actuators to perform their tasks upon receipt of control commands. Additionally, filter the data for local use and send the remaining to the next higher level.

The visualization and reporting Human-Machine-Interaction (HMI) and M2M are performed on second and third levels. It is important to consider the interaction times for real-time analytics (seconds to minutes) and for transactional analytics (even days). Hence, Fog must support various types of storage on the levels.

Figure 2.3 shows that the time scale becomes longer on the higher level with wider geographical coverage[BFR12] while the global coverage is provided by Cloud which behaves as a long-term storage repository.

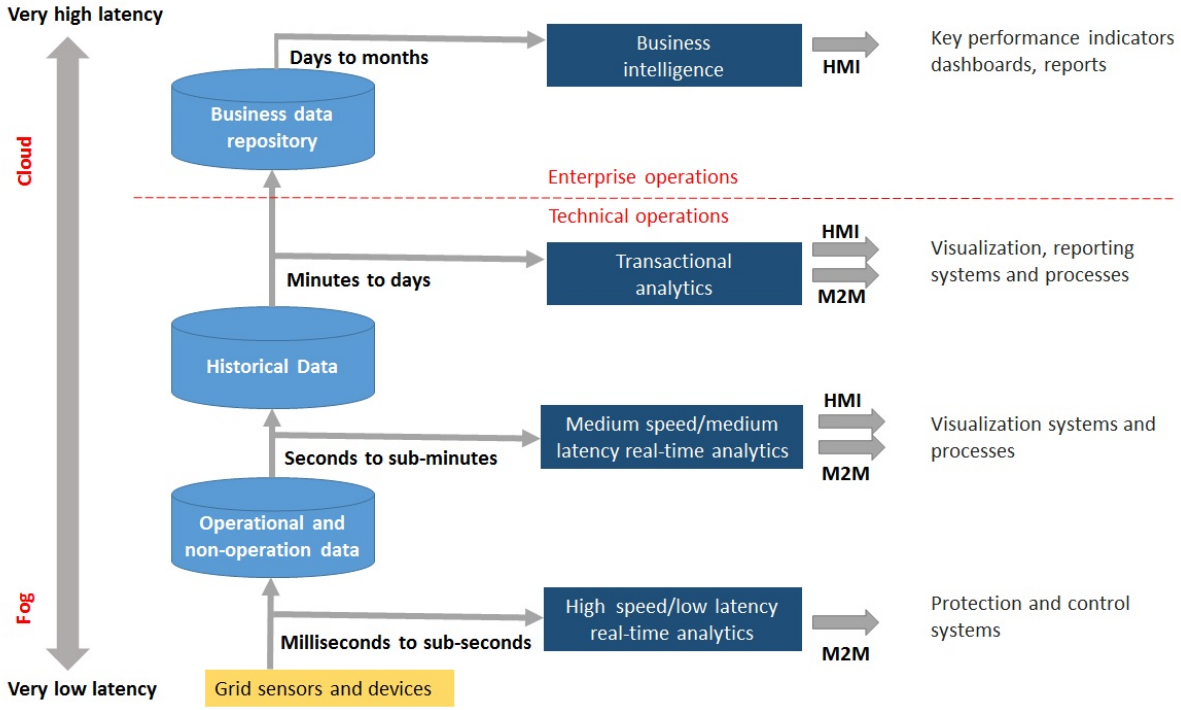


Figure 2.3: Analytics performed on Fog Computing [BFR12]

2.4 Application Domains: Fog and Cloud

Fog Computing provides location and context based services through Fog nodes. It is an ideal platform to provide safety, mobility, location awareness, low latency and heterogeneity. In this context, a thorough literature is reviewed in order to discuss the best use cases involving IoT, Fog and Cloud concepts [BFR12; LGZ15]. These use cases are outlined below.

Smart Traffic Lights and Connected Vehicles

Ambulances are used to transfer patients to the nearest health care centers and provide first aid services, if necessary. This transfer must be done as quickly as possible to avoid patient's critical conditions. Therefore, roads should have less traffic allowing ambulances to pass through. How about busy lanes surrounded by vehicles? One solution would be to change street lights automatically based on the video captured that senses the ambulance flashing lights, allowing other vehicles to free up the lane.

In context to road accidents and its prevention, smart traffic lights communicate with the local sensors installed along the roadways, detecting the presence of the pedestrians and bikers. Additionally, distance is measured between the pedestrian's location and approaching vehicles and a warning signal is sent to the vehicles. Fog nodes perform real-time analytics e.g., changing the timings of the cycles in response to traffic conditions. Finally, data is collected from Fog servers/nodes and sent to Cloud where long term analytics are performed.

Tourist Locations

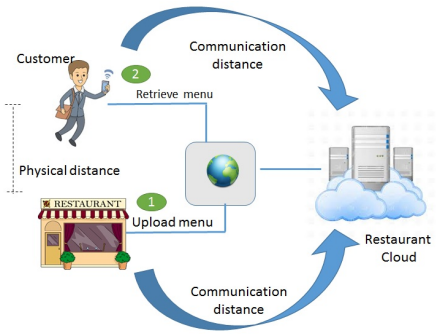
Tourists are interested to know about certain attractive locations. Therefore, information should be delivered at the right place and on the right time. A location-based tourist application enabled by Fog Computing can be beneficial for the tourist. Fog nodes can be deployed in a particular tourist location. For instance, at the entrance or at sightseeing location of a scenery park. A pre-cache information can be stored by the Fog nodes including maps and tourist guide while the other nodes can interact with sensor network for environment monitoring providing alerts to the tourist.

Restaurants

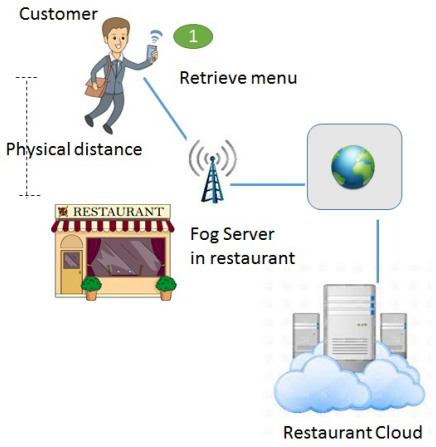
People are often interested to know about the kind of meals available at a particular restaurant. Also, when the user is inside the restaurant, the food servers (waiters) sometimes takes a time to explain what is currently available with respect to the time. Figure 2.4 depicts the scenario how Fog and Cloud environments behave when a localized request about meals is made. Figure 6.8a shows that a customer is near the restaurant and requests localized meal menu. However, the request takes time as it fetches the

response from the cloud which is location unaware of the user and presents a pool of information to the user.

In figure 6.8b, a Fog Computing environment is placed by the restaurant which allows users to access localized menu information stored on Fog servers.



(a) Cloud Computing



(b) Fog Computing

Figure 2.4: Fog and Cloud Computing Scenario

3 Related Work

This chapter delivers understanding about current work and associated technologies in IoT domain. It describes different middlewares, standards, and protocols currently used, focusing features related to the discovery of devices followed by a technology comparison.

3.1 Middlewares and Frameworks

Middlewares play an important role in the deployment of IoT infrastructure. They are often bundled with components such as different services and protocols. There is no doubt that IoT devices are heterogeneous in nature, providing different functionalities and capabilities. Therefore, an integration between heterogeneous devices and middlewares is required which enables seamless access to the device data, encapsulating low-level communication aspects. In this section, a discussion on selected solutions is carried out, highlighting the aspects of the discovery of IoT devices. At the same time, an analysis of the state-of-the-art protocols, promoting IoT together with middleware solutions is carried out.

3.1.1 Global Sensor Network - GSN

Global Sensor Network (GSN) [1] offers middleware platform services to ease flexible integration and network deployment [AKM06]. The project was initiated in 2005 with a concept that different sensors in a deployment require a software system which gathers data from sensors. Hence, GSN provides an abstraction layer to hide implementation and complicated structures of the sensors through a concept called virtual sensors.

Figure 3.1 describes that GSN is built on container-based architecture which provides hosting and managing support for virtual sensors at runtime. GSN architecture comprises of several layers. At the lower level, several virtual sensors are hosted in the

[1] https://github.com/LSIR/gsn/wiki/GSN-in-a-nutshell

GSN container which are managed by Virtual Sensor Manager (VSM), a component which is responsible for providing the sensor data and necessary underlying infrastructure through provided set of wrappers. The sensor discovery and live data feed are accomplished by these wrappers. The top three layers provide access to GSN container functions through web services. In contrast, our approach is based on location aware services which allow users to discover devices for a particular location on the fly.

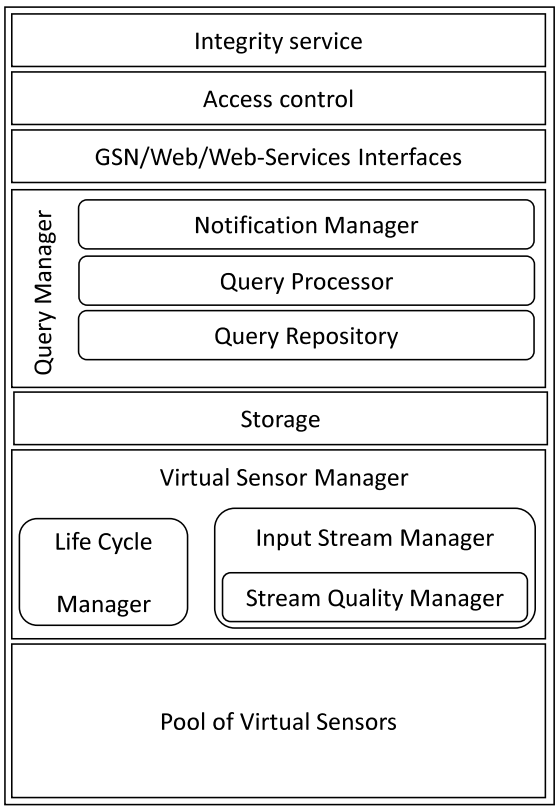


Figure 3.1: Global Sensor Network-GSN Architecture [AKM06]

3.1.2 OpenIoT

OpenIoT [2], a joint contribution by the European Union's Seventh Framework Programme is a open source middleware, based on Utility-based IoT model in a cloud platform,

[2]https://github.com/OpenIotOrg/openiot/wiki

allowing information about sensors, actuators and smart devices. OpenIoT leverages X-GSN as discussed in [CJA14], an extension of the GSN project in order to embed semantic information on virtual sensors by SSN ontology. X-GSN facilitates the device discovery and helps to interpret and understand the device data easily.

Figure 3.2 shows the architecture of the OpenIoT middleware. Sensor data management is provided using X-GSN technologies while the semantic data management is available through Linked Sensor Middleware (LSM) for storing and processing data streams.

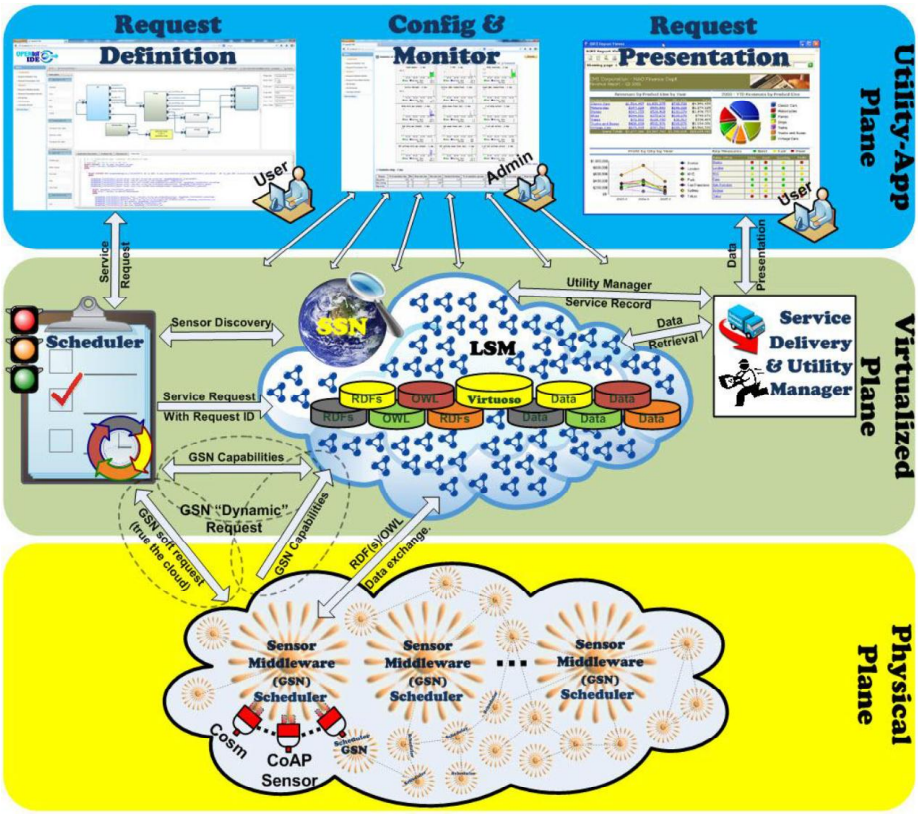


Figure 3.2: OpenIoT Architecture
[CJA14]

3.1.3 Xively

Xively [3] is a commercial IoT platform offering services for developing applications which require IoT devices to be connected to the Internet. The project aims to provide

[3] www.xively.com

flexible communication with each other through a common ground. It leverages industry standards communication methods namely Message Queue Telemetry Transport (MQTT), REST, HyperText Transport Protocol (HTTP) and sockets. Libraries are provided for various multiple platforms e.g., Arduino, Raspberry, C, Java, Android to communicate with the low-end devices. Xively offer Web-based interfaces through which several environments can be configured and managed.

Figure 3.3 shows high-level Xively architecture where devices including things, legacy applications are connected to Xively platform through its provided APIs. On the other hand, our solution also applies the same approach to connect devices. However, automatic context based discovery of devices is the key differentiator of our solution.

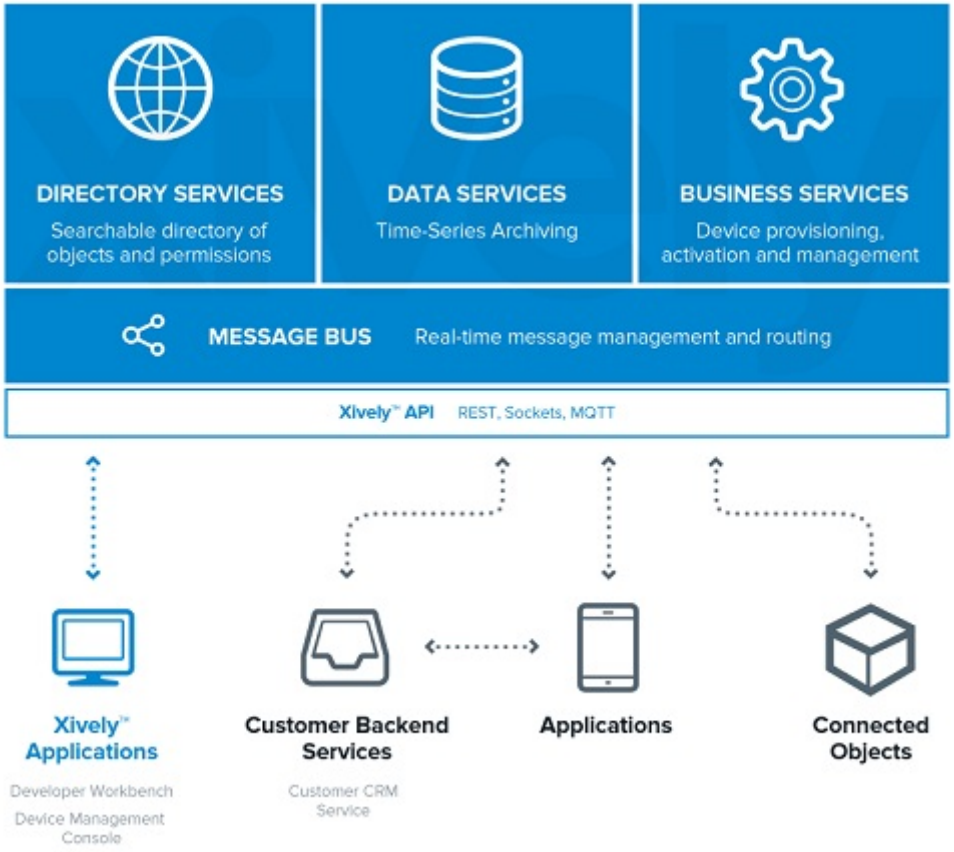


Figure 3.3: Xively Commercial Platform Architecture

3.1.4 Dynamix

Ambient Dynamix [4] is an open source context-based framework that runs as a background service on Android devices. Its lightweight pluggable context discovery mechanism helps in understanding the environment of the user. Dynamix is enriched with context discovery services and it is achieved through provided plugins which are automatically discovered. These plugins are installed on the user's device to interact with the physical world, enabling sensing and actuation tasks. A large collection of plugins are available and more can be build using open Software Development Kits (SDK).

In Figure 3.4, customized drivers helps to communicate with the sensors while pre-defined plugins stored in the plugin repository assist in understanding the contextual information of the user's environment.

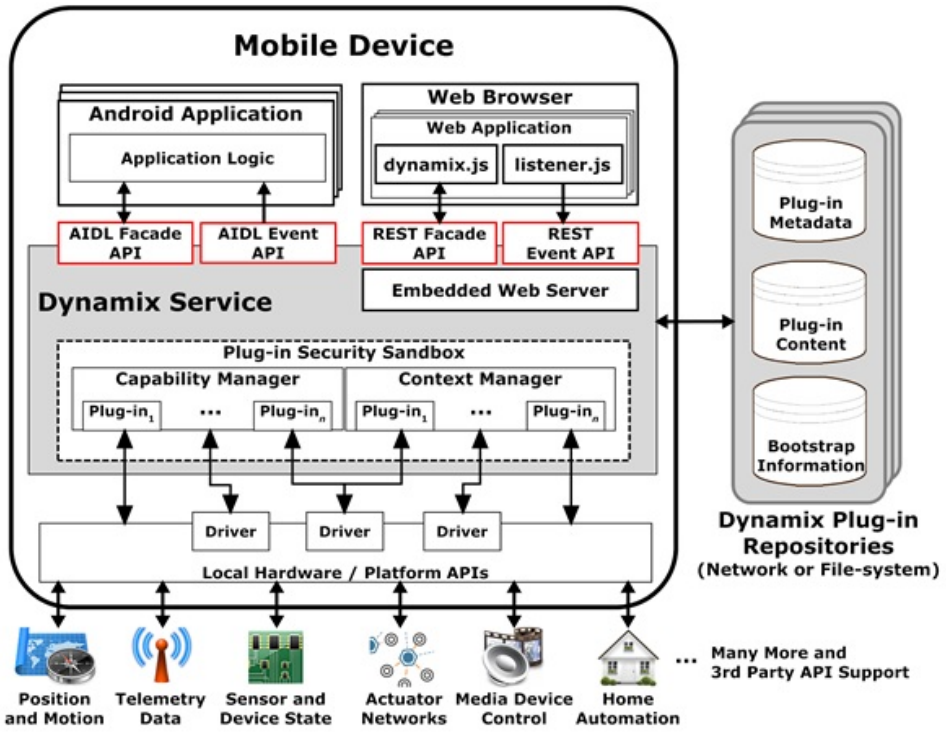


Figure 3.4: Dynamix Context based Architecture

[4] http://ambientdynamix.org

3.1.5 Eclipse IoT Frameworks and Services

Eclipse Foundation [5] provides set of services and frameworks for the development of IoT. Eclipse Kura, a Java/OSGi based framework that provides underlying support for hardware, management of network configurations, interaction between M2M/IoT platforms though its robust APIs which are not generally available in embedded agents. Kura provides advanced device and remote management features at the gateway level. Beside its basic gateway connectivity, network configuration and delivery services, its remote management capabilities help to monitor devices at the edge, providing configuration and deployment benefits. Additionally, Eclipse Kura also includes switching serial on or off, wifi management and remote data processing. Figure 3.5 illustrates Kura architecture.

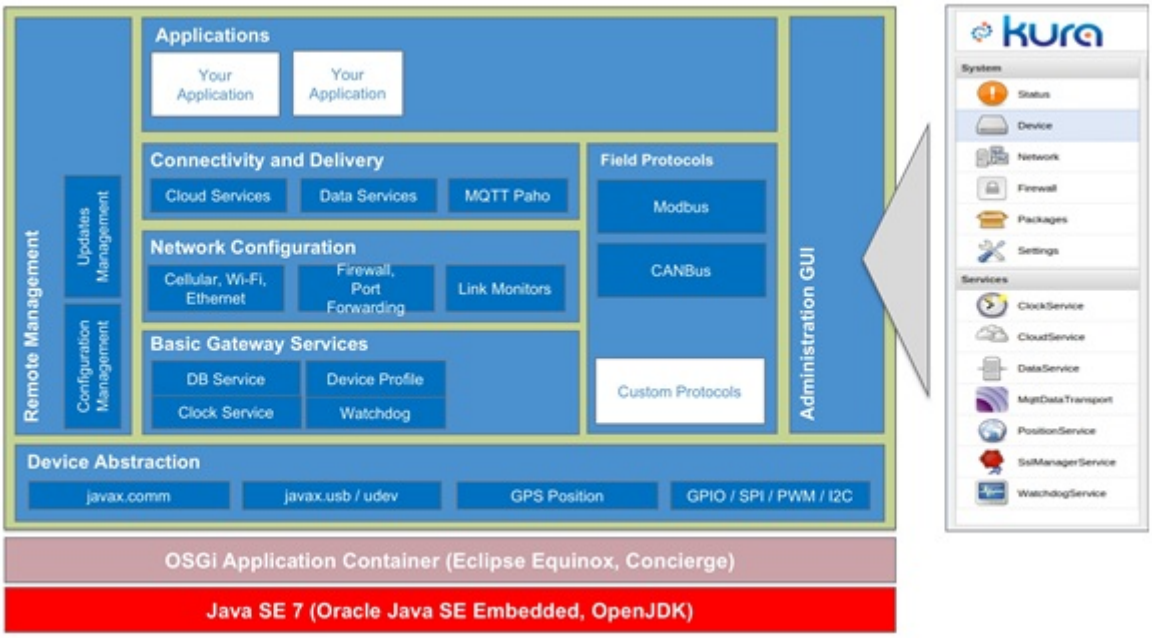


Figure 3.5: Eclipse Kura Architecture

Two of the popular projects, built on open standards are Mosquitto and Paho, offered by eclipse community. Eclipse Mosquitto, an open source message broker is pervasively used to transmit sensor data via a publish-subscribe protocol MQTT. Publishers usually send messages to a queue or a topic endpoint without knowing the destination of

[5] www.iot.eclipse.org

messages whereas, the subscribers receive the messages (published) by waiting for the information in which they are interested in (queue or topic). Eclipse Paho is the first open source MQTT client implementation. It's Java based version is used to connect several MQTT brokers and offers synchronous and asynchronous APIs to give full control to the developers in order to implement MQTT logic in their business applications.

3.2 Messaging Standards and Protocols

Communication in IoT networks is done through messaging using standards and protocols. Variety of applications leverage appropriate features offered by higher level protocols for IoT . For instance, SNMP and DDNS protocols can be used fo managing and configuring home devices. Therefore, a clear understanding is required to select available standards and protocols for desired solution. In this section, a discussion on mostly commonly used standards and protocols has been done e.g., MQTT, CoAp, HTTP and XMPP.

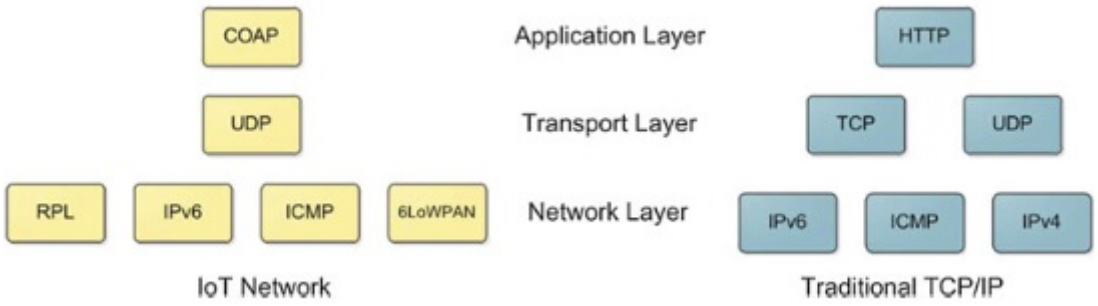


Figure 3.6: Common Protocols of IoT versus traditional TCP/IP [ISE15]

3.2.1 MQTT

Message Queue Telemetry Transport abbreviated as *MQTT*, is a lightweight messaging protocol was introduced by IBM in 1999 and became the OASIS standard in 2013 [KCVA15]. It is a publish/subscribe messaging protocol in contrast with request/response, in which client request the server for some data. However, publish/subscribe systems publishes the content to all its subscribers who are interested in the particular content. It is designed for low-end network devices, minimizing network bandwidth and ensuring message delivery.

Figure 3.7 describes the components of a publish/subscribe architecture which consists of publishers, mostly the low-end sensors/devices that connect to the message broker, sends data and sleep, a message broker which receives the data published by devices and subscribers who are listening on particular data. The notion of topics is used in the message broker as it behaves as the central data repository and classifies sensor data in topics and pass them to subscribers. TCP/IP Port 1883 is used for MQTT protocol communication.

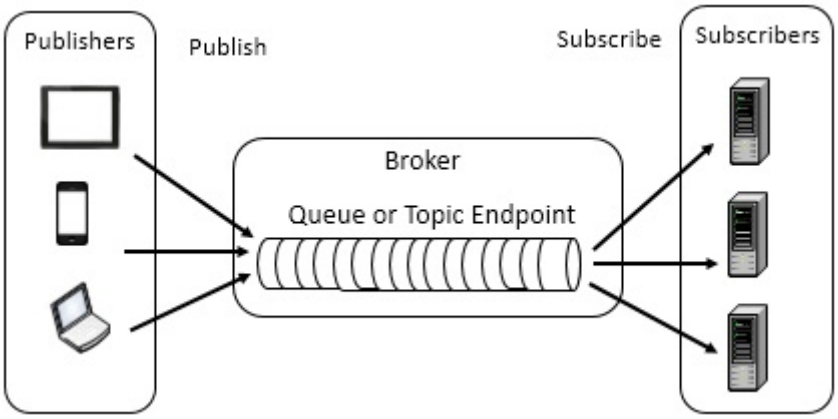


Figure 3.7: MQTT Architecture [ISE15]

3.2.2 CoAP

Constrained Application Protocol (CoAp) is a synchronous request/response application layer protocol. It was designed by Internet Engineering Task Force (IETF) and adheres to the concepts of the HTTP protocol. Similar to HTTP, CoAP is document transfer in nature, utilizing GET, POST, PUT and DELETE commands, making it interoperable for resource constrained devices. CoAp works on UDP based application protocol which removes TCP overhead and bandwidth requirements. The protocols aims to interact with M2M applications [CoAp16]. It also supports dynamic discovery of IoT devices. It is evident that clients (sensors) and server (e.g IoT platform) require native support for CoAp protocol. However, such support is not provided by most of the sensor manufacturers.

3.2.3 XMPP

Extensible Messaging and Presence Protocol (XMPP), a IETF standardized messaging protocol initially designed for instant messaging applications. The protocol is quite old

and has been used in various internet applications. Recently, it has again received an importance in IoT applications with its easy to use XML data format. XMPP provides publish/subscribe and request/response messaging systems and runs on TCP. Therefore, it is up to application developer which messaging system is applicable for desired application. The protocol also supports low latency small message formats which are well suited for near real-time applications [BTDA13].

3.2.4 RESTful Web Services over HTTP

RESTful services abbreviated as *Representational State Transfer* is an architectural style rather than a protocol. REST requires use of HTTP methods to interact with the message-oriented system using simplified synchronous request response model. REST supports Extensible Markup Language (XML) and Javascript Object Notation (JSON) as the commonly used types. Most IoT cloud platforms are highly dependent on REST due to its ease of use interactions, caching, content negotiation and authentication mechanisms [UZL11]. RESTful Web Services are efficient and fast and is being used very widely in the internet world. REST consists of the basic principles discussed by [FRT00], [PCL08].

- **Stateless:** RESTful Web Services needs to scale to meet high performance demands, decrease request response time and fulfill requests as quickly as possible. To cope such demands, intermediary load balancing, fail over and gateway servers are placed that transfer requests to one another and send the response back. REST requests are stateless, meaning that each request consists of self-contained and sufficient information so that intermediary servers can forward and route these requests without any state being saved locally in the entire request path. HTTP and body of the request contains parameters, context and data need to be fulfilled by appropriate servers.

- **Uniform Interface:** RESTful Web Services require the use of the HTTP protocol as the de-facto standard. The protocol provides generic interfaces as described in RFC 2616 [6]. The REST design guidelines recommend the explicit use of HTTP methods e.g., GET, POST, PUT and DELETE. Create, retrieve, update and delete (CRUD) operations are mapped to HTTP generic methods, to manipulate resources in a uniform manner. Each method has its own behavior, standards and status codes. POST transfer a new state onto resource. GET retrieve the current state of the resource. PUT is used to change or update the state of the resource. DELETE is used to delete or remove a resource.

- **Representations:** Representations in REST refers the format and structure of messages exchanged in request/response fashion. These representations can be in variety of formats e.g., JSON, XML, PDF. Typically, clients are not aware of the internal format

[6] https://www.ietf.org/rfc/rfc2616.txt

of a resource available on the server. Therefore, clients negotiate the server to provide an appropriate format of a response at the time of the request is made. The agreement on the format of a resource between the server and the client is referred as *Content Negotiation*. Usually, Content-Type and Accept headers are used. Content-Type header provides the information about the format e.g., application/json being used in HTTP message whereas the Accept header is used by clients, which tells the server to send a response back in a format that client expects. Upon unavailability of the requested format at the server, a response code of 406 (Not Acceptable) is sent to the client.

- **Resource Addressability:** RESTful Web Services exposes resources (data and functionality) via Universal Resource Identifier (URI). Each resource can be accessed using directory structured like URI that are unique and exactly one for each resource. The hierarchical structure of URI helps to understand specific resources and can be accessed anytime even if implementation details changes, allowing resources to be straightforward, predictable and easily understandable.

3.3 Technology Comparisons

Nearly, all discussed middlewares utilize some sort of agents, gateways, wrappers and standard protocols to enable device connection. Xively supports RESTful services, sockets and MQTT protocol while GSN uses customized wrappers along with HTTP. On the other hand, OpenIoT and Dynamix leverage semantic annotations through X-GSN, customized drivers and plugins respectively. With respect to communication techniques, OpenIoT and Xively employs client/server, GSN works on peer-to-peer and Dynamix uses plug and play communication models. All of the surveyed proposals provides device discovery features. Table 3.1 illustrates the comparisons among different IoT middleware technologies.

Table 3.2 depicts the most common protocols used in IoT environments and their comparisons. It can be obverse that CoAp and REST/HTTP possesses request/response approach. However, CoAp is the only protocol that works on UDP which makes it lightweight and reduces communication overhead. REST/HTTP can suitable if constrained communication and power consumption are not considered, allowing interaction with the internet using HTTP protocol. In contrast, MQTT is the most efficient protocol if power is considered. The importance of the MQTT can be determined that Facebook messenger and other known players use MQTT which proves that MQTT is a reliable publish/subscribe protocol especially when a huge amount of information is to be updated.

Middleware	Communiation Model	Discovery	Device Connection
GSN	Peer-to-peer	Yes	Wrappers, HTTP and UDP generic
OpenIoT	Client/Server	Yes	X-GSN
Xively	Client/Server	Yes	RESTful API, Sockets, Websockets, MQTT
Dynamix	Plug and Play	Yes Context discovery	Customized plugins

Table 3.1: Internet of Things Middlewares and Frameworks Comparison

Protocol	Transport	Messaging	Architecture
MQTT	TCP	Publish/Subscribe	Tree based
CoAp	UDP	Request/Response	Tree based
REST	HTTP	Request/Response	Client Server
XMPP	TCP	Publish/Subscribe Request/Response	Client Server

Table 3.2: Protocols and Standards Comparison

4 Requirements and Use Case

This chapter highlights the concept of Context-Aware Discovery Service for Internet of Things (CADsIoT) by defining context and context-aware system. Furthermore, functional and non-functional requirements are identified. Finally, a motivational end-to-end use case related to logistics is discussed, showing how the automatic discovery of devices based on context can help in reducing process cycles and enable better decisions.

4.1 Overview

The evolution of IoT has made a significant impact on infrastructures, business models, and industry standards. Current trends shows that devices and sensors deployments has been increased at large extend and predicts significant growth in coming years [SGP10]. Dealing with billions of devices which will be connected with network of networks, distributed across everywhere, data management and availability of such heterogeneous devices requires a sophisticated solution that can simplify not only managing devices but also enables efficient discovery of the devices on the fly. Attaching context-aware computing techniques can significantly provide flexibility in device management, discovery and reading valuable data from sensors.

Although, successful prototypes and solutions are implemented by researchers and engineers using context-aware computing techniques in which data sources are fairly static and remained unchanged. However, such developed solutions are not suitable in collecting information from billions of devices, connected to the internet. Hence, context-aware solutions will play an important role in discovering devices, their management and data availability. It is important to describe context-aware terminology used in this thesis. Several definitions have been proposed for context by Dey [DAB01], Hull [HRP97], Brown [BP95] and Ward [WAH97]. However, these definitions were unable to deal with identification of the context at a large scale and hence, the following definition for context is defined:

> "*Context is any information that can be used to characterize 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.*"[AGD99]

Similarly, the understanding about context awareness was too specific and do no comply on a broader level. Hence, Perera [PZA14] and Abowd [AGD99] defined context awareness as:

> "*A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task.*"[PZA14]

In this thesis, the above discussed definitions are considered where the location is used as a primary category of context and do not focus on entire context-aware computing platform.

The CADsIoT is a concept that emphasizes to deal with large amount of devices and sensors installed in IoT environments, streamlining the process of registration, management and dynamic discovery of devices based on contextual information [1]. The solution allows registration of devices and sensors, attaching them to particular context and leverage subscription feature to enable dynamic discovery based on attached context. Additionally, notifications are triggered when new devices are discovered. To understand the concept better, an example scenario is discussed in section 4.3.

4.2 Requirements

This section details the requirements of the proposed solution. These requirements are carried out by considering the aspects of device discovery features, together with contextual information. Solution requirements are categorized as functional and non-functional requirements. Functional requirements often called capabilities, are the ones which must be met by the proposed system. Non-functional requirements are often termed as constraints or qualities on the system functionality e.g., scalability, heterogeneity, context, security and privacy.

Requirements for collecting data from physical sensors e.g., temperature, motion, smoke detectors are out of the scope of this thesis. However, dealing with the sensor is discussed in several literatures. Some of them has been identified in this section as below.

Bhatt in his literature [BAJ16] presented a solution for connecting home appliances with IoT, enabling monitoring of devices deployed in a home environment. Bhatt showed

[1] The term contextual information and context refers to the current location of devices, sensors and users, collected by Global Positioning System

designing and implementation process in a home automation scenario, utilizing MQTT, middleware technologies including techniques to read sensor data.

Silvia [SGA15a] investigates the use of IoT in health discipline with wireless sensor networks for Ambient Assistant Living (AAL) environments. The literature proposed an implementation on how the data e.g., temperature and humidity can be collected from devices deployed in the living environments. This data helps in monitoring indicators for patients having respiratory problems.

Thakre in [TSJ14] presented a prototype of alert generation system based on the temperature sensors, transmitting data at regular intervals. The system allows monitoring of abnormalities found in the equipments and take necessary actions based on the timely notification received by the alert system. Hence, techniques on how to read sensor data along with middleware technologies are explained.

4.2.1 Functional Requirements

The proposed solution aims to provide a service which enables users to discover devices and sensors deployed in IoT environments. The discovery is done based on the context, allowing users to automatically find devices with respect to their current location. In order to describe the functional requirements, use case diagram in figure 4.1 is depicted which shows major functionalities covered by the system. These functionalities can be provided by the proposed solution, which includes registration, context attachment, subscribe and discover devices. The functionalities are further discussed in this section.

1. **Registration:** Registering IoT assets e.g., devices, sensors, context in a device registry is one of the fundamental requirement in IoT based solutions. Interaction with these assets can be done via simple protocols such as HTTP. Operations such as addition, modification, retrieval and deletion of assets can be performed. Moreover, meta-data for each type of asset can be defined such as device identifier, description, creation and modification date, current status and context of the device.

2. **Attach Context:** Context plays integral part in the disruptive change in IoT environments. As discussed earlier in section 4.1, the thesis uses location as a primary category of context and do not focuses on the entire context-aware computing platform. Context defines the current location of devices and sensors, expressed in GPS. Users and applications can attach context with devices based on the fact that appropriate sensors are attached with devices available in the same vicinity. For instance, sensors are deployed in a university building on different floors and these sensors are connected with atleast one device via internet. Whenever sensor

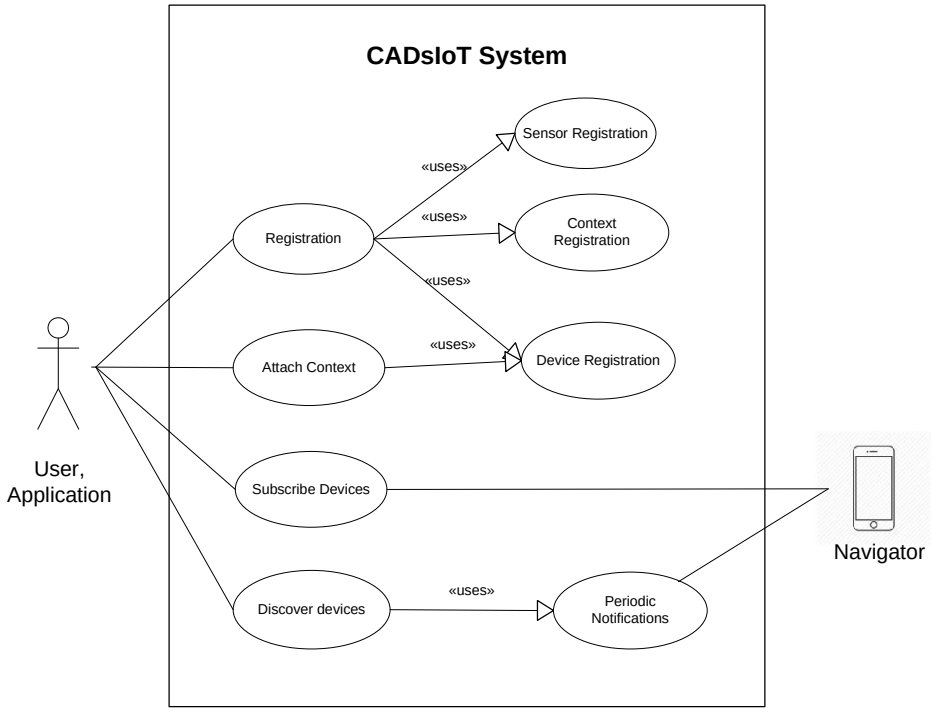


Figure 4.1: Solution Functionalities

location is requested, the device context is fetched and associated sensors provide their location information where they are placed inside the building.

3. **Subscribe Devices:** Use case deals with the subscription of devices and sensors. A subscription is a collection of devices and sensors in which user is interested to allow periodic monitoring of devices. Normally, users who subscribe to devices are called subscribers. Subscriptions are applied on sensors directly. However, its parent device is also subscribed based on the inheritance rule. These subscriptions are used by clients to facilitate automatic discovery of devices and sensors based on context.

4. **Discover Devices:** Discovering heterogeneous devices is a challenging task. CADsIoT locates nearby devices in a region based on user current location. This use case allows periodic notifications which are provided by the clients. Assume, a security guard is interested to know about the indoor temperature of the building. Using the client, firstly he subscribed to the temperature device and then reaches the region where temperature device is located. The region is the spherical distance from the location of the device and the current location of the user. Hence, as soon security guard enters the region lets say at a distance of 500m, real-time notification is triggered on the client application that a new device is discovered along with its context and the discovered device is displaced on the map.

4.2.2 Non-Functional Requirements

1.Heterogeneity: The amount of devices connecting to the internet and contributing to IoT environments is increasing. These devices and sensors are designed by different vendors with proprietary technologies and standards. According to the European Union, IoT solutions should provide context-aware services by 2020 [VFG11]. Hence, CADsIoT enables a context-aware discovery service in which newly added devices are located based on the subscribed context. An ideal device discovery service should be able to discover devices according to their capabilities e.g., type of sensors, services offered.

2. **Scalability and Extensibility:** It is evident that more and more devices and sensors will be used in the coming years by different industries for monitoring and tracking purposes. This requirement fosters more sophisticated solutions utilizing the IoT. New and useful functionalities will be added without altering components. CADsIoT solution is built on industry standards and allows developers to scale and extend to cater future needs.

3. **Security and Privacy:** As Internet of Things connects several devices together, this gives a way for potential loopholes and entry points from where a malicious attack can be made. The devices in IoT are supposed to obtain and share a huge amount of sensitive data and this is where the main security risk arises. Most of the IoT devices uses the normal public internet, we can visualize the risk of sensitive data flowing around and doubling the risk for consumers and providers. A recent study was conducted [SMD14] and the results suggested that almost 70 percent of IoT devices contains security vulnerabilities in the form of password security, encryption and authentication. Hence, designing efficient security and privacy strategies for discovering devices accordingly is challenging.

4. **Context:** Context information is highly important to understand IoT environments. It characterizes the situation of an entity e.g., person, place or object relevant to the interaction [BSM11]. The frequency of changing the physical positions and the contextual information are related to each other. An ideal approach should be taken to collect data from sensors for understanding the environment better. Hence, CADsIoT leverages location e.g., GPS as a primary category of contextual information in order to discover nearby devices.

5. **Usability:** The usability of CADsIoT system allow users to subscribe devices, display attached sensors and discovery of devices without any programming efforts. The user enters only the meta data required for the registration and enables subscription to discover devices dynamically. However, the service also provides flexible REST APIs, enabling developers and communities with the rich set of functionalities offered by CADsIoT solution.

4.3 Use Case

Internet of Things promises its benefits for businesses, enterprises, and consumers. While some businesses are still considering to employ IoT solutions, transportation and logistic industry is way ahead than others and actually the real beneficiary of IoT [DHL-Cisco15]. Logistics providers deal with millions of shipments moved from one place to another and share various real-time information provided by the vast network of devices and sensors, spread across the geographical regions, enabling higher levels of operational efficiency and automated services for their customers. Due to the economy of devices and sensors, logistic providers strive to leverage full IoT capabilities to provide value-added services to end users, not only to earn business benefits but also enables IoT potential to reshape logistic industry over the next decade.

To better understand CADsIoT solution and its features, an example scenario is taken from logistic industry due to its emerging information networks, deployed across several geographical regions. It is the requirement of IoT applications to dynamically integrate the network of "*Things*" into emerging information networks. Ideally, IoT application should provide feasible and scalable architecture, powered by enabling technologies such as cloud computing. Catering such integration in which "*Things*" are dynamically discovered and configured is a challenging task. In this thesis, aspects of device discovery based on contextual information are taken into consideration.

4.3.1 Background

With the evolution of e-commerce, parcel volumes are increasing their popularity and thus, are in high demands. This trend envisions that traditional letter volume will be minimized and more parcels and packages will be moving around. As a result, companies are planning to provide efficient services to cater future customer requirements. With this vision in mind, Company A, a global logistic provider headquartered in Germany with offices worldwide, is interested to improve its business process and to streamline inefficiency encountered during shipment to end users. The company follows a regular process which is partially manual, consumes more time and do not provide context-based information to facilitate better decisions and efficient package delivery.

Imagine, region A comprises of several recipients and five packages should be delivered within two hours. Although, the recipients belongs to the same region A, however, they are distant to each other e.g., 200m, 300m etc. The delivery centers [2] are equipped with smart postal boxes which include sensors giving information about the status of the box, the current temperature inside the box (in the case of environment friendly items).

Figure 4.2 shows company A current process in which two simple cases are considered.

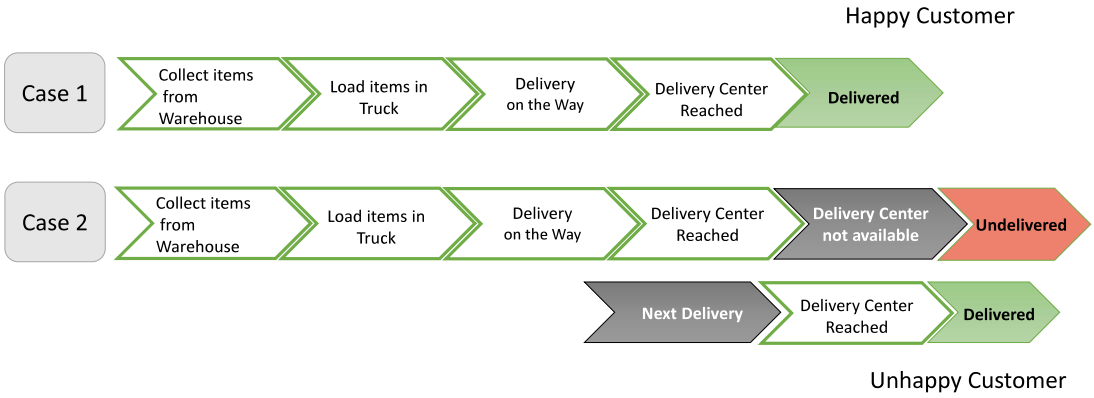


Figure 4.2: Company A Current Process

[2] the term delivery centers, recipients, and receivers mean end users who have ordered items and have smart postal boxes installed

Case 1 - Happy Customer

This case includes the normal process of company A in which items are delivered smoothly and the customer is also satisfied. The process is detailed as follows.

Step 1 - Collect item from Warehouse: In this step, all ordered items from different suppliers are collected and sorted in the warehouse. These items should be delivered in a single day according to the coverage region. It is assumed that each truck is responsible for delivering items in defined regions as per business policy.

Step 2 - Load items in the Truck: Items are loaded into corresponding boxes, available in the truck, allowing the delivery person to locate items as quick as possible at the time of delivery.

Step 3 - Delivery on the way: Now, the truck has left the warehouse and out for delivery in region A, Region B and so on. At this time, the delivery person carries a handheld device which contains information abut the delivery centers and respective recipients.

Step 4 - Reach Delivery Center: Shipment is ready to be delivered to the recipient. However, no information about the availability of the delivery center/recipient is communicated. Delivery person should go and check physically the availability of smart postal box and decide that the items should be delivered or not.

Step 5 - Delivered: Based on the information collected in the previous step, requested items are delivered as the smart post box is available. Once delivered, relevant status details are entered in the handheld to keep track of the shipment and then upload the information to centralized shipment tracking system of company A.

Case 2 - Unhappy Customer

In contrast to case 1, this case is slightly different where shipments need to be delivered at two delivery centers which are available in region A. At the end of the process, one of the delivery center receives requested items. Steps 1 to 4 are executed as described in case 1 in section 4.3.1. However, remaining steps are detailed as follows:

Step 5 - Delivery center unavailable: In this step, the delivery person came to know that the smart postal box is not available and its inactive at delivery center A. This activity urge the delivery person to go physically and check if the smart postal box is available. Additionally, the availability of the recipient is also ensured as the ordered items are sensitive. Hence, the items should not be delivered.

Step 6 - Next Delivery: This step plans the item to be delivered in region A. The truck took its way towards the next region and finally arrived in delivery center B.

Step 7 - Delivered: The requested items are now delivered as the smart postal box and recipient are available. However, the recipient claims that the items are not delivered on the time promised by company A. Therefore, the customer is unhappy and not satisfied with the service.

Business challenges posed by current process

Major problems are highlighted during the current shipment process followed by company A.

- No real-time contextual information to decide, items should be delivered or not
- Physical effort to examine availability of smart postal boxes
- Increase maintenance cost of the truck such as fuel, time etc
- Decrease in daily delivery targets
- Items not delivered to a customer in a timely manner

4.3.2 Solving the Business Problem

Future computation, storage, and communications services will be ubiquitous and highly distributed. Things such as devices, sensors, machines, smart objects when integrated together, will create a decentralized and loosely coupled architecture, connected by dynamic networks. Notably, the data streams coming from these devices will contribute to the Big data paradigm. Prior to dealing with data, *Things* should be dynamically discovered and configured which provide comfort in device discovery and data collection challenges. Therefore, the thesis aims to solve the problem of discovering devices while leveraging contextual information.

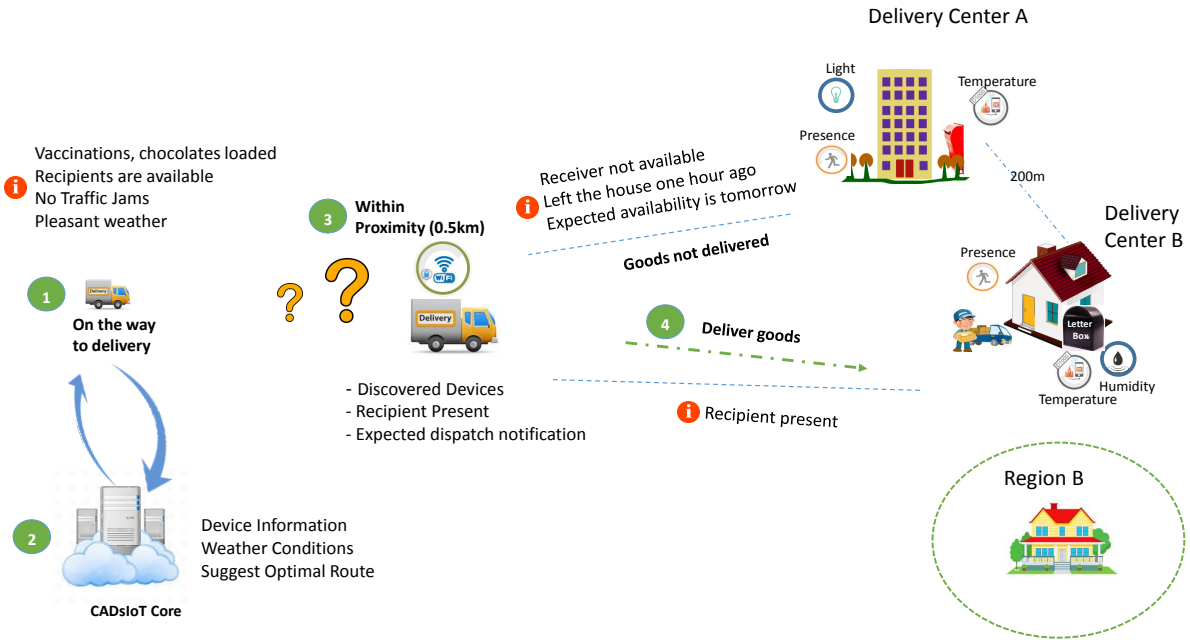


Figure 4.3: CADsIoT proposed solution as per given scenario

Figure 4.3 illustrates the end-to-end solution of the discussed scenario in section 4.3.1. As Company A needs to streamline inefficiencies in the process, case 2 (Unhappy Customer) is considered as most of the dependencies lies in case 2.

1 - At the Warehouse

As soon as the ordered items details are conveyed to the delivery person, relevant device information, placed at delivery centers are also fetched which are called subscriptions. These subscriptions include information about available devices and sensors e.g., device status, availability, location etc. placed at recipient's shipping location.

2 - Out for delivery

Besides delivery centers and recipients information, the delivery person also has the knowledge about smart postal boxes placed at delivery centers. Now the truck is on the way for daily deliveries, context based environmental information such as weather conditions are collected. These information suggests truck to take best route in order to avoid traffic jams and can be pushed to delivery person mobile device.

3 - Within Proximity

This step is of utmost importance as it enables the delivery person for efficient decision making. Context-based information is collected to decide whether the delivery should be done or not. The decision to deliver items are based on contextual information such as delivery person current location, presence of recipient, smart postal box status. The truck reaches the proximity of the delivery centers A and B as they are the recipients of the shipments. The following two situations arise which are also depicted in figure 4.4.

Delivery Center A

As soon as the truck reaches the proximity of delivery center A in region A, in our case 0.5km distant from delivery center A, the solution notifies the delivery person that he has arrived in the region A based on the subscription done in step 1. Also, nearby devices are discovered and displayed on the map. At this time, client can make a connection with the delivery center A to read smart postal box (device) availability. If the smart postal box device is not available, delivery person skips data collection e.g., recipients availability directly from the current location (delivery center A). Since the smart postal box is not active and the recipient is not present, the delivery cannot be done to the delivery center A and hence, delivery center B is the next delivery point.

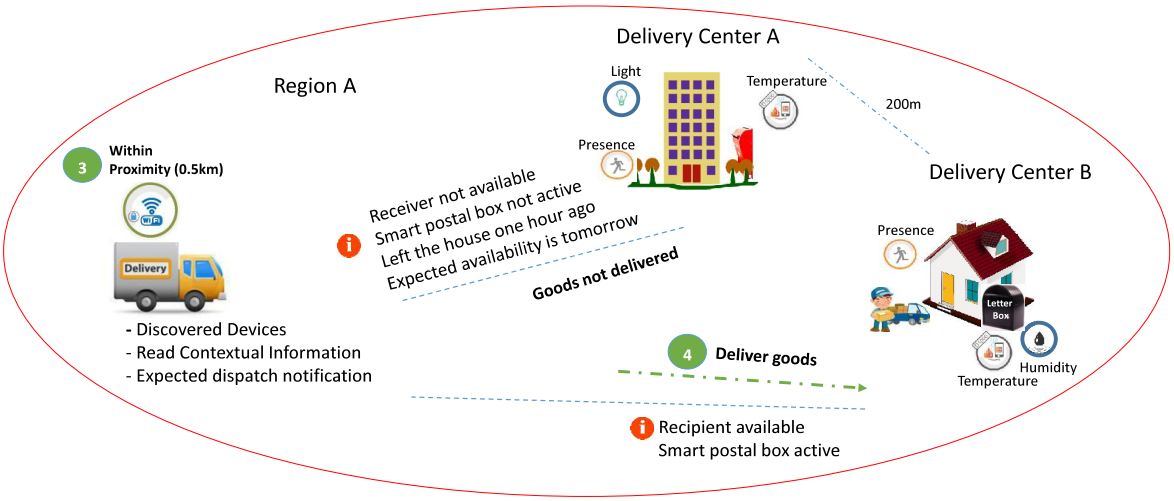


Figure 4.4: Truck within Region A Proximity

Delivery Center B

Now, the next delivery of items is targeted to delivery center B. In fact, delivery center B is located in region A at a distant of 200m from delivery center A. Therefore, the devices are already discovered while the truck enters in a region A. *Navigator* only makes a secure connection with delivery center B, to read smart post box (device) availability. Based on the contextual information read from the delivery center B, the smart postal box is active and the recipient is also present. Hence, the delivery can be made to delivery center B.

4.3.3 Benefits

CADsIoT helps company A not only decrease inefficiencies but also reduce unnecessary steps in the shipment process by leveraging context-based information for efficient decision making.

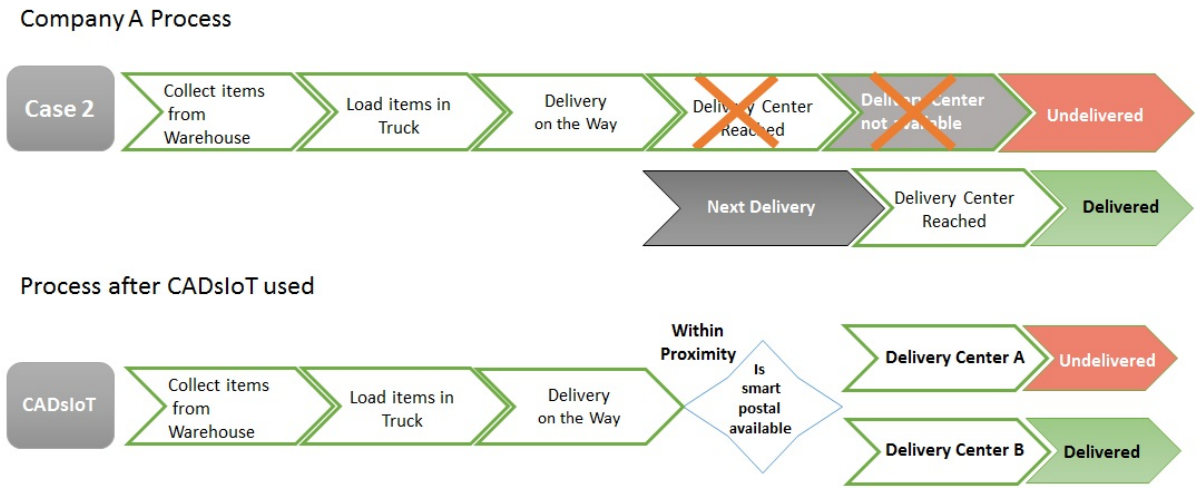


Figure 4.5: Company A Process Comparison

Figure 4.5 depicts the comparison between the previous process and the process when CADsIoT is employed. As shown, once the truck reaches the delivery center's region, the solution discovers smart postal boxes while remaining within the proximity and query them for their availability. In case, the devices are not available, the delivery is scheduled to next recipient. With the newly employed process supported by contextual information, Company A can now make better decisions and meet their defined targets. Additionally, some of the benefits are also listed below:

- **Better decisions:** Delivery person can now plan to dispatch goods to other recipients on time.
- **Saves time:** as the delivery person only concentrates where the smart postal boxes are available.
- **Save Energy and heavy luggage:** With the pre-informed information about the receiver, the delivery person does not have to carry the goods and walk.
- **Increase in daily delivery targets:** More packages can be delivered to receivers.
- **Saves Cost:** This will reduce costs by optimizing the truck's route and enables for more efficient delivery.

5 CADsIoT: Proposed Solution

This chapter portrays the design aspects of CADsIoT solution. It describes the overall architecture, comprising of *CADsIoT Core* and *Navigator* and the services available at different layers. Finally, the specification of *CADsIoT Core* resources, representations, URI patterns and interactions are also discussed.

5.1 Architecture

CADsIoT provides a discovery service for IoT environments. This section details the architectural view of CADsIoT. As discussed earlier, this thesis focuses on the discovery of IoT devices with the proposed architecture.

CADsIoT comprises of two main components: a backend service called *CADsIoT Core* and a smartphone client *Navigator*. The backend service acts as the centralized repository for devices, sensors, contextual information and other associated assets. *Navigator* communicates with the backend service via RESTful APIs. Both these components are discussed in this section.

5.1.1 CADsIoT Core

CADsIoT Core has a muti-layered architecture, built on RESTful Web Services. It includes loose coupling principle which provides benefits in the same way as in most dynamic web applications. With its loosely coupled architecture, business rules defined via business logic are separated from the type of database being used. The REST based implementation provides operations to manage IoT assets e.g., devices, sensors, context and subscriptions. *CADsIoT Core* facilitates to create, modify and manage assets via simple HTTP methods.

Figure 5.1 depicts five layered architecture, describing various components and services at each layer supported by *CADsIoT Core*.

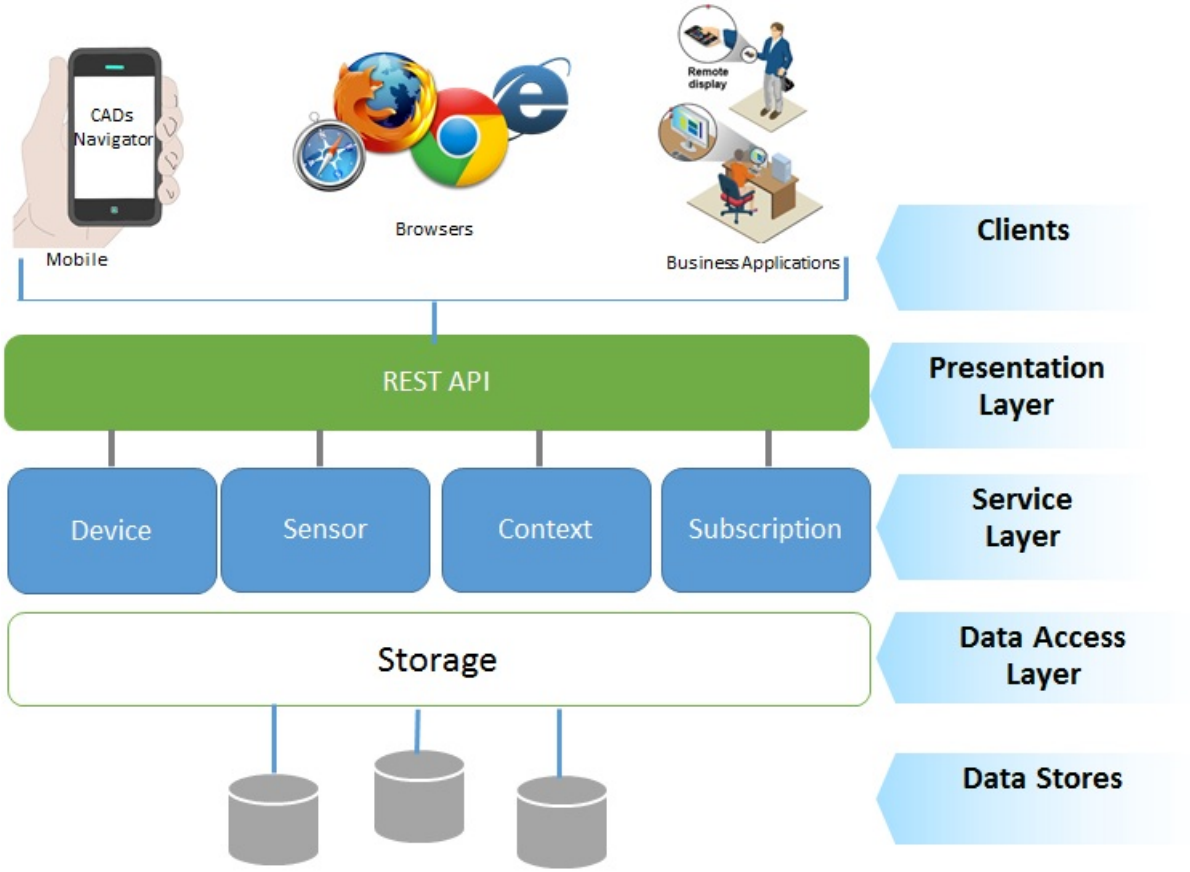


Figure 5.1: CADsIoT Core Architecture

Presentation Layer

CADsIoT Core exposes REST-based interfaces that allow *Navigator* and external REST clients to access corresponding assets. Device, sensor, context and subscription information can be managed via HTTP requests. The self-contained requests uses HTTP methods e.g., POST, GET, PUT and DELETE which are mapped to the corresponding operations of the resources and then requests are delegated to the Service Layer.

Service Layer

The service layer defines business rules for each type of service and acts as a mediator between presentation layer and persistence layer to exchange data. This layer comprises of four core services available in *CADsIoT Core* component: device, sensor, context and subscription. Services can be invoked through provided HTTP interfaces through presentation layer. Requests sent through uniform interfaces are mapped to corresponding services, perform business rules (business logic) and sends required functionality to the requester. These services are implemented as individual service components, leading to flexible and scalable architecture, enabling developers and system integrators to leverage devices and sensors functionalities through provided RESTful APIs.

Data Access Layer

Data access layer is an abstraction layer which provides a seamless interface to access data from one or more data stores. This layer hides the complexity of the underlying storage mechanism being used. The layer is responsible for fulfilling data related requests, requested by appropriate services, defined at the service layer. The real benefit of this layer is to separate business logic from different data store technologies.

Data Stores

This layer usually consists of persistence storages, directory servers, legacy and enterprise systems. IoT data is typically distributed across various data storage mechanisms e.g., Binary Large Object (BLOB) for unstructured data, Key-value storages, relational database management systems. When combining, a consolidated and integrated view of data is presented to appropriate request/system. Through *CADsIoT Core* flexible architecture, enterprise data stores can be accessed through unified RESTful APIs, encapsulating the complexities of underlying storage mechanisms.

5.1.2 CADsIoT Navigator

CADsIoT Navigator is an Android based prototype that is capable of discovering devices and sensors automatically (discussed in the common layer of Navigator architecture, 5.1.2) deployed in a particular environment, regardless of their heterogeneity. *Navigator* interacts with *CADsIoT Core* to receive registered devices, contextual data and subscription information to provide real-time notifications which tells, where the devices are installed. In general, *Navigator* implementation demonstrates the feasibility of the proposed approach rather than developing a complete IoT application. Figure 5.2 depicts the high-level architecture of *Navigator*, outlining various building blocks.

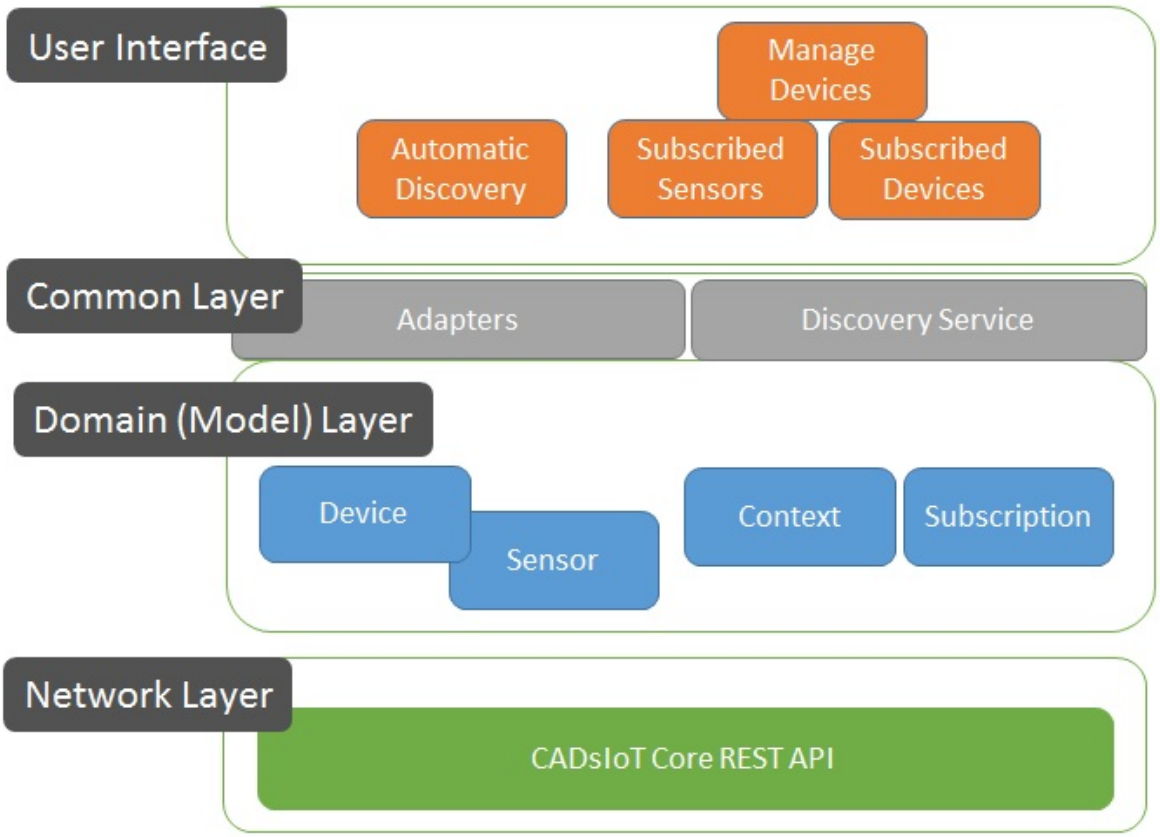


Figure 5.2: Navigator Architecture

User Interface Layer

User Interface Layer also called presentation layer provides a user experience in order to interact with different services provided by the application. This layer consists of several components which are native to Android platform and serves as the building blocks. These components include activities, fragments, layouts, services when integrated together, provides a rich user interface. *Navigator* leverage Android platform user interface assets to enable seamless interaction with devices and sensors for dynamic discovery as discussed in the common layer of Navigator architecture. The interfaces for devices, sensors, subscriptions helps to perform various associated tasks using a consistent user experience.

Common Layer

Common layer deals with various Android components e.g., adapters, services. Adapters are data storage holders which bind to specific UI to display data on the interface. Android services are application components that perform long running requests and do not provide any user interface. These services have its own lifecycle and can be invoked by different application components e.g., activities, fragments. *Navigator* provides a discovery service that resides between the user interface and domain layer to discover devices automatically. The service uses proactive Location Based Services (LBS), better known as Geofencing [GSD15] which allows remote monitoring of objects. Navigator's discovery service notifies the user about the location-based information provided by GPS if the user enters or leaves dedicated circular regions, called geofence. It then matches the positioning information provided by mobile devices with the devices and sensors locations (fetched by backend service), represented by geofences and trigger notifications about the discovered devices.

Domain (Model) Layer

One of the important layer is a domain layer which interacts with the network layer to request data and includes business logic and data model. Respective services at this layer correspond to *CADsIoT Core* (backend service) and request required data. The services work with data models for devices, sensors, context and subscriptions. Once required information is retrieved into appropriate data models, objects are sent to upper layers, where information can be served by data adapters and associated application components.

Network Layer

The network layer as the name says, performs network related tasks required by the application. In *Navigator* architecture, this layer is responsible for communicating with the backend service in order to perform HTTP operations. Google Volley [1] is used as an HTTP library to enable this interaction. It also manages scheduling of requests and provides caching facilities, eliminating developers to write caching code again. Upon successful communication with backend service, raw data is parsed to appropriate data models. Furthermore, mappers e.g., date formats are also considered at this layer.

[1] https://developer.android.com/training/volley/index.html

5.2 CADsIoT Core REST API

CADsIoT Core is a RESTful application and exposes data and functionality as resources, enabling consumers to perform create, retrieve, update and delete actions using HTTP. This section outlines the design and development considerations of *CADsIoT Core*. The most common ingredients used for *CADsIoT Core* development are resources along with its representations, URIs and interactions are considered in this section.

5.2.1 Resources

CADsIoT Core deals with four main resources to fulfill the requirements. These resources comply with the two basic characteristics, defined by RESTful Web Services: a) URI is required to access each resource b) each resource has at least one representation for the external world. Figure 5.3 is a Entity Relationship Diagram (ERD) which helps in designing REST resources and their relationships. Furthermore, description of each resource is also outlined in this section.

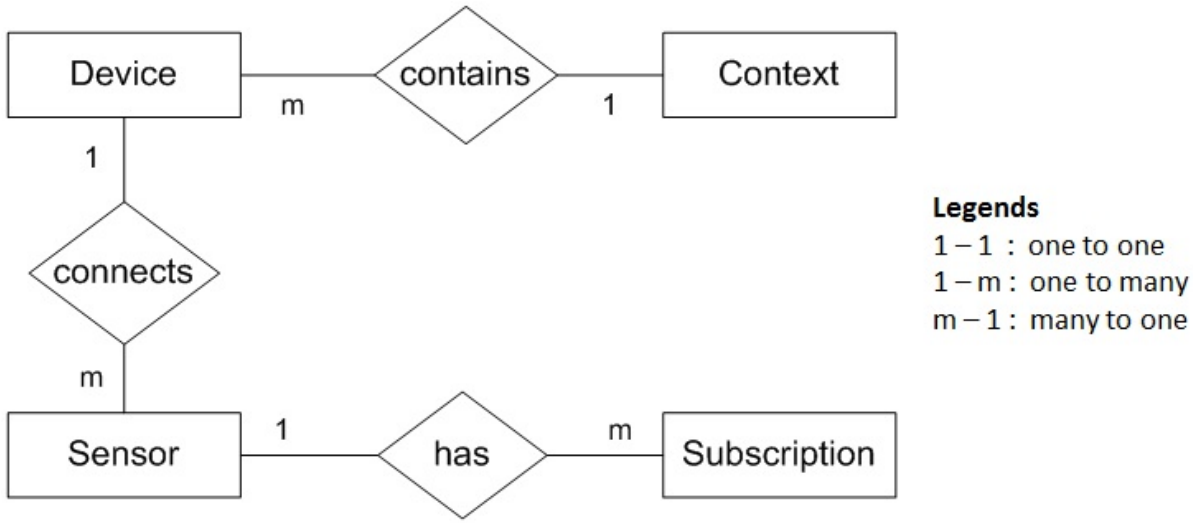


Figure 5.3: ER-Diagram for Resource Design

Device Resource : A device resource is used to create, retrieve, update and delete devices from the system. Devices have a physical context which specifies the physical location where they are found. The contextual object consists of GPS and some relevant information. It is assumed that devices connect to one or more sensors in the system. Table 5.1 depicts properties of device resource.

Attributes	
id	**long** Unique identifier generated by the system
name	**string** Name of the device
description	**string** Description of the device
manufactureName	**string** Device manufacture name
creationDate	**date** System generated date date of the device entered in the system
modificationDate	**date** Date of device modification in the system
active	**boolean** Represents current status of the device
networkAddress	**string** local IP address

Table 5.1: Device Resource properties

Sensor Resource : A sensor resource represents an object that is capable of providing data and is connected to a device or microcontroller e.g., Raspberry, Arduino etc. Likewise devices, sensors also have physical locations which show the indoor locations where the sensors are installed. Normally, the global contextual information e.g., city, area, street is determined by its associated device. It is assumed that sensors do not work alone by themselves but requires a device which signals to perform some action e.g., reading data. Table 5.2 depicts properties of sensor resource.

Attributes	
id	**long** Unique identifier generated by the system
name	**string** Name of the sensor
description	**string** Description of the sensor
manufactureName	**string** Sensor manufacture name
creationDate	**date** System generated date of the sensor entered in the system
modificationDate	**date** Date of sensor modification in the system
active	**boolean** Represents current status of the sensor
location	**string** name of the place where sensor is place e.g Building8/Floor2/Room22
category	**string** Type of sensor e.g temperature, motion, sound

Table 5.2: Sensor Resource properties

Context Resource : Context resource represents a physical location e.g., building, school, office, universities, warehouses etc. The context is identified mainly using GPS coordinates that can be used to discover relevant devices in a particular region. Context provides location information to devices. It is assumed that a particular device has exactly one context at a given time.

Attributes	
id	**long** Unique identifier generated by the system
name	**string** Name of the context
description	**string** Description of the context
streetName	**string** name of the street where context is found
streetNo	**string** Street number
city	**string** City of the context
zipcode	**string** Zip code of the context
latitude	**double** Denotes latitude in decimal value
longitude	**double** Denotes longitude in decimal value
creationDate	**date** System generated creation date of the context
modificationDate	**date** Date of context is modified on in the system

Table 5.3: Context Resource properties

Subscription Resource : Subscription resource allows sensors to be subscribed. A subscriber can be a system, service or a user who is interested in interacting with subscribed sensors. However, *CADsIoT Core* subscription module considers users as the subscribers. When a sensor is subscribed, the device attached to subscribed sensors is also subscribed based on the parent-child rule.

Attributes	
id	**long** Unique identifier generated by the system
name	**string** Name of the subscriber
description	**string** Description of the subscriber
email	**string** email address of subscriber
creationDate	**date** System generated creation date of the subscription
modificationDate	**date** Subscription modification date in the system
sensorId	**long** sensor unique identifier to be subscribed

Table 5.4: Subscription Resource properties

5.2.2 Representations

Data requires some representation to flow on the internet and is often expressed in some type of format. These formats are classified as MIME types which are standards and conventions to exchange data over the internet in a convenient manner. In REST world, resources have data associated with them and are commonly expressed in two media types: JSON and XML. *CADsIoT Core* also leverages JSON representation due to its expressiveness and broader availability.

Device Resource : Listing 5.1 depicts complete representation in JSON which includes context and sensor objects.


```
{
"id": 1,
"name": "Raspberry PI 2",
"description": "This device is capable to provide sound and motion information",
"manufactureName": "Element Design",
"creationDate": "2016-08-27T10:47:042Z",
"modifiedDate": "2016-08-29T13:45:025Z",
"active": true,
"networkAddress": "192.168.0.9",
"contextInfo": {
      "id": 2,
      "name": "FIUS",
      "description": "University",
      "streetName": "university street",
      "streetNo": "38",
      "city": "Stuttgart",
      "zipcode": "70569",
      "country": "Germany",
      "lat": 48.744552,
      "lon": 9.106863,
      "creationDate": "2016-08-24T17:00:015Z",
      "modifiedDate": "2016-08-24T17:00:015Z",
      "deviceId": null
},
"sensors": [
      {
            "id": 2,
            "name": "Temperature Sensor",
            "description": "Temperature Sensor",
            "manufactureName": "Texas Instruments",
            "creationDate": "2016-08-23T14:53:006Z",
            "modifiedDate": "2016-08-23T14:53:006Z",
            "active": true,
            "location": "Building38/Floor1/Room23",
            "category": "temperature",
            "connectionType": "wired",
            "topicId": "topicId",
            "device": {
                  "id": 1
            }
      }
  ]
}
```


Listing 5.1: Device Representation in JSON

Sensor Resource : Listing 5.2 depicts the complete JSON representation which also includes subscriptions objects.


```
{
      "id": 2,
      "name": "Temperature Sensor",
      "description": "Temperature Sensor",
      "manufactureName": "Texas Instruments",
      "creationDate": "2016-08-23T14:53:006Z",
      "modifiedDate": "2016-08-23T14:53:006Z",
      "active": true,
      "location": "Building38/Floor1/Room23",
      "category": "temperature",
      "deviceId": null,
      "subscriptionId": null,
      "connectionType": "wired",
      "topicId": "topicId",
      "subscriptions": [
      {
            "id": 7,
            "name": "Allan Mark",
            "description": "Allan Mark",
            "email": "allan.mark@gmail.com",
            "creationDate": "2016-09-24T14:27:031Z",
            "modifiedDate": "2016-09-24T14:27:031Z",
            "sensorId": 2
      },
      {
            "id": 12,
            "name": "Karim Khan",
            "description": "Karim Khan",
            "email": "karim.khan@gmail.com",
            "creationDate": "2016-09-24T15:52:055Z",
            "modifiedDate": "2016-09-24T15:52:055Z",
            "sensorId": 2
      }
      ]
}
```


Listing 5.2: Sensor Representation in JSON

Context Resource : Listing 5.3 depicts the complete JSON representation.

```
{
      "id": 1,
      "name": "University of Stuttgart Campus Vaihingen",
      "description": "State University",
      "streetName": "Pfaffenwaldring",
      "streetNo": "7",
      "city": "Stuttgart",
      "zipcode": "70569",
      "country": "Germany",
      "lat": 48.745727,
      "lon": 9.104995,
      "creationDate": "2016-08-24T17:00:015Z",
      "modifiedDate": "2016-08-24T17:00:015Z",
      "deviceId": null
}
```

Listing 5.3: Context Representation in JSON

Subscription Resource : Listing 5.4 depicts the complete JSON representation. Subscriptions are user objects which represents the relationship with the sensors.

```
{
      "id": 4,
      "name": "Andreas Schmid",
      "description": "Andreas Schmid",
      "email": "andreas.schmid@gmail.com",
      "creationDate": "2016-09-20T17:00:022Z",
      "modifiedDate": "2016-09-20T17:00:022Z",
      "sensorId": 4
}
```

Listing 5.4: Subscription Representation in JSON

5.2.3 URI Patterns

The addressability aspect of RESTful APIs is covered by resource identifiers often called URIs. Interesting information provided by the server is exposed as resources, can be located using URIs. Every resource has its own unique identifier for its identification. These URIs typically conforms as Universal Resource Locator (URL) in RESTful world, consisting of the unique pattern, scheme, authority, path, query and fragment [MR97]. Since resources do have lifecycle and changes over time, defining URL patterns is very important. This enables locating resources directly through static paths anytime, enabling predictability, hierarchical structure for better understandability and usability.

CADsIoT Core REST resources follows industry best practices for defining URI patterns [MM12]. REST clients e.g., *Navigator* request resources using defined URI patterns with JSON payload. Table 5.5 illustrates URI patterns for corresponding resources available in *CADsIoT Core*.

Resource	URI	Description
Device	/devices	URI pointing to collection of devices
	/devices/{id}	URI identifying particular device given by device id
Sensor	/sensors	URI pointing to collection of sensors
	/sensors/{id}	URI identifying particular sensors given by sensor id
Context	/locations	URI pointing to collection of context
	/locations/{id}	URI identifying particular context given by context id
Subscription	/subscriptions	URI pointing to collection of subscriptions
	/subscriptions/{id}	URI identifying particular subscription given by subscription id

Table 5.5: CADsIoT Core REST URI Patterns

5.2.4 Interactions

Resources are managed by uniform interfaces required by RESTful architecture. Each representation implements create, retrieve, update and delete (CRUD) operations corresponding to HTTP verbs – POST, GET, PUT, DELETE. Table 5.6 depicts the commonly used operations and their relationships with HTTP verbs.

Operations	HTTP Verbs	Description
Create	POST	Use to create new resources in a collection
Retrieve	GET	Use to retrieve a required representation of resource
Update	PUT	Use to create or update resources
Delete	DELETE	Use to delete resource

Table 5.6: HTTP Verbs and CRUD Operations Relationships

This section provides an understanding on how create, retrieve, update and delete (CRUD) operations can be performed to manipulate CADsIoT resources using HTTP verbs. However, the complete listing of request and response for each type of resource is discussed in REST API documentation, Appendix A.

Creating Resources

HTTP verb "*POST*" is frequently used to create resources. In fact, it adds the new resource within the entire collection and a new identifier (ID) is assigned by the service. On successful creation of a resource, the server sends back a "*Location*" header containing a link of the newly created resource with response code 201 (CREATED). According to HTTP specification [FRJ14], POST should be used to make non idempotent requests as it is neither safe and nor idempotent. Performing two identical POST requests will result in creating two resources with the same information.

The examples show adding new CADsIoT asset e.g., devices, sensors, context and subscriptions in the system using HTTP POST method. On successful creation, the server returns a response code of 201 (CREATED) and a "*Location*" header containing a URL, embed with the newly created ID of the resource. The POST request accepts JSON as payload to define respective resource properties in the system. The POST request follows a URL pattern which typically includes baseURL[2] and resourceURI[3] e.g., baseURL/resourceURI. Here are the examples of individual resource types defined in CADsIoT system.

Examples:

Device — POST baseURL/devices
Sensor — POST baseURL/sensors
Context — POST baseURL/locations
Subscription — POST baseURL/subscriptions

Retrieving Resources

Representation of a resource can be fetched or read using HTTP verb "*GET*". The method returns the collection of a resource or a particular resource in an accepted representation e.g., JSON or XML with a response code of 200 (OK). When a required representation is not found, it often returns 404 (NOT FOUND) or 400 (BAD REQUEST) to the client. According to HTTP design specification, GET allows reading data only and considered as a safe operation as it does not modify or alter data in the system. Making several GET

[2]Represent the actual URL where the service is deployed e.g http://www.example.com/
[3]Represent URI of the target resource

requests to the same URI will result in the same output as with single request. Hence, GET operation is idempotent.

CADsIoT Core leverage HTTP GET to read required information from the system. Upon completion of a request, the server sends a JSON representation of a required resource with a response code of 200 (OK). If a resource is not found, a response code of 404 (NOT FOUND) is returned to the client. Additionally, GET requests also accept query parameters which allow filtered data to be returned by the server by following typical URL patterns e.g., baseURL/resourceURI/ and baseURL/resourceURI/id. Some of the examples of GET requests defined in CADsIoT system are listed below.

Examples:

Device — GET baseURL/devices
Sensor — GET baseURL/sensors/2
Context — GET baseURL/locations
Subscription — GET baseURL/subscriptions/1

Modifying Resources

HTTP verb "*PUT*" is mainly used to update existing resources in the system. It takes a newly updated representation of the original resource in the body of the request. Sometimes, PUT is also used to create a new resource in case if the existing resource is not available. In a such a case, a new resource gets created with provided representation along with the resource identifier (ID) in the request URI. On successful update, a response code of 200 (OK) or 204 (NO CONTENT) is returned by the server. Due to network requirements, the body in the response is optional.

CADsIoT Core provides partial and complete modification of a resource. Partial update refers to modify parts (not complete) of the resource whereas complete (full) modification allows replacing the existing state with the newly updated representation of the original resource. HTTP PUT is used to provide complete modification semantics against a specified request URI which accepts a complete representation of a resource. However, HTTP POST is used to partially update the resource in the proposed system. If the update is successful, 200 (OK) is returned by the server with the status description of the resource representation in the response body. To avoid the creation semantics of PUT operation, the system returns 404 (NOT FOUND) if the requested resource does not exist.

Examples:

Device(full update) — PUT baseURL/devices/1
Device (partial update) — POST baseURL/devices/2
Context — PUT baseURL/locations/2

Subscription — PUT baseURL/subscriptions/1

Deleting Resources

Resources can be deleted using HTTP verb "*DELETE*". It allows the resource to be deleted identified by a URI. Once the server has successfully deleted the resource, 200 (OK) response code with the status description of the representation or 204 (NO CONTENT) without the response body is returned. According to HTTP specification [FRJ-RFC14], deleting a resource is idempotent. If several delete requests are performed against the same URI, the result will be the same since the resource has already been deleted in the first request. Additionally, a response code of 404 (NOT FOUND) is sent to the client.

CADsIoT Core uses HTTP DELETE to remove a resource from the system. Upon successful deletion, 200 (OK) response code is returned by the server. If the resource is not found, a response code of 404 (NOT FOUND) is sent to the client. Some of the examples are listed below.

Examples:

Device — DELETE baseURL/devices/1
Sessor — DELETE baseURL/sensors/2
Context — DELETE baseURL/locations/2
Subscription — DELETE baseURL/subscriptions/1

6 CADsIoT: Implementation and Validation

This chapter outlines technical details of the CADsIoT solution. Starting with core features, it explains how the proposed concept is transformed into technical specifications, resulting in two prototypes: *CADsIoT Core* and *Navigator*. Furthermore, necessary technical concepts and anatomy of each prototype, leading to validation are also considered in this chapter.

6.1 Core Features

CADsIoT is designed to be simple and overcomes device discovery pitfalls based on particular context. It leverages state-of-the-art tools and technologies to implement and validate the proposed concept as discussed in Chapter 4 and Chapter 5. The prototype comprises of two main implementations i.e., building a scalable backend service notably, *CADsIoT Core* and a smart mobile implementation serving as a client called *Navigator*. Both prototypes contribute to achieve the desired goal. As we move further, the concepts discussed will get clarified in architecture and validation section. Figure 6.1 shows the four major features provided by CADsIoT.

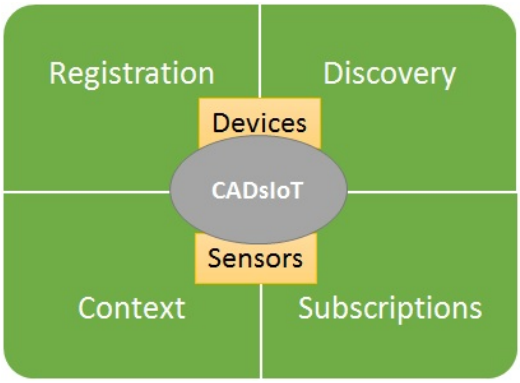


Figure 6.1: CADsIoT Core Features

6.1.1 Registration

Registration and connection of devices is a vital process for any IoT enabled application. Users continuously search access to devices for weather, surveillance, agricultural and other context data. Hence, CADsIoT provides a registration module where IoT assets e.g., devices, sensors and contextual information can be registered and managed. Figure 6.2 shows two registered devices with their current status, deployed location and the number of sensors attached to the particular device.

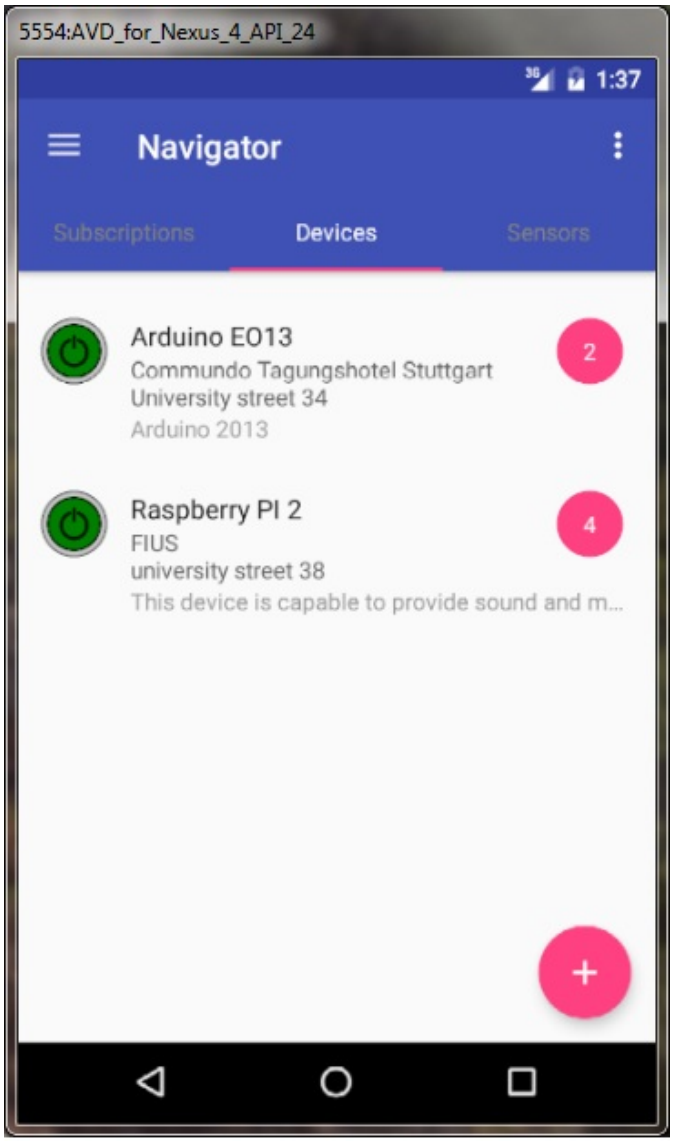


Figure 6.2: Registered Devices

6.1.2 Subscriptions

Subscriptions facilitate control over particular assets in which the user is interested in. These subscriptions are applied on sensors which are attached to respective devices. If a user subscribes to a particular sensor, then its associated device will automatically be subscribed, following a parent-child relationship principle. Let say, a user wants to know information e.g., temperature, aggregated from a group of sensors, he can do so by subscribing respective sensors and perform the desired operation. Figure 6.3 shows that a particular user has subscribed to three different sensors.

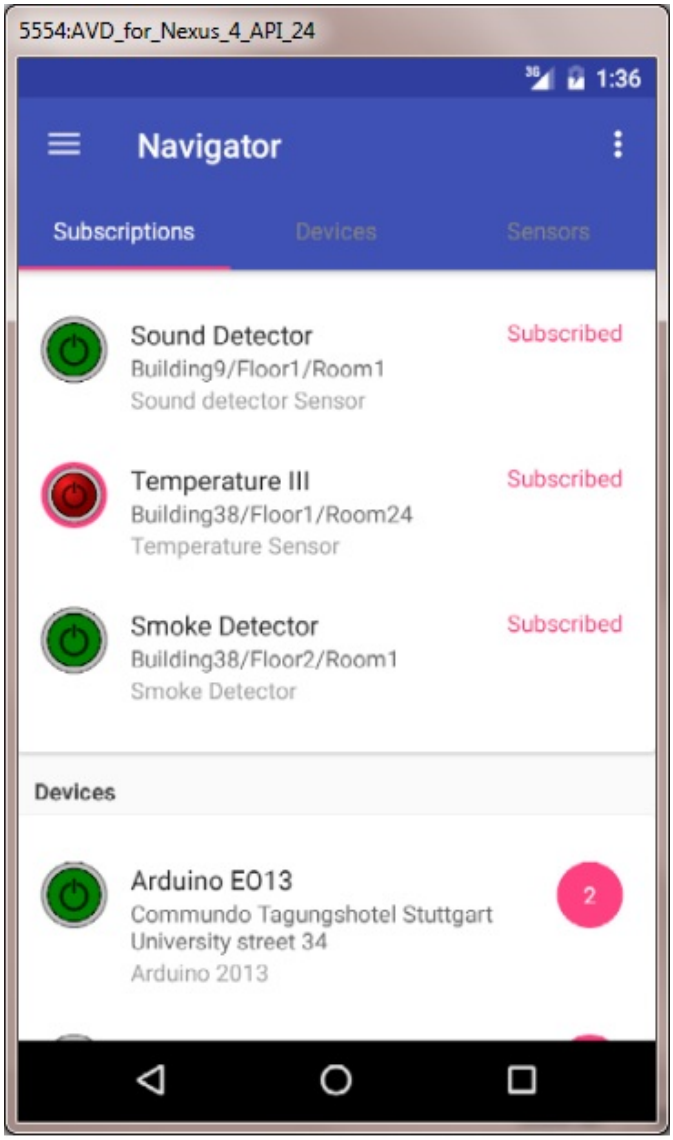


Figure 6.3: Sensors and associated devices subscriptions

6.1.3 Device Discovery

Device discovery is a crucial step for the resource usage and data generated by IoT environments. This is because, these devices are deployed in heterogeneous and widely distributed environments, where discovering and management of devices is challenging. CADsIoT leverages location information through GPS as a primary category of contextual information, in order to discover nearby devices. *Navigator* communicates with *CADsIoT Core* to collect registered devices, context and subscription information and allows dynamic discovery based on Geofencing approach as discussed in the Navigator architecture. Figure 6.4 depicts discovery of devices found in a region.

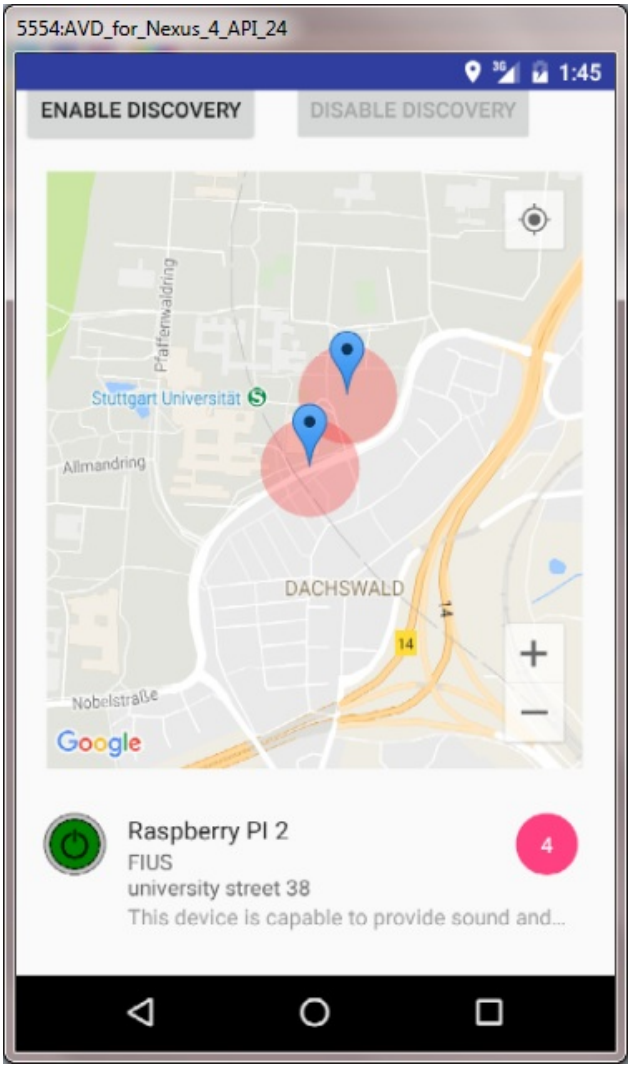


Figure 6.4: Discovered devices as the user entered the region

6.1.4 Real Time Notifications

CADsIoT facilitates users to know which devices are available in the current region. This information is provided by *Navigator* application. The devices are located by matching user's current context and device context, facilitated by real-time notification and a map. Furthermore, whenever the user reaches the same region again, appropriate notifications are triggered. Figure 6.5 illustrates notification feature provided by CADsIoT via *Navigator*.

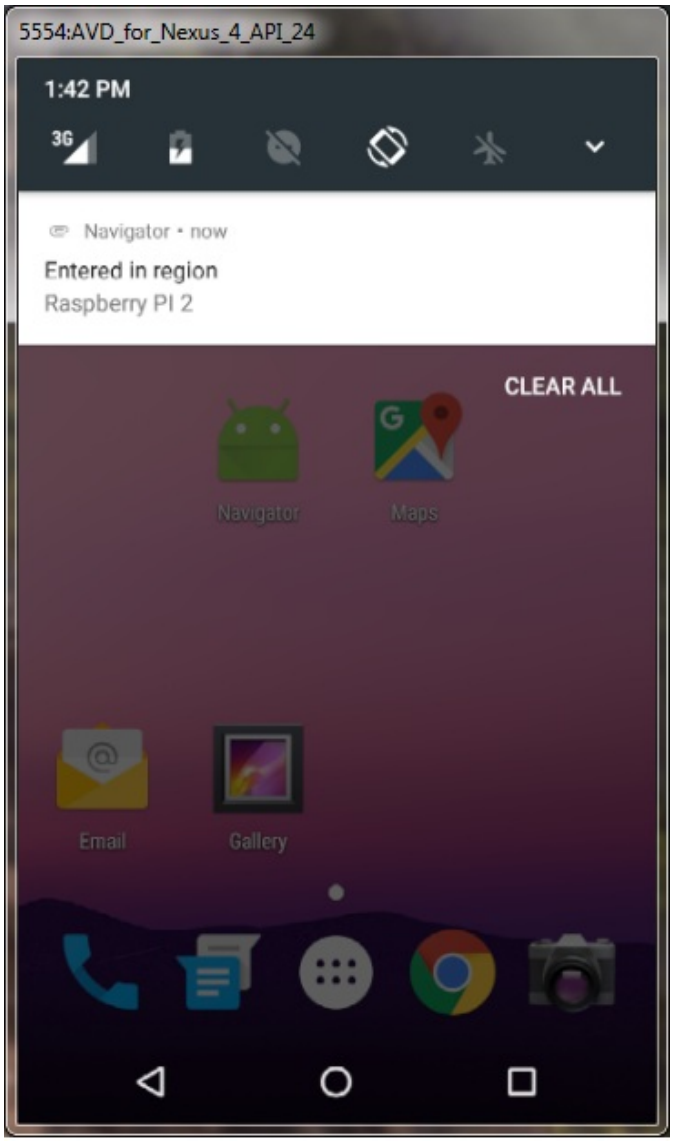


Figure 6.5: Real Time Notification

6.2 CADsIoT Core: Backend Service

CADsIoT Core is a service that provides a set of functionalities to enable device registration and discovery features in an IoT environment. Due to a variety of tools and technologies currently available, the selection for its implementation includes ample amount of research. *CADsIoT Core* is built on REST, an architectural style to develop scalable and reliable applications. The REST style is often used with HTTP, an application level protocol that provides operations to exchange representation between clients and servers. In contrast to the techniques such as COBRA, SOAP, REST is lightweight, consumes less bandwidth, allows many different data formats, scalability and flexibility to write web services which require small learning curve. Therefore, keeping in view to build a scalable architecture, additional set of technologies are also used together with Java programming environment and are discussed below. It should be noted that detail explanation of technologies can be reviewed in the given references.

6.2.1 Java Persistence API (JPA)

Almost every application requires objects to be persisted in some storage devices e.g., database management system. Traditional CRUD operations on relations which are represented in the form of tables in relational database systems are performed using SQL and JDBC. These techniques become even worse if complex SQL queries are written as the complexity of the application increases. As a result, extraordinary efforts are required to manage the application. Treating queries as objects combining with object oriented principles can certainly provide an abstraction between application code and persistence mechanism. In order to provide abstraction to perform database operations, a solution called Object/Relational Mapping (ORM) is commonly used in the industry. This model forms a bridge to map database relations to objects using the specification JSR-000338[1] and tools like Hibernate [2]. Java Persistence API(JPA) is a standard Java API that simplifies data persistence and allows database relations to be mapped as Java objects. It provides various annotations e.g., @Entity, @ID, @Column to persist data using services of the entity manager.

[1] http://download.oracle.com/otndocs/jcp/persistence-2.0-fr-oth-JSpec/
[2] an open source implementation of JPA, http://hibernate.org/orm/

6.2.2 Java API for RESTful Web Services (JAX-RS)

The Java API for RESTful Web Services (JAX-RS) [3] is a java programming API specification (JSR-311) that allows creating web services using REST architectural pattern. Its uses annotations to help in mapping resources (java classes) as a web resource e.g., @Path, @Produces, @Consumes. Jersey [4] is a reference implementation of JAX-RS and is used to hide low-level details and exposed resources in several representations such as JSON, XML etc. JAX-RS provides the semantics of annotations e.g., @GET, @POST etc while Jersey facilitates with request parsing, invoking right methods etc. *CADsIoT Core* provides JSON as the primary media type. It uses Jersey and Jackson API [5] to convert objects from Java and JSON (vice versa). Jersey also supports Spring constructs, dependency Injection in particular [6]. Spring encapsulates transactions, persistence, and security efficiently and its integration with jersey allows Spring beans to be used as JAX-RS components (resources, provides). Spring provides annotations such as @Component, @Repository, @Service that enable JAX-RS to work with Spring functionality.

6.2.3 Anatomy of Components

Earlier, the high-level architectural details in section 5.1.1 are discussed. This section covers the internal application design and implementation of *CADsIoT Core*. The implementation consists of several components which ensures its loosely coupled design, separating business logic from the type of storage mechanism being used.

To better understand interactions by different HTTP clients e.g., Postman, OKHTTPClient and even *Navigator* (discussed later), a pictorial representation is done to explain each component and its responsibility. Each resource e.g., device, sensor, subscription and context follows consistent interaction pattern. REST clients requests for services which are responsible to provide relevant functions to fulfill the request. These functions collaborate with the data access layer to retrieve persistence information from the datastore. For the sake of simplicity, only one resource type with important component details are highlighted here. The complete listing of resources, methods, response codes are discussed in REST API documentation Appendix A.

Figure 6.6 shows the application process and how different components interact together when a new request is received. These details are discussed below:

[3] https://jax-rs-spec.java.net/
[4] https://jersey.java.net/
[5] https://jersey.java.net/documentation/latest/media.html
[6] (http://projects.spring.io/spring-framework/)

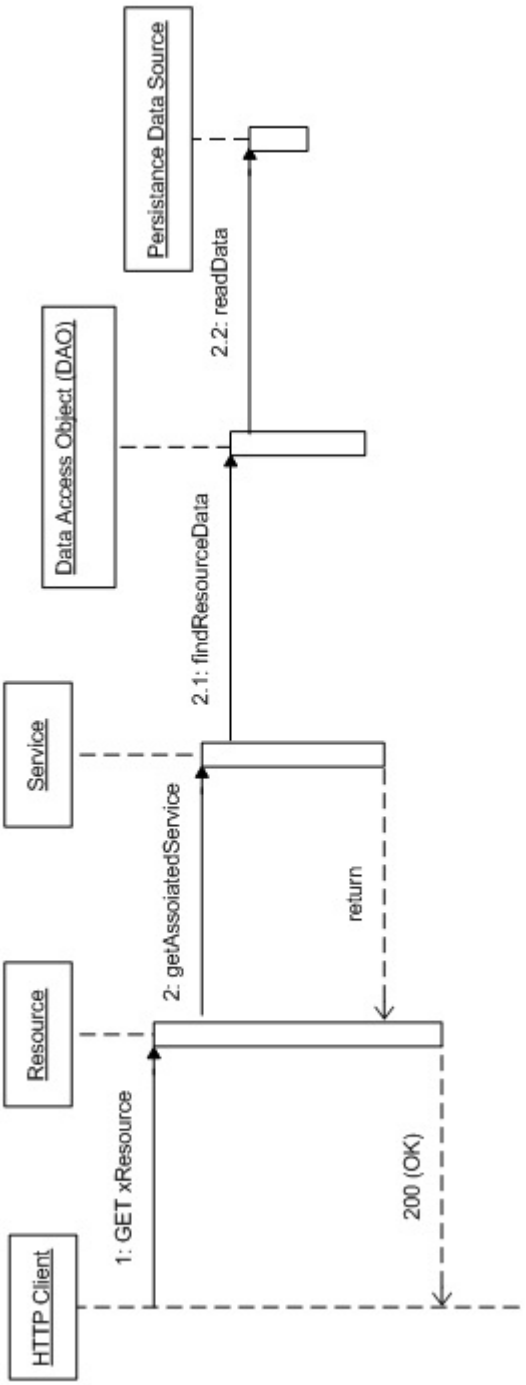


Figure 6.6: CADsIoT Core Internal Components Communication

HTTP Client

Any HTTP client can perform operations on resources within *CADsIoT Core* using four basic HTTP operations: GET, POST, PUT and DELETE. In figure 6.6, the client is interested to get a collection of e.g., sensor resource. In such a case, a GET request is issued by HTTP client with a specific URL http://127.0.0.1:8080/discover-iot/rest/api/v1/sensors. Listing 6.1 shows a self-contained request for a sensor resource and invokes GET method. Upon completion of a request, a response code of 200 (OK), including a collection of sensors represented in JSON is sent back to the requester.

```
GET /discover-iot/rest/api/v1/sensors/ HTTP/1.1
Host: 127.0.0.1:8080
Cache-Control: no-cache
```

Listing 6.1: Sensor HTTP GET Request

```
@GET
@Produces({MediaType.APPLICATION_JSON})
public List<Sensor> getSensors(@Context UriInfo info) throws AppException {
      List<Sensor> sensorList = null;
      //find all sensors at uri /sensors
      if (info.getQueryParameters().size() == 0){
            log.info("Getting all sensors...");
            sensorList = sensorService.getSensors();
      }//  /sensor?<queryString>
      else if (info.getQueryParameters().size() > 0 ){
            Map<String, String> filterMap = new HashMap<String, String>();
            filterMap.put("subscriber",info.getQueryParameters()
                                                .getFirst("subscriber"));

            if (info.getQueryParameters().getFirst("lat") != null &&
                        info.getQueryParameters().getFirst("lon") != null){
                  filterMap.put("lat",info.getQueryParameters().getFirst("lat"));
                  filterMap.put("lon",info.getQueryParameters().getFirst("lon"));
            }
            sensorList = sensorService.filterSensors(filterMap);
      }
      log.info("returning available sensors");
      return sensorList;
}
```

Listing 6.2: Code Snippet for handling GET Request

Listing 6.2 shows the corresponding implementation of reading sensors in a sensor resource. It can be noted that certain annotations are injected which are applied on the

method level. However, there are also class level annotations which are necessary to bind corresponding services with the resources.

- @GET – correspond to HTTP GET requests.
- @Produces – defines the response of this request to be produced. In our case, the response is presented in JSON representation.

Services

CADsIoT Core services reside at the service layer discussed in section 5.1.1. These services are defined by business rules (logic). Once requests are accepted by appropriate resources, associated services invoke the desired operation. In our case, sensor service is responsible for providing a collection of sensors.

```
@Override
public List<Sensor> getSensors() throws AppException {
      List<SensorEntity> sensors = sensorDao.findAllSensors();
      if(sensors == null){
            throw new AppException(Response.Status.NOT_FOUND.getStatusCode(),
            404,
            "No sensors found in the database",
            "Please add some sensors in the database",
            Constants.HELP_URL);
      }
      return getSensorsFromEntities(sensors);
}
```

Listing 6.3: Code Snippet for Sensor Service

Listing 6.3 shows the corresponding method invoke by the resource. It should be noted that sensor service is automatically bound to the sensor resource through spring provided dependency injections. The service talks to the persistence layer, responsible to fetch required data from the data store. Upon receipt, relevant Java Persistence API (JPA) entities are collected and transform to normal Plain Old Java Objects (POJOs). These POJOs simplify JPA and JSON annotations, making code more cleaner.

Data Access Object(DAO)

CADsIoT uses Data Access Object on a persistence layer, a design pattern responsible to abstract the details of the persistence mechanism. This pattern allows changes to be made only at the persistence level if underlying storage mechanism changes.

```
@Override
public List<SensorEntity> findAllSensors() {
      String sqlString = "SELECT s FROM SensorEntity s";
      TypedQuery<SensorEntity> query = entityManager.createQuery(sqlString,
          SensorEntity.class);
      return query.getResultList();
}
```

Listing 6.4: Code Snippet for Sensor DAO

Listing 6.4 shows to fetch sensors from the data store. JPA entities are user defined classes and maps the properties as defined in the database. Hibernate, a reference implementation of JPA is used as a provider to implement more granularity and leverage hibernate specific features e.g., lazy and eager loading when dealing with relationships.

As discussed above, every resource, component communicates in the same fashion as shown in figure 6.6. The complete listing of resources, methods, response codes are discussed in REST API documentation Appendix A.

6.3 Navigator: Smartphone Client

Navigator is Android based mobile application acting as the client in order to interact with the backend service. *Navigator's* contribution to dynamically discover devices is of significant importance. It allows users to discover devices and sensors automatically and also serves as HTTP client as it interacts with *CADsIoT Core*.

This section highlights the implementation aspects of *Navigator* which consists of Android native libraries, services and their interactions in the application. Our focus is to describe main components/libraries adhering to the proposed concept rather than discussing the complete Android platform and known services which can be reviewed through Android documentation.

6.3.1 Discovery Service

As discussed in section 5.1.2, Geofencing is an approach to monitor geographical areas remotely and triggers notifications when a mobile object enters or exits these areas [RFD09]. Monitoring and tracking of objects are surrounded by location services. This is one of the reasons, most context-aware applications make use of the location features provided by mobile objects.

Navigator also makes use of geofencing approach, supporting the discovery of devices and sensors with automatic notifications. *Navigator* discovery service is a background service and is implemented using Android native Geofence libraries [7]. This service defines circular regions, also called fences based on the certain radius with respect to device contexts (locations). The user subscribes to interested devices which he wants to be notified upon entering the device region.

Listing 6.5 depicts how the service responds to transitions. The background service will get handled once the user enters in the device region and identifies the type of transitions e.g., entering a region, staying in a region, exiting from a region. Based on the transition type, it triggers real-time notification providing information of the discovered devices, which are also shown on the map.

[7] https://developer.android.com/training/location/geofencing.html

```
@Override
protected void onHandleIntent(Intent intent) {
      Log.d(TAG, "onHandleIntent hasBundle: "+intent.hasExtra("myBundle"));
      Bundle bundle = intent.getExtras().getBundle("myBundle");
      Log.d(TAG, "devices List: "+bundle.getParcelableArrayList("devices"));
      ArrayList<Device> devices = bundle.getParcelableArrayList("devices");
      GeofencingEvent geofencingEvent = GeofencingEvent.fromIntent(intent);

      if(!geofencingEvent.hasError()) {
            int transition = geofencingEvent.getGeofenceTransition();
            String notificationTitle;

            switch(transition) {
                  case Geofence.GEOFENCE_TRANSITION_ENTER:
                        notificationTitle = getString(R.string
                              .geofence_transition_notification_text_entered);
                        Log.v(TAG, "Region Entered");
                        break;
                  case Geofence.GEOFENCE_TRANSITION_DWELL:
                        notificationTitle = getString(R.string
                              .geofence_transition_notification_text_dwell);
                        Log.v(TAG, "Dwelling in Region");
                        break;
                  case Geofence.GEOFENCE_TRANSITION_EXIT:
                        notificationTitle = getString(R.string
                              .geofence_transition_notification_text_exited);
                        Log.v(TAG, "Geofence Exited");
                        break;
                  default:
                        notificationTitle = "Geofence Unknown";
                  }
            Log.d(TAG, "getTriggeringGeofences: "+ getTriggeringGeofences(intent) );
            sendNotification(this, getTriggeringGeofences(intent),
                notificationTitle, devices);
      }
}
```

Listing 6.5: Code Snippet for Discovery Service

6.3.2 Google Volley API

Navigator communicates with *CADsIoT Core* to exchange information on the network layer. Volley API is used to enable this interaction. Volley is a Google offering which performs HTTP based operations in an efficient manner. It provides networking for Android application easier as a lot of boilerplate code is also catered by the API. It also manages scheduling of requests and provides caching facilities, eliminating developers to write caching code again. A detailed discussion about the API is out of the scope of this research. However, readers can refer to Google Android documentation [8].

6.3.3 Anatomy of Components

Likewise any other Android application, *Navigator* comprises of several internal application components. These components interact each other at different levels. While developing the application, the focus to achieve the desired goal i.e., discovering devices based on context automatically is considered rather focusing on user experience.

This section outlines the basic components pattern of *Navigator* application such as activities, fragments, adapters, listeners and communication with *CADsIoT Core* backend service. *Navigator* can be extended by making use of its reusable components to incorporate more features. Figure 6.7 shows application components involved in *Navigator*.

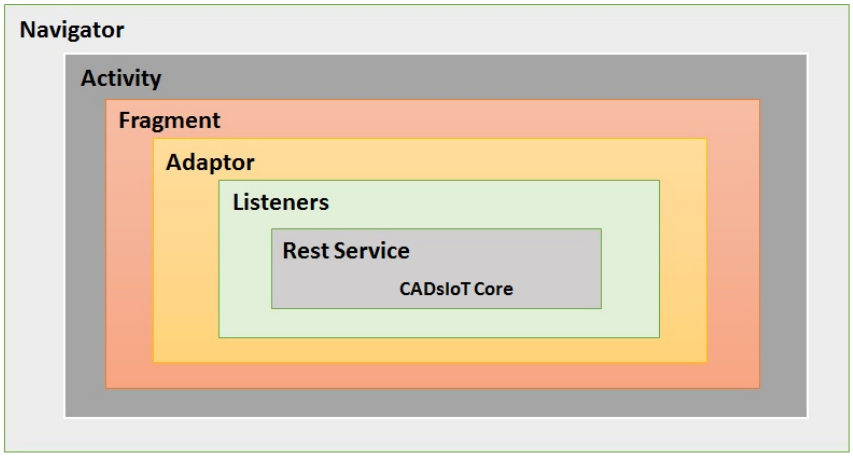


Figure 6.7: Navigator Application Components

[8] https://developer.android.com/training/volley/index.html

The top level component in an Android application are activities. These activities are the entry points to serve different functions that a user can perform. They have their own lifecyle and have different states e.g., onCreate, onResume, onDestroyed. *Navigator* makes use of the device, sensor, discovery activities which serve as the user interface and displays relevant data.

Navigator activities contain fragments. These fragments are a portion of user interfaces that reside in an activity. *Navigator* internal application architecture make use of these fragments to avoid reinventing the wheel and connect corresponding fragments with designated activity.

Adapters are data holders which provide data access and acts as a bridge between the fragments (a viewable component) and the data items. Adapters work on a collection of data items as well as single data item which contributes to a viewable component.

Navigator makes use of listener pattern which allows asynchronous callbacks when an event occurs. Volley API as discussed earlier, communicates with *CADsIoT Core* backend service in an asynchronous manner. Volley acting as a REST library sends requests to backend service and register appropriate listeners along-with callback functions. Upon receipt of HTTP response, parsing of data into appropriate model e.g., device, sensor, context and subscription is performed. This parsed data, in the forms of domain objects are passed to associate callbacks (implements as part of the listener pattern). These callbacks are responsible for loading data in corresponding adapters.

6.4 Validation

The section outlines the validation of proposed solution as discussed in the requirements section, Chapter 4. Two prototypes constitute to CADsIoT solution: *CADsIoT Core* and *Navigator* which needs to be validated. For simplification reasons, REST APIs provided by backend service and discovery service developed by *Navigator* are considered for validation as both correlates with the functional requirements. The complete details of REST endpoints are out of the scope of this section. Readers can refer to REST API documentation, Appendix A.

Registration

One of the requirement, as mentioned earlier, is to have a device registry, acting as the centralized repository where all devices and sensors are registered and maintained. Hence, CADsIoT Core provides registration module along with metadata information. Table 6.1 and 6.2 depicts the REST endpoints with respect to device and sensors. Additionally, listing 6.6 and 6.7 shows the response of HTTP requests.

Device URI – /devices			
	HTTP Method	**Message Body**	**Response Code**
Add Device	POST	Yes	201(Created)
Retrieve Device	GET	No	200(OK)
Update Device	PUT	Yes	201(Created)
Delete device	DELETE	No	204(No Content)

Table 6.1: Device Registration REST Interactions

```
GET /discover-iot/rest/api/v1/devices HTTP/1.1
Host: 127.0.0.1:8080
Content-Type: application/json
Cache-Control: no-cache

{
"id": 7,
"name": "Air Pollutant Device",
"description": "Air Pollutant Device",
"manufactureName": "Element Design",
"creationDate": "2016-11-23T12:32:047Z",
"modifiedDate": "2016-11-23T12:32:047Z",
"active": true,
"networkAddress": "192.168.5.8",
"contextInfo": {
      "id": 5,
      "name": "Trattoria Da Franco",
      "description": "Resturant",
      "streetName": "Adolf-Engster-Weg",
      "streetNo": "10",
      "city": "Stuttgart",
      "zipcode": "70569",
      "country": "Germany",
      "lat": 48.748605,
      "lon": 9.082285,
      "creationDate": "2016-11-23T14:33:018Z",
      "modifiedDate": "2016-11-23T14:33:018Z",
      "deviceId": null
      },
"sensors": []
}
```

Listing 6.6: Devices Request Response using GET

Sensor URI – /sensors			
	HTTP Method	**Message Body**	**Response Code**
Add Sensor	POST	Yes	201(Created)
Retrieve Sensor	GET	No	200(OK)
Update Sensor	PUT	Yes	201(Created)
Delete Sensor	DELETE	No	204(No Content)

Table 6.2: Sensor Registration REST Interactions

```
GET /discover-iot/rest/api/v1/sensors HTTP/1.1
Host: 127.0.0.1:8080
Content-Type: application/json
Cache-Control: no-cache

{
"id": 3,
"name": "Temperature III",
"description": "Temperature Sensor",
"manufactureName": "ABM Electric Ltd",
"creationDate": "2016-08-23T14:53:031Z",
"modifiedDate": "2016-08-23T14:53:031Z",
"active": false,
"location": "Building38/Floor1/Room24",
"category": "temperature",
"deviceId": null,
"subscriptionId": null,
"connectionType": "wired",
"topicId": "topicId",
"subscriptions": [
            {
            "id": 13,
            "name": "Bernd Wagner",
            "description": "Bernd Wagner",
            "email": "bernd.wagner@gmail.com",
            "creationDate": "2016-10-05T16:00:048Z",
            "modifiedDate": "2016-10-05T16:00:048Z",
            "sensorId": 3
            }
      ]
}
```

Listing 6.7: Sensors Request Response using GET

Attach Context

CADsIoT Core supports attaching context with devices and eventually with sensors. The context details primarily define the description of the place where devices are actually placed. It is assumed that sensors e.g., motion sensor, temperature sensor are attached to the particular device. Locating device context allows sensor positions to be located. CADsIoT provides partial update facility via HTTP POST to attached particular context with a device. Multiple different devices can have the same context. Assuming context is already created (ID=5) but it is not yet attached to any device. Listing 6.8 describes this behavior and shows how HTTP clients can attach context with devices.

```
POST /discover-iot/rest/api/v1/locations/5 HTTP/1.1
Host: 127.0.0.1:8080
Content-Type: application/json
Cache-Control: no-cache
Postman-Token: 71fcaece-e377-fd10-66a4-ab1524cc9b0d

{   "deviceId": 7 }
```

Listing 6.8: Attach Context using Partial Update

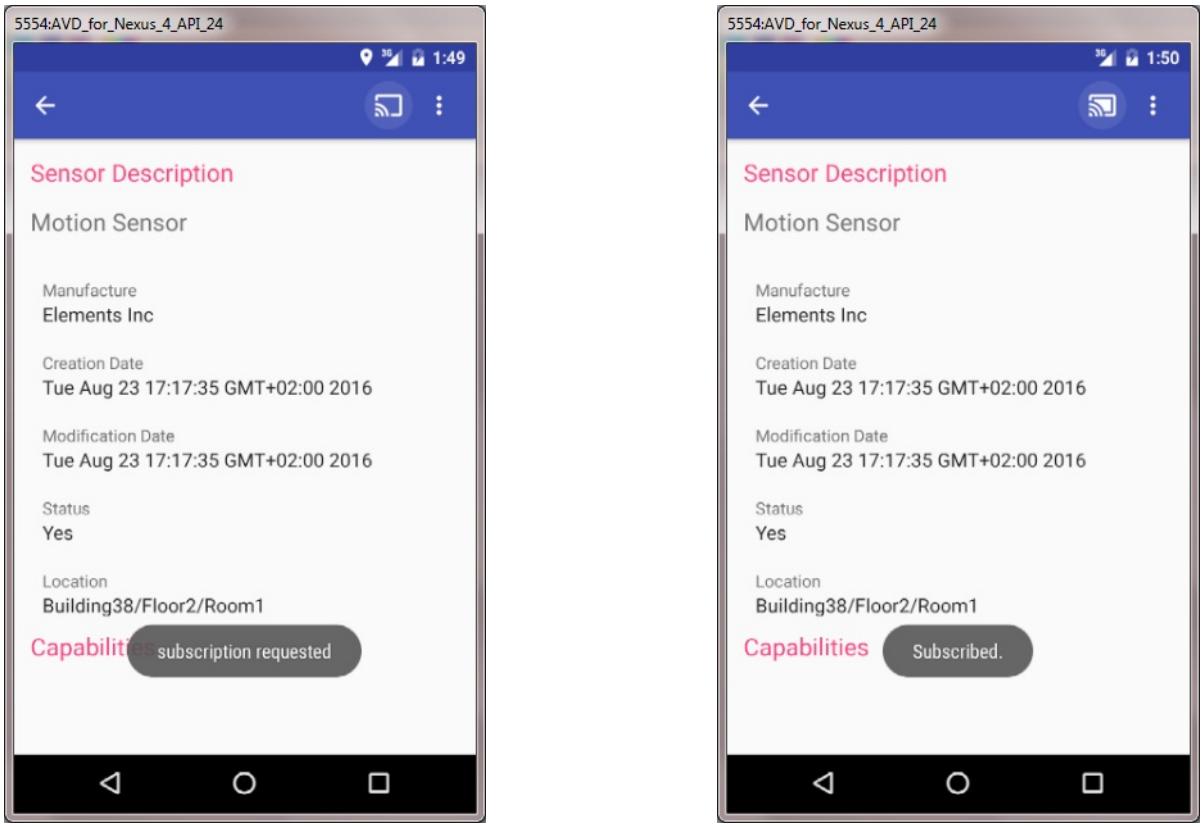
Subscribe Devices

Device and sensor subscription requirement is fulfilled by *CADsIoT Core* and *Navigator*. The subscription is basically a user object who is interested to be informed by certain devices. Subscription of devices and the discovery service communicates each other to enable locating devices automatically. CRUD operations can be performed on subscriptions via REST endpoints. Additionally, *Navigator* allows any device to be subscribed or unsubscribed. Listing 6.9 shows subscription request along with message body and figure 6.8 illustrates how the same operation can be performed by *Navigator*.

```
POST /discover-iot/rest/api/v1/subscribers HTTP/1.1
Host: 127.0.0.1:8080
Content-Type: application/json

{
"name": "Bernd Wagner",
"description": "Bernd Wagner",
"email": "bernd.wagner@gmail.com",
"creationDate": null,
"modifiedDate": null,
"sensorId": 4
}
```

Listing 6.9: Subsription Request



(a) Subscription Requested **(b)** Subscription Completed

Figure 6.8: Device and Sensor Subscription

Discover Devices

One of the core requirement and heart of this thesis is, how to discover devices based on context. The requirement is fulfilled by *CADsIoT Core* and *Navigator*. The backend service provides REST endpoints which give subscribed nearest devices and sensors using Harvestine formula [9]. On the other hand, *Navigator* utilizes geofencing approach as discussed in section 6.3.1. The discovery service implemented in *Navigator* enables users to locate devices when they are entering a region. The region is defined by latitudes, longitudes, and a radius. The service triggers real-time notifications when the user enters in a region and informs about the context (location) of the device.

[9] determines distance between two GPS coordinates https://en.wikipedia.org/wiki/Haversine_formula

```
GET /discover-iot/rest/api/v1/sensors?lat=48.745&amp;lon=9.1065 HTTP/1.1
Host: 127.0.0.1:8080
Content-Type: application/json
Cache-Control: no-cache
```

Listing 6.10: Nearest Devices

Listing 6.10 shows HTTP request for nearby sensors based on current location.

7 Discussion and Future Work

This chapter summaries the overall work done in this thesis, allowing readers to understand different aspects in a convenient manner. Later, a brief overview of possible future work and open challenges related to the proposed concept are also highlighted.

7.1 Discussion

This work has presented a concept on how heterogeneous IoT devices can be discovered based on the context with the help of possible scenarios and applications. As a result, two prototypes are designed and implemented. The work contributes to different aspects of the concepts and hence laid down in three parts, each of these are discussed below.

In the first part of the thesis, a background about the general concepts involved in Internet of Things has been discussed as a starting point. Herein, the motivation behind proposing context-aware discovery service followed by thesis objective, the evolution of Internet of Things and how cloud platforms and their extensions such as Fog Computing can help IoT by outlining possible application scenarios has been examined. Then, the state-of-the-art technologies has been reviewed, focusing on available middlewares and frameworks, communication standards and protocols. This extensive research of various literature and journals benefited to restructure the idea of context-aware discovery service by reviewing concepts, reusable design and enabling technologies, used in practice.

The second part of the thesis contributes to realize the proposed concept (CADsIoT) by defining requirements, motivational use case followed by technical design requirements. Herein, functional and non-functional requirements are defined emphasizing on the key specifications (adheres to the CADsIoT solution) rather than complete context-aware computing platform. However, the functional requirements are provided by the system. The work ensures that sensor data manipulation e.g., reading data from sensors is kept aside and not part of the thesis. However, relevant literature reviews dealing with sensor data are highlighted. Furthermore, a real life scenario is considered in the logistic industry with various illustrations, facilitating readers to understand the possible application of the concept.

In the last part of the thesis, the design and implementation aspects of CADsIoT comprises of CADsIoT Core, a backend service and Navigator, a smartphone application has been introduced. The required functionalities, layered architecture details for each component and the formulation of *CADsIoT Core* REST API design guidelines are explained. Later, technical details for both components, highlighting technologies and libraries used in order to realize the concept i.e., discovering devices based on context automatically. The explanations are backed up by code snippets, interaction diagrams, depicting the implementation of the concept, both at backend and client levels. Finally, the two prototypes are validated based on the functional requirements discussed in section 4.2 and associated REST endpoints.

All in all, talking about the world of devices where millions of devices will contribute to businesses growth, CADsIoT approach can play a significant role in discovering devices automatically, adhering to device availability based on the context.

7.2 Future Directions

This section briefly highlights current challenges and possible future directions which are closely related to the work presented in this research.

Context Discovery

Context based applications are gaining more and more attraction. As discussed in section 4, context describes the current situation of an object. CADsIoT leverages location as the primary category of context. However, an ideal context-aware application considers different levels of context e.g., who, where, when and what of objects to understand the change of current environment [SGA15b]. Combining such levels of context awareness in CADsIoT concept can significantly improve device discovery features. Sensor data is another important aspect to enrich IoT data with contextual information. Automatic tagging on such data according to different application domains provides a better understanding of sensor data but leads to a challenging task. The current development in semantic technologies and context-aware computing leads to future directions.

Security and Privacy

As Internet of Things connects several devices together, this gives a way for potential loopholes and entry points from where a malicious attack can be made. Current research efforts are not capable to deal with IoT security issues. Discovery of devices based on context is a cumbersome process i.e, securing all the components involved in the discovery process. Security model should be made at each level of participating entities dealing with authorization to discover devices, security threats to a network,

enabling user trust. Roman [RNL11] describes certain security and privacy aspects in IoT environments. Data security is another important direction to protect devices from vulnerabilities. Devices in IoT are supposed to obtain and share a huge amount of sensitive data and this is where, the main security risk arises. Most of the IoT devices uses the normal public internet, we can visualize the risk of sensitive data flowing around and doubling the risk for consumers and providers.

Standardization

Internet of Things ecosystem involves several enabling technologies. The integration of these technologies opens new doors to the researchers. Multiple contributions has been made to bring up standards in order to realize IoT paradigm. However, there is still a necessity to bring architectures, standards and APIs to interconnect smart objects which are heterogeneous in nature. By now, most of the smart objects are connected through web interfaces provided by cloud community which increases the rapid development process but at the same time, introduces network load, latency and data processing overhead. Enabling efficient machine-to-machine communications will reduce additional overheads.

A CADsIoT Core REST API Documentation

Base URL : http://127.0.0.1:8080/discover-iot/rest/api/v1/

A.1 Device Resource

	URI	Method	Response Code	Description
1	/devices	GET	200 (OK)	Returns list of all available devices
2	/devices/{id}	GET	200 (OK)	Returns device based on the id
3	/devices	POST	201 (CREATED)	Add new device in the collection of devices
4	/devices/{id}	PUT	201 (CREATED)	Full Update - Modify device (all properties required) in the data store based on the id
5	/devices/{id}	POST	200 (OK)	Partial Update - Modify some properties (no all) of device
6	/devices/{id}	DELETE	204 (NO CONTENT)	Delete device from the data store
7	/devices?subscriber=<user_email>	GET	200 (OK)	Returns all devices which user has subscribed to

Table A.1: Devices REST Endpoints

A.1.1 Device- Show All devices

```
o     URL: /devices
o     Method: GET
o     URL Params: None
o     Data Params (Payload): No
o     Success Response:
      o     Code: 200
      o     Content: {
                  "id": 1,
                  "name": "Video Surveillance",
                  "description": "Microcontroller device for Video Surveillanc",
                  "manufactureName": "Element Design",
                  "creationDate": "2017-01-04T11:34:051Z",
                  "modifiedDate": "2017-01-04T11:34:051Z",
                  "active": true,
                  "networkAddress": "192.168.0.6",
                  "contextInfo": null,
                  "sensors": []
                  }
o     Error Response: None
o     Sample Call:
            GET /discover-iot/rest/api/v1/devices HTTP/1.1
            Host: 127.0.0.1:8080
            Content-Type: application/json
```

Listing A.1: Show ALL Devices

A.1.2 Device- Show Device

```
o     URL: /devices/{id}
o     Method: GET
o     URL Params:
      o     Required: id=[integer]
o     Data Params (Payload): None
o     Success Response:
      o     Code: 200
      o     Content: {
                  "id": 1,
                  "name": "Video Surveillance",
                  "description": "Microcontroller device for Video  Surveillanc",
                  "manufactureName": "Element Design",
                  "creationDate": "2017-01-04T11:34:051Z",
                  "modifiedDate": "2017-01-04T11:34:051Z",
                  "active": true,
                  "networkAddress": "192.168.0.6",
                  "contextInfo": null,
                  "sensors": []
                  }
o     Error Response:
      o     Code: 404
      o     Content: {
                  "status": 404,
                  "code": 404,
                  "message": "The device you requested with id 15 was not found in the
                      database",
                  "link": null,
                  "developerMessage": "Verify the existence of the Device with the id 15
                      in the database"
                  }
o     Sample Call:
      GET /discover-iot/rest/api/v1/devices/1 HTTP/1.1
      Host: 127.0.0.1:8080
      Cache-Control: no-cache
```

Listing A.2: Show Device

A.1.3 Device- Add new Device

```
o     URL: /devices
o     Method: POST
o     URL Params: None
o     Data Params (Payload):
      o     Payload: {
                  "name": "Video Surveillance",
                  "description": "Microcontroller device for Video Surveillanc",
                  "manufactureName": "Element Design",
                  "active": true,
                  "networkAddress": "192.168.0.6"
                  }
o     Success Response:
      o     Code: 201
      o     Content: {
                  "id": 1,
                  "name": "Video Surveillance",
                  "description": "Microcontroller device for Video  Surveillanc",
                  "manufactureName": "Element Design",
                  "creationDate": "2017-01-04T11:34:051Z",
                  "modifiedDate": "2017-01-04T11:34:051Z",
                  "active": true,
                  "networkAddress": "192.168.0.6",
                  "contextInfo": null,
                  "sensors": []
      }
      o     Header: Location -- http://127.0.0.1:8080/discover-iot/rest/api/v1/devices/1
o     Error Response: None
o     Sample Call:
      POST /discover-iot/rest/api/v1/devices HTTP/1.1
      Host: 127.0.0.1:8080
      Content-Type: application/json
      Cache-Control: no-cache
```

Listing A.3: Add New Device

A.1.4 Device- Modify Device

```
o     URL: /devices/{id}
o     Method: PUT
o     URL Params:
      o Required: id=[integer]
o     Data Params (Payload):
o     Payload: {
                  "name": "Video Surveillance",
                  "description": "Microcontroller device for Video Surveillance",
                  "manufactureName": "Element Design",
                  "active": true,
                  "networkAddress": "192.168.0.6"
                  }
o     Success Response:
      o     Code: 201
      o     Content: {"message": "Device has been updated"}
      o     Header: Location -- http://127.0.0.1:8080/discover-iot/rest/api/v1/devices/1
o     Error Response:
      o     Code: 404
      o     Content: {
                  "status": 404,
                  "code": 404,
                  "message": "The device you requested with id 2 was not found in the
                      database",
                  "link": null,
                  "developerMessage": "Verify the existence of the Device with the id 2
                      in the database"
                  }
o     Sample Call:
      PUT /discover-iot/rest/api/v1/devices/2 HTTP/1.1
      Host: 127.0.0.1:8080
      Content-Type: application/json
```

Listing A.4: Modify Device

A.1.5 Device- Partial Modify Device

```
o     URL: /devices/{id}
o     Method: POST
o     URL Params:
      o Required: id=[integer]
o     Data Params (Payload):
o     Payload: {
                  "active": true,
                  "networkAddress": "192.168.0.2"
                  }
o     Success Response:
      o     Code: 200
      o     Content: {"message": "Device has been successfully updated" }
o     Error Response:
            o     Code: 404
            o     Content: {
                        "status": 404,
                        "code": 404,
                        "message": "The device you requested with id 2 was not found in
                            the database",
                        "link": null,
                        "developerMessage": "Verify the existence of the Device with the
                            id 2 in the database"
                        }
o     Sample Call:
      POST /discover-iot/rest/api/v1/devices/1 HTTP/1.1
      Host: 127.0.0.1:8080
      Content-Type: application/json
```

Listing A.5: Partial update Device

A.1.6 Device- Delete Device

```
o     URL: /devices/{id}
o     Method: DELETE
o     URL Params:
      o Required: id=[integer]
o     Data Params (Payload): None
o     Success Response:
      o     Code: 204
o     Error Response:
      o     Code: 404
      o     Content: {
                  "status": 404,
                  "code": 404,
                  "message": "The device you requested with id 2 was not found in the
                      database",
                  "link": null,
                  "developerMessage": "Verify the existence of the Device with the id 2
                      in the database"
                  }
o     Sample Call:
      DELETE /discover-iot/rest/api/v1/devices/1 HTTP/1.1
      Host: 127.0.0.1:8080
      Content-Type: application/json
```

Listing A.6: Delete Device

A.2 Sensor Resource

	URI	Method	Response Code	Description
1	/sensors	GET	200 (OK)	Returns list of all available sensors
2	/sensors/{id}	GET	200 (OK)	Returns sensor based on the id
3	/sensors	POST	201 (CREATED)	Add new sensor in the collection of sensor
4	/sensors/{id}	PUT	201 (CREATED)	Full Update - Modify sensor (all properties required) in the data store based on the id
5	/sensors/{id}	POST	200 (OK)	Partial Update - Modify some properties (no all) of sensor
6	/sensors/{id}	DELETE	204 (NO CONTENT)	Delete sensor from the data store
7	/sensors?subscriber =<user_email>	GET	200 (OK)	Returns all sensors which user has subscribed to
8	/sensors?subscriber =<user_email>&lat= <lat>&lon<lon>	GET	200 (OK)	Returns all nearest sensors based on current latitudes and longitudes

Table A.2: Sensors REST Endpoints

A.2.1 Sensor- Show All sensors

```
o     URL: /sensors
o     Method: GET
o     URL Params: None
o     Data Params (Payload): No
o     Success Response:
      o     Code: 200
      o     Content: {
                  "id": 1,
                  "name": "Humidity Sensor",
                  "description": "Humidity Sensor (multi purpose)",
                  "manufactureName": "ABM Electric",
                  "creationDate": "2017-01-04T14:34:043Z",
                  "modifiedDate": "2017-01-04T14:34:043Z",
                  "active": true,
                  "location": "Building38/Floor1/Room23",
                  "category": "temperature",
                  "deviceId": null,
                  "connectionType": "wired",
                  "topicId": "demoTopic",
                  "subscriptions": []
                  }
o     Error Response: None
o     Sample Call:
      GET /discover-iot/rest/api/v1/sensors HTTP/1.1
      Host: 127.0.0.1:8080
      Content-Type: application/json
```

Listing A.7: Show ALL sensor

A.2.2 Sensor- Show Sensor

```
o     URL: /sensors/{id}
o     Method: GET
o     URL Params:
      o     Required: id=[integer]
o     Data Params (Payload): None
o     Success Response:
      o     Code: 200
      o     Content: {
                  "id": 1,
                  "name": "Humidity Sensor",
                  "description": "Humidity Sensor (multi purpose)",
                  "manufactureName": "ABM Electric",
                  "creationDate": "2017-01-04T14:34:043Z",
                  "modifiedDate": "2017-01-04T14:34:043Z",
                  "active": true,
                  "location": "Building38/Floor1/Room23",
                  "category": "temperature",
                  "deviceId": null,
                  "connectionType": "wired",
                  "topicId": "demoTopic",
                  "subscriptions": []
                  }
o     Error Response:
      o     Code: 404
      o     Content: {
                  "status": 404,
                  "code": 404,
                  "message": "The sensor you requested with id 15 was not found in the
                      database",
                  "link": null,
                  "developerMessage": "Verify the existence of the sensor with the id 15
                      in the database"
                  }
o     Sample Call:
      GET /discover-iot/rest/api/v1/sensors/1 HTTP/1.1
      Host: 127.0.0.1:8080
      Cache-Control: no-cache
```

Listing A.8: Show Sensor

A.2.3 Sensor- Add new Sensor

```
o     URL: /sensors
o     Method: POST
o     URL Params: None
o     Data Params (Payload):
      o     Payload: {
                  "name": "Humidity Sensor",
                  "description": "Humidity Sensor (multi purpose)",
                  "manufactureName": "ABM Electric",
                  "creationDate": null,
                  "modifiedDate": null,
                  "active": true,
                  "location": "Building38/Floor1/Room23",
                  "category": "temperature",
                  "connectionType": "wired",
                  "topicId": "demoTopic"
                  }
o     Success Response:
      o     Code: 201
      o     Content: {
                  "id": 1,
                  "name": "Humidity Sensor",
                  "description": "Humidity Sensor (multi purpose)",
                  "manufactureName": "ABM Electric",
                  "creationDate": "2017-01-04T14:34:043Z",
                  "modifiedDate": "2017-01-04T14:34:043Z",
                  "active": true,
                  "location": "Building38/Floor1/Room23",
                  "category": "temperature",
                  "deviceId": null,
                  "connectionType": "wired",
                  "topicId": "demoTopic",
                  "subscriptions": []
                  }
      o     Header: Location -- http://127.0.0.1:8080/discover-iot/rest/api/v1/sensors/1
o     Error Response: None
o     Sample Call:
      POST /discover-iot/rest/api/v1/sensors HTTP/1.1
      Host: 127.0.0.1:8080
      Content-Type: application/json
      Cache-Control: no-cache
```

Listing A.9: Add New Sensor

A.2.4 Sensor- Modify Sensor

```
o     URL: /sensors/{id}
o     Method: PUT
o     URL Params:
      o Required: id=[integer]
o     Data Params (Payload):
      o     Payload: {
                  "name": "Humidity Sensor",
                  "description": "Humidity Sensor (multi purpose)",
                  "manufactureName": "ABM Electric",
                  "creationDate": "2017-01-04T14:34:043Z",
                  "modifiedDate": "2017-01-04T14:34:043Z",
                  "active": true,
                  "location": "Building38/Floor1/Room1",
                  "category": "temperature",
                  "connectionType": "wired",
                  "topicId": "demoTopic",
                  }
o     Success Response:
      o     Code: 201
      o     Content: {"message": "Sensor has been updated"}
      o     Header: Location -- http://127.0.0.1:8080/discover-iot/rest/api/v1/sensors/1
o     Error Response:
      o     Code: 404
      o     Content: {
                  "status": 404,
                  "code": 404,
                  "message": "The sensor you requested with id {id} was not found in the
                      database",
                  "link": null,
                  "developerMessage": "Verify the existence of the sensor with the id
                      {id} in the database"
                  }
o     Sample Call:
      PUT /discover-iot/rest/api/v1/sensors/1 HTTP/1.1
      Host: 127.0.0.1:8080
      Content-Type: application/json
```

Listing A.10: Modify Sensor

A.2.5 Sensor- Partial Modify Sensor

- Updates sensor resource partially.
- Attach sensor to devices.

```
o     URL: /sensors/{id}
o     Method: POST
o     URL Params:
      o Required: id=[integer]
o     Data Params (Payload):
      o     Payload: { "deviceId": 1 }
o     Success Response:
      o     Code: 200
      o     Content: {"message": "sensor has been successfully updated"}
o     Error Response:
      o     Code: 404
      o     Content: {
                  "status": 404,
                  "code": 404,
                  "message": "The sensor you requested with id {id} was not found in the
                      database",
                  "link": null,
                  "developerMessage": "Verify the existence of the sensor with the id
                      {id} in the database"
                  }
o     Sample Call:
      POST /discover-iot/rest/api/v1/sensors/1 HTTP/1.1
      Host: 127.0.0.1:8080
      Content-Type: application/json
```

Listing A.11: Partial Update Sensor

A.2.6 Sensor- Delete Sensor

```
o     URL: /sensors/{id}
o     Method: DELETE
o     URL Params:
      o Required: id=[integer]
o     Data Params (Payload): None
o     Success Response:
      o     Code: 204
o     Error Response:
      o     Code: 404
      o     Content: {
            "status": 404,
            "code": 404,
            "message": "The sensor you requested with id 2 was not found in the database",
            "link": null,
            "developerMessage": "Verify the existence of the sensor with the id 2 in the
                database"
            }
o     Sample Call:
      DELETE /discover-iot/rest/api/v1/sensors/1 HTTP/1.1
      Host: 127.0.0.1:8080
      Content-Type: application/json
```

Listing A.12: Delete Sensor update Device

A.2.7 Sensor- Show Subscribed Sensors

```
o     URL: /sensors?subscriber={email_address}
o     Method: GET
o     URL Params:
      o Required: subscribers=[email_address]
o     Data Params (Payload): None
o     Success Response:
      o     Code: 200
      o     Content : [
                  {
                  "id": 1,
                  "name": "Humidity Sensor",
                  "description": "Humidity Sensor (multi purpose)",
                  "manufactureName": "Texas Instruments",
                  "creationDate": "2017-01-04T14:34:043Z",
                  "modifiedDate": "2017-01-04T15:30:059Z",
                  "active": false,
                  "location": "Building9/Floor1/Room1",
                  "category": "temperature",
                  "deviceId": 1,
                  "subscriptionId": null,
                  "connectionType": "wired",
                  "topicId": "topicId",
                  "subscriptions": [
                                    {
                                    "id": 1,
                                    "name": "Bernd Wagner ",
                                    "description": "Bernd Wagner (Stuttgart)",
                                    "email": "bernd.wagner@gmail.com",
                                    "creationDate": "2017-01-04T16:25:004Z",
                                    "modifiedDate": "2017-01-04T16:29:011Z",
                                    "sensorId": 1
                                    }
                  ]
                  }
            ]
o     Error Response: None
o     Sample Call:
      GET /discover-iot/rest/api/v1/sensors?subscriber=bernd.wagner@gmail.com HTTP/1.1
      Host: 127.0.0.1:8080
      Content-Type: application/json
```

Listing A.13: Show Subscribed Sensors

A.2.8 Sensor- Show Nearest Sensors

```
o     URL: /sensors?subscriber={email_address}&lat={lat}&lon={lon}
o     Method: GET
o     URL Params:
      o Required: subscribers=[email_address] ,
      o Optional : lat=[double], lon=[double], this gives all nereast (subscribed) senors
          within 1km region
o     Data Params (Payload): None
o     Success Response:
      o     Code: 200
      o     Content : [
                  {
                  "id": 1,
                  "name": "Humidity Sensor",
                  "description": "Humidity Sensor (multi purpose)",
                  "manufactureName": "Texas Instruments",
                  "creationDate": "2017-01-04T14:34:043Z",
                  "modifiedDate": "2017-01-04T15:30:059Z",
                  "active": false,
                  "location": "Building9/Floor1/Room1",
                  "category": "temperature",
                  "deviceId": 1,
                  "subscriptionId": null,
                  "connectionType": "wired",
                  "topicId": "topicId",
                  "subscriptions": [
                              {
                              "id": 1,
                              "name": "Bernd Wagner ",
                              "description": "Bernd Wagner (Stuttgart)",
                              "email": "bernd.wagner@gmail.com",
                              "creationDate": "2017-01-04T16:25:004Z",
                              "modifiedDate": "2017-01-04T16:29:011Z",
                              "sensorId": 1
                              }
                        ]
                  }
            ]
o     Error Response: None
o     Sample Call:
      GET /discover-iot/rest/api/v1/sensors?subscriber=bernd.wagner@gmail.com
            &lat=48.744552&lon=9.106863 HTTP/1.1
      Host: 127.0.0.1:8080
      Content-Type: application/json
```

Listing A.14: Show Nearest Sensors based on current location

A.3 Context Resource

	URI	Method	Response Code	Description
1	/locations	GET	200 (OK)	Returns list of all available context
2	/locations/{id}	GET	200 (OK)	Returns context based on the id
3	/locations	POST	201 (CREATED)	Add new context in the collection of context
4	/locations/{id}	PUT	201 (CREATED)	Full Update - Modify context (all properties required) in the data store based on the id
5	/locations/{id}	POST	200 (OK)	Partial Update - Modify some properties (no all) of context
6	/locations/{id}	DELETE	204 (NO CONTENT)	Delete context from the data store

Table A.3: Contexts REST Endpoints

A.3.1 Context- Show All Contexts

```
o     URL: /locations
o     Method: GET
o     URL Params: None
o     Data Params (Payload): No
o     Success Response:
      o     Code: 200
      o     Content: {
                  "id": 1,
                  "name": "Trattoria Da Franco",
                  "description": "Hotel",
                  "streetName": "Adolf-Engster-Weg",
                  "streetNo": "10",
                  "city": "Stuttgart",
                  "zipcode": "70569",
                  "country": "Germany",
                  "lat": 48.748605,
                  "lon": 9.082285,
                  "creationDate": "2017-01-04T16:03:023Z",
                  "modifiedDate": "2017-01-04T16:03:023Z",
                  "deviceId": null
                  }
o     Error Response: None
o     Sample Call:
      GET /discover-iot/rest/api/v1/locations HTTP/1.1
      Host: 127.0.0.1:8080
      Content-Type: application/json
```

Listing A.15: Show ALL Context

A.3.2 Context- Show Context

```
o     URL: /locations/{id}
o     Method: GET
o     URL Params:
      o     Required: id=[integer]
o     Data Params (Payload): None
o     Success Response:
      o     Code: 200
      o     Content: {
                  "id": 1,
                  "name": "Trattoria Da Franco",
                  "description": "Hotel",
                  "streetName": "Adolf-Engster-Weg",
                  "streetNo": "10",
                  "city": "Stuttgart",
                  "zipcode": "70569",
                  "country": "Germany",
                  "lat": 48.748605,
                  "lon": 9.082285,
                  "creationDate": "2017-01-04T16:03:023Z",
                  "modifiedDate": "2017-01-04T16:03:023Z",
                  "deviceId": null
                  }
o     Error Response:
      o     Code: 404
      o     Content: {
                  "status": 404,
                  "code": 404,
                  "message": "The ContextInfo you requested with id {id} was not found in
                      the database",
                  "link": null,
                  "developerMessage": "Verify the existence of the ContextInfo with the
                      id {id} in the database"
                  }
o     Sample Call:
      GET /discover-iot/rest/api/v1/locations/1 HTTP/1.1
      Host: 127.0.0.1:8080
      Cache-Control: no-cache
```

Listing A.16: Show Context

A.3.3 Context- Add New Context

```
o     URL: /locations
o     Method: POST
o     URL Params: None
o     Data Params (Payload):
      o     Payload: {
                  "name": "Trattoria Da Franco",
                  "description": "Hotel",
                  "streetName": "Adolf-Engster-Weg",
                  "streetNo": "10",
                  "city": "Stuttgart",
                  "zipcode": "70569",
                  "country": "Germany",
                  "lat": 48.748605,
                  "lon": 9.082285,
                  "creationDate": null,
                  "modifiedDate": null,
                  }
o     Success Response:
      o     Code: 201
      o     Content: {
                  "id": 1,
                  "name": "Trattoria Da Franco",
                  "description": "Hotel",
                  "streetName": "Adolf-Engster-Weg",
                  "streetNo": "10",
                  "city": "Stuttgart",
                  "zipcode": "70569",
                  "country": "Germany",
                  "lat": 48.748605,
                  "lon": 9.082285,
                  "creationDate": "2017-01-04T16:03:023Z",
                  "modifiedDate": "2017-01-04T16:03:023Z",
                  "deviceId": null
                  }
      o     Header: Location -- http://127.0.0.1:8080/discover-iot/rest/api/v1/locations/1
o     Error Response: None
o     Sample Call:
      POST /discover-iot/rest/api/v1/locations HTTP/1.1
      Host: 127.0.0.1:8080
      Content-Type: application/json
```

Listing A.17: Add New Context

A.3.4 Context- Modify Context

```
o     URL: /locations/{id}
o     Method: PUT
o     URL Params:
      o Required: id=[integer]
o     Data Params (Payload):
      o     Payload: {
                  "name": "Trattoria Da Franco",
                  "description": "Comnunity Hotel",
                  "streetName": "Adolf-Engster-Weg",
                  "streetNo": "10",
                  "city": "Stuttgart",
                  "zipcode": "70569",
                  "country": "Germany",
                  "lat": 48.748605,
                  "lon": 9.082285,
                  "creationDate": null,
                  "modifiedDate": null
                  }
o     Success Response:
      o     Code: 201
      o     Content: {"message": "context Info has been updated"}
o     Error Response:
      o     Code: 404
      o     Content: {
                  "status": 404,
                  "code": 404,
                  "message": "The ContextInfo you requested with id {id} was not found in
                      the database",
                  "link": null,
                  "developerMessage": "Verify the existence of the ContextInfo with the
                      id {id} in the database"
                  }
o     Sample Call:
      PUT /discover-iot/rest/api/v1/locations/1 HTTP/1.1
      Host: 127.0.0.1:8080
      Content-Type: application/json
```

Listing A.18: Modify Context

A.3.5 Context- Partial Modify Context

- Updates context resource partially.
- Attach context to devices.

```
o     URL: /locations/{id}
o     Method: POST
o     URL Params:
      o Required: id=[integer]
o     Data Params (Payload):
      o     Payload: {
                  "description" : "Community Hotel Stuttgart",
                  "deviceId": 1
                   }
o     Success Response:
      o     Code: 200
      o     Content: {"message": "Context Info has been successfully updated"}
o     Error Response:
      o     Code: 404
      o     Content: {
                  "status": 404,
                  "code": 404,
                  "message": "The ContextInfo you requested with id {id} was not found in
                      the database",
                  "link": null,
                  "developerMessage": "Verify the existence of the ContextInfo with the
                      id {id} in the database"
                  }
o     Sample Call:
      POST /discover-iot/rest/api/v1/locations/1 HTTP/1.1
      Host: 127.0.0.1:8080
      Content-Type: application/json
```

Listing A.19: Partial Update Context

A.3.6 Context- Delete Context

```
o     URL: /locations/{id}
o     Method: DELETE
o     URL Params:
      o Required: id=[integer]
o     Data Params (Payload): None
o     Success Response:
      o     Code: 204
o     Error Response:
      o     Code: 404
      o     Content: {
                  "status": 404,
                  "code": 404,
                  "message": "The context you requested with id {id} was not found in the
                      database",
                  "link": null,
                  "developerMessage": "Verify the existence of the sensor with the id
                      {id} in the database"
                  }
o     Sample Call:
      DELETE /discover-iot/rest/api/v1/locations/1 HTTP/1.1
      Host: 127.0.0.1:8080
      Content-Type: application/json
```

Listing A.20: Delete Context

A.4 Subscription Resource

	URI	Method	Response Code	Description
1	/subscriptions	GET	200 (OK)	Returns list of all available subscriptions
2	/subscriptions /{id}	GET	200 (OK)	Returns subscription based on the id
3	/subscriptions	POST	201 (CREATED)	Add new subscription in the collection of subscription
4	/subscriptions /{id}	PUT	201 (CREATED)	Full Update - Modify subscription (all properties required) in the data store based on the id
5	/subscriptions /{id}	POST	200 (OK)	Partial Update - Modify some properties (no all) of subscription
6	/subscriptions /{id}	DELETE	204 (NO CONTENT)	Delete subscription from the data store
7	/subscriptions?subscriber=<user_email>	GET	200 (OK)	Returns all subscriptions which user has subscribed to

Table A.4: Subscriptions REST Endpoints

A.4.1 Subscription - Show All Subscriptions

```
o     URL: /subscriptions
o     Method: GET
o     URL Params: None
o     Data Params (Payload): No
o     Success Response:
      o     Code: 200
      o     Content: {
                  "id": 1,
                  "name": "Bernd Wagner",
                  "description": "Bernd Wagner",
                  "email": "bernd.wagner@gmail.com",
                  "creationDate": "2017-01-04T16:25:004Z",
                  "modifiedDate": "2017-01-04T16:25:004Z",
                  "sensorId": null
                  }
o     Error Response: None
o     Sample Call:
      GET /discover-iot/rest/api/v1/subscriptions HTTP/1.1
      Host: 127.0.0.1:8080
      Content-Type: application/json
```

Listing A.21: Show ALL Subscriptions

A.4.2 Subscription- Show Subscription

```
o     URL: /subscriptions/{id}
o     Method: GET
o     URL Params:
      o     Required: id=[integer]
o     Data Params (Payload): None
o     Success Response:
      o     Code: 200
      o     Content: {
                  "id": 1,
                  "name": "Bernd Wagner",
                  "description": "Bernd Wagner",
                  "email": "bernd.wagner@gmail.com",
                  "creationDate": "2017-01-04T16:25:004Z",
                  "modifiedDate": "2017-01-04T16:25:004Z",
                  "sensorId": null
                  }
o     Error Response:
      o     Code: 404
      o     Content: {
                  "status": 404,
                  "code": 404,
                  "message": "The subscriber you requested with id {id} was not found in
                      the database",
                  "link": null,
                  "developerMessage": "Verify the existence of the subscriber with the id
                      {id} in the database"
                  }
o     Sample Call:
      GET /discover-iot/rest/api/v1/subscriptions/1 HTTP/1.1
      Host: 127.0.0.1:8080
      Cache-Control: no-cache
```

Listing A.22: Show Subscription

A.4.3 Subscription- Add New Subscription

```
o     URL: /subscriptions
o     Method: POST
o     URL Params: None
o     Data Params (Payload):
      o     Payload: {
                  "name": "Bernd Wagner",
                  "description": "Bernd Wagner",
                  "email": "bernd.wagner@gmail.com",
                  "creationDate": null,
                  "modifiedDate": null
                  }
o     Success Response:
      o     Code: 201
      o     Content: {
                  "id": 1,
                  "name": "Bernd Wagner",
                  "description": "Bernd Wagner",
                  "email": "bernd.wagner@gmail.com",
                  "creationDate": "2017-01-04T16:25:004Z",
                  "modifiedDate": "2017-01-04T16:25:004Z",
                  "sensorId": null
                  }
      o     Header: Location --
          http://127.0.0.1:8080/discover-iot/rest/api/v1/subscriptions/1
o     Error Response: None
o     Sample Call:
      POST /discover-iot/rest/api/v1/subscriptions HTTP/1.1
      Host: 127.0.0.1:8080
      Content-Type: application/json
```

Listing A.23: Add New Subscription

A.4.4 Subscription- Modify Subscription

```
o     URL: /subscription/{id}
o     Method: PUT
o     URL Params:
      o Required: id=[integer]
o     Data Params (Payload):
      o     Payload: {
                  "name": "Bernd Wagner ",
                  "description": "Bernd Wagner (Stuttgart)",
                  "email": "bernd.wagner@gmail.com",
                  "creationDate": null,
                  "modifiedDate": null
                  }
o     Success Response:
      o     Code: 201
      o     Content: {"message": "Subscription 1 has been updated"}
o     Error Response:
      o     Code: 404
      o     Content: {
                  "status": 404,
                  "code": 404,
                  "message": "The subscriber you requested with id {id} was not found in
                      the database",
                  "link": null,
                  "developerMessage": "Verify the existence of the subscriber with the id
                      {id} in the database"
                  }
      o     Header: {Location --
          http://127.0.0.1:8080/discover-iot/rest/api/v1/subscriptions/1
o     Sample Call:
      PUT /discover-iot/rest/api/v1/subscriptions/1 HTTP/1.1
      Host: 127.0.0.1:8080
      Content-Type: application/json
```

Listing A.24: Modify Subscription

A.4.5 Subscription- Partial Modify Subscription

- Updates subscription resource partially.
- Attach subscription to sensors.

```
o     URL: /subscriptions/{id}
o     Method: POST
o     URL Params:
      o Required: id=[integer]
o     Data Params (Payload):
      o     Payload: { "sensorId": 1 }
o     Success Response:
      o     Code: 200
      o     Content: {"message": "subscription 1 has been successfully updated"}
o     Error Response:
      o     Code: 404
      o     Content: {
                  "status": 404,
                  "code": 404,
                  "message": "The subscriber you requested with id {id} was not found in
                      the database",
                  "link": null,
                  "developerMessage": "Verify the existence of the subscriber with the id
                      {id} in the database"
                  }
o     Sample Call:
      POST /discover-iot/rest/api/v1/subscriptions/1 HTTP/1.1
      Host: 127.0.0.1:8080
      Content-Type: application/json
```

Listing A.25: Partial Update Subscription

A.4.6 Subscription- Delete Subscription

```
o     URL: /subscriptions/{id}
o     Method: DELETE
o     URL Params:
      o Required: id=[integer]
o     Data Params (Payload): None
o     Success Response:
      o     Code: 204
o     Error Response:
      o     Code: 404
      o     Content: {
                  "status": 404,
                  "code": 404,
                  "message": "The resource you requested with id {id} was not found in
                      the database",
                  "link": null,
                  "developerMessage": "Please verify existence of data in the database
                      for the id - {id}""
                  }
o     Sample Call:
      DELETE /discover-iot/rest/api/v1/subscriptions/1 HTTP/1.1
      Host: 127.0.0.1:8080
      Content-Type: application/json
```

Listing A.26: Delete Subscription

A.4.7 Subscription- Show All Subscriptions by subscriber

```
o     URL: /subscriptions?subscriber={email_address}
o     Method: GET
o     URL Params:
      o Required: subscribers=[email_address]
o     Data Params (Payload): None
o     Success Response:
      o     Code: 200
      o     Content : {
                  "id": 1,
                  "name": "Bernd Wagner ",
                  "description": "Bernd Wagner (Stuttgart)",
                  "email": "bernd.wagner@gmail.com",
                  "creationDate": "2017-01-04T16:25:004Z",
                  "modifiedDate": "2017-01-04T16:29:011Z",
                  "sensorId": 1
                  }
o     Error Response: None
o     Sample Call:
      GET /discover-iot/rest/api/v1/subscriptions?subscriber=bernd.wagner@gmail.com
          HTTP/1.1
      Host: 127.0.0.1:8080
      Content-Type: application/json
```

Listing A.27: Show All Subscriptions By Subscriber

List of Abbreviations

Abbreviation	Meaning	First occurrence
AAL	Ambient Assistant Living	41
BLOB	Binary Large Object	55
CADsIoT	Context-Aware Discovery Service for Internet of Things	3
CC	Cloud Computing	15
CoAp	Constrained Application Protocol	34
ERD	Entity Relationship Diagram	58
GPS	Global Positioning System	16
GSN	Global Sensor Network	27
HMI	Human-Machine-Interaction	24
HTTP	HyperText Transport Protocol	30
IETF	Internet Engineering Task Force	34
IoT	Internet of Things	3
IP	Internet Protocol	22
JPA	Java Persistence API	80
JSON	Javascript Object Notation	35
LBS	Location Based Services	57
LSM	Linked Sensor Middleware	29
M2M	Machine-to-Machine	15
MQTT	Message Queue Telemetry Transport	30
NIST	National Institute of Standards and Technology	21
ORM	Object/Relational Mapping	76
OSI	Open Systems Interconnection	21
POJOs	Plain Old Java Objects	80
REST	Representational State Transfer	3
RFID	Radio Frequency Identification Device	15
SDK	Software Development Kit	31
URI	Universal Resource Identifier	36
URL	Universal Resource Locator	65

(continued)

Abbreviation	Meaning	First occurrence
VSM	Virtual Sensor Manager	28
XML	Extensible Markup Language	35
XMPP	Extensible Messaging and Presence Protocol	34

Bibliography

[AG-EPoSS08] G. H. A. Bassi. *Internet of Things in 2020: A Roadmap for the future*. 2008. URL: http://www.sztaki.hu/~pbakonyi/bme/kieg/Internet-of-Things_in_2020_EC-EPoSS_Workshop_Report_2008_v3.pdf (visited on 08/13/2016) (cit. on p. 19).

[AGD99] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, P. Steggles. "Towards a better understanding of context and context-awareness." In: *International Symposium on Handheld and Ubiquitous Computing*. Springer Berlin Heidelberg. 1999, pp. 304–307 (cit. on p. 40).

[AIA10] L. Atzori, A. Iera, G. Morabito. "The Internet of Things: A Survey." In: *Comput. Netw.* 54.15 (Oct. 2010), pp. 2787–2805. ISSN: 1389-1286. DOI: 10.1016/j.comnet.2010.05.010. URL: http://dx.doi.org/10.1016/j.comnet.2010.05.010 (cit. on p. 19).

[AKA14] M. Aazam, I. Khan, A. A. Alsaffar, E. N. Huh. "Cloud of Things: Integrating Internet of Things and cloud computing and the issues involved." In: *Proceedings of 2014 11th International Bhurban Conference on Applied Sciences Technology (IBCAST) Islamabad, Pakistan, 14th - 18th January, 2014*. Jan. 2014, pp. 414–419. DOI: 10.1109/IBCAST.2014.6778179 (cit. on pp. 16, 20).

[AKM06] K. Aberer, M. Hauswirth, A. Salehi. "A Middleware for Fast and Flexible Sensor Network Deployment." In: *Proceedings of the 32Nd International Conference on Very Large Data Bases*. VLDB '06. Seoul, Korea: VLDB Endowment, 2006, pp. 1199–1202. (Visited on 08/08/2016) (cit. on pp. 27, 28).

[BAJ16] A. Bhatt, J. Patoliya. "Cost effective digitization of home appliances for home automation with low-power WiFi devices." In: *Advances in Electrical, Electronics, Information, Communication and Bio-Informatics (AEEICB), 2016 2nd International Conference on*. IEEE, 2016, pp. 643–648. URL: http://ieeexplore.ieee.org/abstract/document/7538368/ (visited on 12/04/2016) (cit. on p. 40).

[BFR12] F. Bonomi, R. Milito, J. Zhu, S. Addepalli. "Fog Computing and Its Role in the Internet of Things." In: *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*. MCC '12. New York, NY, USA: ACM, 2012, pp. 13–16. ISBN: 978-1-4503-1519-7. DOI: 10.1145/2342509.2342513. URL: http://doi.acm.org/10.1145/2342509.2342513 (visited on 08/03/2016) (cit. on pp. 22–25).

[BP95] P. J. Brown. "The stick-e document: a framework for creating context-aware applications." In: *ELECTRONIC PUBLISHING-CHICHESTER-* 8 (1995), pp. 259–272. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.8.7472&rep=rep1&type=pdf (visited on 12/06/2016) (cit. on p. 39).

[BSM11] Soma Bandyopadhyay, Munmun Sengupta, Souvik Maiti, Subhajit Dutta. "Role Of Middleware For Internet Of Things: A Study." In: *International Journal of Computer Science & Engineering Survey* 2.3 (Aug. 2011), pp. 94–105. ISSN: 09763252. DOI: 10.5121/ijcses.2011.2307. URL: http://www.airccse.org/journal/ijcses/papers/0811cses07.pdf (visited on 12/04/2016) (cit. on p. 44).

[BTDA13] S. Bendel, T. Springer, D. Schuster, A. Schill, R. Ackermann, M. Ameling. "A service infrastructure for the Internet of Things based on XMPP." In: *2013 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*. Mar. 2013, pp. 385–388. DOI: 10.1109/PerComW.2013.6529522 (cit. on p. 35).

[CC-NIST11] G. Swenson. *Final Version of NIST Cloud Computing Definition Published*. Text. Oct. 2011. URL: https://www.nist.gov/news-events/news/2011/10/final-version-nist-cloud-computing-definition-published (visited on 09/27/2016) (cit. on p. 22).

[CJA14] J.-P. Calbimonte, S. Sarni, J. Eberle, K. Aberer. "XGSN: an open-source semantic sensing middleware for the web of things." In: 2014. URL: http://ceur-ws.org/Vol-1401/tc-ssn2014-complete.pdf#page=53 (visited on 08/08/2016) (cit. on p. 29).

[CoAp16] *CoAP — Constrained Application Protocol | Overview*. 2016. URL: http://coap.technology/ (visited on 08/10/2016) (cit. on p. 34).

[DAB01] A. K. Dey, G. D. Abowd, D. Salber. "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-aware Applications." In: *Hum.-Comput. Interact.* 16.2 (Dec. 2001), pp. 97–166. ISSN: 0737-0024. DOI: 10.1207/S15327051HCI16234_02. URL: http://dx.doi.org/10.1207/S15327051HCI16234_02 (cit. on p. 39).

[DHL-Cisco15] DHL, Cisco Consulting Services. “Internet of Things In Logistics: A Report.” In: (2015). URL: http://www.dpdhl.com/content/dam/dpdhl/presse/pdf/2015/DHLTrendReport_Internet_of_things.pdf (visited on 01/02/2017) (cit. on p. 44).

[FRJ-RFC14] R. Fielding, J. Reschke. “Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content.” In: (2014). URL: http://tools.ietf.org/html/rfc7231 (visited on 12/21/2016) (cit. on p. 69).

[FRJ14] R. Fielding, J. Reschke. “Hypertext transfer protocol (HTTP/1.1): Message syntax and routing.” In: (2014). URL: http://tools.ietf.org/html/rfc7230 (visited on 12/20/2016) (cit. on p. 67).

[FRT00] R. T. Fielding. “Architectural styles and the design of network-based software architectures.” PhD thesis. University of California, Irvine, 2000. URL: http://jpkc.fudan.edu.cn/picture/article/216/35/4b/22598d594e3d93239700ce79bce1/7ed3ec2a-03c2-49cb-8bf8-5a90ea42f523.pdf (visited on 12/12/2016) (cit. on p. 35).

[Gartner16] *Gartner: 21 Billion IoT Devices To Invade By 2020*. 2016. URL: http://www.informationweek.com/mobile/mobile-devices/gartner-21-billion-iot-devices-to-invade-by-2020/d/d-id/1323081 (visited on 08/06/2016) (cit. on p. 16).

[GSD15] S. R. Garzon, B. Deva. “Infrastructure-Assisted Geofencing: Proactive Location-Based Services with Thin Mobile Clients and Smart Servers.” In: IEEE, Mar. 2015, pp. 61–70. ISBN: 978-1-4799-8977-5. DOI: 10.1109/MobileCloud.2015.31. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7130870 (visited on 01/01/2017) (cit. on p. 57).

[HRP97] R. Hull, P. Neaves, J. Bedford-Roberts. “Towards situated computing.” In: *Wearable Computers, 1997. Digest of Papers., First International Symposium on*. IEEE, 1997, pp. 146–153. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=629931 (visited on 12/06/2016) (cit. on p. 39).

[ISE15] U. Isikdag. *Enhanced Building Information Models*. SpringerBriefs in Computer Science. Cham: Springer International Publishing, 2015. ISBN: 978-3-319-21824-3. URL: http://link.springer.com/10.1007/978-3-319-21825-0 (visited on 08/10/2016) (cit. on pp. 33, 34).

[JCIoT13] Jim Chase. *The Evolution of the Internet of Things*. White Paper. Texas Instruments, 2013. URL: http://www.ti.com/lit/ml/swrb028/swrb028.pdf (cit. on p. 20).

[JM16] J. McKendrick. *Fog Computing: a New IoT Architecture*. Web Resource. RTInsights, 2016. URL: https://www.rtinsights.com/what-is-fog-computing-open-consortium/ (cit. on p. 23).

[KCVA15] V. Karagiannis, P. Chatzimisios, F. Vázquez-Gallego, J. Alonso-Zárate. "A Survey on Application Layer Protocols for the Internet of Things." In: *Transaction on IoT and Cloud Computing* 1.1 (Jan. 2015). DOI: 10.5281/zenodo.51613. URL: http://dx.doi.org/10.5281/zenodo.51613 (cit. on p. 33).

[KD08] R. van Kranenburg, S. Dodson. *The Internet of Things: A Critique of Ambient Technology and the All-seeing Network of RFID*. Network notebooks. Institute of Network Cultures, 2008. ISBN: 9789078146063. URL: https://books.google.de/books?id=PilgkgEACAAJ (visited on 08/13/2016) (cit. on p. 19).

[KSR12] R. Khan, S. U. Khan, R. Zaheer, S. Khan. "Future Internet: The Internet of Things Architecture, Possible Applications and Key Challenges." In: IEEE, Dec. 2012, pp. 257–260. ISBN: 978-0-7695-4927-9. DOI: 10.1109/FIT.2012.53. URL: http://ieeexplore.ieee.org/document/6424332/ (visited on 01/01/2017) (cit. on p. 20).

[LGZ15] T. H. Luan, L. Gao, Z. Li, Y. Xiang, G. Wei, L. Sun. "Fog Computing: Focusing on Mobile Users at the Edge." In: *arXiv:1502.01815 [cs]* (Feb. 2015). arXiv: 1502.01815. URL: http://arxiv.org/abs/1502.01815 (visited on 08/03/2016) (cit. on p. 25).

[MM12] M. Massé. *REST API design rulebook: [designing consistent RESTful Web Service Interfaces]*. eng. Beijing: O'Reilly, 2012. ISBN: 978-1-4493-1050-9 (cit. on p. 66).

[MR97] R. Moats. *URN syntax*. Tech. rep. 1997. URL: https://www.rfc-editor.org/rfc/pdfrfc/rfc2141.txt.pdf (visited on 12/20/2016) (cit. on p. 65).

[PCL08] C. Pautasso, O. Zimmermann, F. Leymann. "Restful web services vs. big'web services: making the right architectural decision." In: *Proceedings of the 17th international conference on World Wide Web*. ACM, 2008, pp. 805–814. URL: http://dl.acm.org/citation.cfm?id=1367606 (visited on 12/12/2016) (cit. on p. 35).

[PI11] P. Parwekar. "From Internet of Things towards cloud of things." In: *2011 2nd International Conference on Computer and Communication Technology (ICCCT)*. Sept. 2011, pp. 329–333. DOI: 10.1109/ICCCT.2011.6075156 (cit. on p. 15).

[PZA14] C. Perera, A. Zaslavsky, P. Christen, D. Georgakopoulos. “Context Aware Computing for The Internet of Things: A Survey.” In: *IEEE Communications Surveys & Tutorials* 16.1 (2014), pp. 414–454. ISSN: 1553-877X. DOI: 10.1109/SURV.2013.042313.00197. URL: http://ieeexplore.ieee.org/document/6512846/ (visited on 12/05/2016) (cit. on p. 40).

[RFD09] F. Reclus, K. Drouard. “Geofencing for fleet & freight management.” In: *Intelligent Transport Systems Telecommunications,(ITST), 2009 9th International Conference on*. IEEE, 2009, pp. 353–356. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5399328 (visited on 12/26/2016) (cit. on p. 81).

[RNL11] R. Roman, P. Najera, J. Lopez. “Securing the internet of things.” In: *Computer* 44.9 (2011), pp. 51–58. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6017172 (visited on 01/01/2017) (cit. on p. 93).

[SGA15a] M. P. Silva, A. L. Goncalves, M. Dantas, B. Vanelli, G. Manerichi, S. A. d. Santos, M. Ferrandim, A. Pinto. “Implementation of IoT for Monitoring Ambient Air in Ubiquitous AAL Environments.” In: IEEE, Nov. 2015, pp. 158–161. ISBN: 978-1-5090-0182-8. DOI: 10.1109/SBESC.2015.37. URL: http://ieeexplore.ieee.org/document/7423232/ (visited on 12/04/2016) (cit. on p. 41).

[SGA15b] S. Sukode, S. Gite, H. Agrawal. “Context Aware Framework in IoT: A survey.” In: *International Journal* 4.1 (2015). URL: https://www.researchgate.net/profile/Shilpa_Gite/publication/273456830_International_Journal_of_Advanced_Trends_in_Computer_Science_and_Engineering/links/55090dba0cf26ff55f849128.pdf (visited on 01/01/2017) (cit. on p. 92).

[SGP10] H. Sundmaeker, P. Guillemin, P. Friess, S. Woelfflé, European Commission, Directorate-General for the Information Society and Media. *Vision and challenges for realising the Internet of things*. English. OCLC: 847355368. Luxembourg: EUR-OP, 2010. ISBN: 978-92-79-15088-3 (cit. on pp. 19, 39).

[SMD14] S. Mansfield-Devine. “Lack of security in Internet of Things devices.” In: *Network Security* 2014.8 (2014). ISSN: 1353-4858. DOI: http://dx.doi.org/10.1016/S1353-4858(14)70075-3. URL: http://www.sciencedirect.com/science/article/pii/S1353485814700753 (cit. on p. 43).

[TSJ14] S. S. Thakre, P. S. Jain. “Temperature Monitoring and Alert Generation System–An IoT Implementation.” In: (2014). URL: https://www.ijsr.net/archive/v4i11/NOV151294.pdf (visited on 12/04/2016) (cit. on p. 41).

[UZL11] B. Upadhyaya, Y. Zou, H. Xiao, J. Ng, A. Lau. “Migration of SOAP-based services to RESTful services.” In: *2011 13th IEEE International Symposium on Web Systems Evolution (WSE)*. Sept. 2011, pp. 105–114. DOI: 10.1109/WSE.2011.6081828 (cit. on p. 35).

[VFG11] O. Vermesan, P. Friess, P. Guillemin, S. Gusmeroli, H. Sundmaeker, A. Bassi, I. S. Jubert, M. Mazura, M. Harrison, M. Eisenhauer, et al. “Internet of things strategic research roadmap.” In: *O. Vermesan, P. Friess, P. Guillemin, S. Gusmeroli, H. Sundmaeker, A. Bassi, et al., Internet of Things: Global Technological and Societal Trends* 1 (2011), pp. 9–52. (Visited on 12/04/2016) (cit. on p. 43).

[WAH97] A. Ward, A. Jones, A. Hopper. “A new location technique for the active office.” In: *IEEE Personal Communications* 4.5 (1997), pp. 42–47. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=626982 (visited on 12/06/2016) (cit. on p. 39).

All links were last followed on Jan 2017.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature