A MESSAGE BROKER BASED PLATFORM AS A SERVICE INFRASTRUCTURE FOR CONTEXT-AWARE APPLICATION DEVELOPMENT

AYKUT GÜNER

JULY 2019

A MESSAGE BROKER BASED PLATFORM AS A SERVICE INFRASTRUCTURE FOR CONTEXT-AWARE APPLICATION DEVELOPMENT

A THESIS SUBMITTED TO

GRADUATE SCHOOL

IZMIR UNIVERSITY OF ECONOMICS

BY

AYKUT GÜNER

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

IN THE GRADUATE SCHOOL

JULY 2019

M.S. THESIS EXAMINATION RESULT FORM

Approval of the Graduate School

[illegible]

Prof. Dr. M. Efe Biresselioğlu
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

[illegible]

Prof. Dr. Cem Evrendilek
Head of Department

We have read the thesis entitled **"A Message Broker Based Platform as a Service Infrastructure for Context-Aware Application Development"** completed by **AYKUT GÜNER** under supervision of **Asst. Prof. Dr. Kaan Kurtel** and **Asst. Prof. Dr. Ufuk Çelikkan** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

[illegible]	[illegible]
Asst. Prof. Dr. Ufuk Çelikkan Co-Supervisor	Asst. Prof. Dr. Kaan Kurtel Supervisor

Examining Committee Members	**Date:** 01/08/2019
Asst. Prof. Dr. Kaan Kurtel Dept. of Software Engineering, IUE	[illegible]
Asst. Prof. Dr. Ufuk Çelikkan Dept. of Software Engineering, IUE	[illegible]
Assoc. Prof. Dr. Adil Alpkoçak Dept. of Computer Engineering, DEU	[illegible]
Assoc. Prof. Dr. Hüseyin Akcan Dept. of Software Engineering, IUE	[illegible]
Asst. Prof. Dr. Kamil Erkan Kabak Dept. of Industrial Engineering, IUE	[illegible]

ABSTRACT

A MESSAGE BROKER BASED PLATFORM AS A SERVICE INFRASTRUCTURE FOR CONTEXT-AWARE APPLICATION DEVELOPMENT

AYKUT GÜNER
M.S. in Computer Engineering
Graduate School
Supervisor: Asst. Prof. Dr. Kaan Kurtel
Co-Supervisor: Asst. Prof. Dr. Ufuk Çelikkan
July 2019

Context-awareness is a property which enables applications to adapt their operations into their environment without user intervention. Context-aware computing involves complex interactions between data providers and data requestors making context-aware software application development time consuming, difficult and costly. To ease context-aware application development, presence of a platform is very beneficial to handle the vital and mundane tasks that take place between data providers and data requestors such as storage and formatting of the context data.

The main contribution of this study is to propose and implement a service based infrastructure for Context Aware Computing. Our primary motivation building such a platform is to provide a reference implementation to software community that eases development of context aware applications. The platform, called as “A Platform for Context Aware Application Development-PCAD”, is service based and provides its functionality via a set of services namely, Context Modeling and Reasoning, Security, Rule, Data Management, Alarm and Notification, Transformation and Reporting Service. In the design and implementation of the platform, Node.js, NoSQL, MQTT, RESTful and several other toolkits are used. These frameworks and toolkits enable rapid and efficient development.

Keywords: Context Awareness, Context Aware Computing, Middleware, Publisher Subscriber, NoSQL, MQTT, RESTFul.

ÖZ

YAYINCI/ABONE MESAJLAŞMA TEKNİĞİNE DAYALI, DURUM FARKINDA UYGULAMALARIN GELİŞTİRİLMESİNE YÖNELİK SERVİS MİMARİSİNE SAHİP BİR PLATFORM

AYKUT GÜNER
Bilgisayar Mühendisliği, Yüksek Lisans
Lisansüstü Programlar Enstitüsü
Tez Danışmanı: Dr. Kaan Kurtel
İkinci Tez Danışmanı: Dr. Ufuk Çelikkan
Temmuz 2019

Durum farkındalık, uygulamaların, kullanıcı müdahalesi olmadan işlemlerini kendi ortamlarına uyarlamasını sağlayan bir niteliktir. Bilgi ve iletişim teknolojisinde son zamanlarda popüler olan durum farkındalık, veri sağlayıcıları ile veri istemcileri arasında karmaşık bir etkileşim göstermektedir. Bu da, durum farkında uygulama geliştirmeyi zaman alıcı, zor ve maliyetli hale getirmektedir. Veri sağlayıcıları ile veri istemcileri arasında bir platformun varlığı, taraflar arasındaki etkileşimin yönetilerek, uygulamalarının geliştirilmesini kolaylaştıracak ve durum verilerinin depolanması ve formatlanması gibi olağan fakat yapılması elzem görevlerin yerine getirilmesini sağlayacaktır.

Bu çalışmanın temel katkısı, durum farkında uygulamalar için servis tabanlı "*A Platform for Context Aware Application Development-PCAD*" olarak adlandırılan bir platform önermesi ve yazılım topluluğuna durum farkında uygulama geliştirmesini kolaylaştıran bir örnek sunmaktır. Platformun servisleri; Durum Modelleme ve Anlamlandırma, Güvenlik, Kural, Veri Yönetimi, Alarm ve Bildirim, Dönüşüm ve Raporlamadır. Tasarım ve uygulama sürecinde Node.js, NoSQL, MQTT, RESTful ve çeşitli kütüphaneler kullanılmıştır. Bunlar, PCAD'in hızlı ve verimli bir şekilde geliştirilmesinde yardımcı olmuştur.

Anahtar Kelimeler: Durum Farkındalık, Durum Farkında Hesaplama, Katmanlı Mimari, Yayımcı Abone, NoSQL, MQTT, RESTFul.

ACKNOWLEDGEMENT

Firstly, I would like to thank to my advisors, Asst. Prof. Dr. Ufuk Çelikkan and Asst. Prof. Dr. Kaan Kurtel for their patience, excellent guidance and encouragement.

This work is supported by Scientific and Technological Research Council of Turkey TÜBİTAK, Grant No: 114E938. Therefore, I would like to thank to TÜBİTAK.

I am very thankful to my wife Çisel Güner for her patience, my brother Aytuğ Güner for his support and my parents Hediye and Hayati Güner for their unconditional trust and encouragement all the time.

To my little boy*Alaz*...

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

Context-awareness is a relatively new approach in information and communications technology (ICT) offering unlimited application options. Basically, the whole idea that the context-awareness revolves around is that applications need to conceptually be aware of their presence and status and as well as be respondent to environmental stimuli. This situation affects business processes and causes radical changes. Internet of Things (IoT) applications and the technologies based on the fourth industrial revolution (Industry 4.0) are both at the center of this change.

The primary motivation of this thesis is to propose a message-broker based platform that employs a service based architecture to provide an infrastructure for developing context-aware applications. The platform fulfills the following requirements of the context-aware applications: efficient data exchange, security and privacy, context and data management, event based notification, real-time support(soft). The platform is designed to be extensible allowing new requirements to be added without making fundamental changes by playing the role of a middleware between data provider and data consumer actors. The platform increases data delivery speed and enables sensors to send their data easily and securely. Driven by these motivations, the platform called PCAD (A Platform for Context-Aware Application Development) is implemented with a focus on context-aware computing by using current technologies such as Node.js, NoSQL, MQTT and RESTful (Representational State Transfer) web-services.

The contribution of this thesis is to propose and implement a novel context-aware application development architecture based on publisher-subscriber pattern employing a service oriented paradigm. In addition, the ideas presented in this thesis may lead other researchers to develop other context-aware application architectures by integrating several state of the art technologies to provide a complete solution in this domain.

The thesis is organized as follows. Chapter 2 introduces terminology and discusses background information on context, context-awareness, context modeling, context delivery concepts, publisher-subscriber pattern and its implementation, MQTT. A review of similar architectures and frameworks are also given here. Chapter 3 explains the platform, its design and architectural aspects. Implementation details of the platform are given in Chapter 4. Implementation of each service and their interactions are stated in this chapter. Platform tests and their results are presented in Chapter 5. The thesis ends with the Conclusion Chapter.

Chapter 2

Background and Terminology

Plenty of research was conducted on context aware computing. Research show that the term context is understood tacitly by most of the people, however it is hard to define its exact meaning. In order to clarify and define the meaning of context, they mostly use synonyms or examples.

Schilit and Theimer [1] describe the term context as identities of nearby people and objects, their location and changes to those objects. Similarly, Brown et al. [2] define context as information, which can be location, the time of day, season of the year, temperature surrounding the user. Similar to Brown definition, Ryan et al. [3] define context in terms of the user's identity, environment, location and time. Dey [4] considers context as the user's emotional state, location, focus of attention, objects, time or date of the day, and people in the user's environment. According to Abowd et al. [5], defining context by examples is difficult to apply, and they want to decide whether a type of information not listed in the definition is context or not. In these circumstances, Dey defines context as follows: "*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*" [6].

The term context-aware is appeared in the study by Schilit and Theimer [1] for the first time. Schilit and Theimer define context awareness as the ability of a mobile user's applications to discover and react to changes in the environment they are situated in. Later, Ryan et al. [3] define it as follows: "*context awareness is the ability of the computer to sense and act upon information about its environment*". Hull et al. [7] had brought a similar definition.

Context aware applications need to adapt their actions to the current context without requiring user interventions. These applications provide more usable and effective way to their users by taking context into account. Abowd et al. [5] stated that "*previous definitions of context-aware computing fall into two categories: using context and adapting to context*". Explaining context awareness through "*context use* depends on how information such as location, temperature, identity etc. is used to provide more usable, effective and intelligent software or applications to users. Pascoe et al. [8] define context-aware computing as having the ability of detecting, sensing, interpreting and responding based on context obtained by sensors or devices. They use context to automate some of the fieldworker's activities such as automatically entering the user's location and let them to use it on an application. Rekimoto et al. [9] use context information to create more usable and intelligent augmented reality system. Their wearable system has a screen that contains context-aware panes and reflects user's current physical contexts: location and object. They feed these screen several sensors or devices like beacons and GPS devices. Salber et al. [10] define context awareness as follows: "*Context-aware computing aims to provide maximal flexibility of a computational service based on real-time sensing of any of these forms of context*". They categorize context to be physical, emotional, intentional or historical information about users or devices.

The term " *adapting to context* depends upon changes of a real time application or adaptations based on the application and user contexts. Schilit et al. [11] define context awareness as the application's or software's ability to adapt and react based on user's context attributes such as location, nearby people, and accessible devices. Another definition about adapting itself based on context is stated by Brown et al. [2]. They classify a context aware application as an application that changes its behavior according to the user's context. The application uses context for presenting more relevant outcome to the user. These definitions can not cover context awareness with all aspects. Dey et al.[5] state a more comprehensive definition 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.*"

2.1 Context Modeling and Reasoning

A context model is needed to define and store context data. Strang and Linnhoff-Popien [12] summarize the context modeling approaches, which are based on the data structures used for representing and exchanging contextual information in the respective system. After three years, Bolchini et al. [13] analyze and compare available context-aware data models.

A context model is needed to transform the existing context data into a form so it can be machine processable and one can reason about it. At this point, modeling a context provides a uniform representation of contexts to facilitate decision making, therefore, requests can be handled in a simple, expressive and generic manner. "Context modeling" enriches information, thereby, it allows development of quality applications. Baldouf et al. [14] summarize models that can be used for modeling context in the following:

- Key-Value models. It is the simplest structure for modeling contextual data. This model is used in various service frameworks in order to describe the capabilities of a service by using key-value pairs. However key-value models lack of ability to model sophisticated data according to [12].
- Markup scheme models. Markup schemes consist of hierarchical data structure. Tags are used to represent attributes and content. User Agent Profile (UAProf) [15], which is represented with RDF/S is an example of markup scheme model.
- Graphical models. Another well-known and commonly used model is the

Unified Modeling Language (UML). UML is also suitable for modeling context.

- Object oriented models. In computer science, object-oriented techniques are used in various fields. Using this technique in context modeling gives the ability to use power of object orientation (e.g., inheritance, encapsulation). Baldouf et al. [14] explain such a use case within an example system named Hydrogen [16] which uses an object-oriented modeling.

- Logic based models. In logic models, defining context model is done by facts, expressions, and rules. After that a logic based system allows to populate or update facts. In order to determine new facts based on existing rules in the systems, the interference technique, which is also called reasoning, can be used.

- Ontology based models. Concepts and relationships are represented by ontologies. Since, ontologies have high and formal expressiveness and opportunity for implementing ontology reasoning process, they are very promising instrument for modeling contextual data.

Korpiää et al. [17] express that ontologies are the most suitable models for fulfilling requirements such as simplicity, flexibility, inference, genericity and efficiency. The conclusion of the evaluation is that ontologies are the most expressive models with the Resource Description Language (RDF) and the Web Ontology Language (OWL). However, to develop flexible and useable context ontologies that cover a wide range of possible contexts is a challenging task

Resource Description Framework (RDF) and ontologies are instruments used when we want to describe, link and impose a structure on data. Semantic Sensor Network (SSN) is one such promising ontology. The SSN ontology is devised to describe sensors, their observations, procedures, feature of interest, the samples, observed properties and actuators. In the official document of SOSA [18] is stated that "*SSN follows a horizontal and vertical modularization architecture by including a lightweight but self-contained core ontology called SOSA (Sensor, Observation, Sample, and Actuator) for its elementary classes and properties*".

SOSA is an expansion of SSN in a way that provides widened infrastructure for target audience and application areas that can make use ontologies. Figure 2.1 shows how SOSA and SSN define and link entities to each other. Note that in the figure, there is always a path from one entity to another. So that, structuring data this way allows to establish a cause-effect relationship among data.

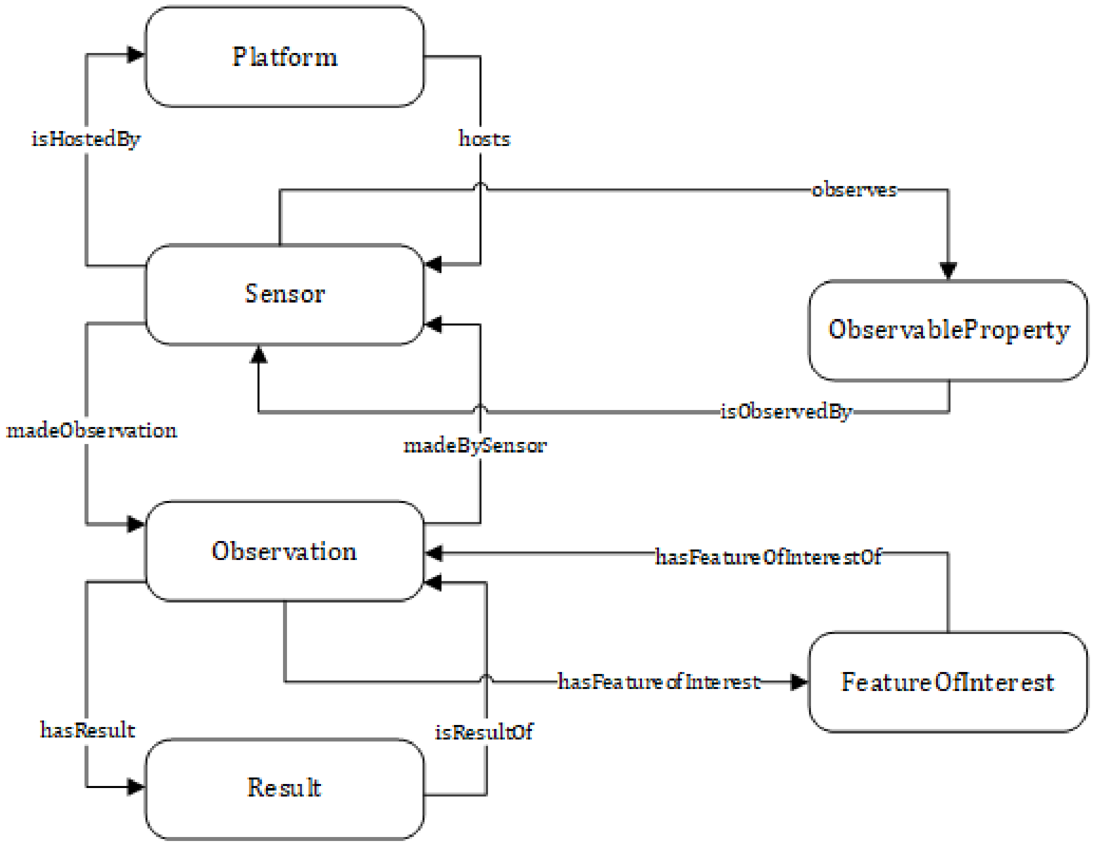


Figure 2.1: Overview of SOSA definition and linkage

2.2 Context and Sensor Data Delivery

Bellavista et al. [19] state that publisher-subscriber based transfer mechanisms are scalable solutions to distribute huge amounts of context data. Happ et al. [20] research requirements of IoT platform and they come up with a comparison of communication protocols AMQP, MQTT, XMPP and ZeroMQ. All these protocols are based on publish/subscribe architecture and mainly responsible for delivering messages or data from publishers to subscribers. They tested performance of these protocols based on latency, throughput for different sensor types. They state that Message Queuing Telemetry Transport (MQTT) is intentionally designed to transport sensor-like data and gains reputation as being de-facto standard in the field. Since MQTT is a pure publish/subscribe protocol, it is impossible to contact connected clients directly.

Official MQTT specification describes MQTT as follows: "*MQTT is a Client Server publisher-subscriber messaging transport protocol. It is lightweight, open, simple, and designed so as to be easy to implement. These characteristics make it ideal for use in many situations, including constrained environments such as for communication in Machine to Machine (M2M) and Internet of Things (IoT) contexts where a small code footprint is required and/or network bandwidth is at a premium*" [21]. The protocol runs over TCP/IP and provides one-to-many message transportation. Delegating context delivery to another party like MQTT, allows decoupling of context delivery from context acquisition. Due to minimal packet overhead and easy implementation process for both the client and server side, MQTT distinguished itself as the right way for transporting data when compared to HTTP.

The publisher/subscriber pattern is a different way of client-server communication. In client-server communication, clients connect directly to server, however in publisher/subscriber pattern, entities do not know each other. Both sides are named as *client* but they have different roles. One is subscriber(s) that receives message and the other is publisher that sends a message. Communication and transportation of data between them is handled by a broker. Main responsibility of a broker is filtering messages that come from publisher and distribute them to subscriber(s) that has interest. Filtering can be done with different ways. However, in MQTT, the common and default filtering mechanism is **topic** (subject) based. In this type, messages that come from publisher are filtered based on a *topic* (subject) and delivered to subscriber(s), which has an interest in this *topic*. The definition of a *topic* can be stated as follows: *Topic* is a string, which is used by the broker to filter messages for subscribers. *Topics* consist of one or more levels. Levels (topics) are case sensitive and separated by a forward slash (/).

In terms of context delivery, it is important to point out that the message is delivered accurately and completely to both senders and receivers. MQTT has three different message delivery types that are formally named as Quality of Service (QoS) in order to handle various situations like the one above. When an application process depends on a message, and it is not able to handle duplication of the message, and it wants to ensure that each message delivered to it should be sent only once, the application must use QoS level 2. QoS 2 is the highest level in MQTT and guarantees that every message is sent or received exactly once. Thus, this level is the safest but the slowest one. A more faster one is QoS level 1. QoS 1 also guarantees the message delivery but duplication of message may occur. The fastest one is QoS level 0. QoS 0 does not guarantee message delivery. In this level, message is sent and forgotten immediately without any concern about delivery. Furthermore, QoS levels of sending and receiving messages may be different and is subject to client. In particular, publisher might decide to send message with QoS 2 where subscriber might subscribe itself with QoS 0. Then broker receives message with QoS 2 but sends it to subscriber with QoS 0.

Eventually, MQTT comprises a variety of advantages for context delivery like decoupling context acquisition and distribution, ensuring message delivery, filtering, asynchronicity for sending and receiving messages. Usage of MQTT is discussed in Chapters 3 and 4.

2.3 Reference Architectures and Frameworks

In IoT, there are several architectures, applications and usage areas. The thesis mainly focuses on data traffic of IoT and its architecture. Data traffic occurs when providers and requestors communicate. The flow is generally from providers to requestors. Most of the time, providers are low hardware capable devices like sensors. They have limited power to operate for both sending data and sensing things that are responsible. HTTP is a high level protocol and has bigger footprint on the network communication for these type of devices. Luoto and Systä [22] stated that MQTT is more suitable for IoT devices with limited resources. Therefore, MQTT is used in practical IoT implementations.

Collina et al. [23] propose Qest that is a broker that consists of REST and MQTT. Qest exposes an MQTT topic as a REST resource. They expose resources through REST end points. Topics on MQTT are populated by REST end points. Collina et al. delineate the process as follows. *"Firstly it modifies the broker semantic to retain and syndicate the last payload seen on a topic, secondly it exposes that payload as a REST resource"*.

Sentilo [24] offers a very similar architecture to solve the high data traffic problem. It acts like a middleware between applications and providers with variety of services like alert, security, real-time storage and audit. Applications and providers communicate through Sentilo without knowing each other.

Gaia [25] is a middleware inspired from operating system design and called as Gaia Operating System. Gaia works similar to an OS. Gaia OS kernel layer has five services: Space Repository Service, Event Manager, Context Service, Presence Service and Context File System. The space repository services provides information storage and retrieval functionality about entities, which are hardware and software that are registered to Gaia OS. The event service is responsible for distributing events between suppliers and consumers. The context service provides querying and registration functionality to applications so that they gain ability to adapt themselves to their environment. The presence service detects presence of entities.

Hydrogen [16] is a three-layered architecture for a context aware mobile application development. Its architecture involves three layers: Adapter, Management, and Application layers. These layers are responsible for separating interaction, storage and maintenance processes from the applications themselves.

CASS [26] is a middleware platform implemented on the top of SQL database. CASS implements its functionality by four major components. *Sensor Listener* receives data from sources. *Context Retriever* fetches context data from context storage. *Rule Engine* is responsible for managing behaviors based on processed rules. *Interpreter* converts data into suitable and usable form, for example converting raw data to Celsius.

Context Aware Broker [27] CoBrA's architecture includes Semantic Web languages such as RDF and the Web Ontology Language OWL to define ontologies of context. CoBrA enforces the privacy policies on its central component *broker agent*.

CORTEX [28] is an another architecture based on object oriented design. It is basically based on Sentient Object Model. A sentient object encapsulates entities Sensory capture, Context hierarchy and Inference engine.

Context Toolkit framework [29] proposes a widget based approach and defines a widget as a software component that allows applications to access context information. The framework uses XML schema for context modeling.

Freeband [30] project offers a web-service based approach to facilitate development and deployment of mobile context-aware application. It claims that using web services technologies such as SOAP, WSDL, UDDI, XML eases to development of context-aware applications.

Octopus [31] is a middleware to ease the development of context-aware applications especially for home/office domain. It provides services to developers for developing sensor or application software where developers have minimal knowledge for sensors.

A Context Gathering Framework (CaSP)[32] is based on a service approach and has six services for sensing, modeling, association, storage and retrieval. It uses XML as messaging format. It also provides application programming libraries (API) to ease development of context-aware applications. This architecture is similar to the one that is proposed and implemented in this thesis.

Our parallel work described in [33], is another implementation based on the same principles discussed in this thesis. It is also a service based architecture which uses the Actor model whereas the implementation described here uses publisher/subscriber model. We shall call the actor based implementation as PCAD/A and the platform implementation presented in this thesis as PCAD/PS in order to differentiate these two from each other.

Table 2.1 compares the platforms surveyed in this section based on several criteria. Since, Qest classifies itself as a context broker, it is excluded from the table. The three criteria - architecture, data delivery method and context modeling - are mainly used to compare other platforms with each other. The remaining three criteria - security and privacy, event notification and real time support are included because PCAD/PS has provided these features to fulfill the requirements of the context-aware applications. The different architecture types used in the comparison are as follows:

- Component based: Architecture is based on loosely coupled major components.
- Distributed: System component can be distributed.
- Service based: Platform functionality is provided by set of services that can be work together or alone.
- Centralized: Platform provides a middleware to applications to be developed on top of it.

Data delivery methods are classified as query based and publisher/subscriber pattern based. The four context modeling techniques used in Table 2.1 are Key-value, object, ontology, markup and relational modeling techniques. The details of these techniques are given in Section 2.1, Context Modeling and Reasoning. For the other criteria, if the tools supports it, a Yes is entered, otherwise a No indicates an unsupported feature. When a piece of information is not available for a particular tool/application/framework, a dash (-) is used.

Table 2.1: Comparison of platforms

Platform	Architecture	Data Exchange	Modeling	Security and Privacy	Event Notification	Real Time Support
Context Toolkit	Component based	Query	Key-Value	-	No	No
Gaia	Service based	Query	Ontology	Yes	Yes	No
Hydrogen	Centralized	Query	Object	-	Yes	Yes
Sentilo	Centralized	Query, Pub/Sub	Key-Value	Yes	Yes	Yes
CASS	Centralized	Query	Relational	-	Yes	Yes
CoBrA	Distributed	Query	Ontology	Yes	Yes	No
CORTEX	Centralized	Query	Object	Yes	Yes	Yes
Freeband	Centralized	Query	Ontology	-	Yes	No
Octopus	Distributed	Pub/Sub	-	-	Yes	Yes
CaSP	Service based	Query, Pub/Sub	Markup	-	Yes	Yes
PCAD/A	Service based	Query	Relational	Yes	Yes	Yes
PCAD/PS	Service based	Query, Pub/Sub	Relational	Yes	Yes	Yes

Chapter 3

The PCAD Platform

The architecture - A Platform for Context-Aware Application Development (PCAD) proposed in [34] forms the basis of the platform described in this thesis. PCAD proposes a "*novel software platform based on the notion of context-awareness which allows rapid and easy development of context aware applications*".

3.1 General Overview of the Platform

The platform is inspired from OS service and layering concept. Silberschatz et al.[35] identifies a set of operating system services as shown in Section 3.1. Operating systems provide APIs or libraries to users and programs in order to provide a way to use services. OS system structure involves two crucial design principles: layering and modularization. A Layered architecture offers several advantages such as operation decoupling, simplicity and easy debugging. Since each layer is decoupled from the one above along an API boundry, when a change occurs in one layer, does not affect to layers above. Only the n+1st and n-1st layer may get affected. Modularization enables linking new services to system. A typical example of modularization is the Pluggable Authentication Modules (PAM) framework found on UNIX-like operating systems, which allows different authentication modules to be plugged into the framework. PAM-aware applications can dynamically utilize different authentication mechanisms as long as they use a standard application programming interface in their interaction with the PAM framework. PCAD provides a similar method like the PAM modules in developing Context-Aware Applications. Applications and sensors can plug themselves into the platform dynamically using the services of the communication binding layer in a similar way to the PAM module which can be plugged into the operating system using the authentication framework.

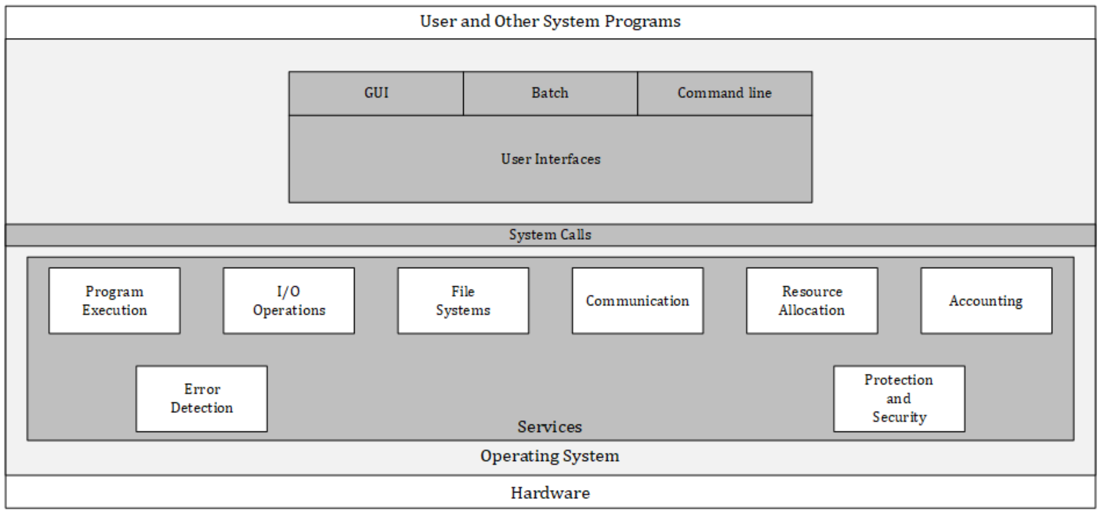


Figure 3.1: Operating System Services
Source: Taken from Silberschatz, A., et. al. [35]

PCAD emphasizes security and privacy, to be extensible, simple, generic and service based infrastructure for context-aware application development. The platform fulfills its operations through a service based architecture similar to the OS services. Figure 3.2 presents the proposed architecture and mapping of the PCAD layers to OS layers.

PCAD acts as an orchestrator when connecting sensors and application programs to each other via its services and platform bindings. PCAD services can either run independently or in combination with other services when fulfilling a user request. Even if a user does not directly interact with a service itself, the user uses these services by API calls or libraries. PCAD determines which service to be used based on user requests.

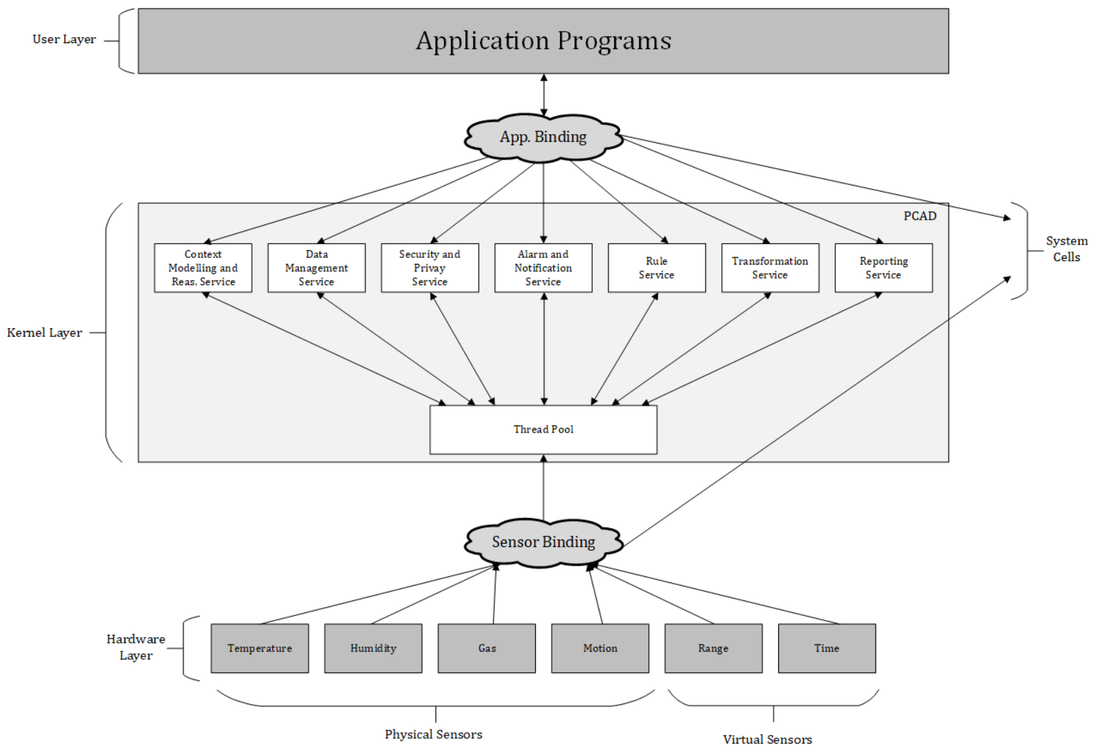


Figure 3.2: PCAD Middleware Layers
Source: Adapted from Celikkan, U., and Kurtel, K., [34]

3.2 PCAD Services

PCAD has adapted a service based computing model for fulfilling incoming requests. Based on the type of the request, the platform uses one or more services to fulfill the request. These services are listed in Table 3.1 and their functions are explained in Sections 3.2.1 through 3.2.7. Implementation and usage of these services are explained in Chapter 4.

Table 3.1: PCAD Services

Service	**Responsibility**
Data Management Service (DMS)	Stores and distributes context data. It supports various database systems
Security and Privacy Service (SPS)	Authenticates every request and determines authorizations based on Role and Attribute Based Access Control principle
Rule Service (RS)	Process and filters context data based on provided rules
Alarm and Notification Service (ANS)	Informs requestors when their request is fulfill
Transformation Service (TS)	Validates that the incoming request is well-formed and if necessary performs conversion between XML and JSON
Reporting Service (RpS)	Generates reports based on context data
Context Modeling and Reasoning Service (CMRS)	Transforms raw data into context data, so that context reasoning can be performed

3.2.1 Context Modeling and Reasoning Service

A context model addresses classified context data elements (time, location of user, etc.) in order to meet the requirements of a user. Therefore, context modeling is an essential part of any context-aware system. Context Modeling and Reasoning Service of the platform is responsible for transforming raw sensor data to linked data in RDF format as described in Section Context Modeling and Reasoning.

Several attributes of a sensor including but not limited to location, surroundings or time can be part of a context. Some of these attributes are obtained from sensor's meta-data and others are actual sensor readings. Creating a context enriches raw data by giving a way of reasoning and meaning to it. Creation, transformation and linkage of context data is a time consuming process. Rather than a default conversion of raw data into context data immediately, modeling of the context starts when it is requested. This means that, data is not stored as modeled context data in the database by default. This service does not perform any modeling in this current version of the platform. It is just a pass-through service, a placeholder for our future release. Data arriving to this service exit the service unmodified.

3.2.2 Data Management Service

Data Management Service is responsible for the management of the platform data such as storage and retrieval. Platform stores its data in a database. Although there are various database management systems to use, this thesis focuses on Relational Database Management System (RDBMS) and Document Database Management System (NoSQL). In RDBMS, data is stored in a tabular form of rows that represent records and columns that represent attributes. Searching, inserting, updating and deleting records are fulfilled by usingStructured Query Language (SQL). In the other alternative -i.e NoSQL database- data is stored in the form of *documents*. These documents contains key-value pairs. Each document represents the data itself, its attributes and relations. This structure of Document Database gives a flexibility for storing new attributes without the need to create a new column unlike the way it is done RDBMS. Insertion and retrieval of data from NoSQL database is faster too and this situation is particularly important for context-aware applications that use large data. As a result, NoSQL database is chosen as the database of choice. However, the platform can work with different RDBMS (i.e. MySQL) and NoSQL (i.e MongoDB) databases by employing a database adapter. Figure 3.3 represents structure of Data Management Service. Usage and implementation details are explained in Section 4.1.

NoSQL performs well and is a fast database. However, some operations may take time under heavy load. In order to provide a suitable solution to such situations, the platform incorporates parallelization of data streaming and database operations. Details of this structure is explained in Section 3.2.4.

3.2.3 Security and Privacy Service

Security and Privacy Service is responsible for protecting the platform resources and determine which user can perform what operation on data like inserting new resource, reading or viewing resource attributes. Every user that wants to use the platform must authenticate. Authentication process verifies a user's identity using username/password pair. Following a successful authentication, the platform

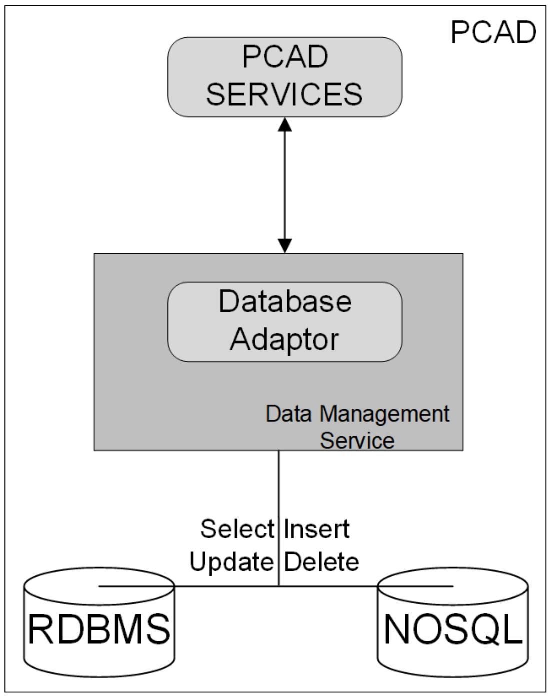


Figure 3.3: Data Management Service

generates an access token which is required in every subsequent requests. The access token becomes user's identity and is also used when authorizing request of the user that is sent to the platform.

Platform contains three types of users described as follows:

- System Administrator: any person who manages platform, authorize user's access requests,
- Requestor: any person or application that request data from the platform,
- Provider: any physical or virtual sensor or any other platforms or applications that provide data to the platform.

Users are further assigned roles which determines if an authorization should be granted or not.

There are various Access Control mechanism implementations, but the common fundamental intention of all is to control access to resources based on the role of entities. RBAC functions are adequate for most cases. However, a resource gets complex because of its attributes. The attributes and environmental conditions are used to define policies on subjects and resources. Therefore, any access control mechanism should be capable of taking care of these complex resources as well. For this reason Attribute Based Access Control (ABAC) is proposed in the literature. Hu et al. [36] defines ABAC as follows: "*An access control method where subject requests to perform operations on objects are granted or denied based on assigned attributes of the subject, assigned attributes of the object, environment conditions, and a set of policies that are specified in terms of those attributes and conditions.*" Hence, we designed the platform to use Role and Attribute Based Access Control mechanisms together.

In order to manage access to complex resources, the platform employs Attribute and Role Based Access Control together and defines three built-in roles corresponding to three different kinds of user. These roles are "System Administrator", "Data Requestor" and "Data Provider". These roles specify operations and access rights on the resources. Each role is made of triplets, which are called *Access Control List Triplets (ACLT)* and grants access to the attributes of a resource. Each ACLT specifies a resource name, resource attribute and an operation applied to the attribute. Consequently, a user with this role is granted to perform the specified operations on the resource attributes. The definition of a Role in terms of its ACLTs is given below:

$$Role = \{(Res, A, C)_1,\ (Res, A, C)_2 \ldots (Res, A, C)_n\}$$

where:

Res: Resource Name

A: Resource attribute

C: An operation from (**C**reate, **R**ead, **U**pdate, **D**elete)

Role inheritance in ABAC provides hierarchical and flexible role definition. This allows the derivation of new access control policies from the existing ones via inheritance. As mentioned previously, every user is assigned a predefined role by the system during registration and can be changed dynamically later via user interface or an API call.

3.2.4 Alarm and Notification Service

Alarm and Notification Service is used for fulfilling a request of the requestor. When a requestor makes a request to the platform, this request is handled by the Transformation Service first and the bulk of the work is done by Alarm and Notification Service thereafter. When ANS takes control, it is responsible for distributing jobs to other services for fulfilling the request. This service is also responsible for notifying requestors that have interest about some situations. Requestor can get data from the platform in two major ways. A requestor can either make a one time request which is composed of sending request and getting a response immediately after which the transaction will be over, or requestor can ask data continuously from the platform. Figure 3.4 shows these delivery methods of the platform and their data sources used.

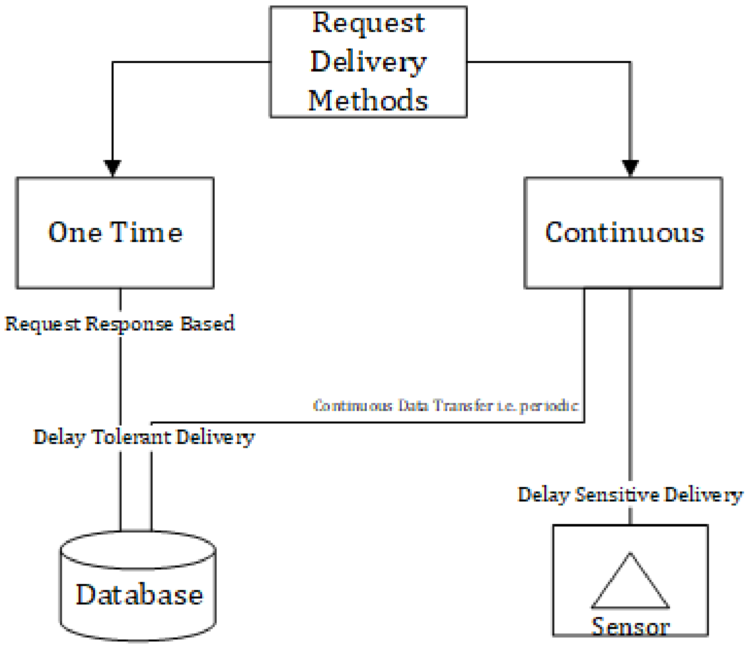


Figure 3.4: Data Delivery Mechanisms

When a requestor wants to get continuous data transfer, then requestor should use the services of MQTT. MQTT is a publisher/subscriber mechanism and is explained in Section 2.2. In continuous data transfer, the communication channel between the requestor and the platform is up all the time. The requestor has two options on how to use this channel. In the first one, requestor can get data directly from a sensor whenever it is available. In this method, the sensor software interrupts the platform and pushes data to platform, hence it is called "push method". Another name we give to it is "Delay Sensitive Delivery", because the data sent by the sensor is delivered to the requestor without any delay. In the second one, the requestor can get data in fixed periodic intervals from the database. Hence, the name of the method is proposed as "pull method" or "Delay Tolerant Delivery", because the platform periodically polls the database, pulls the data and sends it to the requestor. In periodic data request method, requestor will not get any data until the time period expires. In addition to continuous data delivery, requestors may want to get data only once. If requestor asks such data from the platform, the data will be retrieved from the database using a query and send to the requestor. "One time" data request uses RESTful services for requesting data and getting response whereas "continuous data" delivery methods uses MQTT.

When requestor asks data periodically, the requestor can get either new data or stale data that is received in the previous period. It is up to requestor to handle stale data.

In Delay Tolerant Delivery, since the data is acquired from the database this gives the platform a chance to convert raw data into context data. However, the downside of the Delay Tolerant Delivery method is that, since this process takes time, the requestor must wait for some period of time. Hence, it is called as Delay Tolerant Delivery. If the requestor wants something fast or if the application is sensitive to delays then the requestor asks data directly from sensor and delivered immediately. In this case the requestor receives only raw data, in the meantime however, this raw data visits the services of the platform and is converted to context data and ultimately written to the database.

In Delay Sensitive and Delay Tolerant Delivery methods, requestors connect to MQTT as a subscriber using the given topic. In the meantime, providers and the platform are connected to MQTT as publisher. Providers have private topic for each sensor they have and the topic is only usable between the platform and sensor itself. The platform creates a new topic for requestors when they request data from provider directly. Overview of this mechanism is demonstrated in Figure 3.5.

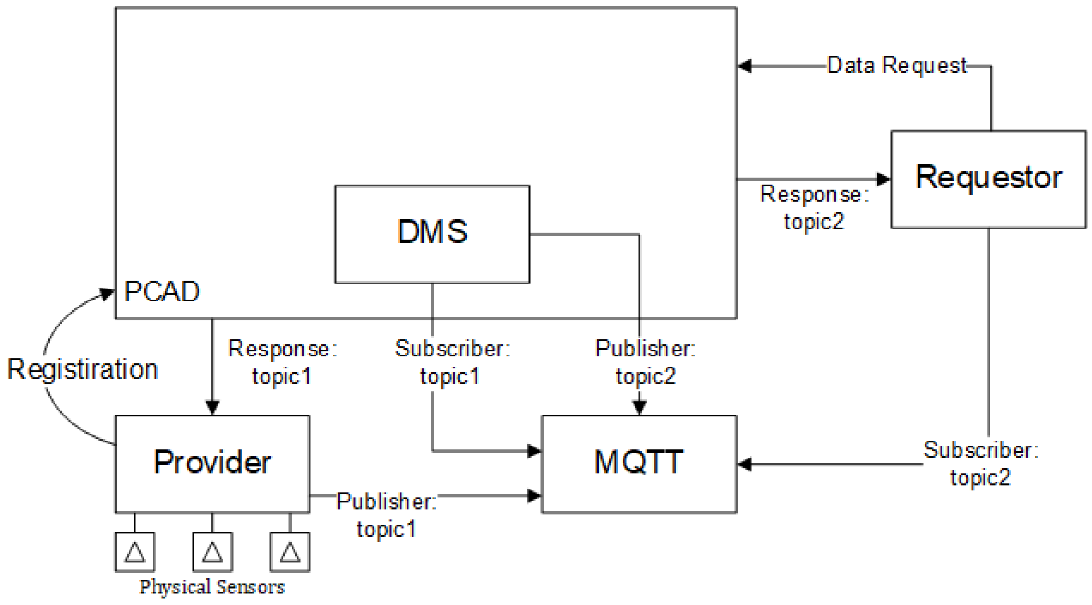


Figure 3.5: General View of Publisher/Subscriber Mechanism

3.2.5 Rule Service

The Rule Service (RS) is responsible for delivering context data based on conditions. RS performs this task according to the rules specified by the requestor. Rule Service can process aggregated context data that is obtained from different data sources. Once requestor requests context data using a rule, Rule Service registers the rule to the rule engine for the purpose of evaluating the rule against currently provided context data. When conditions specified in the rule are met, the platform starts sending data to requestor through the Alarm and Notification Service.

Following example demonstrates the working principle of RS. Assume that a requestor wants to perform a task such as turning air conditioner on when current room temperature and humidity values are less than 10 and greater than 50 respectively. In order to get notified, the requestor can define a rule with these constrains: “if *temperature* < 10 and *humidity* > 50 then notifyForUndesiredConditions”. Once both conditions are met, the platform sends a notification that includes current value of these sensors. This rule contains a conjunct (and). A rule that is a combination of conjuncts and/or disjuncts is called a compound rule.

If requestor wants to get sensor data directly from two different sensors using a rule, there may be an anomaly while getting data. If we refer to the above example, when temperature sensor provides data but humidity sensor does not, the rule fails immediately. In order to rectify the situation, the platform incorporated a timeout value when processing rules. When a sensor that participates in the rule provides data to the platform, the corresponding rule is fired, meaning the sub-rule that matches to sensor sending the data is executed immediately. The result will be either success or failure. However, the other sub-rule goes to a timeout period because the sensor corresponding to the other sub-rule has not sent any data. If at the end of the timeout period, the sensor still has not sent any data, the sub-rule fails. Thus, the overall rule result is evaluated according to the short circuit evaluation rules. On the other hand, if the rule is tagged with a database option, then the database will be consulted immediately. Now the timeout value indicates the time frame that sensor value will be acquired in the past. BNF notation for the new rule definition is given in Figure 3.6. The above described anomaly can be avoided by modifying the existing rule according to the new rule definition:

$$(Temp\ [(true,0)(false,0)]\ < 10)\ \&\ (Hum\ [(true,10)(true,60)]\ > 50)$$

⟨*Rule*⟩	::=	⟨*SubRule*⟩⟨*LogicalOperator*⟩⟨*SubRule*⟩
⟨*SubRule*⟩	::=	⟨*SensorId*⟩⟨*RequestDef*⟩⟨*ComparisonOperator*⟩ ⟨*Threshold*⟩
⟨*SensorId*⟩	::=	number
⟨*RequestDef*⟩	::=	[(⟨*sensor*⟩,⟨*timeout*⟩), (⟨*database*⟩,⟨*timeout*⟩)]
⟨*sensor*⟩	::=	'`true`' | '`false`'
⟨*timeout*⟩	::=	second
⟨*database*⟩	::=	'`true`' | '`false`'
⟨*ComparisonOperator*⟩	::=	'>' | '<' | '`!=`' | '`=`' | '≤' | '≥'
⟨*Threshold*⟩	::=	number
⟨*LogicalOperator*⟩	::=	'`and`' | '`or`'

Figure 3.6: Rule BNF Notation

This rule starts to run when the platforms get temperature data that is less than 10. Then it waits for 10 seconds for humidity sensor, if humidity data is not provided in 10 seconds, the data is acquired from the database at most 60 seconds old. After these, the rule is evaluated according to the short circuit evaluation rules and then the service returns a response to the requestor if conditions are met.

3.2.6 Transformation Service

The platform provides Programming Language Libraries in JAVA and Node.js as well as two different data format: JSON and XML for users and devices to interact with the platform as shown in Figure 3.7. Both data formats can be used in RESTful API invocations. Usage and details of the libraries are explained in Section 4.5.

Distinction between JSON and XML is made by header parameters. The platform enforces that any request to the platform must have these header parameters. A requestor or provider is also able to send data with one format

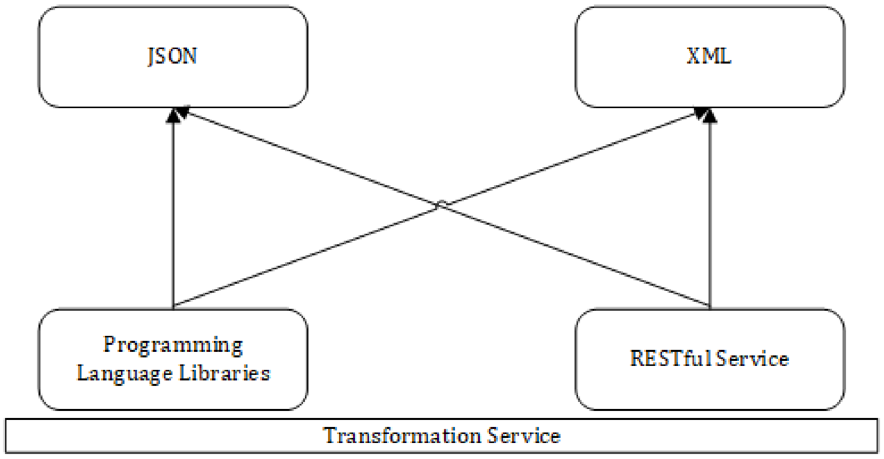


Figure 3.7: Transformation Service Overview

and get data in another format by setting header parameters appropriately. One requestor might want to send request payload in JSON format, and wants to receive response in XML format. To do so, requestor should set related parameters correctly.

3.2.7 Reporting Service

Reporting service (RpS) is responsible of data visualizing and composed of two different modules. RpS is available via user interface of the platform and all authorization applies to logged-in user. The first module of RpS draws charts using real time data. This means, when new data is created by the platform, related chart(s) updates itself without user intervention.

Second module shows user filtered results. User can filter any entity related to resources such as time, location, status. This type of chart does not update itself and waits for user intervention for getting the last available data.

Chapter 4

Design and Implementation

This chapter explains implementation of the platform. Choosing a proper framework, being fault tolerant and meeting performance needs are important criteria considered during implementation. In order to service a large number of requestors and providers, the platform should be responsive and non-blocking. These requirements are implicitly satisfied by Node.js features. The platform should also support different databases. The use of LoopBack framework enables speedy development. MQTT is used as the data delivery protocol as it is fault-tolerant. In the implementation of the platform, two external libraries are used. These are JSON rule engine [37] and, ABAC helper [38]. Table 4.1 shows the decisions for implementing the platform. In this section, design and implementation details of the services is explained. The Context Modeling and Reasoning Service and the proposed solution to the anomaly described in Rule Service will be implemented in the future version of the platform.

Table 4.1: Implementation decisions

Framework	: Node.js, LoopBack
External Libraries	: ABAC Helper, JSON Rule Engine
Languages	: JavaScript, HTML, Java
Operation System	: Linux
Database	: MongoDB
Communication Interfaces	: RESTful API, MQTT

Figure 4.1 provides a general overview of the platform emphasizing the data flow between the requester and the provider. The platform provides two connection types to the requestors. First one is used for continuous data streaming between a provider and a requestor using MQTT. This type of connection is represented using black lines in Figure 4.1. The dashed lines represent "subscriber" channels and normal lines represent "publisher" channels with broker. The other connection type is used to request one time data and perform other administrative operations such as new sensor registration, user creation or authentication. This connection type is represented as gray lines in Figure 4.1. HTTP protocol and RESTful API is used for this type of communication. The "PCAD Services" in the Figure are responsible for fulfilling the requests. In the following sections these services are explained in detail.

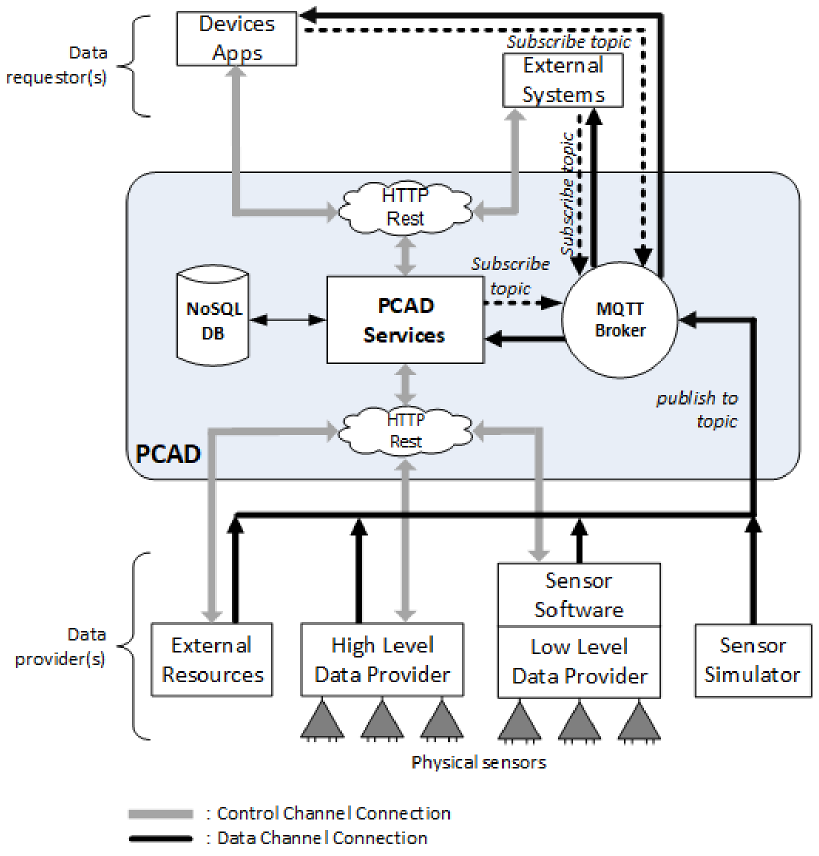


Figure 4.1: General view of the platform
Source: Adapted from Guner, A., et. al. [39]

4.1 Data Management Service

Data Management Service is responsible for performing database operations on resources and managing MQTT topics on the broker like creating, closing and giving access to it. Data Management Service follows the model-view-controller paradigm. Resources of the platform have corresponding models, which are used in database operation **C**reate, **R**ead, **U**pdate, **D**elete and abbreviated as CRUD.

In software development, generally speaking a model corresponds to a table in the database and an attribute corresponds to a column in the same table. This mechanism is good for defining models easily and retrieving data with its relations. Even though relational model and structured format are good perspectives, applications that use relational databases need code changes in order to use, manipulate and represent newly created column (attribute). Some applications may need attributes that are stored/retrieved dynamically without needing code changes. Consider the following example: sensor data on the platform have value and timestamp attributes. After a while, some sensors may gain an ability to provide location of the data. To store the location attribute in the relational database (except JSON supported databases), a new column should be added to the corresponding table and related code changes must be done. However, non-relational (NoSQL) databases provide such ability dynamically. In NoSQL databases, in order to add a new attribute to the database, it is enough to place the new attribute in the to-be-inserted object into the document. Therefore, NoSQL database is selected as the data source of the platform and MongoDB is used as an open source NoSQL database.

As stated, models are good for performing database operation at the application level. The platform has several models to represent application logic and CRUD operations. Every model corresponds to a related document (a table in Relational Databases) in MongoDB. Table 4.2 presents the documents and their descriptions. Tables 4.3 to 4.4 demonstrate the document structure. There are also some other built-in models that helps the platform to run. However, they are omitted.

Table 4.2: Database Documents and Descriptions

users	Users document contains user related information such as name, email, role, status. Any data provider is also stored in this document.
sensors	Sensors document stores metadata of any kind of data provider like physical or virtual ones and sensor belongs to user in *users* document.
roles	This document stores predefined and newly created roles. Roles are kept as triple which is explained in Security and Privacy Service
sensor_data	Any data provided by data provider is stored in this document.

The platform uses juggler [40], an ORM library that provides a set of interfaces for interacting with databases. Another advantage of using such library is that, it provides an abstract layer to models in order to use various relational and non-relational databases without fundamental code change.

Table 4.3: Database Models and Structures

Users	Roles
`"id": {`	`"id": {`
`    "type": "number",`	`    "type": "number",`
`    "required": true`	`    "required": true`
`},`	`},`
`"username": {`	`"role": {`
`    "type": "string",`	`    "type": "string",`
`    "required": true`	`    "required": true`
`},`	`},`
`"password": {`	`"resource": {`
`    "type": "string",`	`    "type": "string",`
`    "required": true`	`    "required": true`
`},`	`},`
`"name": {`	`"action": {`
`    "type": "string",`	`    "type": "string",`
`    "required": true`	`    "required": true`
`},`	`},`
`"surname": {`	`"attributes": {`
`    "type": "string",`	`    "type": "string",`
`    "required": true`	`    "required": true`
`},`	`}`
`"email": {`	
`    "type": "string",`	
`    "required": true`	
`},`	
`"role_id": {`	
`    "type": "number",`	
`    "required": true`	
`},`	
`"register_date": {`	
`    "type": "DateString",`	
`    "required": false`	
`}`	
`"valid": {`	
`    "type": "boolean",`	
`    "required": true`	
`},`	

Table 4.4: Database Models and Structures

Sensors	Sensor Data
`"id": {`	`"id": {`
`    "type": "number",`	`    "type": "number",`
`    "required": true`	`    "required": true`
`},`	`},`
`"user_id": {`	`"sensor_id": {`
`    "type": "number",`	`    "type": "number",`
`    "required": true`	`    "required": true`
`},`	`},`
`"type": {`	`"data": {`
`    "type": "string",`	`    "type": "string",`
`    "required": true`	`    "required": true`
`},`	`},`
`"metric": {`	`"timestamp": {`
`    "type": "string",`	`    "type": "DateString",`
`    "required": true`	`    "required": true`
`},`	`}`
`"valid": {`	
`    "type": "boolean",`	
`    "required": true`	
`},`	
`"period": {`	
`    "type": "string",`	
`    "required": true`	
`},`	
`"register_date": {`	
`    "type": "DateString",`	
`    "required": false`	
`}`	

4.2 Security And Privacy Service

Security and Privacy Service is responsible for mainly two tasks: authentication and authorization to control data access in the platform. To manage security and privacy, token based authentication and authorization is used. The general concept behind the token based authentication is to allow users to use an "access token" instead of providing their username/password pair for each request. Username/password pair is provided only when obtaining the initial token. Once the access token is acquired, each subsequent request except registration, log-in and log-out, should accompany this token. Access token is supplied to the platform in two ways. First is to insert the token as a URL parameter named as "access_token" as shown in Listing 4.1. The second, which is more compact and secure, is to supply the token as a header parameter again named as "access_token" and shown in Listing 4.2.

```
curl -X GET
--header 'Accept: application/json'
'http://DUMMY_PCAD_URL/api/v1/sensors?access_token=<
    OBTAINED_ACCESS_TOKEN>'
```

Listing 4.1: Sending Access Token as URL Parameter

```
curl -X GET
--header 'Accept: application/json'
--header 'access_token: <OBTAINED_ACCESS_TOKEN>'
'http://DUMMY_PCAD_URL/api/v1/sensors'
```

Listing 4.2: Sending Access Token as Header Parameter

After creating an account and gaining 'Access Token', this token is used for authentication. The platform creates an MQTT topic for providers. This topic is used for receiving data from a provider. Sending data via MQTT also requires an authentication. When providers initiate connection with the "CONNECT" message to the broker, they should send their obtained "Access Token" as a "Client Identifier". When MQTT broker gets a CONNECT message with the "clientId" parameter, it authenticates the provider. Opening a connection to

MQTT broker with “clientId” parameter is demonstrated in Listing 4.3. If authentication is successful then the broker returns “CONNACK” message with status code zero. Otherwise, one of the status codes given in Table 4.5 is returned.

```
const options={
    clientId:'<OBTAINED_ACCESS_TOKEN>',
    ...
};
var client = mqtt.connect("mqtt://DUMMY_PCAD_URL",options)
```

Listing 4.3: Sending Access Token as Header Parameter

Table 4.5: MQTT Connection Return Messages

Status Code	**Status Definition**
0	Connection accepted
1	Connection refused, unacceptable protocol
2	Connection refused, identifier rejected
5	Connection refused, not authorized

Requestors use RESTful endpoints via HTTP by providing previously acquired “Access Token”. Requestors use the same procedure specified for providers when they need to subscribe to MQTT topic. Note that subscribing to an MQTT topic is needed for continuous data transfer.

Users of the platform must have the right authorizations to access resources. As described in Section 3.2.3, gaining access to a resource on the platform is controlled by a combination of Role Based and Attribute Based Access Control Mechanisms. With this combination, the platform gains flexibility in defining and managing access policies. Resources of the platform are sensors and their data. Attribute is a property of the resource and operation is the action that is taken on that attribute. So that this set becomes a definition of a role on the platform. A role can inherit from other role(s). Consider the following scenario: System Administrator wants to create a role that has read access to attributes A_1, A_2, and A_3 of resource Ω. There are already two existing roles related to resource Ω as defined below:

$$\begin{aligned} Role: \ A &= \{(\Omega, A_1, Read),\ (\Omega, A_2, Read)\} \\ Role: \ B &= \{(\Omega, A_3, Read)\} \end{aligned}$$

Now, the System Administrator can define the desired role either by creating a new one or extending it from roles A and B. In the first method, the new role C is created as a stand-alone role by providing a brand new definition which is given below.

$$Role: \ C = \{(\Omega, A_1, Read),\ (\Omega, A_2, Read),\ (\Omega, A_3, Read)\}$$

The second and more flexible solution is to create the new role C by extending from role A and B. Then, role C is defined in terms of role A and B and denoted as;

$$C = A \cup B$$

In the second method when a new right is added to either role A or B, C gains the same right instantly. In the example above, when a requestor with a role A wants to access to resource Ω, the platform filters result object and send only values of A_1 and A_2 attributes, and when a second requestor having role C wants to access the same resource, platform returns result with value of A_1, A_2 and A_3 attributes of Ω.

With these information, explaining authorization is much simpler. Consider the same Role A given above and a requestor. When requestor requests access to Resource Ω, the platform queries the database and gets the access rights of the requestor. If an ACLT exists corresponding to the desired action and attribute in the Role Set for the requested resource, the user will be able get response to its request. Otherwise, the platform will return the response shown in Listing 4.4. When requestor gets data from MQTT same procedures follow. Before pushing data to topic, the platform filters result object according to access rights of requestor and then pushes the filtered result.

```
"error": {
    "statusCode": 401,
    "name": "Error",
    "message": "Authorization Required",
    "code": "AUTHORIZATION_REQUIRED"
}
```

Listing 4.4: Error Message for Unauthorized Access

The platform has built-in role definitions. These are “requestor”, “provider” and “admin”. “requestor” or “provider” role is assigned to users when (s)he registers himself/herself to the platform. These roles are stored in “Roles” document and given in Listing 4.5.

In the role definition given below “role” represents the name of a role (i.e. provider), “resource” is the model name of the “resource”(i.e. “sensors” or “sensorData”). The key “action” defines an operation on the “resource”. It consists of one of the four “CRUD” actions and one of the “own” or “any” keywords which indicates the possession of the “resource”. Having access right to “attribute” of the “resource” is defined by “attributes” key. When defining “attributes”, access right “*” covers all attributes of the “resource”. Exclamation mark “!” defines an access restriction to the “attribute”. For example

```
{
    role: 'provider', resource: 'sensors', action: 'create', ownership: '
        own', attributes: '*, !id, !valid, !userId, !registerDate'
}
```

means that, anyone having the role “provider” can create a sensor with its attributes except that “id”, “valid”, “userId”, “registerDate” attributes.

```
[
    { role: 'requestor', resource: 'sensorData', action: 'read',
        ownership: 'any', attributes: '*, !id' },
    { role: 'requestor', resource: 'sensors', action: 'read',
        ownership: 'any', attributes: '*, !valid, !userId, !
        registerDate' },
    { role: 'provider', resource: 'sensorData', action: 'read',
        ownership: 'own', attributes: '*, !id' },
    { role: 'provider', resource: 'sensorData', action: 'create',
        ownership: 'own', attributes: '*, !id, !timestamp' },
    { role: 'provider', resource: 'sensors', action: 'create',
        ownership: 'own', attributes: '*, !id, !valid, !userId, !
        registerDate' },
    { role: 'provider', resource: 'sensors', action: 'read',
        ownership: 'own', attributes: '*, !id, !valid, !userId, !
        registerDate' },
    { role: 'provider', resource: 'sensors', action: 'delete',
        ownership: 'own', attributes: '*, !id, !valid, !userId, !
        registerDate' },
    { role: 'provider', resource: 'sensors', action: 'update',
        ownership: 'own', attributes: '*, !id, !valid, !userId, !
        registerDate' },
    { role: 'admin', resource: 'sensorData', action: 'read',
        ownership: 'any', attributes: '*' }
    { role: 'admin', resource: 'sensors', action: 'read', ownership:
        'any', attributes: '*' }
]
```

Listing 4.5: Built-in Role Definitions

4.3 Alarm and Notification Service

The behavior described in Alarm and Notification Service of Chapter 3 is captured in seven interaction mechanisms. These mechanisms are grouped into two based on their delivery type: Delay Sensitive and Tolerant. First group provides one time data delivery and the second one provides continuous data delivery. Figure 4.2 demonstrates the groups and their subgroups with two properties: data source and rule inclusion. Following section will explain the seven interaction mechanisms in detail.

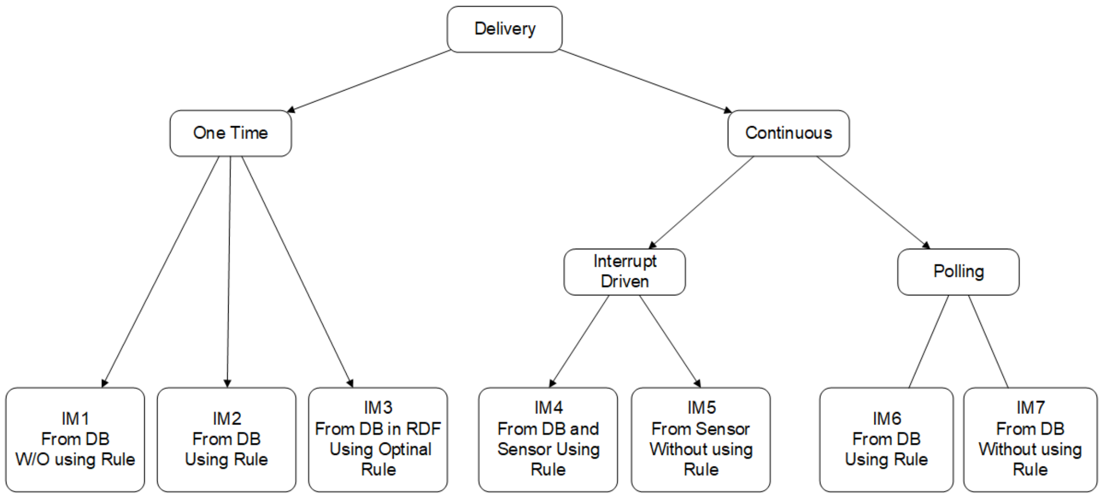

Delivery
One Time
Continuous
Interrupt Driven
Polling
IM1 From DB W/O using Rule
IM2 From DB Using Rule
IM3 From DB in RDF Using Optinal Rule
IM4 From DB and Sensor Using Rule
IM5 From Sensor Without using Rule
IM6 From DB Using Rule
IM7 From DB Without using Rule

Figure 4.2: Data Delivery Methods

Requestor-Platform Interaction Mechanism

Interaction mechanisms between the platform and providers are explained in this section. These interaction mechanisms demonstrates the Application Binding layer shown in Figure 4.3. In one time data delivery mechanisms, data request interface is RESTful web-services. In these interaction methods data sent to requestor is already processed and stored into database by the platform services. So, data is retrieved from the database and sent to requestor.

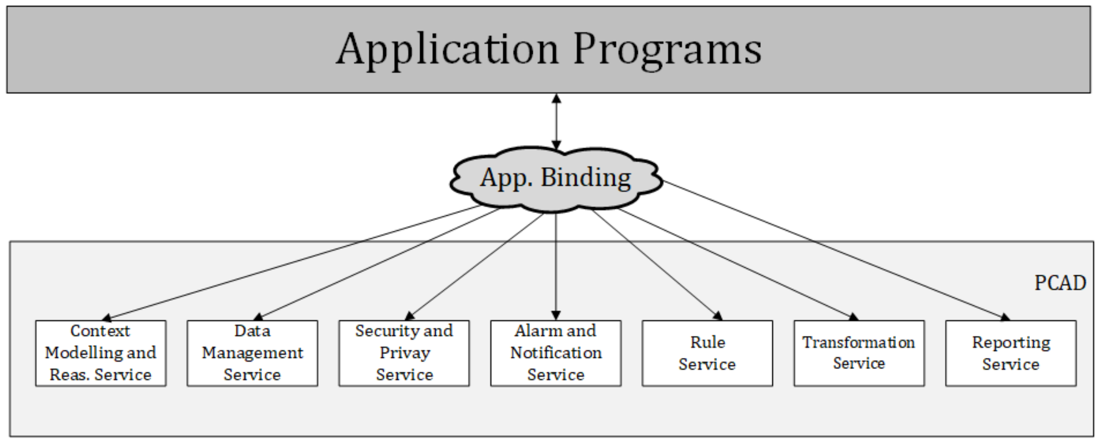


Figure 4.3: PCAD Requestor Binding Layer

1. Interaction Mechanism 1 (IM1):
   In Interaction Mechanism 1, requestor requests context data without any filter. Platform queries the database for the requested context data and send it to requestor as a response.

2. Interaction Mechanism 2 (IM2):
   Interaction Mechanism 2 is similar to IM1 except that IM2 includes a filter as described in Section 3.2.5. This filter is applied to data retrieval step and the filtered data is sent to the requestor.

3. Interaction Mechanism 3 (IM3):
   In this interaction mechanism, requestor receives data in RDF format.

Second group of interaction mechanisms provides continuous data transfer. In these interaction mechanisms, data request interface is also RESTful web-services. However, requestor gets data by subscribing to the related MQTT topic as described in Section 2.2. Some interaction mechanisms send data in raw format. This means that data is sent as it is collected from sensor without any extra processing done on it. In the remaining interaction mechanism, data is processed by the services of the platform to make it more suitable to context aware computing.

4. Interaction Mechanism 4 (IM4):
   Requestor requests data to be delivered directly from a sensor without any rule. When requested sensor data becomes available, the platform pushes raw data into an MQTT topic so it is transmitted to requestor. In parallel, the platform processes the raw data and saves it to the database. This interaction mechanism sends raw data to the requestor.

5. Interaction Mechanism 5 (IM5):
   In this interaction mechanism, requestor requests data directly from sensor by specifying a rule. When rule conditions are met, the platform pushes requested data into MQTT topic which eventually transmitted to the requestor. However, due to lack of available data on the server at the time of the request, the request may fail and no data is sent. This situation is explained in detail in Rule Service Section.

6. Interaction Mechanism 6 (IM6):
   In this interaction mechanism, requestor wants to get data periodically from the database. The platform registers the request to Alarm and Notification Service. Within periodic intervals, the platform pushes data into MQTT topic, which is consumed by the subscriber (requestors). The requestors may get stale data if no new data has been written to the database between two data delivery interval.

7. Interaction Mechanism 7 (IM7):
   This interaction mechanism is similar to IM6 except that it uses a rule when streaming data.

Figure 4.4 shows the data path for the continuous and one time data requests methods. On the left side, the requestor gets data from the MQTT broker. IM4 to IM7 is placed on the left side. On the right side, the requestor gets data as response as the result of its request. IM1 to IM3 is placed here.

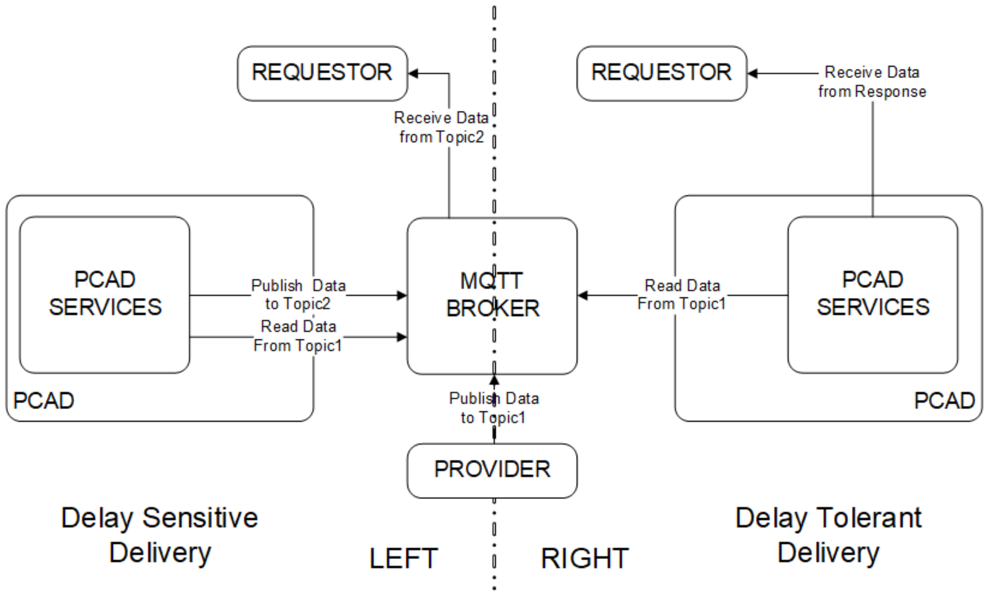


Figure 4.4: Delay Sensitive and Delay Tolerant

Provider-Platform Interaction Mechanism

In addition to application binding layer, the platform has a sensor binding layer as demonstrated in Figure 4.5. Even though providers are classified as "users" in the platform, actually they are data creators as having physical or virtual sensors.

First of all, a provider should register to the platform by using either the graphical user interface or RESTful API. After the registration and approval by the "System Admin", provider gets the "Access Token" and acquires the right to add a new sensor to the platform. If the provider chooses to use RESTful API to add a new sensor, the provide should add a "Access Token" to the add request. The provision of the "Access Token" is implicit, if the provider chooses to use GUI. Regardless of using a GUI or REST, providers obtain the followings:

- A unique sensor ID
- A unique MQTT Topic

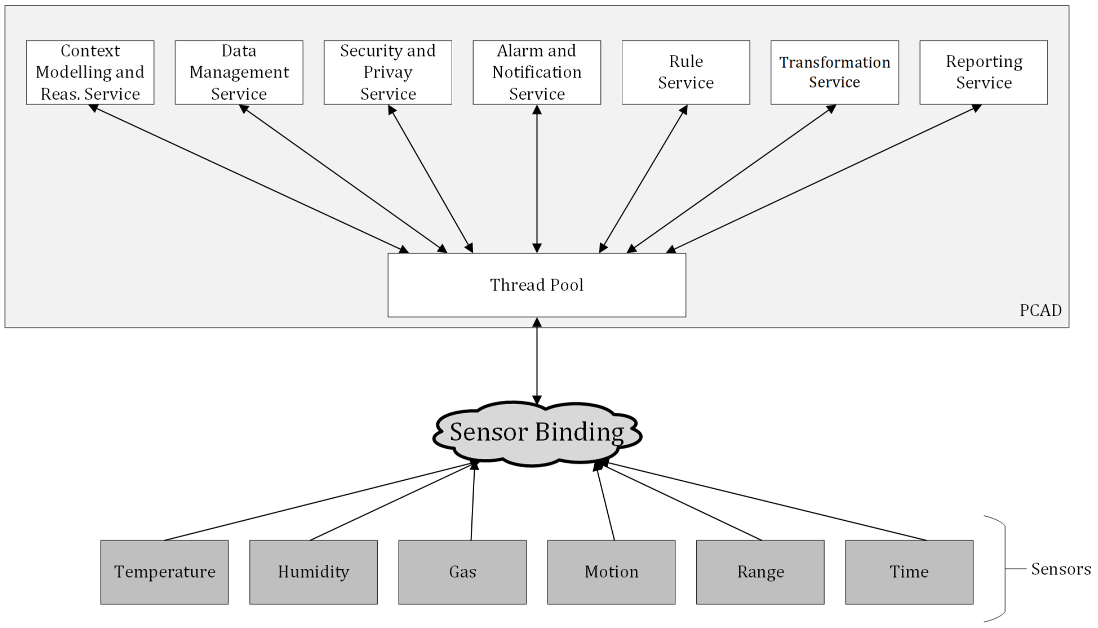


Figure 4.5: PCAD Provider Binding Layer

Data is provided to the platform either by using RESTful API or MQTT. Even though the platform does not restrict the use of RESTful API for data provision, using MQTT protocol is more suitable than using HTTP. When using RESTful API the provider should use obtained sensor ID and the "Accees Token" to provide data. Listing 4.6 demonstrates an RESTful API request.

```
curl -X POST
--header 'Content-Type: application/json'
--header 'Accept: application/json'
--header 'access_token:<OBTAINED_ACCESS_TOKEN>'
--data '{
    'data': 'string',
    'timestamp': 'dateTime'
  }'
  'http://DUMMY_PCAD_URL/sensors/<OBTAINED_SENSOR_ID>/data'
```

Listing 4.6: Providing Data Using RESTful API

When using MQTT, provider should connect to a given MQTT "Topic". As stated, MQTT broker also enforces authentication. To get authenticated, provider should use the obtained "Accees Token" when the provider wants to connect to the broker. Listing 4.7 demonstrates opening connection to broker and sending data to the platform in JAVA language.

```
String publisherId = "<OBTAINED_ACCESS_TOKEN>";
String TOPIC = "<OBTAINED_MQTT_TOPIC>";
String DATA = "<SENSOR_DATA>";
IMqttClient publisher = new MqttClient("tcp://DUMMY_PCAD_URL:PORT",
    publisherId);
MqttConnectOptions options = new MqttConnectOptions();
options.setCleanSession(true);
publisher.connect(options);
publisher.publish(TOPIC,DATA);
```

Listing 4.7: Providing Data Using MQTT

Interaction Mechanisms Service Bindings

While requestors interact with the platform by using one of the interaction mechanisms, the request is handled by the platform services. The two services, Security and Alarm and Notification, are used for all requests and by all interaction mechanisms. When a request arrives to the platform, first ANS calls Transformation Service in order to parse the request body and the elements. After that, the ANS calls the SPS for security check. The SPS checks security and creates an ACLT object for future use. These operations are common for all interaction mechanisms.

Interaction Mechanisms 1, 2, and 3 take data directly from the database. So, these mechanisms use Data Management Service. In addition, IM2 includes rule object and IM3 returns data in form of RDF. Therefore, IM2, IM3 use Rule Service and Context Modeling and Reasoning Service respectively. IM4, IM6 and IM7 are continuous data deliver mechanisms and their source of data is also the database. So, these interaction mechanisms also use DMS. IM5 uses continuous data delivery mechanism. However, its source of data is sensor. So IM5 is the only one that does not use DMS. Overall service usage of Interaction Mechanisms is represented in Table 4.6.

Table 4.6: Interaction Mechanism Service Usage

Interaction Mechanism	**RS**	**DMS**	**ANS**	**CMRS**	**SPS**	**TS**
IM1	-	X	X	-	X	X
IM2	X	X	X	-	X	X
IM3	-	X	X	X	X	X
IM4	X	X	X	-	X	X
IM5	-	-	X	-	X	X
IM6	-	X	X	-	X	X
IM7	X	X	X	-	X	X

Interaction Mechanism 6 demonstrates the operation of second group of the interaction mechanisms and is given as an example to clarify the sequence of events that take place. These sequence of events are demonstrated in Figure 4.6.

1. Requestor requests data by using RESTful API.

2. After taking the request, ANS calls Transformation Service for transforming request body to make it usable by the platform services followed by a call to the Security and Privacy Service.

3. Security and Privacy Service checks authentication and access rights of requestor via the supplied “Access Token”. Then, SPS creates an ACLT for filtering data that will be sent eventually.

4. Since the request is handled using continuous data delivery, the Alarm and Notification Service calls the Data Management Service to create a topic on the MQTT broker. After that, DMS registers itself as a publisher for that topic and call Transformation Service for the purpose of sending the topic and MQTT URL to the requestor. In the meantime, the Alarm and Notification Service saves the topic for future use. The DMS remains in subscriber state all the time for all incoming data from all sensors in order receive data from them and make them available for the platform.

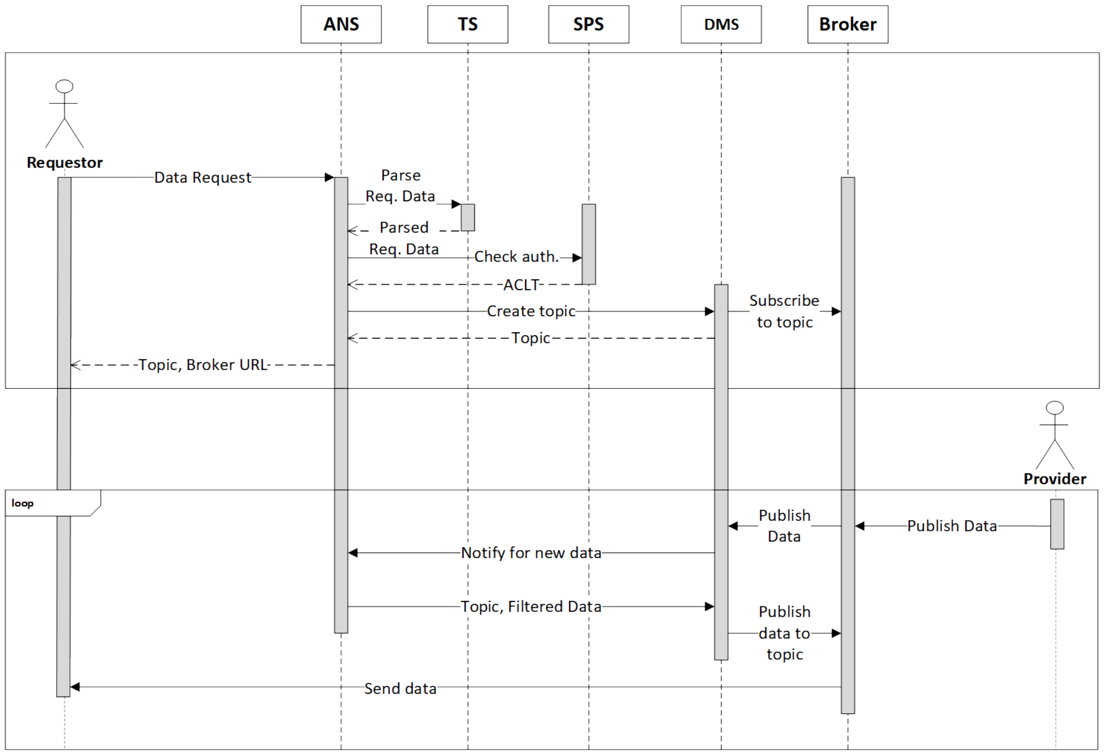


Figure 4.6: Sequence Diagram for IM6

5. When requestor receives the topic and MQTT URL, it connects to MQTT as a subscriber for that topic and waits for incoming data.

6. When data associated with a topic is received by the platform DMS notifies the Alarm and Notification Services.

7. DMS filters data attributes based on ACLT, and pushes the data into the topic. At that time, requested data is published to the requestor. The requestor can either waits for future incoming data, or close connection with the broker. At the same time, the data is processed by the platform services.

Endpoints

The complete list of RESTful API endpoints is given in Table 4.7 and their payloads is represented in Table 4.8.

Table 4.7: RESTful API Endpoints

URL	**Method**	**Payload**	**Description**
/users	POST	User	Creates a new users.
/users/login	POST	UserCredentials	Logs-in Provider, Requestor and Admin.
/users/logout	POST	User	Creates a new users.
/users	GET POST	- User	Returns users Creates new user
/users/{id}	GET DELETE PUT	- - User	Returns sensor Deletes user Updates user
/sensors	GET POST	- Sensor	Returns sensors Creates new sensor
/sensors/{id}	GET DELETE PUT	- - Sensor	Returns sensor Deletes sensor Updates sensor
/sensors/{id}/data	GET POST	- SensorData	Returns sensor data Creates sensor data
/sensors/{id}/stream	GET POST	- Rule	Send raw data using MQTT Send data when rule conditions are satisfied
/roles	GET POST	- Role	Returns roles Creates new role
/roles/{id}	GET DELETE PUT	- - Role	Returns role Deletes role Updates role
/reports/{id}	GET	-	Returns sensor information

Table 4.8: Endpoint Payloads

User	Sensor

`{`
`  "firstname": "string",`
`  "lastname": "string",`
`  "username": "string",`
`  "company": "string",`
`  "email": "string",`
`  "password": "string"`
`}`
	`{`
	`  "type": "string",`
	`  "unit": "string",`
	`  "latitude": "string",`
	`  "longitude": "string"`
	`}`

Role	SensorData

`{`
`  "role": "string",`
`  "resource": "string",`
`  "action": "string",`
`  "attributes": "array"`
`}`
	`{`
	`  "data": "string",`
	`  "timestamp": "string"`
	`}`

Rule	

`{`
`  "periodic": "boolean",`
`  "period": "string",`
`  "rule": "object"`
`}`	

4.4 Rule Service

Data delivery can be achieved in two ways. First one is the one time data delivery (uses filter). In this delivery method, Rule Service acts as data filtering middleware based on filter object, which can be provided as a request parameter. This data filtering can be used only with one time data request methods and filter object should be in JSON format and consist of valid operators given in Table 4.9. When a request is made with the filtering object, the platform queries the database against the given filter. Collected data is returned to requestor. Assume that a requestor wants get data from "Sensor with ID 1" with the following conditions: "data equals to 15 or 23 and timestamp is greater than 2019-01-20". This request is demonstrated in Listing 4.8.

Table 4.9: Supported Operators for Rule and Filter Defining

Operator	**Description**
=, neq	Equality, Inequality (!=)
and, or	Logical AND operator, Logical OR operator
gt, gte	Numerical greater than (>); greater than or equal (≥). Valid only for numerical and date values
lt, lte	Numerical less than (<); less than or equal (≤). Valid only for numerical and date values
between	True if the value is between the two specified values: greater than or equal to first value and less than or equal to second value

```
Filtering Object:
{
    "where": {
        "and": [{
            "or": [{
                "data": "15"
            },
            {
                "data": "25"
            }]
        },
        {
            "timestamp": {
                "gt": "2019-01-20"
            }
        }]
    }
}
```

```
Complete Request URL:
http://DUMMY_PCAD_URL/sensors/1/data?filter={"where": {"and":[{"or": [{"
    data": "15"},{"data": "23"}]},{"timestamp": {"gte": "2019-01-20"}}]}}
```

Listing 4.8: Example Request with Data Filtering

The second and more complex delivery method is the continuous data delivery (uses a rule). In this delivery method, requests are made by RESTful API and data is taken by MQTT protocol. Defining a filter is not possible on the MQTT broker, this process is taken over by the platform. In order to get filtered continuous data, requestor should provide a rule within the request. When platform gets a request with rule, ANS calls the Rule Service. The Rule Service registers the rule to the rule engine [37] on the platform. When any sensor data is provided to the platform, ANS calls Rule Service for executing rules. If all conditions are met then RS notifies ANS and ANS calls DMS to transfer data to requestor by pushing it to MQTT topic. As mentioned in Rule Service there can be an anomaly when checking rule conditions. Resolution of this anomaly is out of the current platform implementation and will be implemented in the future release. So rules can only be executed for only one sensor in the current release.

Rule conditions consist of "*fact*", "*operator*" and "*value*" keywords combined with "***all***" or "***any***" option. These options correspond to "*and*", "*or*" logical operators respectively. "*fact*" represents needed attribute of resource such as "*data*" or "*timestamp*". Allowed "*operator*"s are given in Table 4.10 and "value" is the desired outcome of the "fact". Definition of the rule with these keywords and structure is required by the used Rule Engine. In the future release of the platform, defining rule and filtering result will have same structure and syntax.

Assume that the requestor wants to get data from sensor 1 if the value is between 12 and 15 or value is equal to 20 by using Interaction Mechanism 5. According to these conditions, structure of the rule and the request are demonstrated in Listing 4.9.

Table 4.10: The Platform Rule Operators

Operators	**Description**
String, Numeric and Date Values	
equal	"*Fact*" must be equal to the "*value*"
notEqual	"*Fact*" must not be equal to the "*value*"
Only Numeric Values	
lessThan	"*Fact*" must be less than the "*value*"
lessThanInclusive	"*Fact*" must be less than or equal to the "*value*"
greaterThan	"*Fact*" must be greater than the "*value*"
greaterThanInclusive	"*Fact*" must be greater than or equal to the "*value*"

```
curl -X POST 'http://DUMMY_PCAD_URL/sensors/1/data'
--header 'Content-Type:application/json', 'Accept:application/json'
--header 'access_token:<OBTAINED_ACCESS_TOKEN>'
--data '{
    "conditions": {
        "any": [{
            "all": [{
                "fact": "data",
                "operator": "greaterThan",
                "value": 12
            },{
                "fact": "data",
                "operator": "lessThan",
                "value": 15
            }]
        },{
            "fact": "data",
            "operator": "equal",
            "value": 20
        }]}
}'
```

Listing 4.9: Example Data Request with Rule

4.5 Transformation Service

In order to transfer data between two sides properly, there should be an agreement on the data structure between them. From a RESTful service perspective, JSON is a popular choice for the data format. JSON is self-describing and easy to understand and use. Data format consists of name-value pairs and can represent objects, arrays. On the other hand, XML is used for many years to store, transport and represent data. Transformation Service is responsible of supporting these data formats, translating one into other in the requests and responses.

Distinguishing between these formats are made by providing appropriate header in request. These header parameters are "Content Type" and "Accept". When using JSON as a transfer format these parameters should be set to "application/json". For XML format "application/xml" value should be used. Except continuous data transfer by using MQTT, all API request and response bodies are interchangeable between JSON and XML. Assume that one provider wants to create a new sensor and send related data in JSON format. However, providers want to have them returned instance of the sensor to be in XML format. Listing 4.10 represents the proper header parameter for such request.

```
curl -X POST
--header 'Content-Type: application/json'
--header 'Accept: application/xml'
--data '{
  "type": "string",
  "metric": "string"
  "registerDate": "dateTime"
}'
'http://DUMMY_PCAD_URL/sensors'
```

Listing 4.10: Sending JSON Data Using RESTful service

After receiving this request the platform creates a new sensor metadata in the database and returns the created record. since doubleQuoteAccept parameter is set to "application/xml", response will be in XML format which is given in

Listing 4.11.

```
<?xml version="1.0"?>
<sensor>
  <id>number</id>
  <type>string</type>
  <metric>string</metric>
  <valid>boolean</valid>
  <registerDate>dateTime</registerDate>
</sensor>
```

Listing 4.11: Receiving response in XML Format

Application Programming Libraries (APL)

Some applications may want to use programming library instead of using RESTful services directly. To provide support for such applications, the platform offers Application Programming Libraries (APL) in Java and Node.js. *CRUD* operations can be done using these libraries.

Populating (create, update, delete) methods of APL returns the populated instance where Read method returns either “Object” or “Array of Objects”. Both APL have same function signature. However, Node.js functions return “promise”. JavaScript promises are objects that can produce result at unknown time in the future. State of promises can be either resolved or rejected. Resolved state means that operation is successful and contain desired outcomes such requested objects, response of http request. Promises can return rejected state if any error occurs such as network or database error. Listing 4.12 shows Java (pcad.jar) and Node.js (npm module: pcad) class constructor and method signatures of the platform library. Behind scenes, these libraries create a HTTP request to RESTful API.

```
Class Constructor:
  PCAD()
Methods:
  initPcadInterface(String username, String password, HashMap<String,
      String> option)
  getPcadResource(String resourceName, HashMap<String,String> options)
  getPcadResource(String resourceName, String resourceId, HashMap<String,
      String> options)
  savePcadResource(String resourceName, Object resource, HashMap<String,
      String> options)
  updatePcadResource(String resourceName, Object resource, String
      resourceId, HashMap<String,String> options)
```

Listing 4.12: PCAD Library Class Constructor and Functions

4.6 Reporting Service

Presentation of detailed data is handled by Reporting Service (RpS). Visualization and exposing such information provides a detailed and useful overview of the platform. In user interfaces, RpS provides filterable data visualization via graphics. In these screens, user is able to filter dataset of graphs such as filtering temperature sensors available in the platform. When user is done such filtering, corresponding graph(s) will use the selected sensor(s) as a source.

Using "`/sensors/{id}/stream`" with "*POST*" method is different than others. This endpoint has special payload parameters, which change the operation mode of the endpoint and is used for continuous data delivery methods. First parameter is "*periodic*". This parameter can be either "true" or "false". When it is "true" "*period*" parameter should be defined. This endpoint corresponds to *IM7*. If "*rule*" parameter is added too then it corresponds to *IM6*. If only "*rule*" parameter is set then it corresponds to *IM4*. "*GET*" method of the same endpoint corresponds to *IM5*.

4.7 Design and Implementation Outcomes

An architecture that promotes robustness, extensibility, scalability, adaptability, and privacy allows the creation of software that is reliable, available and serviceable. We set our architectural goals with these aspects in mind. Even though Context Modeling and Reasoning Service has not been implemented yet, extensibility attribute of the architecture allows this service to be easily added to the platform in the future.

Since context data is complex, a flexible access control mechanism is needed. Role based access control, which is adequate enough for a variety of systems can not provide enough flexibility for such complexity. This is the primary reason that the platform uses RBAC and ABAC together.

The platform connects a large number of data providers and data requesters in different ways such as one to many or many to many. A publisher-subscriber pattern is very suitable for such relationships. It also serves the purpose of decoupling publishers and subscribers by involving a broker in between. The platform uses an MQTT protocol based broker to delegate the context delivery tasks to the broker. Use of a broker is especially very beneficial for devices with low hardware capabilities such as controllers housing the sensors. Additionally, Quality of Service (QoS) feature of MQTT protocol provides different levels of guaranteed delivery- an important contributor to the availability aspect of the software. The use of MQTT based broker has shortened development time, at the same time provided many of the desired attributes of a good architectural design in the implementation. The alternative would be to implement everything from scratch.

Using open source libraries in the implementation has speeded up the development process. Open source libraries were preferred because it allows the addition of missing functionality into the library as the source code is available. Moreover, the continuous support of the community in bug fixings and improvements keeps the library up-to-date.

Chapter 5

Tests

In this thesis, three types of tests are planned that give more verification information.

- Database performance test
- Integration test
- GUI test

Performance tests aims to measure performance of the platform. For the Integration Test, Katalon Studio[1] is used to automate test execution for every development cycle. GUI tests are done manually since GUI has few screens.

Database performance test

The database performance test is used to gather information about the potential performance of the database. In order to achieve these goals, data insertion and retrieval performance of MySQL and MongoDB are measured. In order to generate load for the performance test, sensors data are simulated. A Node (sensor.js)

[1]http://katalon.com

application is written to generate random data with a given time interval and send the data to the platform using MQTT. The details of the test environment is presented below:

- Hardware
  - Intel Xeon E3-1220v3 (4 core, 3.1 GHz, 80W), 8GB DDR3 1333 MHz UDIMM, 2 x 1TB 7.2K rpm SATA / 12TB LFF (4x3TB)
- OS and Software
  - Ubuntu KDE 14.04
  - Node.js version: 10.15.3
  - MySQL version: 5.6.27 Ubuntu Release
  - MongoDB version: 4.0.3

To make an informed decision and draw more accurate conclusions about the performance, 3 test case scenarios are devised as shown in Table 5.1. The aim of these test scenarios is to measure how responsive the platform is under the load rather than to compare different databases.

The first test case environment is demonstrated in Figure 5.1. Default configuration for MySQL and MongoDB is used for this test. The results exhibit that, writing sensor data takes 5808 milliseconds on average with MySQL database under load (6000 requests) where it takes 1.56 milliseconds with MongoDB. The graph in Figure 5.2 is drawn using MATLAB tool with the data given in Table 5.2. In Table 5.2 all values are given in millisecond. In this test, the platform that using MongoDB performs much better than the case using MySQL, as the number of data insertion increases. For example, as seen in the Table 5.2, for 1000 records (10 seconds test duration) MySQL needs minimum 42 milliseconds and 84 milliseconds on the average to insert a record where MongoDB does the same in 1 millisecond and in 1.23 milliseconds on the average. For 6000 records MySQL needs minimum 41 milliseconds and MongoDB needs minimum 1 millisecond and 1.56 milliseconds on the average.

Table 5.1: Test Cases

Test case 1

- 1 simulator sensor located on a server sends data every 10ms (100 insert request per second)
- Default MongoDB and MySQL configuration
- Run time 10, 20, 30, 40, 50, 60 seconds

Test case 2

- 4 simulator sensors located on a remote machine send data every 1ms (4000 insert request per second)
- 1 simulator sensor located on a server machine sends data every 10ms (400 insert request per second)
- 1500 users located on remote machine concurrently send data request for getting last value of temperature sensor
- Default MongoDB and MySQL configuration
- Run time: 5 minutes

Test case 3

- Same load and configuration with Test Case 2
- Users request temperature sensor data with the following condition: “data is between 20 and 25”

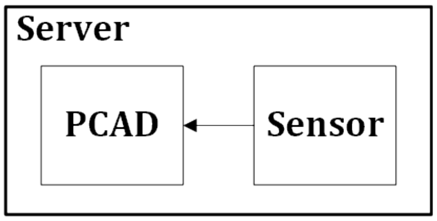


Figure 5.1: Test Case 1 Environment

Table 5.2: Test Case 1 Results

	MongoDB			MySQL		
# of Records	Min (ms)	Max (ms)	Avg. (ms)	Min (ms)	Max (ms)	Avg. (ms)
1000	1	2	1.23	42	730	84
2000	1	2	1.26	33	3252	112
3000	1	3	1.30	24	7051	153
4000	1	3	1.31	38	8831	3124
5000	1	3	1.31	41	13716	4714
6000	1	3	1.56	41	15610	5808

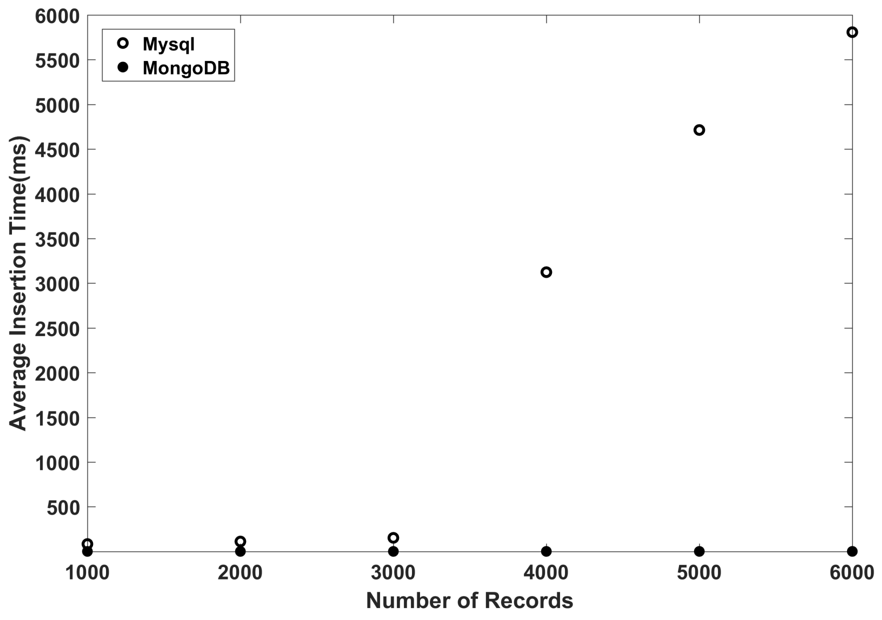


Figure 5.2: Test Case 1 Result: MySQL and MongoDB Insertion Time

In test case 2, an end-to-end performance test is designed and its environment is demonstrated in Figure 5.3. In addition to simulating data generation, data request of the users are simulated too. 4 sensors send data every 1 millisecond and 1 sensor sends data every 10 milliseconds to the platform. During the test, 1500 concurrent users request data from the platform. Response time is calculated as follows:

Response Time = *Time, response is delivered to user* - *Time, user makes a request*

This calculation is shown as $response_{total} = t_1 - t_0$ in Figure 5.3 where t_1 denotes the time a response is delivered to user and t_0 denotes the time when a request is made by the user. At the end of the test, the average response time of the platform is calculated as 630 milliseconds. Data insertion performance of databases are also investigated and results are stated in Table 5.3. As seen from the table an increase in the number of sensor data and requestors does not change the MongoDB performance, thus it does not affect the platform performance.

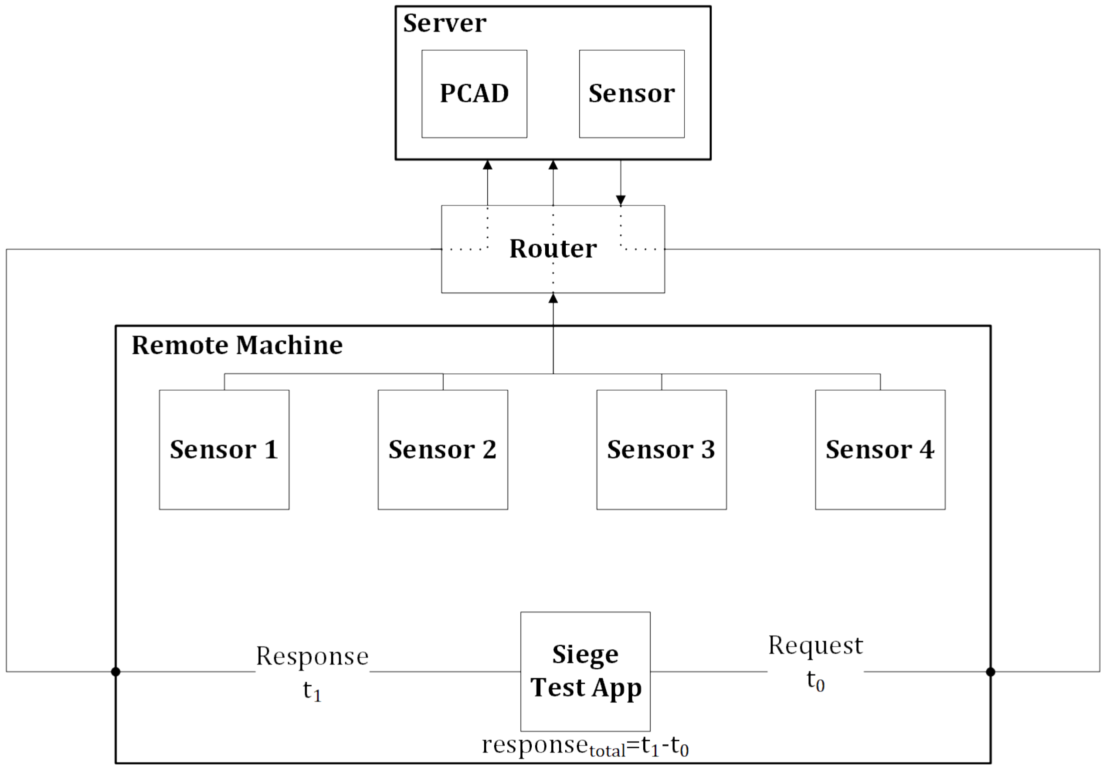


Figure 5.3: Test Case 2 Environment

Table 5.3: Test Case 2 and 3 Results

	Test 2			Test 3		
	Min (ms)	Max (ms)	Avg. (ms)	Min (ms)	Max (ms)	Avg. (ms)
MongoDB	1	3	1.35	1	1197	117
MySQL	32	95813	68216	317	117318	89215

Test case 3 uses the same environment and load with test case 2 except the fact that test case 3 uses a condition when requesting data. In test case 3, users send a query to the platform for temperature sensor data asking for values between 20 and 25. More than 1M records in the database are used in the test. Average response time of the platform is calculated as 1.10 second. Data insertion results are also stated in Table 5.3.

Due to query condition in data request, or increasing the sensor simulator programs increase the insertion times for both databases. The increase is more drastic for the MongoDB. The average insertion times increases nine fold for MongoDB but it is less than half for MySQL. However, using MongoDB performs much better under load for all tests regardless a condition is used or not.

API and Integration Tests

The aim of the API tests is to ensure that each implemented feature does not break the platform's functionality. Since platform provides its functionality via its RESTful web-services, total of 76 API tests are deployed for each endpoint given in Table 4.7. In order to automate these test, Katalon Studio was used. Katalon makes it easy to run all tests automatically after each deployment and provides a record and run functionality. Test cases were prepared based on endpoint calls. Each test have 8 attributes described in Table 5.4. An example of the "sensor create" endpoint test is given in Table 5.5. All the 76 test cases were ran every time when a new feature was developed. Additionally, these tests cover aspects of integration test. As a result, all the test cases are successful.

GUI Tests

The aim of the GUI test is to ensure that the user interface provides the necessary GUI elements to interface with the platform. Each screen shown in Appendix B is tested manually on the Chrome Browser. All screen tests have passed.

Table 5.4: Test Case Attributes

ID	Indicates ID of the test
Aim	Indicates the aim of the test
Method	Indicates the used method in the test and can be a one of the “POST”, “GET”, “PUT”, “DELETE”
URL	Indicates the URL of the endpoint
Input:Query	Indicates the given URL parameter(s)
Input:Header	Indicates the header parameter(s)
Input:Payload	Indicates the given payload data at “POST” or “PUT” request
Response Code	Indicates the returned HTTP code which can be 200, 401, 500 XXXXX
Output	Indicates the returned data from the platform

Table 5.5: Sensor Create Test Case

ID	Test 10
Aim	Creating a new sensor on the platform
Method	POST
URL	`http://DUMMY_PCAD_URL/sensors`
Input:Query	-
Input:Header	access_token=VALID_ACCESS_TOKEN

Input:Payload	`{`
	`"type": "<RANDOM_TYPE>",`
	`"metric": "<RANDOM_METRIC>"`
	`}`

Response Code	200

Output	`{`
	`"id": "<GENERATED_ID>",`
	`"type": "<GIVEN_TYPE>",`
	`"metric": "<GIVEN_METRIC>",`
	`"registerDate": "<DATE_TIME>"`
	`}`

Chapter 6

Conclusion

This thesis offers a context-aware service infrastructure, and its implementation details for context-aware application development platform.

The main contribution of this study is to propose an infrastructure for Context Aware Computing. Also, the study includes implementation of the proposed infrastructure. The motivation behind developing such a platform is to provide a message broker based service infrastructure that is extensible, fast and easy to use. The platform is extensible because it is a service based and has ability to apply new requirements without making fundamental changes. It is fast because it uses MQTT as data delivery method and includes parallelism between data delivery and data process. MQTT provides better performance, low latency and overhead when compared to other communication protocols. Thus, the platform called A Platform for Context Aware Application Development-PCAD tries to ease development of context aware applications with its services. PCAD has inspired by operating system design. In this regard, PCAD offers its functionality via its services. The platform plays a middleware role between data providers and requestors. Middleware approach provides a platform to the providers that they can store their data. The platform also relieves requestors in a way that they obtain context data without concerning about context acquisition and storage. PCAD also enforces authorization and authentication in the aims of protecting resources against unauthorized access. The platform has multiple context services, communications and event handlers with the shared use of sensors and application programs in a wide range of disparate applications. Therefore, we expect that this thesis enhances our experience and contributes to the context-aware computing community and may help developers in building complex software platform developments.

Future Works

The platform can be improved further as listed below:

1. Rule Service will be simplified by giving only one set of syntax including the additions to solve the anomaly stated in Section 3.2.5.
2. Context Modeling Service will be implemented in order to make plain sensor data more suitable for context processing. This service will also include context reasoning functionality.
3. In the concept of scalability and distributed computing, the system is planned to fulfill those needs. The current system design allows the platform to work on a single machine.
4. Rule Service will be more efficient by checking only related rules rather than checking each rule for each provided data.
5. Reporting Service will be enriched via PCAD Query Language (PQL). PQL will provide a mechanism similar to SQL. In PQL, users will be able define filter in terms of logical operator(s), resource attribute(s) and desired value(s).
6. More unit, GUI and integration tests will be implemented.

BIBLIOGRAPHY

[1] Schilit, B. N., and M. M. Theimer, 1994. *Disseminating active map information to mobile hosts*. IEEE Network, vol. 8(5):22–32. ISSN 0890-8044. doi:10.1109/65.313011.

[2] Brown, P. J., J. D. Bovey, and Xian Chen, 1997. *Context-aware applications: from the laboratory to the marketplace*. IEEE Personal Communications, vol. 4(5):58–64. ISSN 1070-9916. doi:10.1109/98.626984.

[3] Ryan, N. S., J. Pascoe, and D. R. Morse, 1998. *Enhanced Reality Fieldwork: the Context-aware Archaeological Assistant*. Gaffney, V., M. van Leusen, and S. Exxon, editors, Computer Applications in Archaeology 1997, British Archaeological Reports. Oxford: Tempus Reparatum, 182–196.
URL http://www.cs.kent.ac.uk/pubs/1998/616

[4] Dey, A. K., 1998. *Context-Aware Computing: The CyberDesk Project*. AAAI 1998.

[5] Abowd, G. D., A. K. Dey, P. J. Brown, et al., 1999. *Towards a Better Understanding of Context and Context-Awareness*. Gellersen, H.-W., editor, Handheld and Ubiquitous Computing. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-540-48157-7, 304–307.

[6] Dey, A., 2001. *Understanding and Using Context*. Personal and Ubiquitous Computing, vol. 5:4–7. doi:10.1007/s007790170019.

[7] Hull, R., P. Neaves, and J. Bedford-Roberts, 1997. *Towards situated computing*. Digest of Papers. First International Symposium on Wearable Computers. 146–153. doi:10.1109/ISWC.1997.629931.

[8] Pascoe, J., N. S. Ryan, and D. R. Morse, 1998. *Human Computer Giraffe Interaction: HCI in the Field*. Johnson, C., editor, Workshop on Human Computer Interaction with Mobile Devices, GIST Technical Report G98-1. University of Glasgow, 182–196.
URL http://www.cs.kent.ac.uk/pubs/1998/617

[9] Rekimoto, J., Y. Ayatsuka, and K. Hayashi, 1998. *Augment-able reality: situated communication through physical and digital spaces*. Digest of Papers. Second International Symposium on Wearable Computers (Cat. No.98EX215). 68–75. doi:10.1109/ISWC.1998.729531.

[10] Salber, D., A. Dey, and G. Abowd, 1998. *Ubiquitous Computing: Defining an HCI Research Agenda for an Emerging Interaction Paradigm*.

[11] Schilit, B., N. Adams, and R. Want, 1994. *Context-Aware Computing Applications*. 1994 First Workshop on Mobile Computing Systems and Applications. 85–90. doi:10.1109/WMCSA.1994.16.

[12] Strang, T., and C. Linnhoff-Popien, 2004. *A Context Modeling Survey*. 34–41.

[13] Bolchini, C., C. Curino, E. Quintarelli, et al., 2007. *A data-oriented survey of context models*. ACM SIGMOD Record, vol. 36:19–26. doi:10.1145/1361348.1361353.

[14] Baldauf, M., S. Dustdar, and F. Rosenberg, 2007. *A Survey on context-aware systems*. Information Systems, vol. 2. doi:10.1504/IJAHUC.2007.014070.

[15] Wapforum. Available from: <http://www.wapforum.org>. [May 28, 2019].

[16] Hofer, T., W. Schwinger, M. Pichler, et al., 2003. *Context-awareness on mobile devices - the hydrogen approach*. 36th Annual Hawaii International Conference on System Sciences, 2003. Proceedings of the. 10 pp.–. doi:10.1109/HICSS.2003.1174831.

[17] Korpipää, P., and J. Mäntyjärvi, 2003. *An Ontology for Mobile Device Sensor-Based Context Awareness*. Lecture Notes in Artificial Intelligence

(Subseries of Lecture Notes in Computer Science), vol. 2680. 451–458. doi: 10.1007/3-540-44958-2_37.

[18] Atkinson, R., R. Garcia-Castro, J. Lieberman, et al. *Semantic Sensor Network Ontology*. Available from: <https://www.w3.org/TR/vocab-ssn/>. [May 28, 2019].

[19] Bellavista, P., A. Corradi, M. Fanelli, et al., 2013. *A Survey of Context Data Distribution for Mobile Ubiquitous Systems*. ACM Computing Surveys, vol. 45. doi:10.1145/2333112.2333119.

[20] Happ, D., N. Karowski, T. Menzel, et al., 2017. *Meeting IoT platform requirements with open pub/sub solutions*. Annales des Télécommunications, vol. 72:41–52.

[21] Banks, A., and R. Gupta. *MQTT Version 3.1.1 Plus Errata 01*. Available from: <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html>. [May 28, 2019].

[22] Luoto, A., and K. Systä, 2018. *Fighting network restrictions of request-response pattern with MQTT*. IET Software, vol. 12(5):410–417. ISSN 1751-8806. doi:10.1049/iet-sen.2017.0251.

[23] Collina, M., G. E. Corazza, and A. Vanelli-Coralli, 2012. *Introducing the QEST broker: Scaling the IoT by bridging MQTT and REST*. 2012 IEEE 23rd International Symposium on Personal, Indoor and Mobile Radio Communications - (PIMRC). ISSN 2166-9589, 36–41. doi:10.1109/PIMRC.2012.6362813.

[24] Sentilo. Available from: <http://www.sentilo.io/wordpress/>. [May 28, 2019].

[25] Román, M., C. K. Hess, R. Cerqueira, et al., 2002. *Gaia: A middleware platform for active spaces*. Mobile Computing and Communications Review, vol. 6:65–67.

[26] Fahy, P., and S. Clarke, 2004. *CASS-Middleware for Mobile Context-Aware Applications*. MobiSys.

[27] Chen, H., T. Finin, and A. Joshi, 2003. *An Intelligent Broker for Context-Aware Systems.* Adjunct Proceedings of UbiComp.

[28] Biegel, G., and V. Cahill, 2004. *A framework for developing mobile, context-aware applications.* Second IEEE Annual Conference on Pervasive Computing and Communications, 2004. Proceedings of the. 361–365. doi: 10.1109/PERCOM.2004.1276875.

[29] Dey, A., and G. Abowd, 2000. *The Context Toolkit: Aiding the Development of Context-Aware Applications.*

[30] Pokraev, S., J. Koolwaaij, M. van Setten, et al., 2005. *Service platform for rapid development and deployment of context-aware, mobile applications.* ISBN 0-7695-2409-5, 646. doi:10.1109/ICWS.2005.106.

[31] Firner, B., R. S. Moore, R. Howard, et al., 2011. *Poster: Smart Buildings, Sensor Networks, and the Internet of Things.* Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems, SenSys '11. New York, NY, USA: ACM. ISBN 978-1-4503-0718-5, 337–338. doi:10.1145/2070942.2070978.
URL http://doi.acm.org/10.1145/2070942.2070978

[32] Devaraju, A., S. Hoh, and M. Hartley, 2007. *A Context Gathering Framework for Context-aware Mobile Solutions.* Proceedings of the 4th International Conference on Mobile Technology, Applications, and Systems and the 1st International Symposium on Computer Human Interaction in Mobile Technology, Mobility '07. New York, NY, USA: ACM. ISBN 978-1-59593-819-0, 39–46. doi:10.1145/1378063.1378070.
URL http://doi.acm.org/10.1145/1378063.1378070

[33] Karaçalık, O., 2018. *An Actor Model Based Platform for Developing Context-aware Applications.* Turkey: CoHE THESIS CENTER.

[34] Celikkan, U., and K. Kurtel, 2015. *A Platform for Context-Aware Application Development: PCAD.* 1481–1488. doi:10.15439/2015F49.

[35] Silberschatz, A., P. B. Galvin, and G. Gagne, 2012. Operating System Concepts. Wiley Publishing, 9th edn. ISBN 1118063333, 9781118063330.

[36] C. Hu, V., D. Ferraiolo, R. Kuhn, et al. *Guide to Attribute Based Access Control (ABAC) Definition and Considerations*. Available from: <https://nvlpubs.nist.gov/nistpubs/specialpublications/NIST.sp.800-162.pdf>. [May 28, 2019].

[37] CacheControl. *Json Rule Engine*. Available from: <https://github.com/CacheControl/json-rules-engine>. [May 28, 2019].

[38] Yıldırım, O. *Role and Attribute based Access Control for Node.js*. Available from: <https://github.com/onury/accesscontrol>. [May 28, 2019].

[39] Guner, A., K. Kurtel, and U. Celikkan, 2017. *A message broker based architecture for context aware IoT application development*. 2017 International Conference on Computer Science and Engineering (UBMK). 233–238. doi: 10.1109/UBMK.2017.8093381.

[40] McKinney, R. *LoopBack Juggler*. Available from: <https://loopback.io/doc/en/lb3/Advanced-topics-data-sources.html>. [May 28, 2019].

Appendix A

Java Application Programming Interface

An application developer who would like to use Java API can download .jar file from github. After adding this library to project, methods described in Table A.1 are available to use.

Table A.1: Interaction Mechanism Service Usage

Name	**Description**
initPcadInterface	It accepts credential of a user with options array which consist of header parameters. This method initializes PCAD interface.
getPcadResource	This method gets data from PCAD for the given resource name.
getPcadResource	This method gets data from PCAD for the given resource name and resource ID.
savePcadResource	This method creates new resource data for the given resource name.
savePcadResource	This method updates existing resource data for the given resource name and resource ID.

PCAD Initializer

```
Signature:
  Object initPcadInterface(String username, String password, HashMap<
      String,String> options)
Description:
  It accepts credential of a user with options array. These options
  consist of header parameters.
Parameters:
  username: Selected username or e-mail of a user in registration process
  password: Selected password of a user.
  Option Values:
    'Accept': Defines desired data format in response of API calls
    'Content Type': Defines sending data format in request
Return Value:
  It returns an object including id, ttl.
    id : access_token that should be used in subsequent API calls.
    ttl: valid time value of given access_token in seconds.
Example Usage:
  HashMap<String, String> options = new HashMap<>();
  options.put("Accept", "application/json");
  options.put("Content Type", "application/json");
  PCAD pcad = new PCAD("<USERNAME>", "<PASSWORD>", options);
  Object pcadObject = pcad.initPcadInterface();
```

Listing A.1: PCAD Initialization

Data Retrieval Function

```
Signature:
  JSONArray getPcadResource(String resourceName, HashMap<String,String>
       options)
Description:
  getPcadResource is used for data retrieval.
Parameters:
  resourceName: Defines the desired resource name such as sensorData.
  options:
    'Accept': Defines desired data format in response of API calls.
    'Content Type': Defines sending data format in request.
    'access_token': Obtained access token in PCAD initialization.
    'filter': JSON filter object
Return Value:
  Returns a JSON array of desired resource objects
Example Usage:
  HashMap<String, String> options = new HashMap<>();
  options.put("Accept", "application/json");
  options.put("Content Type", "application/json");
  options.put("access_token", "<ACCESS_TOKEN>");
  JSONArray pcadObject = pcad.getPcadResource("<RESOURCE_NAME>",
      options);
```

Listing A.2: Data Retrieval from PCAD

Data Retrieval Function with Resource ID

```
Signature:
  JSONObject getPcadResource(String resourceName, String resourceId,
      HashMap<String,String> options)
Description:
  It used for data retrieval with ID of resource.
Parameters:
  resourceName: Defines the desired resource name such as sensorData.
  resourceName: Defines ID desired resource.
  options:
    'Accept': Defines desired data format in response of API calls.
    'Content Type': Defines sending data format in request.
    'access_token': Obtained access token in PCAD initialization.
    'filter': JSON filter object
Return Value:
  Returns a JSON object of desired resource.
Example Usage:
  HashMap<String, String> options = new HashMap<>();
  options.put("Accept", "application/json");
  options.put("Content Type", "application/json");
  options.put("access_token", "<ACCESS_TOKEN>");
  JSONObject pcadObject = pcad.getPcadResource("<RESOURCE_NAME>",
                "<RESOURCE_ID>", options);
```

Listing A.3: Data Retrieval from PCAD with Resource ID

Data Save Function

```
Signature:
  JSONObject savePcadResource(String resourceName, Object resource,
      HashMap<String,String> options)
Description:
  It is used for saving a resource to the PCAD.
Parameters:
  resourceName: Defines name of the resource that want to create in
      PCAD.
  resource: Contains resource attributes
  options:
    'Accept': Defines desired data format in response of API calls.
    'Content Type': Defines sending data format in request.
    'access_token': Obtained access token in PCAD initialization.
Return Value:
  Returns a created instance of the resource as JSONObject.
Example Usage:
  HashMap<String, String> options = new HashMap<>();
  HashMap<String, String> resource = new HashMap<>();
  options.put("Accept", "application/json");
  options.put("Content Type", "application/json");
  options.put("access_token", "<ACCESS_TOKEN>");
  resource.put("<ATTRIBUTE>", "<VALUE>");
  resource.put("<ATTRIBUTE>", "<VALUE>");
  JSONObject pcadObject = pcad.savePcadResource("<RESOURCE_NAME>",
                resource, options);
```

Listing A.4: Save data to PCAD

Data Update Function

```
Signature:
  JSONObject savePcadResource(String resourceName, Object resource,
      String resourceId, HashMap<String,String> options)
  Description:
  It is used for updating a resource to the PCAD.
Parameters:
  resourceName: Defines name of the resource that want to update.
  resource: Contains desired attributes that will be updated.
  options:
    'Accept': Defines desired data format in response of API calls.
    'Content Type': Defines sending data format in request.
    'access_token': Obtained access token in PCAD initialization.
Return Value:
  Returns a update instance of the resource as JSONObject.
Example Usage:
  HashMap<String, String> options = new HashMap<>();
  HashMap<String, String> resource = new HashMap<>();
  options.put("Accept", "application/json");
  options.put("Content Type", "application/json");
  options.put("access_token", "<ACCESS_TOKEN>");
  resource.put("<ATTRIBUTE>", "<VALUE>");
  resource.put("<ATTRIBUTE>", "<VALUE>");
  JSONObject pcadObject = pcad.updatePcadResource("<RESOURCE_NAME>",
                resource, <RESOURCE_ID>, options);
```

Listing A.5: Update data in PCAD

JAVA Example for connection to MQTT

Example of connecting to MQTT using JAVA listed in Listing A.6. In this example, value of variable “topic” is the unique topic for the requestor described in the demonstration of Interaction Mechanism 6 in Section 4.3. This example requires “Eclipse Paho” MQTT library. In this example, when a new data arrives from MQTT, *messageArrived* method is called implicitly as callback method.

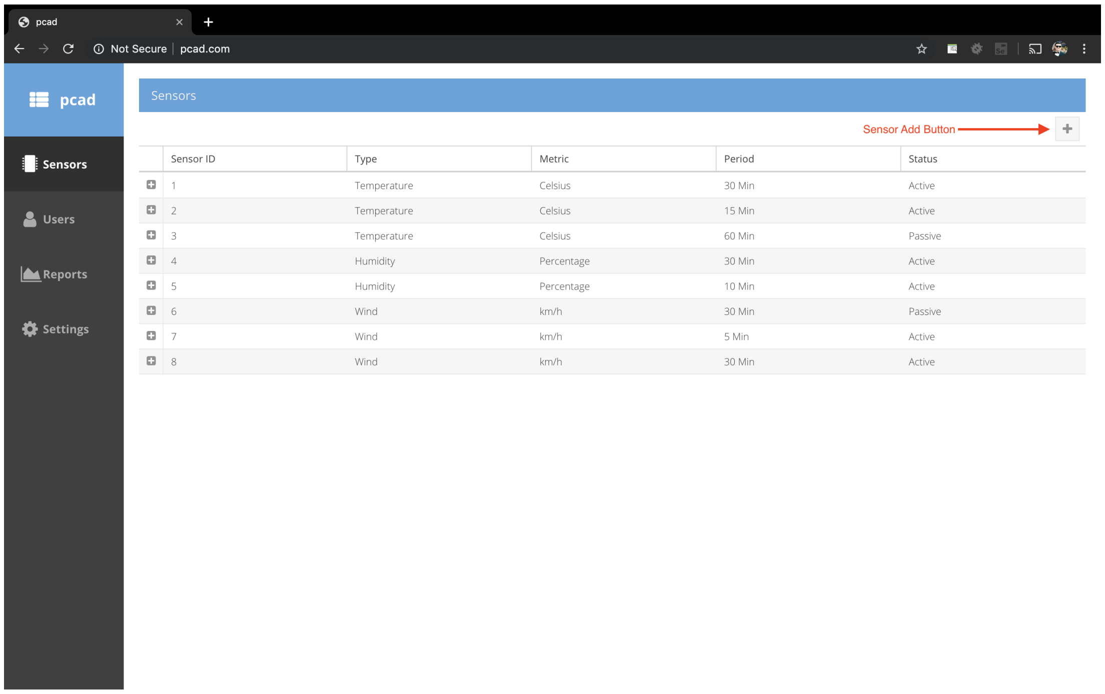
```java
public class pcadClient implements MqttCallback {
  private String topic     = "mytopic";
  private String broker    = "tcp://localhost:1883";
  private String clientId  = "JavaSample";
  private MemoryPersistence persistence = new MemoryPersistence();

  public static void main(String[] args) {
      pcadClient pClient = new pcadClient();
      pClient.subscribe(pClient.topic);
  }

  private void subscribe(String topic){
      MqttClient sampleClient = null;
      MqttConnectOptions connOpts = new MqttConnectOptions();
      connOpts.setCleanSession(true);
      try {
          sampleClient = new MqttClient(broker, clientId, persistence);
          sampleClient.connect(connOpts);
          sampleClient.subscribe(topic);
      } catch (MqttException e) {
          e.printStackTrace();
      }
      sampleClient.setCallback(this);
  }
  @Override
  public void messageArrived(String topic, MqttMessage message) {
      System.out.println(message);
  }
  @Override
  public void connectionLost(Throwable cause) {}
  @Override
  public void deliveryComplete(IMqttDeliveryToken token) {}
}
```

Listing A.6: Connecting as subscriber to MQTT

Appendix B

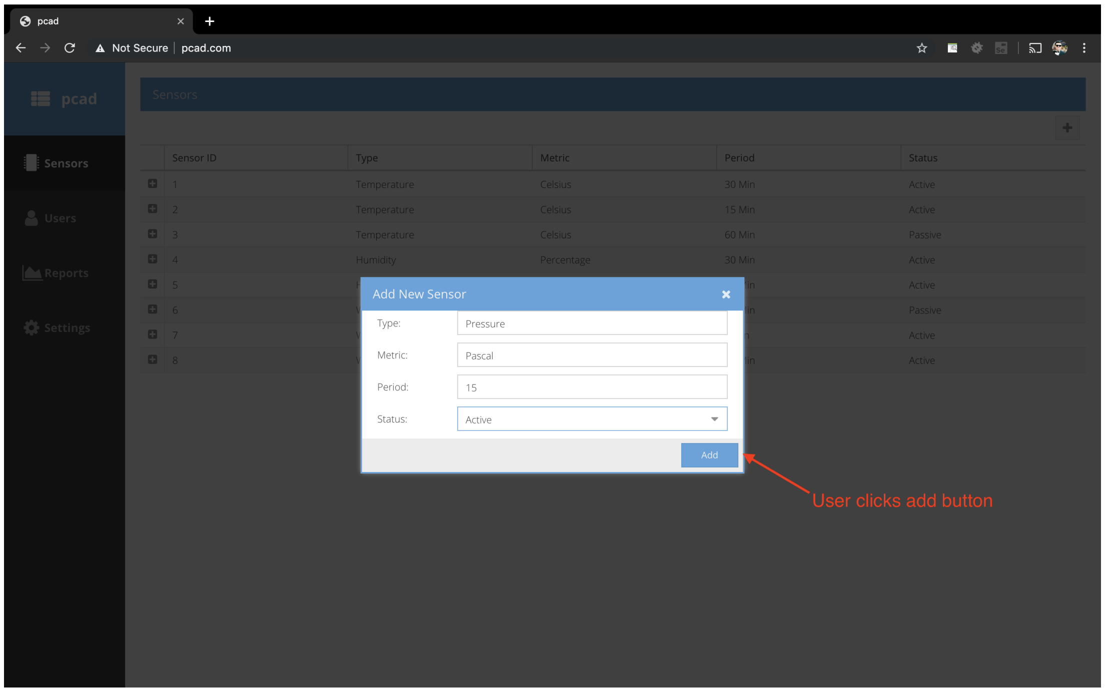
User Interfaces

Sensor Operations



Sensor ID	Type	Metric	Period	Status
1	Temperature	Celsius	30 Min	Active
2	Temperature	Celsius	15 Min	Active
3	Temperature	Celsius	60 Min	Passive
4	Humidity	Percentage	30 Min	Active
5	Humidity	Percentage	10 Min	Active
6	Wind	km/h	30 Min	Passive
7	Wind	km/h	5 Min	Active
8	Wind	km/h	30 Min	Active

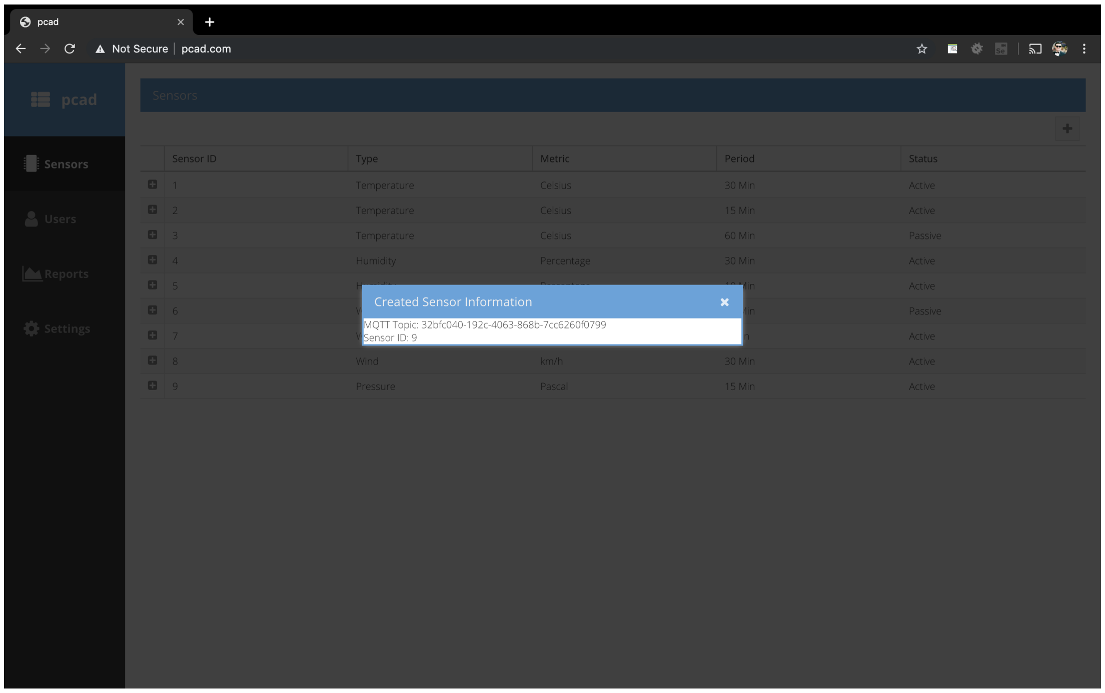
Figure B.1: Listings Sensor



Sensor ID	Type	Metric	Period	Status
1	Temperature	Celsius	30 Min	Active
2	Temperature	Celsius	15 Min	Active
3	Temperature	Celsius	60 Min	Passive
4	Humidity	Percentage	30 Min	Active
5	[illegible]	[illegible]	[illegible]	Active
6	[illegible]	[illegible]	[illegible]	Passive
7	[illegible]	[illegible]	[illegible]	Active
8	[illegible]	[illegible]	[illegible]	Active

Type: Pressure
Metric: Pascal
Period: 15
Status: Active

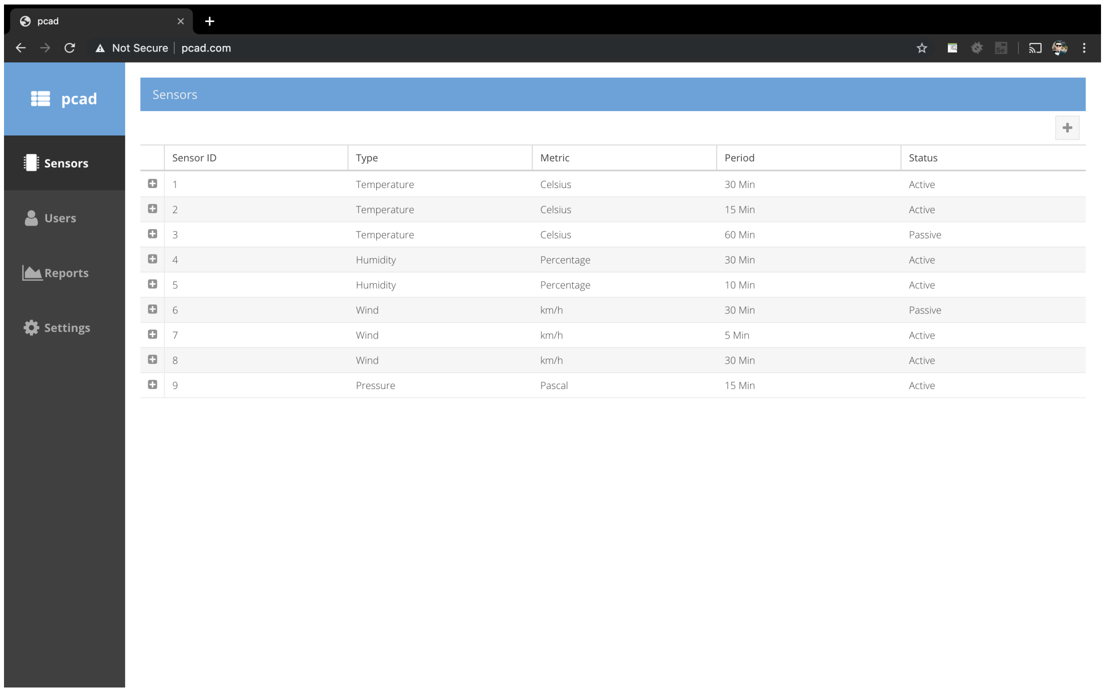
Figure B.2: Adding Sensor



Sensor ID	Type	Metric	Period	Status
1	Temperature	Celsius	30 Min	Active
2	Temperature	Celsius	15 Min	Active
3	Temperature	Celsius	60 Min	Passive
4	Humidity	Percentage	30 Min	Active
5	[illegible]	[illegible]	[illegible]	Active
6	[illegible]	[illegible]	[illegible]	Passive
7	[illegible]	[illegible]	[illegible]	Active
8	Wind	km/h	30 Min	Active
9	Pressure	Pascal	15 Min	Active

MQTT Topic: 32bfc040-192c-4063-868b-7cc6260f0799
Sensor ID: 9

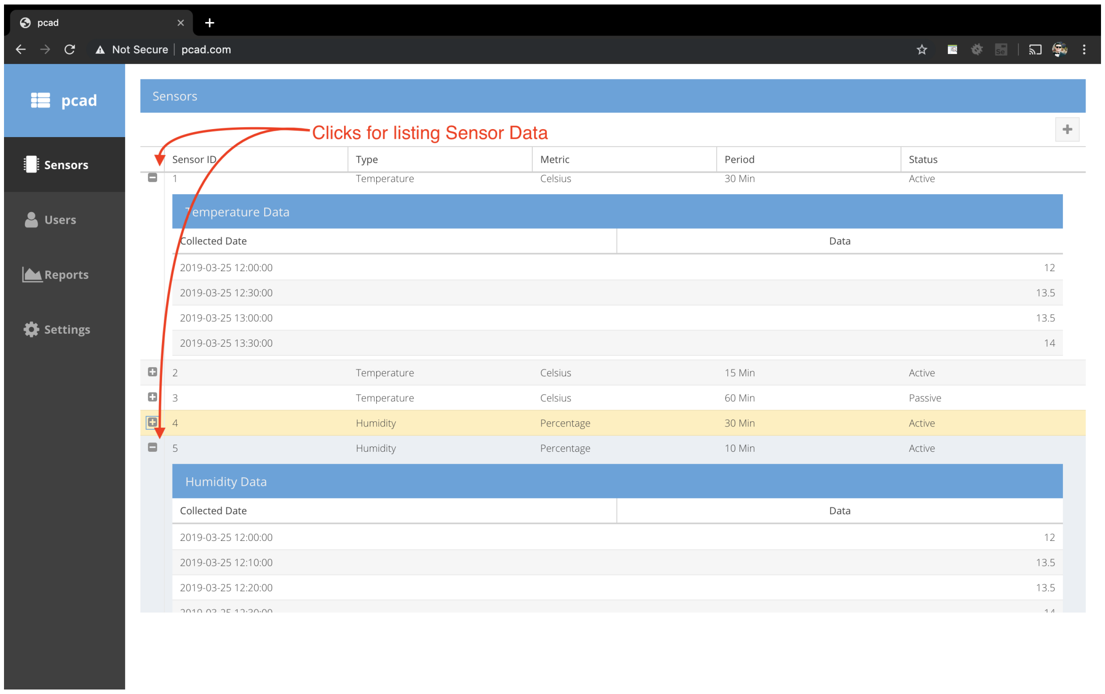
Figure B.3: Created Sensor Information



Sensors

Sensor ID	Type	Metric	Period	Status
1	Temperature	Celsius	30 Min	Active
2	Temperature	Celsius	15 Min	Active
3	Temperature	Celsius	60 Min	Passive
4	Humidity	Percentage	30 Min	Active
5	Humidity	Percentage	10 Min	Active
6	Wind	km/h	30 Min	Passive
7	Wind	km/h	5 Min	Active
8	Wind	km/h	30 Min	Active
9	Pressure	Pascal	15 Min	Active

Figure B.4: Created Sensor



Sensors

Sensor ID	Type	Metric	Period	Status
1	Temperature	Celsius	30 Min	Active

Temperature Data

Collected Date	Data
2019-03-25 12:00:00	12
2019-03-25 12:30:00	13.5
2019-03-25 13:00:00	13.5
2019-03-25 13:30:00	14

Sensor ID	Type	Metric	Period	Status
2	Temperature	Celsius	15 Min	Active
3	Temperature	Celsius	60 Min	Passive
4	Humidity	Percentage	30 Min	Active
5	Humidity	Percentage	10 Min	Active

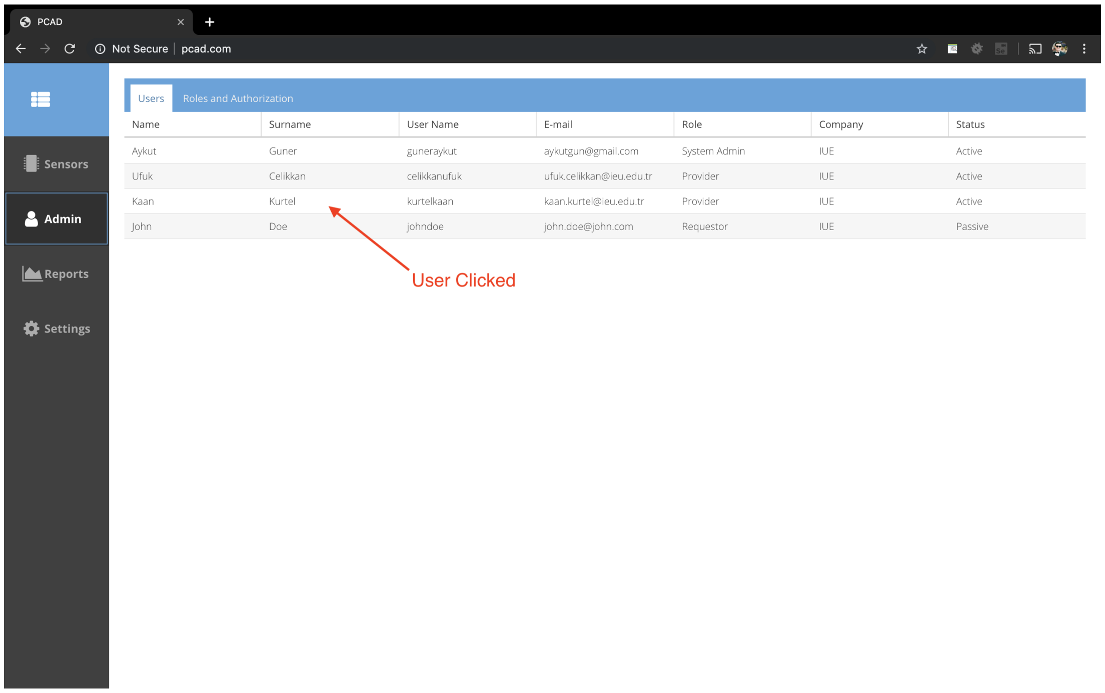
Humidity Data

Collected Date	Data
2019-03-25 12:00:00	12
2019-03-25 12:10:00	13.5
2019-03-25 12:20:00	13.5

Figure B.5: Listing Sensor Data

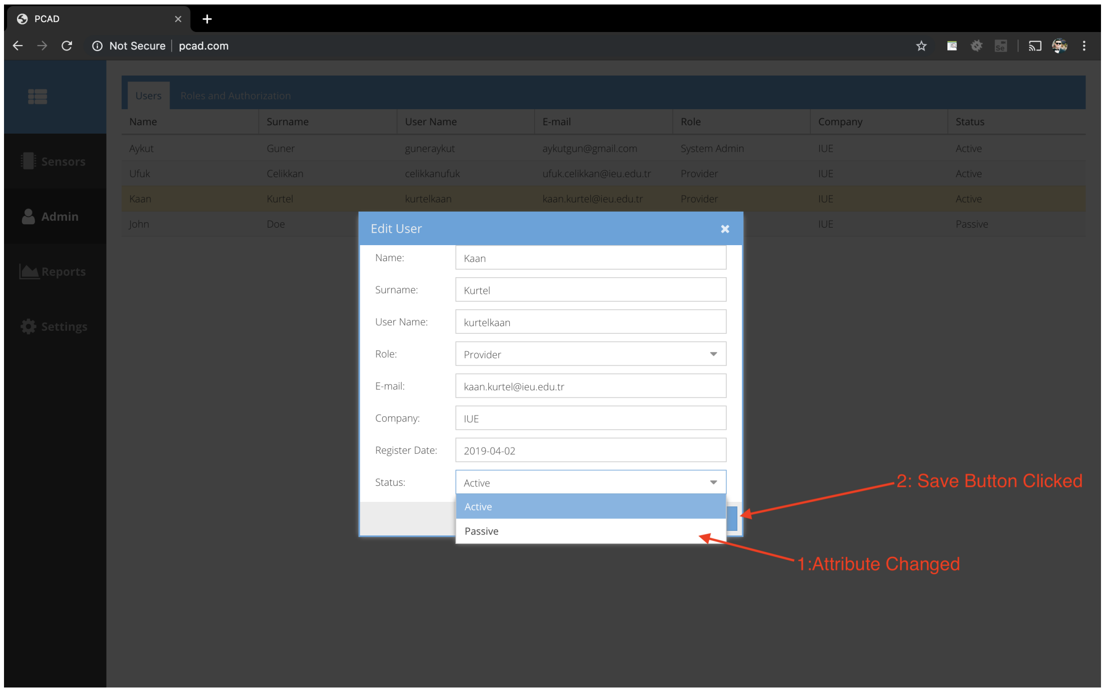
User, Role and Authorization Operations

User Listing and Editing



Name	Surname	User Name	E-mail	Role	Company	Status
Aykut	Guner	guneraykut	aykutgun@gmail.com	System Admin	IUE	Active
Ufuk	Celikkan	celikkanufuk	ufuk.celikkan@ieu.edu.tr	Provider	IUE	Active
Kaan	Kurtel	kurtelkaan	kaan.kurtel@ieu.edu.tr	Provider	IUE	Active
John	Doe	johndoe	john.doe@john.com	Requestor	IUE	Passive

Figure B.6: Listings Users as *System Admin*



Name	Surname	User Name	E-mail	Role	Company	Status
Aykut	Guner	guneraykut	aykutgun@gmail.com	System Admin	IUE	Active
Ufuk	Celikkan	celikkanufuk	ufuk.celikkan@ieu.edu.tr	Provider	IUE	Active
Kaan	Kurtel	kurtelkaan	kaan.kurtel@ieu.edu.tr	Provider	IUE	Active
John	Doe				IUE	Passive

Name:	Kaan
Surname:	Kurtel
User Name:	kurtelkaan
Role:	Provider
E-mail:	kaan.kurtel@ieu.edu.tr
Company:	IUE
Register Date:	2019-04-02
Status:	Active

Active
Passive

Figure B.7: Updating Users as *System Admin*

Role Listing and Creation

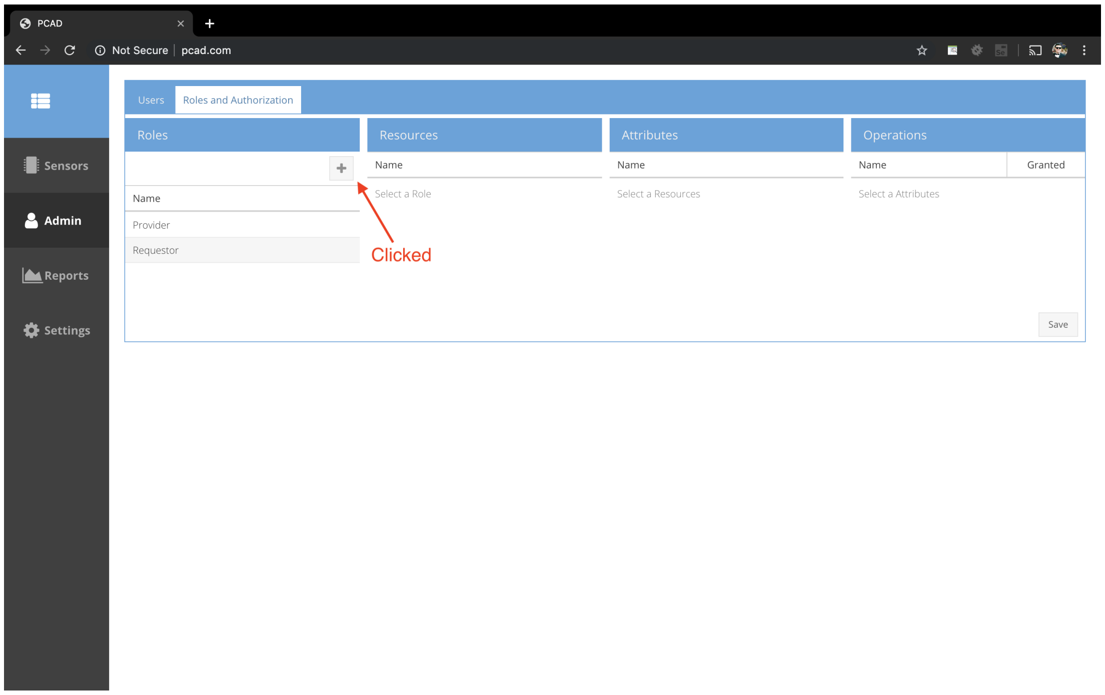


Figure B.8: *System Admin* lists Roles and adds a new Role

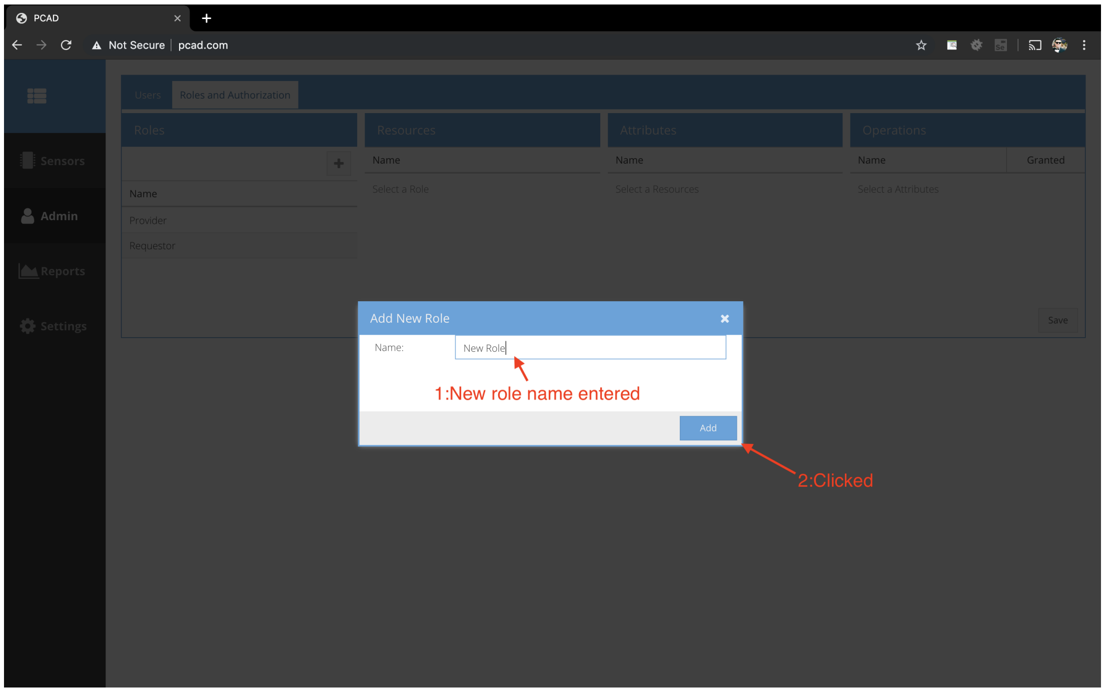


Figure B.9: Adding a Role as *System Admin*

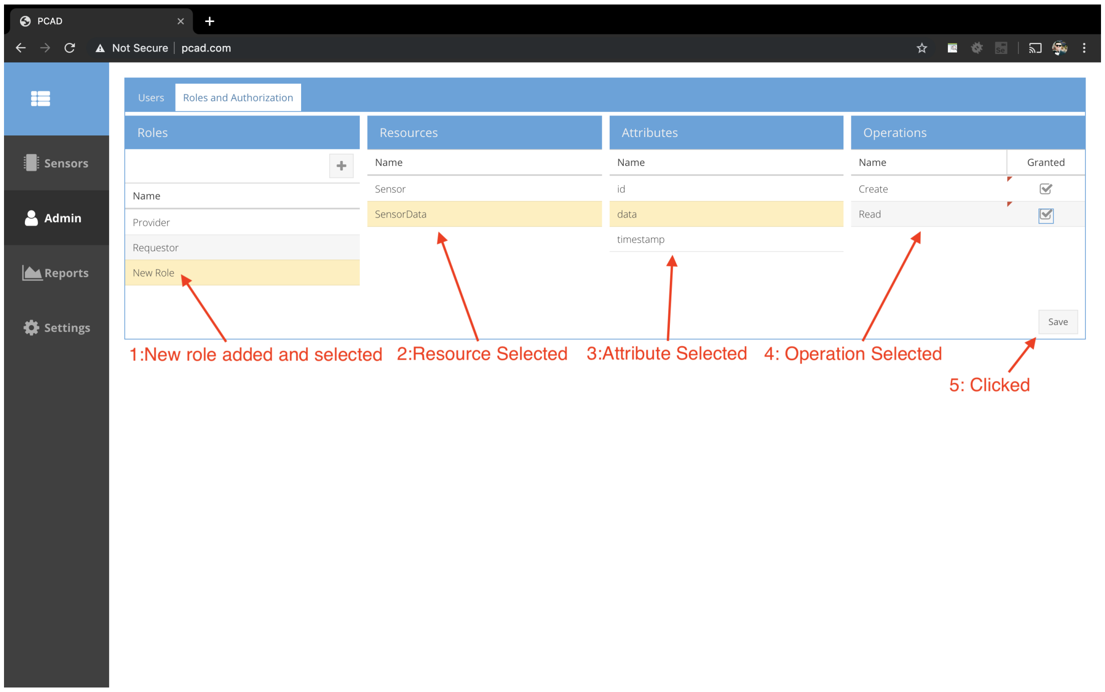


Figure B.10: *System Admin* modifies the new Role

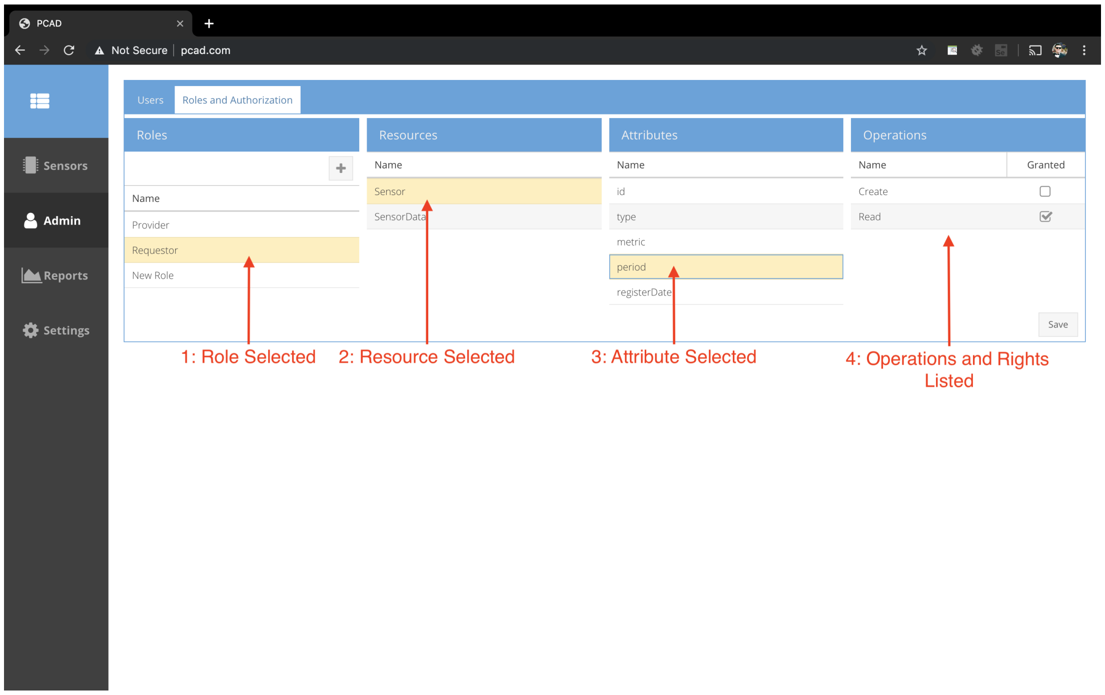


Figure B.11: Listing Role and its Grants as *System Admin*

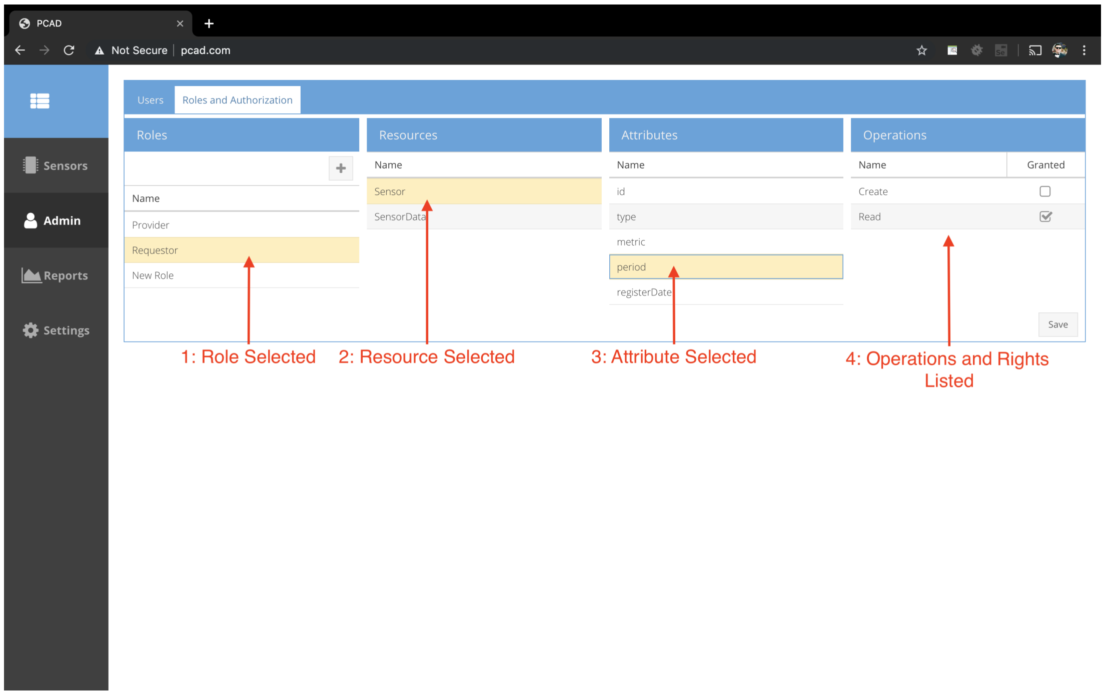


Figure B.12: Listing Role and its Grants as *System Admin*

Appendix C

Installation Guide

PCAD and its toolkits are configured and bundled to run as a Docker Container[1]. This facilitates installation and distribution of PCAD. In order to run PCAD, first Docker[2], Docker Compose[3] and Git[4] should be installed. Afterwards, just few commands are enough to install and use PCAD explained in Appendix C Standalone PCAD Installation.

Installations

This guide gives instructions for Docker, Docker Compose, Git and standalone PCAD installations on macOS Mojave with version 10.14.5, Ubuntu with version 18.04.2 LTS, CentOS with version 7.6.1810 and Windows with version Windows 10 Pro N build (17763.557).

[1]https://www.docker.com/resources/what-container
[2]https://www.docker.com/
[3]https://docs.docker.com/compose/overview/
[4]https://git-scm.com/

macOS Installation

- Go to the link[5] download Docker and follow the instructions on the setup screen
- Go to the link[6] download Git and follow the instructions on the setup screen
- Once both installations are completed, proceed to the Appendix C Standalone PCAD Installation

Ubuntu Installation

- Open a terminal window
- Navigate to desired installation location
- Write the following command

```
sudo apt-get install -y apt-transport-https ca-certificates git curl gnupg-agent software-properties-common \
        && curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add - \
        && sudo apt-key fingerprint 0EBFCD88 \
        && sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" \
        && sudo apt-get update && sudo apt-get install -y docker-ce docker-ce-cli containerd.io && sudo apt install -y docker-compose \
        && git clone https://github.com/guneraykut/pcad-docker.git \
        && cd pcad-docker \
        && sudo docker-compose up -d
```

Listing C.1: Docker, Git and PCAD Installation Commands for Ubuntu

[5]https://hub.docker.com/editions/community/docker-ce-desktop-mac
[6]https://git-scm.com/download/mac

CentOS Installation

- Open a terminal window
- Navigate to desired installation location
- Write the following command

```
yum install -y yum-utils device-mapper-persistent-data lvm2 \
    && yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo \
    && yum install -y git docker-ce docker-ce-cli containerd.io \
    && sudo curl -L "https://github.com/docker/compose/releases/download/1.24.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose \
    && sudo chmod +x /usr/local/bin/docker-compose \
    && sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose \
    && systemctl start docker \
    && git clone https://github.com/guneraykut/pcad-docker.git \
    && cd pcad-docker \
    && docker-compose up -d
```

Listing C.2: Docker, Git and PCAD Installation Commands for CentOS

Windows Installation

- Go to the link[7] download Docker and follow the instructions on the setup screen and do not choose "use windows containers" option
- If Hyper-V was not enabled, setup will give a warning after installation. Follow the instructions on the screen to enable Hyper-V
- Go to the link[8] download Git and follow the instructions on the setup screen

[7]https://hub.docker.com/editions/community/docker-ce-desktop-windows
[8]https://git-scm.com/download/windows

- Once both installations are completed, proceed to the Appendix C Standalone PCAD Installation

Standalone PCAD Installation

This section describes PCAD installation provided that Docker, Docker Compose and Git have already been installed. The steps are listed in Listing C.3.

- Open a terminal window (or PowerShell on Windows)
- Navigate to desired installation location
- Write the following command

```
git clone https://github.com/guneraykut/pcad-docker.git \
&& cd pcad-docker \
&& docker-compose up -d
```

Listing C.3: Standalone PCAD Installation Commands

After the installation, PCAD API's explorer is available at `http://localhost:3000/explorer`. Default installation uses port number 3000 on the host machine. In the case that this port is used by another application, the port must be changed in the "docker-composer.yml" file that is downloaded in the directory "pcad-docker" during the installation. Find the line "3000:3000" and change the port number 3000 on the left side of ":" sign to the new value. Then, the user can connect to PCAD through that port as follows: `http://localhost:<NEW_PORT_NUMBER>/explorer`. This change must be done before "docker-compose up -d" command is executed.