Institute for Visualization and Interactive Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit Nr. 311080009

# Rule Based Inference and Action Selection Based on Monitoring Data in IoT

Arash Fasihi

| | |
|---|---|
| **Course of Study:** | InfoTech |
| **Examiner:** | Prof. Dr. Albrecht Schmidt |
| **Supervisor:** | Dipl.-Inf. Andreas Kopecki |
| **Commenced:** | December 1, 2015 |
| **Completed:** | June 1, 2016 |
| **CR-Classification:** | H.5.2 |

# Acknowledgements

It is with immense gratitude that I acknowledge the support and help of Prof. Dr. Albrecht Schmidt.

I would like to gratefully acknowledge the guidance, help and motivation of my supervisor, Andreas Kopecki.

I would like to thank the HCI department specially Anja Mebus for being so kind and helpful to me, all the time.

# Abstract

The current trend in IoT is to find the ultimate solution to integrate objects to the body of Internet to communicate. Once IoT applications are able to incorporate "Things" effortlessly, handling the transferred data is the major challenge. For IoT platforms, when they are mature enough to plug in things with minimal effort, the future research will be around software frameworks. it is fair that IoT in its early years of existence pay much attention to the engagement of things. However, it is predictable that in the future the trend in IoT researches will fall in software area. A typical IoT platform already includes a software framework to handle and manage data. An IoT software framework is a "Rule-Engine" capable of making decisions based on received data. "Expert Systems" has already been on research to address this problem. However, the emergence of IoT will open new doors to this field. Making rule engines for IoT applications differs in that they will process data that is inherently different.

In this thesis, an IoT software framework with good level of extensibility is offered which allows developers to easily make IoT solutions on top of that. To analyze data streams, there is an interface to host Machine Learning algorithms, together with other interfaces common for an IoT application. To plug in new extensions, the developer is free to develop their own extensions from scratch or to use some other IoT platforms to integrate new modules.

# Contents

# List of Figures

# List of Listings

# 1 Introduction

Currently, the main communication pattern on the Internet is human-human. With the rise of smart objects, intelligent things in physical world are becoming smaller, at the same time smarter. In near future, most objects will have a unique way of identification so that they can be connected to the Internet. So, the Internet will become Internet of Things (IoT). The communication patterns will not be restricted to only human-human. Consequently, human-thing and even thing-thing (also called M2M) are new patterns in connected world [TW10]. The main promise of IoT is to enable a variety of things to be pervasively present around us which through unique addressing schemes, are able to interact with each other and cooperate with their neighbors to reach common goals [Giu10]. Along the same line, in the sophisticated world of IoT the mechanisms managing and utilizing the resulting significant volume of data from the connected objects has yet to match the maturity of the technology itself [AA12].

IoT already includes several domain of applications. These applications can be categorized based on the different network availabilities, scale, coverage and type of user involvement [GKN+11]. IoT applications can be classified into six kind of domains, they are smart homes and smart buildings, automation, mobile communication, smart business, health-care, and utilities [AMN15].

## 1.1 Motivation for IoT Frameworks

Developing IoT applications has been always a matter of challenge due to complexity of areas involved, ranging from Networks and Communication to Data Management and Security [VF13]. To pave the road for IoT application developers,a variety of IoT frameworks and platforms with varying capabilities have come to scene.

The introduction of Web 2.0 in 2004 changed the approach for Web application development, dramatically[1]. Recent efforts for integrating the IoT into the Web 2.0 obtained

[1] http://www.oreilly.com/pub/a/web2/archive/what-is-web-20.html

some considerable results, such as Pachube [Haq04] and SensorMap [NLM+06] , helping the users with providing them with a Web platform capable of visualizing networked things in a similar way as Google Maps does for Points of Interest. It seems that a complete integration of smart things into the Web 2.0 will only be accomplished when users are able to develop, deploy and exploit their own IoT applications as they already do for Website and on-line contents. The key enablers for the success of such an integration are 1) the adoption of IP capable open standards for thing communications , and 2) a user friendly application design framework for smart things. For example, WebIoT is a novel Web application framework for the Internet of Things, characterized by a flexible design and a user friendly interface,that makes it possible to build a very wide spectrum of IoT applications [CDBZ12].

IoT frameworks can help support the communication among things and allow for more sophisticated computation areas like distributed computing and the development of distributed applications. Currently, IoT frameworks seem to focus on real-time data logging solutions like Pachube, offering some basis to work with many things and make them interact. Future developments may result in specific software development environments to develop the software to work with the hardware used in the Internet of Things [KGB].

In terms of the technology and engineering aspects of IoT, currently, there may not be a clear separation between the hardware and software platforms, but it is obvious that the majority of vendors and providers focus on the hardware side[2]. Few vendors in the industry currently offer IoT software platforms, for example, out of the top 100 IoT startups ranked by Mattermark (based on the total funding they received), only about 13 startups provide IoT software platforms[3].

Another important issue is the role of Cloud in IoT. A software framework can be either hosted by an IoT Cloud platform to manage data or work out of the context of Cloud. The both methods have pros and cons.

In IoT world, most IoT platforms host developer's applications to run on Cloud. In fact they are Cloud-based platforms. It is beneficial to rely on Cloud platforms to make an IoT application work. A Cloud platform is a foundation for running applications and storing data. The biggest issue is that it runs in data centers owned by an external service provider, such as Microsoft, Amazon, Google, IBM, Rack Space etc., and it is accessed via the Internet. These are the advantages of using the Cloud platform: Faster deployment of new business capabilities, lower risk business innovation, global scale

[2] https://dzone.com/articles/iot-software-platform-comparison
[3] http://www.forbes.com/sites/louiscolumbus/2015/10/25/the-top-100-internet-of-things-startups-of-2015/

and global reach, and more intelligent IT spending [KGB]. Among featured IoT Cloud platforms we have AWS IoT[4], IBM Bluemix[5], and Xively[6].

However, despite the fact that many believe Cloud is the ultimate computation solution, these are the risks of using the IoT Cloud platform: outsourcing to an external provider, storing data outside your organization (This is clearly a risk but how much risk is appropriate or acceptable is a business decision.), vendor lock-in (so far there is no easy way to port an application developed for one Cloud provider to another Cloud provider.) [KGB].

## 1.2 Scope of Work

In this thesis, we will offer an IoT software framework for IoT application developers to enable them develop their own IoT applications on top of the offered framework. The framework, despite the fact that enables the developer to run their IoT application on their own preferred servers, provides them with the possibility to get the data from Cloud, if necessary. On the other hand, the offered software framework can be a part of IoT Cloud platform for data analysis and management, as well. For this framework we suppose that healthy data streams flow in, without any need for data fusion, filtering and aggregation. In fact, to be able to develop IoT applications using the offered framework one can either have the application run on their own server or have a Cloud platform host the application.

Regardless of the fact that our offered software framework will be hosted by an IoT Cloud platform or not, the goal of this thesis is to come up with a software framework with a plug-in model in order to ease the integration of software modules without any need for reconfiguration of the software architecture. The key aspect for such a framework will be high level of software extensibility, with well-defined interfaces as much as possible. The interfaces will be developed in such a way that integrating new extension of one interface does not affect other extensions. That means for such a software developed on top of this framework, the logic to handle the data stream coming from Hypertext Transfer Protocol (HTTP) channel will be the same for handling the data stream coming from Message Queue Telemetry Transport (MQTT) channel and so on. On the other hand, to analyze the received data, there will be an interface to incorporate Supervised Machine Learning algorithms. This is the responsibility of IoT application developer to plug in suitable algorithm, according to the use-case.

[4] https://aws.amazon.com/iot/
[5] http://www.ibm.com/cloud-computing/bluemix/internet-of-things/
[6] https://www.logmeininc.com/

## Structure

The remaining of the document includes three chapters with this structure:

**Chapter 2 – IoT Applications Basics:** In this chapter, we will have a look at the basics required to develop IoT applications. IoT ecosystem will be briefly discussed in this chapter.

**Chapter 3 – Fundamentals of IoT Platforms and Frameworks:** In this chapter, we will discuss the building blocks for IoT platforms. Then the role of a software frameworks as the rule engine for IoT platforms will be discussed. Finally, we will probe some current approaches to make a rule engine in IoT platforms.

**Chapter 4 – Design:** In this chapter the architecture and the functionality of our framework will be introduced.

**Chapter 5 – Implementation:** In this chapter, an implementation for the design discussed in previous chapter will be introduced.

**Chapter 6 – Summary and Future Work** In this chapter, we will conclude our discussion and then describe possible future improvements for our work.

# 2 IoT Applications Basics

IoT is a communication paradigm that visualizes a near future, in which the objects of everyday life will be equipped with micro-controllers, transceivers for digital communication. Moreover, suitable protocol stacks that will make them able to communicate with one another and with the users are necessary. Therefore, IoT aims at making the Internet even more and more pervasive [ZBC+14].

## 2.1 Concepts

Compared to the traditional Internet, IoT has three outstanding characteristics which we will discuss them briefly in this section. The first obvious characteristic is the overall perception in IoT, that means IoT collects data about objects through sensing devices like Radio-Frequency IDentification (RFID) sensors, two-dimensional code method etc., wherever and whenever needed. The second characteristic for IoT is reliable transmission. The core technology of IoT is nothing but Internet. The Internet of Things relies on the Internet infrastructure to send data about objects to processing centers. The need for accuracy, safety, and real-time transmission is unavoidable. The third characteristic is intelligent processing. The two previous characteristics constitute a network of sensors for IoT applications. However, IoT offers intelligent processing of data about “Things”, as well [XJH13]. IoT has come to extend the Internet into real world by taking in the everyday, everywhere objects. In new usage of Internet, objects are no longer separated from the virtual world but can be controlled from everywhere, anytime. In IoT, objects are equipped with special capabilities to act as access points to the Internet service providers [AMN15].

In the new application of Internet, Internet of Things, so many services with the use of sensors, actuators, smart embedded devices, etc. through unique addressing paradigm, are able to communicate and interact among one another to reach a common goal. The final goal is to make life easy for human. Suppose a situation in which your wallet or watch gives security alarm to you without even your interference [STJ14].

Semantically, when speaking about the Internet of Things, in fact here the word “Internet” means an application of Internet and the word “Things” refers to the information

of Things. The Internet provides an infrastructure to interconnect computers and terminals through interconnected networks. However, what to be transferred through this paradigm is not the “Thing” itself but the information of things is involved. So, we can interpret the “Internet of Things” like “the Internet relating to information of things”. The term IoT is composed of two basic elements. They are “Internet application” and “Thing’s information”. On the other hand, there are already other Internet applications capable of transferring thing’s information. Therefore, there must be something special about IoT which makes it different from other applications like File Transfer Protocol (FTP). We speak here, about those features briefly: It is possible that in other Internet applications thing’s information are uploaded and transferred, however in the especial case of Internet of Things uploaded information is only thing’s information. A thing’s information is a description about the thing entity which has two distinguishing characteristics: (1) small and light to occupy less memory and even weight. (2) simple and concise in order to be read easily. So, there is a need for electronic tags. An electronic tag is a sort of electronic media to store thing’s information with ability to be attached to the thing’s body. For example, two kinds of electronic tags are RFID electronic tags and bar-codes. However, for Internet of Things application RFID tags have some advantages like accepting remote reading, supporting rewriting and so on [HL10a].

## 2.2 Descriptive Model for IoT

IoT can be considered as a new application of Internet not an extension to Internet. The huge difference is that the uploaded data in IoT mostly happens without user inference. To compare it with some other Internet application take FTP or Simple Mail Transfer Protocol (SMTP) as examples in which the data is uploaded by user intention, by contrast, the uploaded information in the Internet of Things is not to be input by somebody but to be read by some machine, namely some readers obtain the thing’s information directly from the thing’s entity, as it is illustrated in the Figure 2.1. It can visually explain the Internet of things working process. A thing’s information according with its thing’s entity is embedded into an RFID electronic tag; an RFID reader reads the thing’s information from the RFID electronic tag by non-contact form, thus the thing’s information is uploaded into Internet; the users over the world can real-timely share the thing’s information [HL10b].

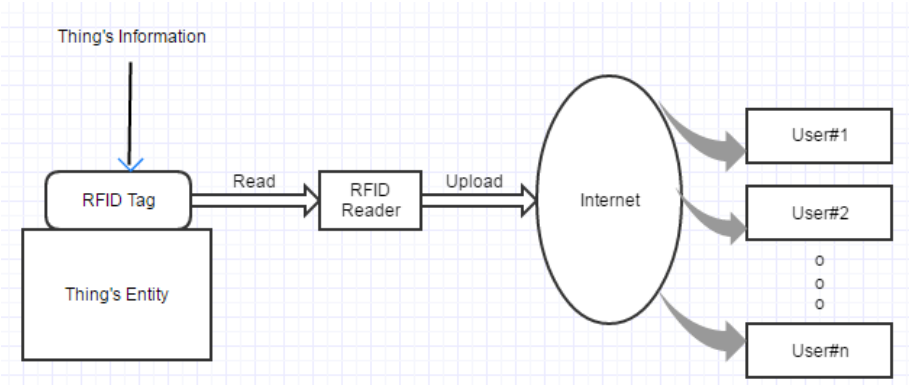


**Figure 2.1:** A Descriptive Model of IoT

## 2.3 From Machine-to-Machine (M2M) to IoT

Machine-to-Machine (M2M) communication and the Internet of Things are two terms that are used in many cases interchangeably. However, IoT comes after M2M communication and in fact, it adds more features to it. M2M can be seen as the solution that allows communication among some devices and a specific application. The communication can be achieved through a wired or wireless network. This architecture allows end users to capture data about events and assets in the environment, such as temperature and pressure. M2M can be applied in a variety of areas like remote monitoring. The Figure 2.2 shows a typical architecture of a M2M solution [HTM+14].

In this architecture, a M2M device is something attached to the object in the environment which is of interest to our business and is equipped with sensing and actuating abilities. Here, we see a conceptual realization of the M2M devices. They can range from small sensors to high-level complex sensing machines. The purpose of the network is to provide a facility for the M2M devices and server-side application to talk among one another. Depending on the type of solution we are working on, the network could be any topology of networks like Wide Area Networks (WANs) and Local Area Networks (LANs). The purpose of service enablement is to provide a generic functionality which can be shared by different applications. It can be just an interface allowing different applications, easily attach to the network in order to communicate with M2M devices. This way development of new applications can be easier. The application is nothing but the implementation of the goal of the whole solution. That means, the application talking to devices through the network, serves the higher level enterprise business process in which it can integrate [HTM+14].

IoT applications initially can be looked at, as the same as M2M communications compatible with the above mentioned ecosystem. In contrast to M2M communications, IoT insists on the connection of sensors and applications through the Internet, and the

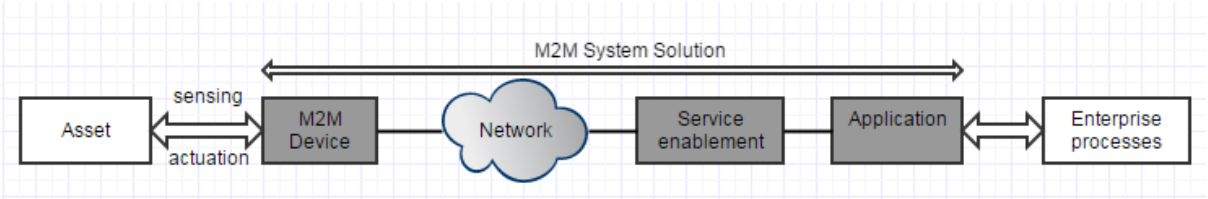


**Figure 2.2:** A Generic M2M System Solution

extensive use of Internet technologies. In fact, IoT tries to enable things to interact via Internet in the same way that humans communicate via Web. That does not necessarily mean that things can not interact via Web. That is what the concept of the Web of Things (WoT) is for. Other than sensors, IoT takes advantage of using other resources of data like Geographic Information Systems (GIS), as well. Even information extracted from social media can be used [HTM+14].

In fact, M2M, is a communication paradigm trying to establish communication to happen between two machines without human interference. IoT is also based on M2M communication paradigm, however it tries to take in almost everything with a data communicating device attached to the body of them, communicating devices like RFID tags and bar-codes, through the Internet infrastructure [AKAH14].

## 2.4 Requirements for IoT Applications

The basic role of IoT is to take every possible object of real life into the Internet. That means, IoT tends to remove the gap between physical world and virtual world. In IoT, we look at the physical items as the physical access points. The idea of having such technology stems from the fact that some other technologies are becoming suitable enough to think about such an innovative technology. For example, think about technologies like Microelectronics, Communications, and Information Technology. The sizes are diminishing, the prices are falling, and the energy consumption is declining. Utilizing these opportunities, objects in physical world can become “smarter” [MF10].

Smart objects can sense their surrounding environment, and through built-in networks they can communicate among each other, and via Internet services they can serve people. As a result, IoT can not be the product of a single capability. In general, several capabilities are involved which we will discuss them here as the basics of IoT [MF10].

**Communication:** This is the fundamental infrastructure for Internet and as a result for the Internet of Things. In IoT, objects need to connect to each other to make use of data

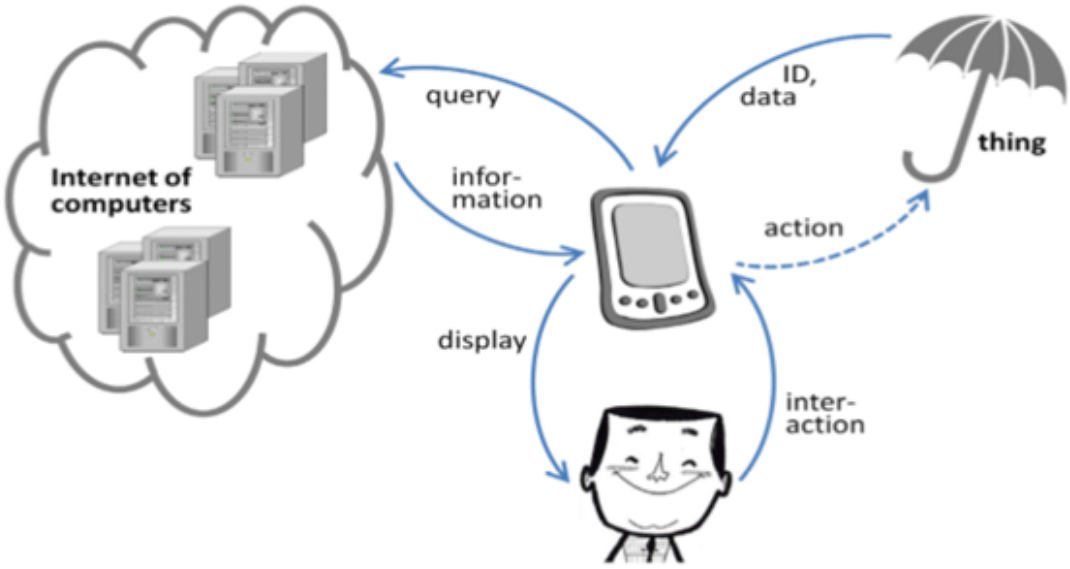


**Figure 2.3:** Smart Phones as a Mediator for IoT Applications, taken from [MF10]

and services. Wireless technologies evolve independently and enhance IoT applications, consequently.

**Identification:** Since objects are not passive elements in IoT, they need to be identified uniquely. If we consider Web of Things (WoT), to identify objects in physical world IPv6 seems to be suitable. RFID and bar-codes are other examples. RFID reader can identify an RFID tag which is attached to the body of a "Thing", and even smart phones can identify bar-codes easily. In more advanced IoT applications, smart phones act like mediator to reach the information of objects as long as they are connected to the network. This concept is shown in Figure 2.3.

**Sensing:** Smart objects collect information about their surrounding environment with sensors, process it to react upon, accordingly.

**Actuation:** Objects can be equipped with actuators to affect their environment. For example, utilizing the sensed electrical signals, they can turn on a light or close a door.

**Localization:** In most IoT applications, there is a need to locate the physical object. In fact, the objects should know their position in the surrounding environment to collect data and react smartly. Therefore, technologies like GPS are involved.

**Embedded Information Processing:** Smart objects are equipped with processor units and memory. This enable them to carry out interpretation on sensor data. This processing power can be duplicated in application back-end, as well.

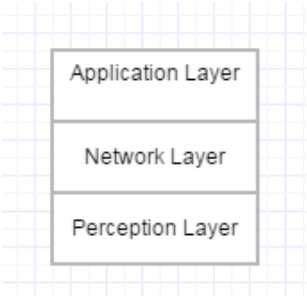


**Figure 2.4:** IoT Three-layer Architecture

**User Interfaces:** Objects in IoT are supposed to interact with users. This can happen through a smart phone or directly. Therefore, innovative interactive paradigms are the matter of significance here. Clearly, most IoT applications are highly interactive systems.

## 2.5 Architecture

Currently, there is a widely accepted three-layer architecture for the Internet of Things. Despite ambiguous definitions of the Internet of Things, the accepted architecture is very clear and understandable as shown in Figure 2.4. In the following, we describe briefly the role of layers in this architecture [MLL+10].

**Perception Layer:** The responsibility of this layer is to identify objects and collect information about them. The perception layer includes 2-D bar-code labels and readers, RFID tags and reader-writers, camera, GPS, sensors, terminals, and sensor networks.

**Network Layer:** This layer transmits and process the information acquired from perception layer.

**Application Layer:** The application layer is a combination of loT's social division and industry demand, to realize the extensive intellectualization. The application layer is the deep convergence of loT and industry technology, combined with industry needs to realize the intellectualized industry.

The suggested three-layer architecture seems to be very simple and concise and is widely accepted by researchers on IoT. However, there are some other architectures discussed by some researchers which show the IoT in more detailed manner. For example, the article [MLL+10] offers an architecture like Figure 2.5. In this architecture the roles of the layers are as following.

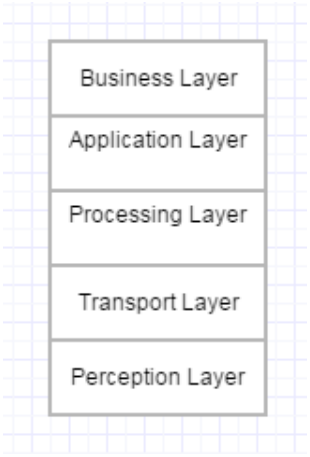


**Figure 2.5:** IoT Detailed Architecture

**Perception:** The task for this layer is to sense the object's properties like temperature, location and so on, through different sensor technologies like RFID, 2-D bar-codes and convert these information into digital format to be able to travel through network.

**Transport:** This layer which is also called network layer is responsible for transmitting the data gathered from perception layer to the processing center.

**Processing:** This layer is in charge of storing, analyzing, and processing the object's information received from the transport layer. Because of the large number of objects and huge amount of data reaching to this layer, different technologies are involved in this layer such as Database technologies, Intelligent Processing, Cloud Computing, and Ubiquitous Computing. In some references, this layer is introduced as the middle-ware layer, as well [AKAH14].

**Application:** By using the already analyzed data from processing layer, this layer provides a variety of applications to ease life for people. Applications like intelligent transportation, logistics management, safety and so on.

**Business:** This layer looks like the manager of the Internet of Things which manages the IoT applications and control their release and charging.

## 2.6 Data Management in IoT

The Internet of Things can be considered as a technology in which interconnected smart objects continuously generate and transmit data through the already existing infrastructure called Internet. When speaking about IoT, there is a tendency among researchers to take much care about developing efficient hardware and communication

tools to reach the ultimate goal of IoT. However, the mechanisms and solutions to handle massive data produced by a variety machines and stations are still to mature. The problem emerges when enterprises are to develop applications to handle the data. The traditional database systems do not seem to be mature enough to be suitable for IoT applications. Traditional data management systems are good at storing, retrieval, and updating data in files and records. In the context of the Internet of Things, data management system should carry out more actions on data before sending them to back-end. They need to summarize data on-line, for off-line analysis. Building and running Big Data analytic applications typically needed for IoT is not easy task to do. Prior to the emergence of IoT, developing data analytics was also a difficult task. However, this can be far more difficult for IoT applications because of some reasons. First of all, the devices to deliver data are completely independent. Moreover, there is not standard way to aggregate the data coming from different sources [AHA13].

When speaking about data in the Internet of Things, "Things" are identifiable objects that talk among one another through exchanging data and affect the surrounding environment by reacting to events and by triggering actions. Here, one can feel the need for having a platform to make the mentioned scenario possible. At the core of this platform there is a huge amount of data produced in real-time manner or stored in fixed storages. This data is converted to useful information through the appropriate applications [VFG+11].

Data management is a big topic to speak about. In this context, we will keep the focus on its role in the Internet of Things. In IoT, data management is a layer between perception layer where objects sense the environment and application layer where the perceived data is analyzed for further applications. The IoT data has a very special characteristic that makes relational database systems somehow a very weak solution. In the simplest case, data from different resources flow to the system indicating that some event is happening or is about to happen. Therefore, before delivering the raw data to decision selection part, there is a need to infer the corresponding event. Moreover, there is always some meta-data about the data generated by "Things"; object identity, location and so on. IoT data will reside in fixed database and be transmit through the network from dynamic and smart objects to concentration storage points until it reaches some centralized data source. Therefore, communication, storage and processing are important factors in data management for IoT [AHA13].

The life-cycle of data in the Internet of Things is illustrated in Figure 2.6. As it is shown, data is collected from data generator and after going through filtering and aggregation, is delivered to back-end to be preprocessed, stored and archived. However, there is a possibility to deliver the aggregated data directly to the user service or application. Finally, the service or the application uses the preprocessed data by querying from back-end. Along the same line, there is a possibility that the device in IoT environment

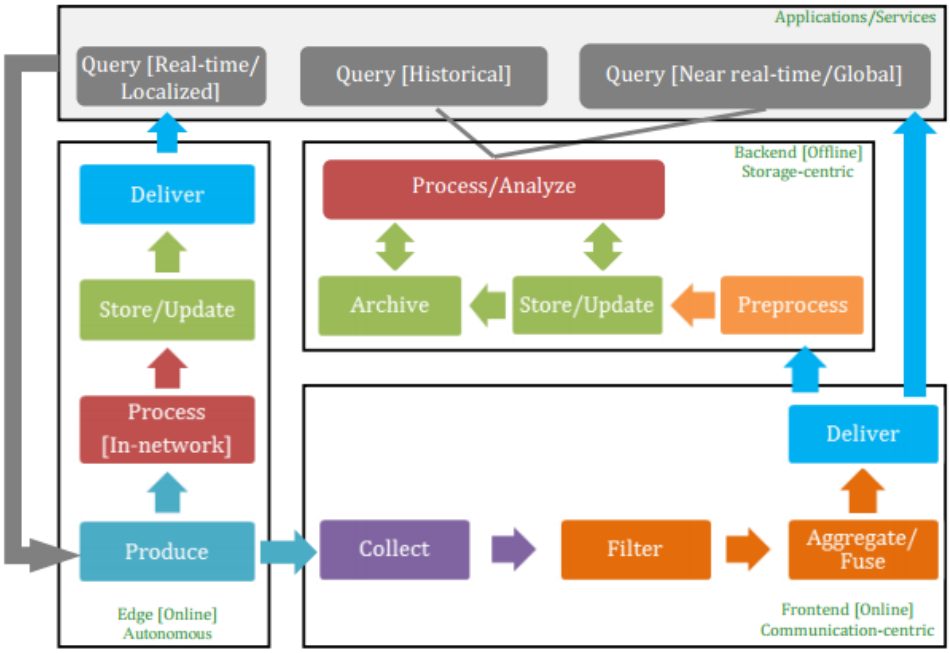


**Figure 2.6:** IoT Data Life-cycle, taken from [AHA13]

locally process and store the data in real-time manner without the need for sending it to back-end for further process [AHA13]. This is possible with smart devices geared with strong processing capabilities. In the following each phase is described in more detail.

**Query:** This part is intended for retrieving real-time data for monitoring purposes or to catch a certain view of the data within the system. The latter is used for more in-depth analysis goals.

**Production:** Data production involves all objects and sensors within IoT context that can produce data both in asynchronous or synchronous fashion.

**Collection:** Data produced in devices has certain time interval. The produced data in devices must be collected and transferred to back-end for further process. The involved technologies are wireless technologies like Zigbee, Wi-Fi and so on.

**Aggregation/Fusion:** The data produced in real-time could have very large volume to store or transfer. Aggregation and fusion are applied on data on real-time to summarize and compress data.

**Delivery:** Once data is collected, filtered, and aggregated and even processed at the edge, it needs to be sent for more in-depth process or final use.

**Preprocessing:** Data in IoT comes from different resources including different formats. The data needs to be preprocessed in order to remove redundancy, fill in the missing data, and aggregate data. In data science, this phase is also called Data Cleaning.

**Storage/Update—Archiving:** This phase is responsible for storage, organization and serving data for processing purposes. Archiving is used for long-term usages. Storage for IoT can come in centralized and decentralized flavors. If we consider Cloud systems, decentralized choices are the best. Traditionally, relational databases are the first choice because of their popularity and high capability in storing data in relational tables. However, NoSQL storage systems are becoming more and more popular because of their ability to keep big data regardless of type consistency and the need for tables.

**Processing/Analysis:** For some applications, task-specific preprocessing may be needed in order to have more meaningful operations take place. This phase is responsible for gaining more insight out of the stored data and predict future.

As it is obvious from the Figure 2.6, the data flow in the Internet of Things environment may go through three different paths. The first one is the path for autonomous systems where the data flows directly from device to application after some processing happening within the network. The second one is the path starting from data collection and after fusion and aggregation the data is delivered to application. Finally, the path that after aggregation, off-line back-end carry out preprocessing, storage and in-depth analysis of data.

Due to the fact that sensors are becoming more and more affordable, IoT data is now becoming more feasible. Therefore, organizations that used to make insight from transactional data, now make insight from IoT data. Not only data is growing in volume, it is also diversifying. That means, the already existing analytical applications lack the ability to cope with IoT data. Decision makers have the data but weak at making fast decisions efficiently. IoT organizations need data management solutions that facilitate fast decisions, no matter how many end points are involved. Under this circumstance, it looks rational to move data management from central data repository toward the edge of network. The trend is to enable the sensors and data producing devices to manage data locally, instead of sending the raw data to central storage and processing unit. Data is managed as soon as produced. This way the streaming data can be aggregated at the edge. Even IoT organizations add intelligence to the edge in order to streamline data processing. More mature organizations try to filter and classify IoT data at the edge which ensure to end up with healthy and organized databases [PBR15].

## 2.7 IoT and Cloud

While devices through IoT ecosystem are connected to each other, they can provide more smart processes and services which can improve the way we live. Under this circumstance, the major challenge is the huge amount of data delivered by these interconnected objects. The inherent problem with IoT is how to make value of the enormous amount of data delivered by so many objects in environment. This will challenge the traditional approach of data management and help to take advantage of methods in Big Data. Cloud Computing is a model for on-demand access to shared pool of resources that can be categorized as infrastructure (IaaS), platform(PaaS), and software (SaaS). Cloud can help to implement IoT applications. Cloud based platforms help in connecting to the things around us (IaaS) at any time, at any place using customized portals and in-built applications (SaaS). Therefore, Cloud acts like a front end to access to the Internet of Things [RSS+12].

We are moving towards Web3 (the Ubiquitous Computing Web). Therefore, the number of already connected devices are growing dramatically and in near future there will be a lot of data, as well. Storing and processing this huge data locally will not be the ultimate solution anymore. So, there will be a need to for huge storages as well as strong computational units. We may safely come to the conclusion that a combination of Cloud Computing and IoT could solve a lot of problems. This new concept is referred as Cloud of Things (CoT) here. However, it is not that easy to bring everything in IoT environment and at the same time, allowing them to have access to resources available in Cloud. Security and privacy are two big issues to be involved. The data security is a very important issue in both IoT and Cloud sides. Suppose that, we are working on smart homes. In this case, the owner of the home may not feel comfortable to send the private data onto the Cloud without being aware of the physical address of the storage where the data is stored and perhaps processed. In the following we discuss some possible issues regarding IoT together with Cloud Computing [AKAH14].

**Protocol Support:** For everything connected to the Internet, there will be different protocols. Some protocols are supported by the gateways some are not. Mapping of protocols in gateways could be a solution.

**Resource allocation:** Depending on the type of sensor being used, the allocation of resources on Cloud must be addressed.

**Identity Management:** In order to have almost everything as a part of Internet, we need proper way of identifying them. IPv6 seems to have enough space to handle it.

**Service Discovery:** The challenge is finding and assigning proper services to IoT nodes. The nodes that can participate in IoT at any time and can leave it at any time, as well.

The nodes that could be mobile ones. For this purpose a uniform way of service discovery may be needed.

**Location of Data Storage:** The location of data, at first glance, may seem to be unimportant issue. However, for some kind of data the physical location also matters. Multimedia data is type of those data. They should be stored in the closest possible distance from user.

In the next section, we will describe KAA open-source Cloud-based IoT platform.

### 2.7.1 KAA Cloud-Based IoT Platform

KAA is a middle-ware platform for building complete end-to-end IoT solutions. The KAA platform provides an open toolkit for the IoT product development. It enables data management for connected objects and back-end infrastructure by providing the server and endpoint SDK components. The SDKs are embedded into the connected object and implement real-time data exchange with the server. KAA SDKs are capable of being integrated with almost any type of connected device or microchip.

The KAA server provides nearly all the back-end functionality needed to operate IoT solutions. It handles all the communication across connected objects, including data consistency and security as well as device interoperability. Moreover, the KAA server has well-established interfaces for integration with data management and analytics systems, as well as with product-specific services. It acts as a foundation for any back-end system that the developers are free to expand and customize to meet the specific requirements of their product. In order to develop an IoT application working on top of KAA platform, there is a considerable amount of freedom to choose the target programing language, data model, and device type. Developers can integrate their own modules, analytics systems, and visualization tools.

For data processing, KAA works based on the log schema which developers design for their applications. On the other hand, KAA server provides developers with APIs for their custom-tailored SDKs. IoT application developers can use these APIs to instruct KAA to deliver the log records to back-end or warehousing systems. A KAA endpoint is tasked to cash the log records (temporary storage of data record) and send them to the server as soon as upload event happens. On the server side, KAA supports a framework of pluggable log appenders that developers use to load the data into database, to send the data to stream processing, or to make it available to customized data processing modules through REST. Therefore, with KAA developers can use their own preferred data analytics.

**Listing 2.1** A KAA SDK in C++

```
#include <memory>

#include "kaa/kaa.hpp"
#include "kaa/IKaaClient.hpp"
#include "kaa/profile/DefaultProfileContainer.hpp"

using namespace kaa;

int main()
{
    kaa::init();
    IKaaClient & kaaClient = Kaa::getKaaClient();

    kaa_profile::Car profile;

    profile.brand = kaa_profile::Brand::Audi;
    profile.model = "A8";
    profile.color = "silver";
    profile.vin = "JTHGL1";

    kaaClient.setProfileContainer(
            std::make_shared<DefaultProfileContainer>(profile));

    Kaa::start();
    Kaa::stop();
}
```

For hardware connectivity, with KAA it is possible to integrate almost any kind of device ranging from those equipped with fully-functional operating system to micro-controllers with limited amount of memory. To integrate a device to the body of KAA, one should use KAA's portable SDKs. They are currently available in Java, C++, and C.

KAA endpoint SDKs are designed to be embedded into client application, easily. They are able to handle client-server communication paradigm, authentication, encryption, and so on. The Listing 2.1 shows a simple SDK written in C++[1].

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

# 3 Fundamentals of IoT Platforms and Frameworks

Currently, IoT platforms play a very significant role in connecting physical and virtual worlds together. In the most primitive form, IoT platforms enable objects in the physical world to connect to each other. In the more precise form, IoT platforms are composed of several blocks: connectivity and normalization, device management, database, processing and action management, analytics, visualization, additional tools, and external interfaces [iot15].

## 3.1 IoT Platform's Making Blocks

**Connectivity and Normalization:** Almost every IoT platform in the most bottom layer includes a connectivity layer. It has the responsibility for bringing different protocols and different data formats into one software interface. This is vital to enable the devices to interact with platform and to enable the platform to receive the data from devices. The connectivity layer makes it possible to have all the data about devices in one place with one format for further analysis. At first glance, the implementation of this layer could seem to be easy, however, consider the situation in which one pressure sensor sends analogue data to the platform for one specific use-case, while some other device like smart phone sends digital data for another use-case. Libraries need to be set up for individual devices. On the other hand, advanced devices usually provide an Application Programming Interface (API) that allows for a standardized communication interface to the platform.

**Device Management:** The Device Management module of an IoT platform makes sure that the connected objects are working properly and its software and applications are updated and running properly.

**Database:** For an IoT platform, one suitable approach to have a database could be a Cloud-based database solution that is distributed across different sensor nodes, for some reasons that are related to the nature of IoT data. For example, one reason is the velocity of data production which results in the fact that many IoT use-cases require the

analysis of streaming data to make instant decisions. Another reason is obviously the high volume of data in IoT use-cases. Moreover, different devices and different sensor types produce very different forms of data which in some cases are very ambiguous and inaccurate.

Typically, the database for IoT platforms should be scalable for big data and should be able to store both structured (SQL) and unstructured data (NoSQL).

**Processing and Action Management:** The data which is captured in connectivity and normalization block and passed to the relevant database is utilized in this module using a rule-based event-action-trigger. This module allows performance of "smart" actions based on specific sensor data. This module is basically nothing but a rule engine that triggers actions based on sensor data using often Condition-Action rules.

**Analytics:** When dealing with data in IoT applications, other than processing and action management, further analysis is sometimes required. Algorithms for advanced calculations and Machine Learning are used to get the most out of the IoT data-stream. These algorithms can be used in off-line manner to get further insight out of data.

**Visualization:** The combination of human eye and brain is still far superior to most analytic and rule-based engines. That is why data visualization is so important: it enables humans to see patterns and observe trends.

**Additional Tools:** Aadvanced IoT platforms often offer additional tools for IoT developers. For example, tools for testing IoT use-cases, or even management-focused tools support the daily operations of the IoT solution. Another tool could be data reporting which extracts data in Comma Separated Values (CSV) files or JavaScript Object Notation (JSON) format.

**External Interfaces:** In general, IoT applications are not stand-alone applications. In IoT-enabled organizations it is very important that the Internet of Things integrates with existing Enterprise Resource Planning (ERP) systems, management tools, manufacturing execution systems and the rest of the wider IT-ecosystem. Built-in APIs, Software Development Kits (SDK), and gateways are possible approaches to the integration of 3rd-party systems and applications. Well-defined external interfaces can help the integration task speed up, dramatically [iot15].

The discussed blocks for an IoT platforms in this selection can be used to put a variety of IoT use-cases in action. For example, suppose that we want to give an IoT-based solution to control water leakage in a washing machine, using the blocks proposed in the above-mentioned IoT platform architecture. The washing machine is geared with sensors to communicate the particular abnormal situation, water leakage. Based on abnormalities in the real-time data stream of these sensors, the machine is stopped before causing damage. Schematically, the Figure 3.1 shows how the IoT platform can

be used to control washing machine in this situation. Both the user and a customer service technician get alerted so they can decide how to address this problem.

In this scenario, the sensor attached to the washing machine transmits lower value than usual which is perceived in connectivity and normalization block of platform (1), then the rule engine indicates three necessary actions: shut down the washing machine, notify the user, and notify the customer service (2). Consequently, the washing machine is shout down remotely (3) and the user gets notified through smart phone (4), customer service gets notified through the Customer Relationship Management (CRM) system (5). The processing and action module plays a crucial role for these use-case. This is where real-time data that surpasses or falls below certain thresholds trigger specific actions [iot15].

## 3.2 Processing and Action Management

In the last section, we had a brief introduction to IoT platforms and its making blocks. As a recap, one of the most important modules in IoT platforms is processing and action management module that hosts a rule engine. In this chapter, we will probe different mechanism to implement such a rule engine with especial look at the state-of-art in IoT.

The Internet of Things provides us with a lot of raw sensor data. However, this kind of data is not mature enough to be used by different applications to serve human. We need mechanisms to gain actionable and contextualized information out of that. Big Data and Data Visualization give us the possibility to understand data deeply by off-line analysis and batch-processing. This is done through the analytics module of an IoT platform as discussed in previous section. However, the essence of IoT is the data concept that is real-time. Real-time stream of data can be analyzed manually for decision making which is not efficient, hence there is a need for automated approaches. Here, the role of Artificial Intelligence (AI) in the Internet of Things comes up[1].

In the Internet of Things, Machine Learning algorithms help entities to get billions of data points and convert them to more meaningful information which can be utilized for future needs. Suppose an IoT environment in which a patent is equipped to a sort of wearable patient monitoring system that is responsible to notify the doctor about possible problems about the patient. Is it possible by analyzing the stream of data manually by a team? Is it possible to write code and create rules to identify abnormal situation? To be able to realize potential problems the data must be analyzed to see

[1] http://www.waylay.io/blog-iot-meets-artificial-intelligence.html

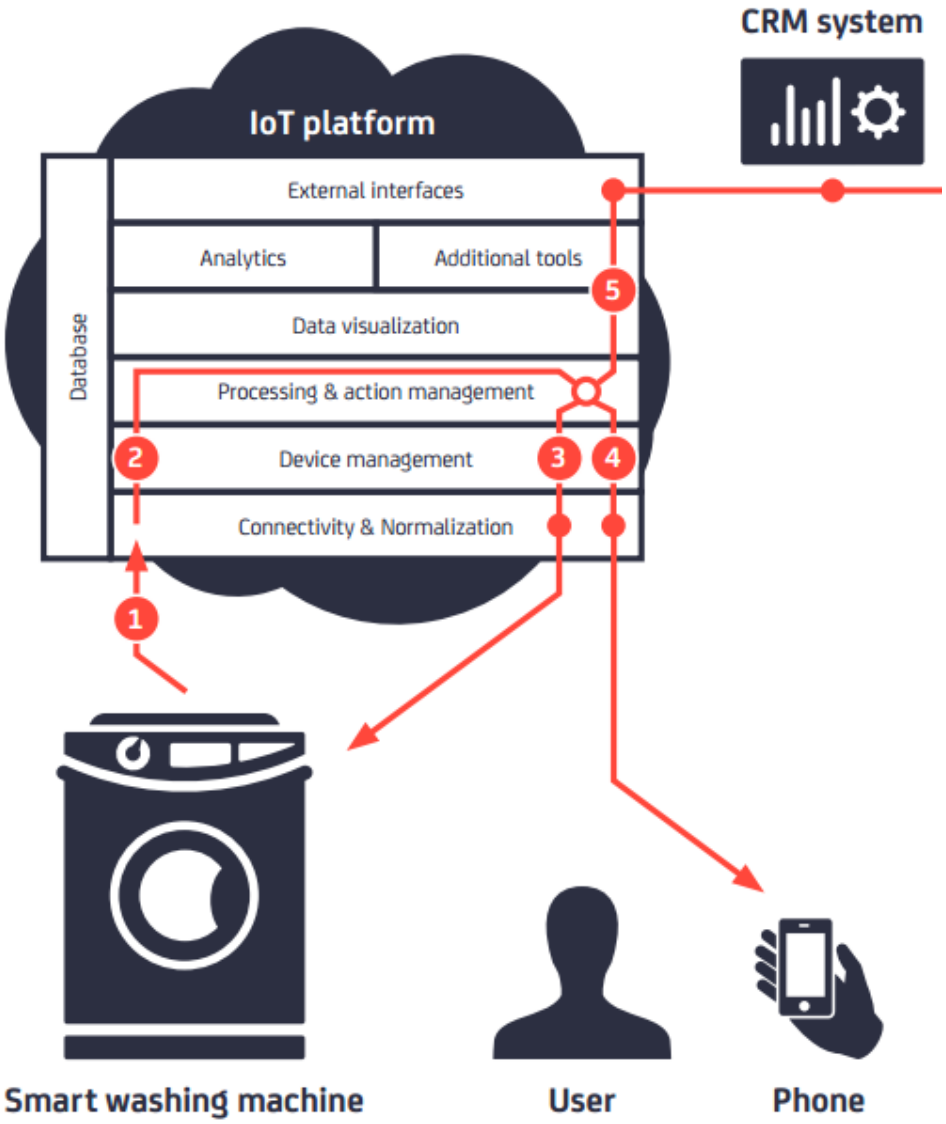


**Figure 3.1:** IoT Platform Triggering and Performing Actions, taken from [iot15]

what is normal and what is not. Similarities, correlations and abnormalities need to be quickly identified based on the real-time streams of data. The system is required to analyze the data and make inference based on previously observed data to react to the current situation and to keep the recently learned information for future use[2].

The power of the Internet of Things comes from the fact that we can make more accurate decision in real-time which enables use-cases like notification, automation, and predictive maintenance. To achieve this goal, Artificial Intelligence offers the concept of rule-based systems or rule engines. An advanced rule engine in IoT environment can ingest the real-time data, reason on that data and call automated actions based on the reasoning[3]. The business rules can flow into the knowledge base of the system through interaction with user. However, Machine Learning offers a better attempt to train the system to infer new rules based on the training dataset or past experiences.

### 3.2.1 Rule-Based Systems Basics

Rule-based systems are in fact practical implementation of intelligent behavior. Of course, they derive from logic. To implement any rule-based engine there is a need to have a logic understood and used. Any rule-based system is based on its own accepted logic. In fact, one of the most important aspect of any Artificial Intelligence activity is how to represent knowledge and one of the major techniques of representing knowledge is logic. In order to express anything we need to get the help from a language. The simplest form of such languages for rule-based systems is “Propositional Logic”. That means we can represent any simple fact through propositional logic. For example, if we say "Temperature is high", we show it with literal like P. If we say "It is sunny", we show it by a literal like Q. Then (P–>Q) which indicates "If the temperature is high, it is sunny" is a new statement, and this is the meaning of propositional logic. The need for this language for intelligent systems is clear. Suppose somebody tells you something in a language. Unless you don’t understand that language, you can’t act accordingly.

First we should understand the role of the knowledge in any intelligent system and how this knowledge can help the behavior of any intelligent system. As it is shown in Figure 3.2. In the context of an intelligent system, the system tries to sense the environment and as soon as accepts input from there, tries to make some decisions and act based on those decisions. The knowledge tells the decision maker what to do. Suppose that the sensed fact by system is temperature. If the temperature goes high and if the proper action is taken, that means the system has the proper knowledge. Therefore

[2] http://www.wired.com/insights/2014/11/iot-wont-work-without-artificial-intelligence
[3] http://www.waylay.io/blog-one-rules-engine-to-rule-them-all.html

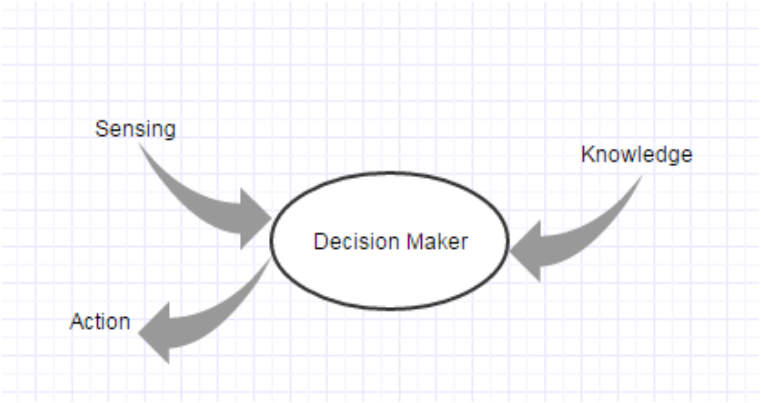


**Figure 3.2:** Intelligent System Behavior

we can see from here that knowledge has important role in demonstrating intelligent behavior.

By the way, it is difficult to have a broad spectrum of knowledge about every possible thing in the environment. We can only expect a machine to demonstrate an intelligent behavior when that machine is left to work in particular environment in particular domain. So the knowledge concept is narrowed down to the domain knowledge. If we think of knowledge we have to first think of the language and then some mechanisms to make sense from the knowledge. The latter is called “Inference Mechanism”.

For a rule-based engine, statements represent assertion knowledge. Such assertions are divided into two categories: rules and facts. Rules are assertions in the from of implications, whereas, facts are assertions that represent domain specific knowledge. For example, the statement "Temperature is high" is a fact. The implication "If temperature is high turn on the light" is a rule. To sum up, that means we can have a Knowledge Base(KB) for our system consisting of facts and rules that are accepted to be true. Rule-based systems are based on rules that say what to DO, given various conditions:

IF < This is the case > THEN < Do this>

A special interpreter controls when rules are invoked. That interpreter is also called inference machine or inference engine. The inference machine which is a computer program, given a new fact or new event sensed, looks at the Knowledge Base (KB) and infer new facts. A rule or set of rules are triggered when their associated conditions are satisfied. To put everything together, a rule-based system consists of a collection of IF-THEN rules, and a collection of facts, and some interpreter controlling the application of the rules, given the facts. This is illustrated in Figure 3.3 [RNI95].

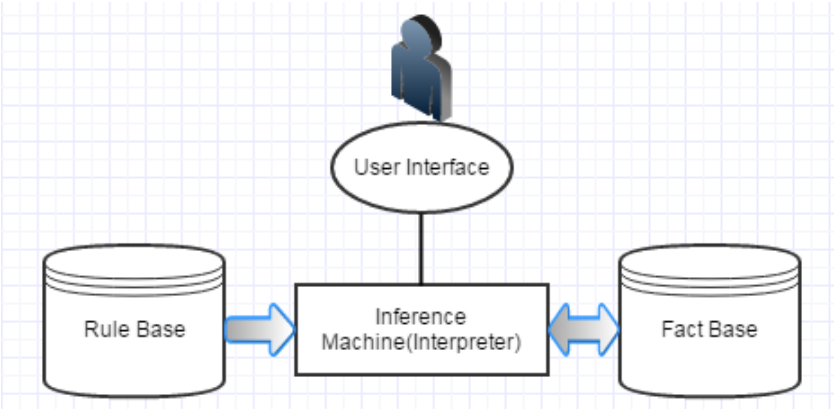


**Figure 3.3:** Rule-Based Systems Architecture

### 3.2.2 Rule Engines Case Study in IoT

In remote health-care monitoring, the collected data from wearable sensors on patent's body must be transferred to a central station for further process. However, two problems are the high load of network and the high ratio of power consumption by transmitter. The solution to these problems can be a rule-based engine to carry out an event-based transmission rather than continuous transmission of data without any preprocessing. The position of rule engine for this problem is shown in Figure 3.4. In this architecture an Electrocardiography (ECG) data acquisition mode is used to collect data and for transmission of data, IEEE 802.15.4 PHY and MAC standards are used. The rule engine basically, consists of two cascading sections. They are decision making (inference algorithm) and transmitter control (action selection). If we consider a static rule engine case, the task of the decision making section is to analyze the features extracted from the collected data and to decide if the data is normal or abnormal. Then a decision is made and sent to the transmitter by the transmitter control. The mechanism of rule engine could be as simple as IF-THEN rules. Moreover, the architecture illustrated in Figure 3.4 shows an example of data processing, close to the edge of network and thereby reducing the load of network and enhancing the performance.

For ECG data collected, the rule engine considers a hard threshold and makes the decision based on that. In fact, based on that threshold the data is classified as normal or abnormal. If the data is classified as the abnormal data, the rule engine switch on the transmitter and sends the sample data to gateway. In fact, the system separates normal data from abnormal data and only sends the abnormal one which is of interest for health caring intension. For example, suppose that the features P, Q, R, S, and T are extracted from ECG data collector and are fed to the rule engine. In the rule engine, decision maker calculates the PR, QRS, QT intervals and then compares them with hard threshold

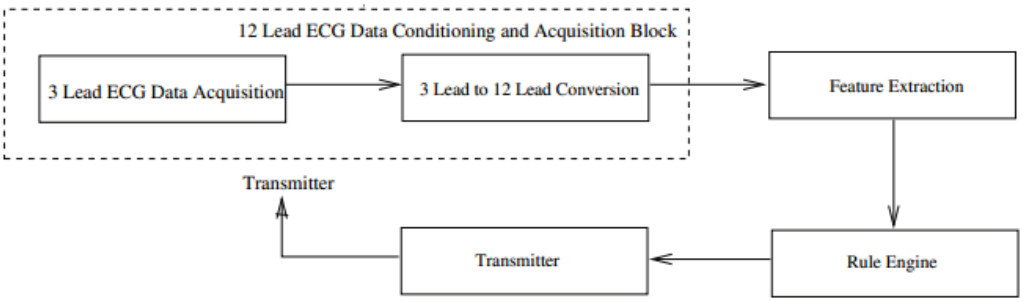


**Figure 3.4:** Rule Engine for Data Transmission, taken from [KRBA14]

and makes a decision. That is, if any of the intervals exceeds the threshold, is classified as abnormal and consequently sent to the corresponding gateway [KRBA14].

### 3.2.3 Rule Engines in IoT

When data stream arrives in any IoT platform, the processing and action management block of the platform is supposed to analyze the stream and take action, accordingly. As already discussed this is where a software framework can be used, like the one we will introduce in the next chapter. The chief functionality of such a framework is to implement a rule-based engine. Currently, in IoT world, there is not an ultimate way of coming up with a comprehensive rule engine. In fact, it depends on the IoT use-case we are working on.

#### 3.2.3.1 Machine Learning Approaches

A learner in its simplest form is a computer program which is said to learn a task given a set of experiences with respect to some criteria or performance metric. A learner is in an environment, trying to learn facts. It has a link to some Knowledge Base (KB) from which it can take and in which it stores the acquired knowledge. The knowledge can be internal data structures and the experience can be percept dataset. In general we have three types of learning methodologies [RNI95]:

**Supervised Learning:** These are the type of learning algorithms in which, the system is given a set of labeled (classified) training examples. These labeled examples include both input and output. Therefore, the system extracts knowledge based on a labeled training dataset.

**Unsupervised Learning:** For this type of learning algorithms, there are no labels given, we only have the examples which are not classified. There are situations which the system wants to learn from them.

**Reinforcement Learning:** In this type of learning algorithms, a sequence of examples is given to system. The system at some points gets rewards or punishments.

All the mentioned Machine Learning algorithms, finally, extract knowledge from the training dataset they are given, and store the learned knowledge within different kinds of data structures. For example, decision trees, neural networks, condition-action rules, rule sets, finite state automata, lisp code, or C code.

concept learning is used to learn the description of a class of objects. We use this description to predict the class of a new object. In classification problems, we have goal concept that we are trying to learn. That is called the target concept. Our guesses of target concept are called hypothesizes. Instances or examples help to learn the goal concept. They are described through a vector of attributes or features. X = <x1, x2,.., xn>. The concept learning problem is narrowed down to find the description of a function (f) that maps feature vectors (X) to a discrete set of k classes.

f: X –> <0, 1, 2, .., k-1>

Often, we have two states of classes like positive or negative. Besides, like any other supervised learning problem, we have often a set of training examples as a pair like (X,f(X)) which help to learn the description of the function. Also, other than training set we have a test set to evaluate our solution. The format that we establish our hypothesis space, depends on the learning approach we take. It can be only a set of rules, or a decision tree or even a neural network. The hypothesis space is defined in terms of attributes (features) in training set. For example, we can assume hypothesizes like this: x1, x1 AND x2, (x1) OR (x2 AND (NOT x3)), and so on. In all different learning approaches, the final step is to select the best possible hypothesis [RNI95].

decision trees are one of the basic approaches to implement a classifier. The technology for building knowledge-based systems by inductive inference from examples has been demonstrated successfully in several practical applications. Results from some studies show ways in which the implementation of a decision tree can be modified to deal with information that is noisy and/or incomplete [Qui86].

decision tree is a classifier for concept learning task. In a decision tree a leaf node is the value of the target attribute(class). A decision node or an internal node specifies some test to be carried out usually on a single attribute value with one branch for each outcome of the test. For example, if we are to classify the climate of a room as good or bad, based on three input attributes like temperature, humidity, and pressure we will end up with a decision tree like the tree shown in Figure 3.5. That is, the input set looks

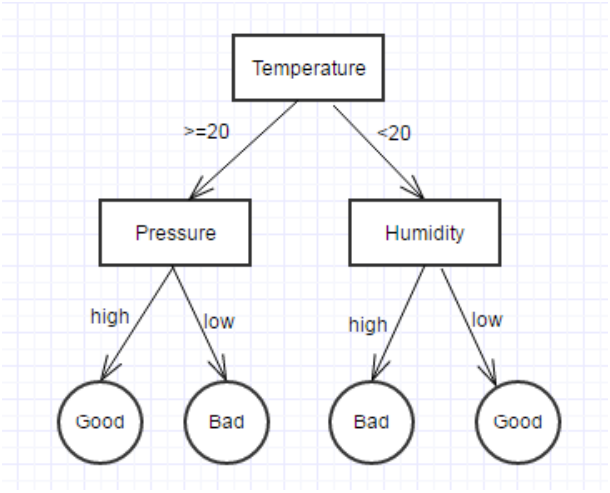


**Figure 3.5:** Decision Tree

like X = <T, P, H> where T stands for temperature, P stands for pressure, and P stands for pressure and X is the target attribute (class). With the given tree, and receiving an input like X = <25, high, low>, our input stream is classified as “Good” [PCZ09].

For decision trees the depth of the tree grows linearly with the number of variables, but the number of branches grows exponentially with the number of states. Decision trees are useful when the number of states per variable is limited (e.g. binary YES/NO) but can become quite overwhelming when the number of states increases. Therefore, for IoT use-case in which we are sure that the number of states is limited, a decision tree can be a good choice[4].

Decision trees are placed among Machine Learning algorithms with weak performance for use-cases with large number of decision states, as already discussed. To overcome this problem by known Machine Learning techniques, neural networks and Bayesian networks are used by some famous IoT vendors. However, some other use methods like flow diagrams and Complex Event Processing (CEP) which we will be speaking about, in next sections.

#### 3.2.3.2 Flow Diagrams

Flow diagrams or pipes are the other alternative in IoT to develop rule engines to handle massive data. They follow the paradigm used in data flow systems, wisely. In a very simple scenario we can consider it as the agents that can be connected together by the

[4] http://www.waylay.io/blog-one-rules-engine-to-rule-them-all.html

system users, enabling a flow of events among agents. While the system is configured and connected, the events begin to flow and consequently the system take actions on behalf of the user. In the world of IoT while many systems are implemented simply with condition-action, some organizations rely on data-flow systems, despite the fact that it is not a new method. For example, WoTKit processor and Node-RED are based on this method which we will go through them in details later [BL14].

Data-Flow Programming (DFP) is type of programming methodologies which tries to implement applications as directed graphs. Within the graph that is similar to a data-flow diagram, the nodes represents either sources, sinks, or processing blocks. The blocks are responsible for the information flow inside the system. The nodes are linked by edges which show the flow of information and its direction. The biggest achievement of this method for processing data lies in its ability to implicit implementation of concurrency. Each node is independent from other nodes and they can process data as soon as it is received without the possibility to create deadlocks as there is no shared space of data among processing blocks. In other programming methods in order to achieve concurrency we may think of semaphores and similar methods which are error prone [Sou12].

As time passes the need for processing large set of data emerges. Therefore, to handle such a big amount of data we might consider multi-processor systems working with multi-thread programming techniques. However, few programmers are comfortable with multi-thread programming due to its complexity and high probability to develop applications with hidden bugs. data-flow programming can pave the road to accomplish parallelism, with less effort. In a data-flow network, nodes are connected through edges. Values are propagated as soon as they are processed to the cascading nodes which result in triggering the new set of computation on them [Sou12]. A simple example comes in Figure 3.6 to show the flow diagram for the statements:

Z = (A * B) + C

W = Z + 4

With all simplicity and efficiency that data-flow diagrams bring to IoT applications, there are some problems to be mentioned. Parsing of the message payload which somehow becomes part of the “template”, and more importantly, it constrains the logic designer to think in linear way, from left to right, following the “message flow”. Interesting problem arises when two inputs come at different times. How long do you wait for the next one to arrive before deciding to move on in decisions? How long the data point/measurement is valid? These are the problems that should be addressed[5].

[5] http://www.waylay.io/blog-one-rules-engine-to-rule-them-all.html

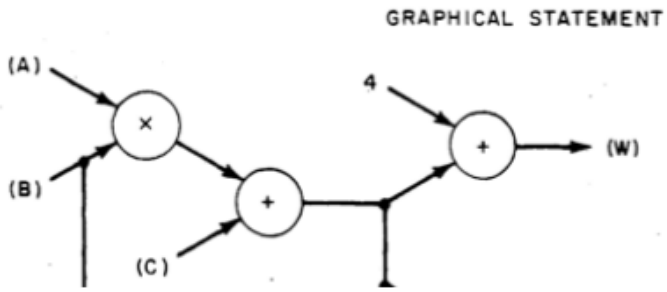


**Figure 3.6:** Data Flow Diagram, taken from [Sou12]

Data-flow diagrams are used in some IoT tools to for data analysis purpose. For example, WoTkit is Web-centric IoT toolkit. An event-based data processing system comes with WoTKit. Sensor data is processed as soon as it arrives from gateways. The idea is to enable user to make an insight of coming sensor data and in some cases a, high level sensor information from low level sensor data in easy way. The interface is a visual programming environment for Java-script based visual languages like Yahoo pipes. The programming paradigm is Data Flow in which the processing pipes are made by user to make meaningful information out of sensor data. A management page provides a list of pipes that the user currently has running. Using this page, users can start, stop and edit the pipes. Administrators can manage all pipes for all users on the system. To develop a new pipe, or edit an existing pipe, the visual programming interface allows users to drag and drop modules to the main pane and then connect them with wires [BL12].

Along the same line, Node-RED is a visual tool for making IoT applications developed by IBM Emerging Technology. Node-RED uses the concept of Data-flow processing to wire up input, output and processing blocks in IoT environment to process massive data and based on that control the things. In Node-RED the processing blocks can be Web services, that is, in each block the computation is left to a Web service that is invoked in proper time. The result is finally an action,for example, sending an email on rainy weather forecast. It is shown in Figure 3.7[6].

FRED is the front-end of the Node-RED enabling multiple users to manage instances of Node-RED in the Cloud environment. In fact, the managing and controlling Node-RED instances is left to be done in Cloud. User can make IoT application visually by connecting together nodes to make the “Flows”. Each node receives input message and produce output message which in turn, can be input for some other node. The flowing messages are Java-script Object Notation Objects (JSON). They must contain at least one payload parameter. A sample message is illustrated in Listing 3.1.

[6] http://developers.sensetecnic.com/article/introduction-to-node-red/

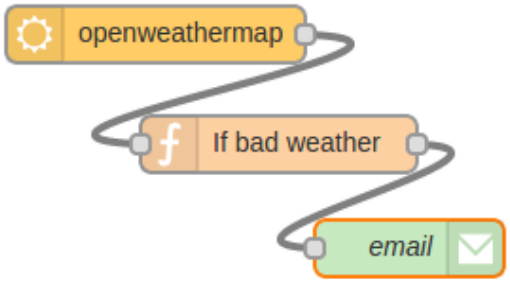


**Figure 3.7:** Node-RED Functionality

**Listing 3.1** Node-RED Message

```
Msg =
{
Payload: massage payload,
Topic: error,
Location: somewhere in space and time
}
```

Nodes in a flow inherit from a base class called “Node”. On instantiation, the nodes subscribe to external services and begin listening for coming data on a port, or they may wait for HTTP requests[7].

#### 3.2.3.3 Complex Event Processing (CEP)

Complex Event Processing is a method of tracking and analyzing streams of data about things that happen, and deriving a conclusion from them. The first person raises this conception is Professor David Luckham from Stanford University [YCL11].

Data analytics like SAP Sybase, HP Vertica, and Action ParAccel MPP have already been in use to utilize big data. However, all of them focus on processing data in off-line manner, and simply they are batch-oriented analytics. When it comes to the Internet of Things, we try to keep our focus on analytics on the live data stream. That means, in IoT we are not interested on storing data after it has been processed, although in some use-cases it is necessary for auditing and establishing a Knowledge Base, based on the observed events. Among common technologies for stream analytics are Event Stream Processing (ESP) and Complex Event Processing (CEP). For example, Twitter Storm and Apache S4 use ESP method and EsperTech Esper uses CEP method [HTM+14].

[7] http://developers.sensetecnic.com/article/introduction-to-node-red/

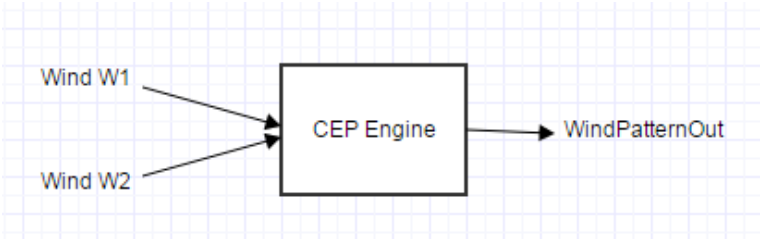


**Figure 3.8:** CEP Engine Example

CEP allows easy matching of time-series data patterns coming from different data sources and it frees developers from the task of handling context locking. However, in some cases it has shown the problem that Decision Trees and Pipelines have shown. That is, modeling is also an issue here to be considered[8].

With the evolution of IoT and advent of Web Services, applications need to make more intelligent decisions using real-time data. Such applications require CEP engines to detect patterns of activity from a variety of data sources and infer events continuously [OBE07]. As an accepted definition: CEP is event processing that combines data from multiple sources to infer events or patterns that suggest more complicated circumstances. The goal of complex event processing is to identify meaningful events (such as opportunities or threats) and respond to them as quickly as possible[9].

CEP engines are, in fact, event-driven information systems that employ techniques such as detecting complex patterns, building correlations, and relationships among many events. From a high abstractive view a typical CEP engine receives a set of input streams from like database files. Many CEP engines use a SQL-like programing language such as Continuous Computation Language (CCL) with an extension for event processing. In the following we show the idea of CEP with a simple example. Suppose that our engine wishes to listen to an input stream called “WindIn” and find the inputs that the wind changes by more than 5 miles per hour in two seconds, then the matching events are put in an output data stream called “WindPatternOut”. The example is shown in Figure 3.8 and the corresponding query comes in Listing 3.2 [OBE07].

CEP engines tend to look at the data to be processed as events. Any phenomenon in physical world can be modeled as an event to be fed into CEP engine. Events have sometimes complicated relationships, even, a single event may consists of several other events. CEP technology can be used to find out these sophisticated relationships and decide on action over them in real-time. In fact, most CEP engines functionally work like reverse databases. In ordinary database systems, the practice is to store data and

[8] http://www.waylay.io/blog-one-rules-engine-to-rule-them-all.html
[9] https://en.wikipedia.org/wiki/Complex_event_processing

**Listing 3.2** Query Representing a sample CEP Engine

```
INSERT INTO WindPatternOut (Location, Speed1, Speed2)
SELECT W1.Location, W1.WindSpeed, W2.WindSpeed
FROM WindIn W1, WindIn W2
MATCHING [2 SECONDS: W1 && W2]
ON W1.Location = W2.Location
WHERE (W1.WindSpeed - W2.WindSpeed) >= 5;
```

make queries against the stored data. Whereas, in CEP engines we store the queries with correlated constrains and then run the data through [JC14].

In IoT the practice of data processing, somehow occurs hierarchical. In this hierarchy, basic events mean atomic events that take place at different times in different places. Complex events are composed of these basic events. In the hierarchical processing model every level only processes their own logic, other complex events logics are kept apart, so it improves efficiency of events processing [Wei12].

Despite all advantages that CEP offers, it is not the ultimate solution for data analysis in IoT. Part of the data in IoT environment is the static description of information. To process this kind of data, there are some distributed computing systems already in use. Hadoop is one the most popular ones. However, the CEP technology if not comparable with off-line batch processors, it has some obvious advantages over already existing data stream processors. In general, data stream processors come in two flavors: Data Stream Processing (DSP) models and Complex Event Processing (CEP) models. DSPs have simple functionality. They integrate data from different input sources to one output stream. The output stream can be stored in database with DBMS and retrieved when needed. On the other hand, CEP method tries to watch the events to filter or calculate the corresponding output. The complex event processing model having better processing power in handling time relationship among different events plays better in IoT environment [CFS+14].

# 4 Design

Previously, in chapter 2 and chapter 3, we talked about IoT basics and the very important role that an IoT software frameworks can play within IoT platforms to achieve the highly automated way of life with minimum risk of human mistakes when it comes to decision making.

To address a problem in real life, we may need to use an IoT solution. In such a solution, having equipments like sensors, actuators, and data processing algorithms in hand, we need an IoT platform to bring them work together. Any IoT platform should give the facility to somehow process the data stream and trigger action, accordingly. To do so, different IoT platforms include a rule-based engine which could be a software-based framework. As discussed in section 3.2.3, there are a variety of approaches to implement a rule-based engine for IoT applications. In the following, we are going to offer a simple architecture for a typical IoT software framework considering vital requirements to allow different IoT solutions to sit on and then in the next chapter, we will show how the implementation is possible. Developing an IoT framework from scratch, capable of incorporating any possible IoT solution needs huge amount of work and time, however the offered framework here despite the fact that is still to mature, can represent the essentials, introducing new features that will be discussed, technically.

## 4.1 Architecture

The Figure 4.1 shows the overall architecture of the system. On the heart of the system sits the Core of the framework, the part that is attached by four interfaces and equipped by a User Interface. In the following we will describe different parts of the framework in great detail.

Before moving into the implementation of the different parts of the system, we have to mention here that the system follows a client-server architecture where the user or possibly the administrator of the application sits on the client side who communicates with the engine on the server side. The communication between the User Interface and the engine relies on the Web protocols and is achieved on top of HTTP. On the other

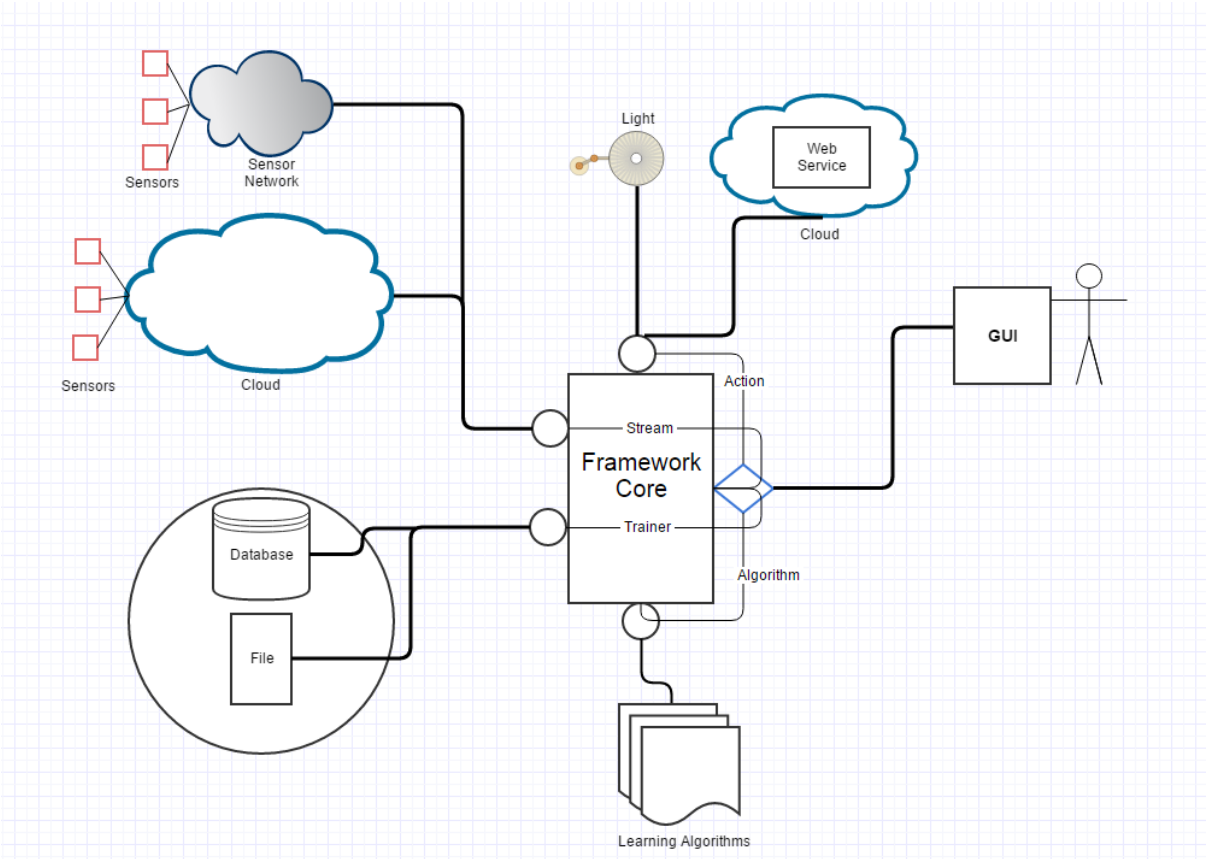


**Figure 4.1:** System Overall Architecture

hand, the data stream can flow into the application using any kind of protocols whether HTTP or for example MQTT.

## 4.2 Functionality

In the last section, we talked about the architecture of the system which we want to develop, in this section we elaborate the functionality of different parts of the system. To begin with, we discuss first the Core of the system. The Figure 4.2 shows the Core of the framework. This part of the system is responsible for connecting different parts together to enable an IoT solution work properly. The parts attached to the Core should be addressed properly. On the boundary of the Core there are four interfaces. These interfaces allow four different components come together, wired up properly and make a typical IoT solution take place. As you can see in the Figure 4.2 there are four interfaces which are shown by small circles. The first one is the interface which allows different Machine Learning algorithms attach to the Core engine for data stream analysis. The

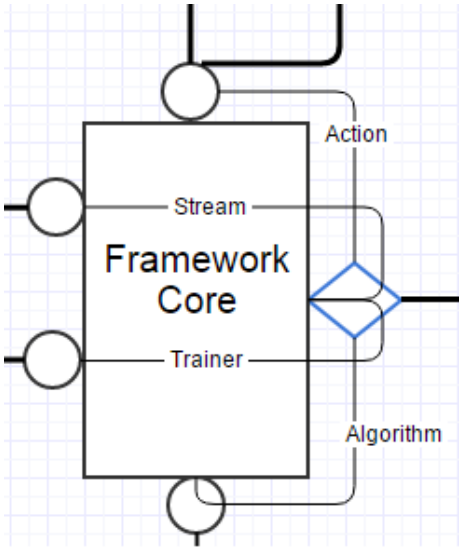


**Figure 4.2:** System Core

second interface is the one for acquiring training dataset to train the learning algorithm before the system starts working.

The third interface is responsible to allow the framework receive input data stream at run-time. The incoming data stream could possibly come from any resources like from devices directly attached to the server, local sensor networks or even somewhere in Cloud. The idea is when the Core engine is running it listens to the data stream source and when a new stream of data arrives, processes it by the already selected and trained Machine Learning algorithm. Finally, having analyzed the input stream the result is ready for decision making which in turn, happens in the last interface to trigger an action. In the simplest possible approach, when we discuss about the Core of our IoT framework, we refer to a rule engine as discussed in chapter 3.

In section 3.2, we discussed that a rule engine in the simplest approach in IoT platforms is implemented with condition-action rules with especial capability of inferring new rules from the already existing rules. The initial rules are defined by application business rules or simply through end-user interactions. This way, the incoming real-time sensor input can be processed and utilized for decision making on near real-time fashion. On the other hand, the events or even the simple sensor stream inputs are stored in the platform repositories for further off-line processing where Machine Learning algorithms and Data Mining techniques are widely involved. However, the offered framework in this chapter base the rule engine directly on Machine Learning algorithms. The part of Core responsible to accept different algorithms is shown in the Figure 4.3.

Using Machine Learning algorithms in the core of IoT frameworks, makes smarter rule engine which can react on unseen events and sensor streams efficiently with high

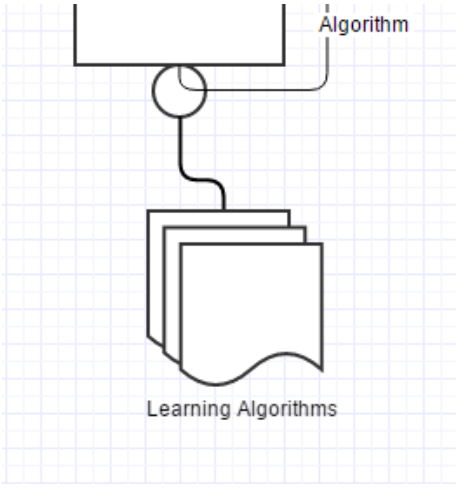


**Figure 4.3:** Machine Learning Algorithms Interface

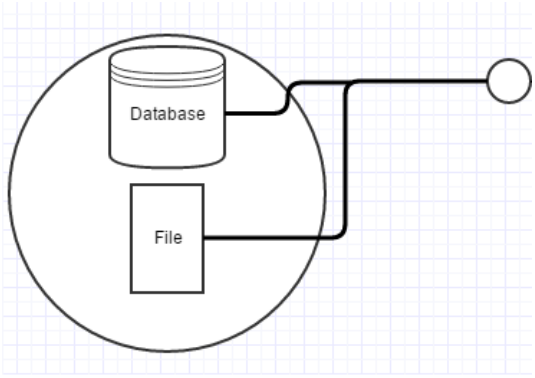


**Figure 4.4:** Learning Algorithms Training Interface

performance. In this regard, the major challenge is training the algorithms which in turn opens new areas depending on the IoT use-case. In the offered framework as you can see in the Figure 4.4 two possible approaches are thought. The first one is simply uploading the training dataset as a text file directly by the user or the administrator of the system before starting the engine. The text file which follows special format is filled by user knowledge of the environment. The same thing can happen also from a local database on the server side. That means we can ask the user through User Interface to enter the training dataset and then store it in the database. Currently, in the implemented version of this architecture as will be discussed later the algorithms are trained by text file uploaded by user. However, the interface to get the algorithms is developed in a way that allows any possible training data source be used.

So far, we have been speaking about the learners and the possible ways to train them. While these steps passed behind, the Core is supposed to listen to the endpoint dedicated

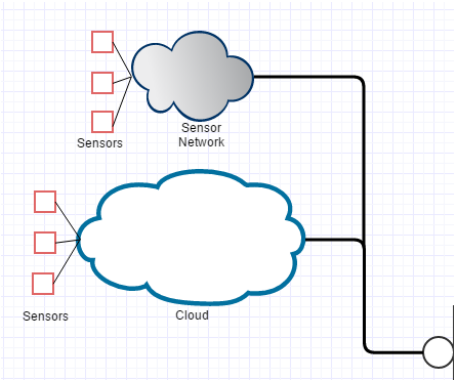


**Figure 4.5:** Data Stream Endpoint Interface

for receiving input data streams. You can see this part of system in the Figure 4.5. Keep in mind that when the engine is running, the mentioned steps will be followed in a pre-designed order. That is, when the Core starts listening to data endpoint, it has been already equipped with a trained Machine Learning model to react on input data.

In this architecture, the interface to receive the input data streams is designed to follow a life-cycle paradigm. That means, having extracted the Machine Learning algorithm model and trained it, the listener to input stream starts receiving data streams until the user through the User Interface decides to stop listening. In the intervening time between start and stop listening, the arrival of an input data stream is treated like an event happens in the data stream interface which is supposed to be handled at the Core of the framework. Handling data at the Core means feeding the data stream to the extracted learner model. In fact, when the Core of the framework is about to handle the fired event in the interface, it has already been equipped with a Machine Learning model preferred by the user through the User Interface. When implementing this architecture, the important issue to keep in mind is addressing the event and its handling mechanism. Therefore, this interface must include an event type subscribed to a call-back function (delegate type) as the handler of the event. This call-back function type only describes the signature of the handler function, the real implementation is left to the Core of the framework.

Moreover, keeping the interface as flexible as possible to be consistent with different sensor input resources can enhance the flexibility of the framework, dramatically. In Figure 4.5 to envision this idea, two different resources are shown. One is a local sensor network, the other one is somewhere in Cloud that collects and delivers data to the framework.

In the implemented version of this architecture, as we will be discussing, the format of sensor inputs will be all finally converted to simple strings through the interface

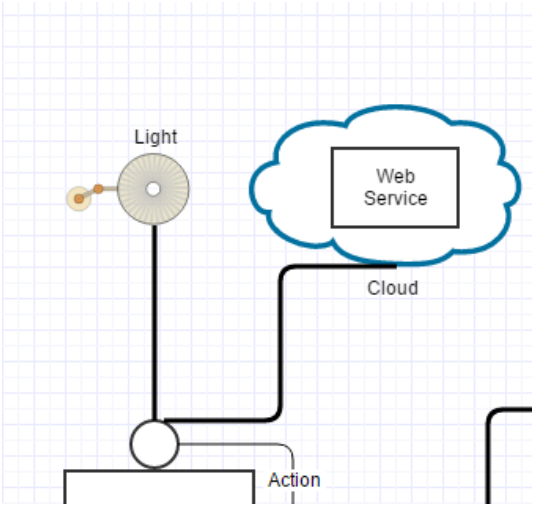


**Figure 4.6:** Action Interface

instances. That means how to convert the incoming input to single separated strings is all left to the writer of the interface instance.

At this point, the input stream flows into the Core of framework through the described interface and there, analyzed to select an action to be taken. The result is delivered to the last interface in this architecture i.e. Action Selection interface, as you can see it in the Figure 4.6.

This is the part of architecture which utilize the analyzed data. It can be used simply to turn on/off a light or a switch on/off a device. If the number of decision states is not binary (true or false) it can be used to change for example the brightness of the light. It all depends on how we define our IoT use-case and how we train our algorithm. However, from a different perspective we can use the result to invoke a Web service or even change the state of an already connected Web service. In fact, this is the part that the raw data is analyzed and could be used in any direction which the owner of the system prefers.

Finally, the whole system is designed to be interactive with user. Interaction with two categories of end users are necessary for this system.

The first category of users is those who only set up the system to run. For example they can decide about the type of Machine Learning algorithm, training dataset resource, data input stream source, and finally the actuator. These users can decide about the type and number of sensors, as well. The second category of users can be considered as the administrators of system. They are responsible for extending and upgrading system. For example, the administrator may wish to add a new input resource extension, or new actuator to trigger action. That means the administrator can add and update any

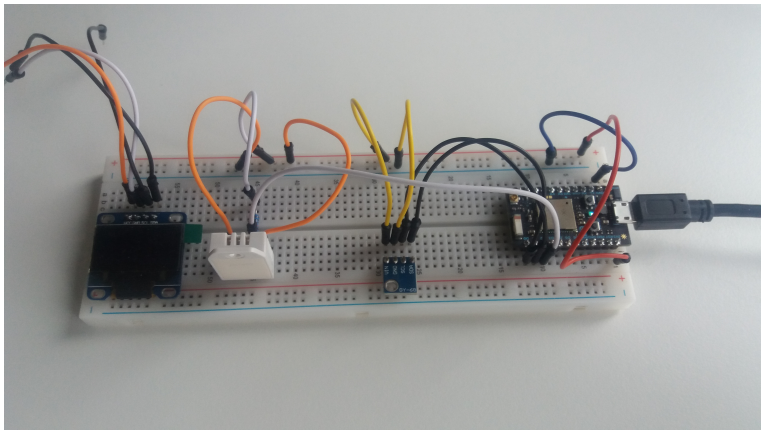

**Figure 4.7:** Photon Development Kit

of four already mentioned interfaces. We refer to these new instances as extensions of interfaces throughout the text. When it comes to extensibility issue, the key point is to make the integration of new extensions as effortless as possible for the extender. This can be achieved through Component Based Software Development (CBSD) methods, as we will be speaking about when describing the implementation. Obviously, a User Interface is required as the one in the shown in the Figure 4.1. As we will be discussing in the implemented version of the architecture, the User Interface is a Web based one.

Up to this point, we have depicted how the system works. Before going to the implementation of the framework, let's exemplify our idea by a simple use-case which helped us to test the functionality of the system. Suppose that we are interested in accomplishing an IoT solution in which three sensors are connected to the system, and continuously deliver data about the temperature, humidity, and pressure of the environment in asynchronous manner. Our hardware development kit to collect sensory data— the Photon (Wi-Fi) is shown in the Figure 4.7. When an abnormal situation occurs depending on the context of the use-case, a light must be turned off or turned on as notification. We used Philips Hue Iris for notification actuator part, as shown in Figure 4.8.

To make this situation be solved using our framework, after setting up interfaces to accept the data and have the system recognize the actuator the only thing remaining is to select a proper Machine Learning algorithm or installing new one. As already discussed in section 3.2.3.1, for this use-case because of the fact that the final state for decision making is narrowed down to only two states (true or false) a decision tree suffices perfectly, regardless of the number of connected sensors. Basically, for all use-cases with limited decision states, decision trees work fine because after training the decision tree, we will have a tree with acceptable depth. For this use-case the decision tree model which we used is Iterative Dichotomiser 3 (ID3). The Figure 4.9 shows how this IoT solution can be achieved with the offered framework. The collected sensor data is not

**Figure 4.8:** Philips Hue Iris

directly delivered to the framework. Preferably, the data first travels to the Cloud-based IoT platform—KAA, and then redirected from Cloud to framework. This way, there is no need to have the sensors geographically close to the server where the framework engine works on. The KAA platform is explained in section 2.7.1.

Moreover, along the same line, with this simple use-case we tested our framework to trigger an action on IFTTT Web-based service, instead of turning on/off a light. IFTTT is a free Web-based service that allows users to create chains of simple conditional statements, called "recipes", which are triggered based on changes to other Web services such as Gmail, Facebook, Instagram, and Pinterest[1]. In our recipe the IF part is the result of data analysis in the Core of framework and THEN part is just sending a notification Email. In IFTTT, Maker Channel allows you to connect IFTTT to your personal projects. With Maker, you can connect a Recipe to any device or service that can make or receive a Web request[2]. In our use-case the result is an Email like the one shown in the Figure 4.10.

[1] https://en.wikipedia.org/wiki/IFTTT
[2] https://ifttt.com/maker

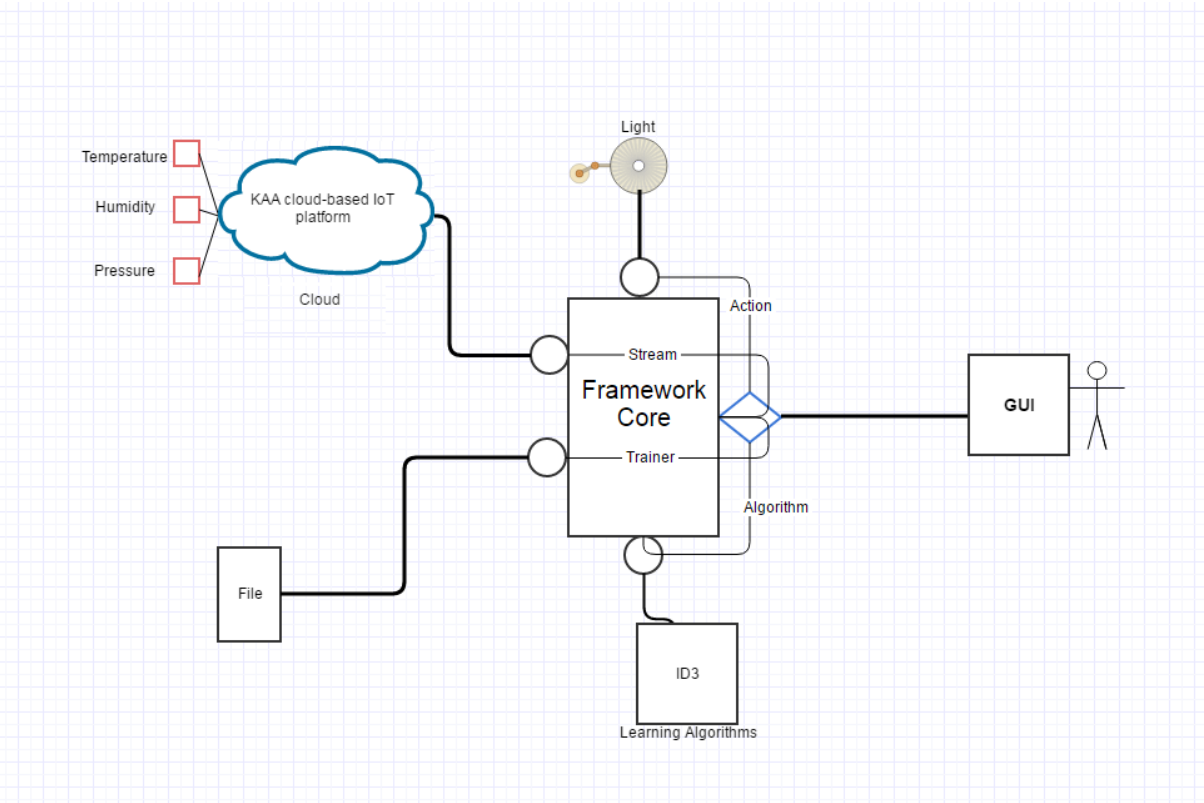


**Figure 4.9:** IoT Use-Case

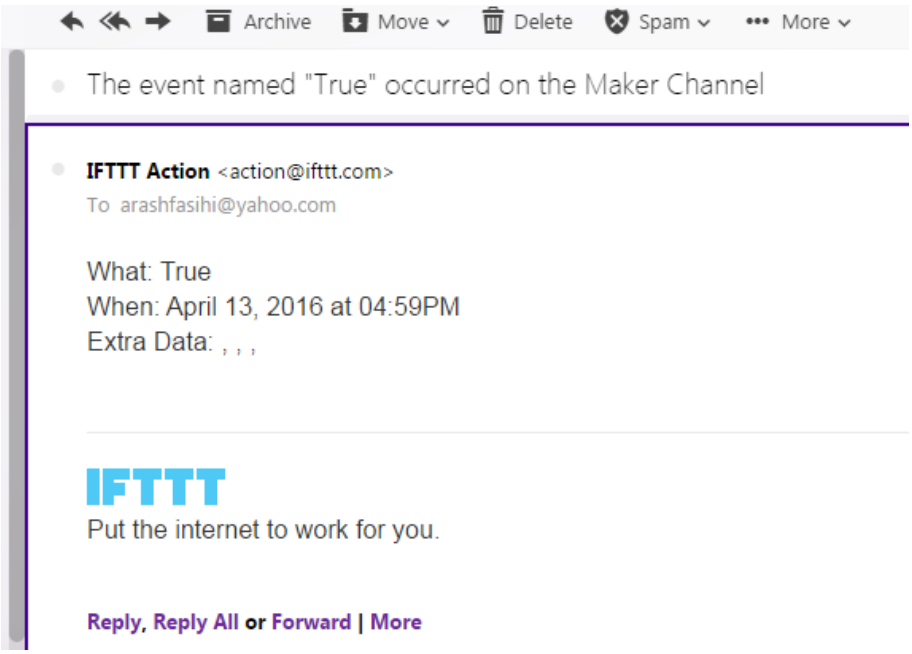


**Figure 4.10:** IFTTT Email Notification

# 5 Implementation

In this chapter we show how the offered architecture in the previous chapter can be implemented. In previous chapter we started explaining from the Core of the system and finished it on the User Interface. Here, we start from the User Interface to have better view on the overall functionality of the software. The programing language used is C-Sharp on Microsoft.NET version 4.6. The User interface is developed using HTML5.

## 5.1 User Interface

The fronted of the software is implemented by HTML5. To keep the flow of our work focused on IoT functionality, in this section we skip HTML coding. As already discussed, the User Interface is composed of two major parts. The first part which is marked as Engine Setup is to start and stop the engine. The second part which is marked as Engine Update is for extending the framework and uploading training dataset.

### 5.1.1 Engine Setup

Engine Setup tab in the User Interface is responsible for setting up the engine. It is illustrated in Figure 5.1 .

The User Interface is developed in such a way that the user can select the desired sensors among all connected sensors. Selecting special sensors automatically narrow down the four interface extensions to be compatible with this setting. For example, if the user selects all the sensors, the Machine Learning algorithm will deliver a classier model for all the sensors, the input stream interface also will receive and delivers all the sensor inputs, and finally the Training Data interface will extract the required information for all the sensors from the text file. The last interface—Action Selection, does not depend on the number of sensors. The Figure 5.2 shows the behavior of the User Interface when the user starts the engine. Apart from the sensor selection part, the User Interface includes four drop-down lists representing options for the four plugged interface extensions attached to the Core of the framework.

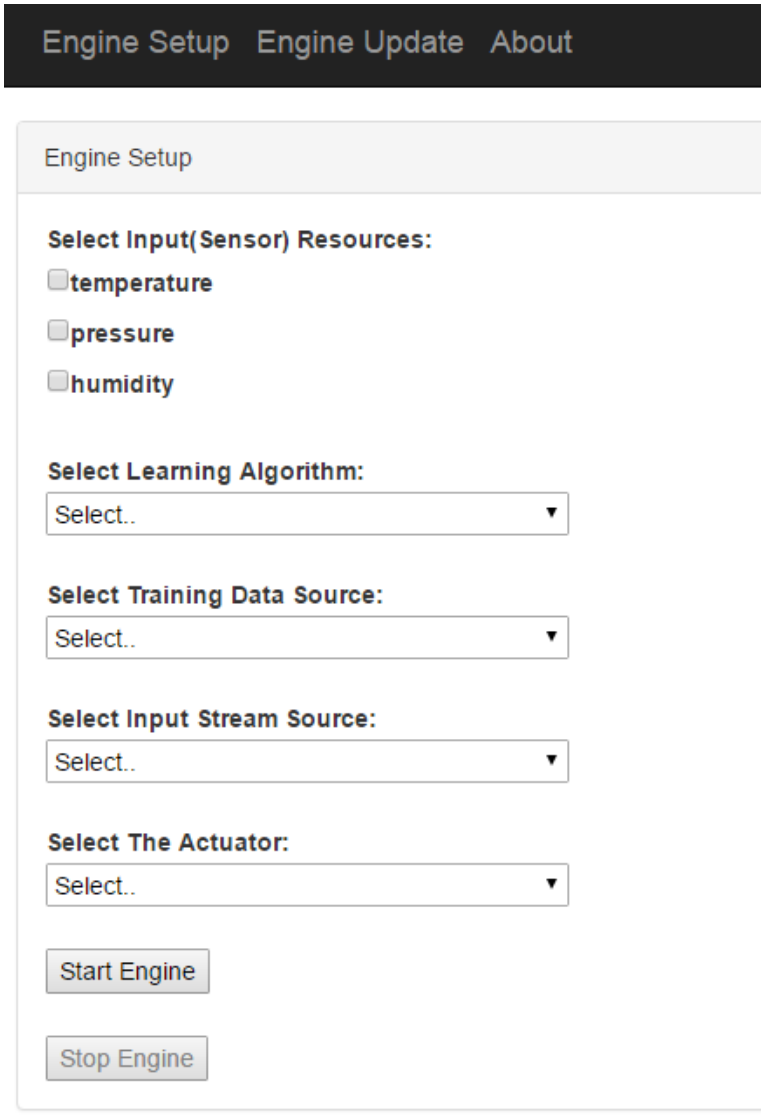


**Figure 5.1:** Engine Setup

As shown in the Figure 5.2, after selecting the proper fields, the Core of the system starts working. While the engine is running, all selection fields in the User Interface freeze until the user again stops the engine. According to the functionality of the system, the user can stop the engine at run-time and change the options and then again start the engine. For example, it is possible for user to stop the running engine which receives data input stream from resource A and again run the engine with option to get the data input stream from resource B. However, forgetting to select one or more fields will result in a notification from User Interface as shown in the Figure 5.3 .

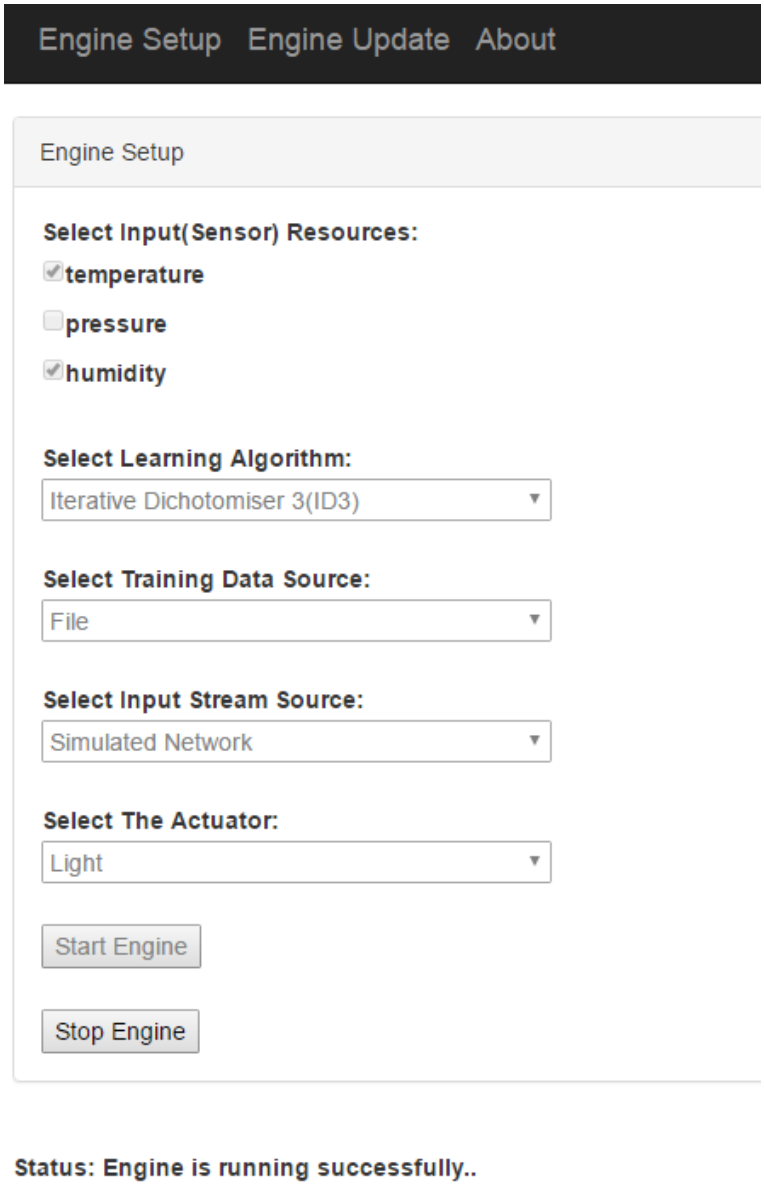


**Figure 5.2:** Engine Running

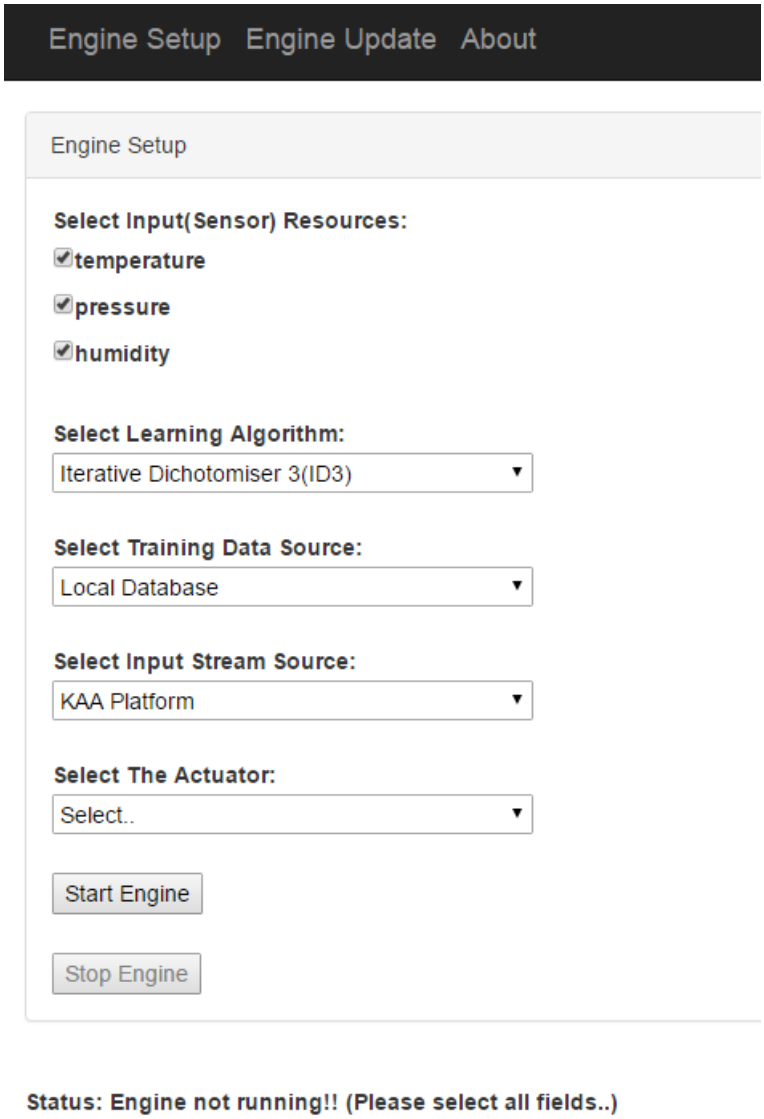


**Figure 5.3:** User Interface Interaction

### 5.1.2 Engine Update

The second tab in the User Interface is Engine Update. Updating the framework means adding new instances of the four interfaces available at the Core of the framework. We also refer to new instances as extensions throughout the text. In fact, this is the place where we want to integrate new modules to our software configuration. For example, if we want to add a new learning algorithm, it is enough to study the related interface and write the new extension i.e. the new algorithm and as it is shown in the Figure 5.4 only upload the DLL file to the server. The same story is true for three other interfaces.

**Listing 5.1** TrainingDataTable Structure

```
public class TrainingData
  {
     public DataTable TrainingDataTable { set; get; }

     public TrainingData() { }
     public TrainingData(int sourceNumber, string target, params string[] sourceList)
     {
        TrainingDataTable = new DataTable();
        for (int i = 0; i < sourceNumber; i++)
        {
           DataColumn column = TrainingDataTable.Columns.Add(sourceList[i],
               typeof(string));
        }
        DataColumn Decisioncolumn = TrainingDataTable.Columns.Add(target,
            typeof(string));
     }
  }
```

The four interfaces work independent of one another. The reason we choose to extend our software by adding already complied DLL files, will be discussed when we explain the structure of the Core. However, the important issue to be careful about is the fact that uploading a DLL file directly to the server from client side compromises the security of the system. To avoid this problem, when the system is in real use, we can leave this part only to the administrators of the system who can enter the Engine Update part by submitting corresponding credentials. The training dataset files can be checked out by the administrators before sitting on the server, as well.

In following, we will be describing the implementation of Core, and attached interfaces.

## 5.2 Training Data Interface

Clearly, this is the interface to get the training dataset to train our learning algorithm. Basically, such an interface can be skipped if the developer of the system wishes to get the training dataset from one single resource. However, there may be use-cases that tend to get the training dataset from new resources. In fact, the mentioned interface makes a variety of IoT solutions possible. Moreover, the interface is written in such a way that takes issues like the number of sensors into consideration. Therefore, the interface includes a data structure to incorporate the training dataset. For that purpose, a data type called TrainingDataTable as shown in Listing 5.1 is introduced.

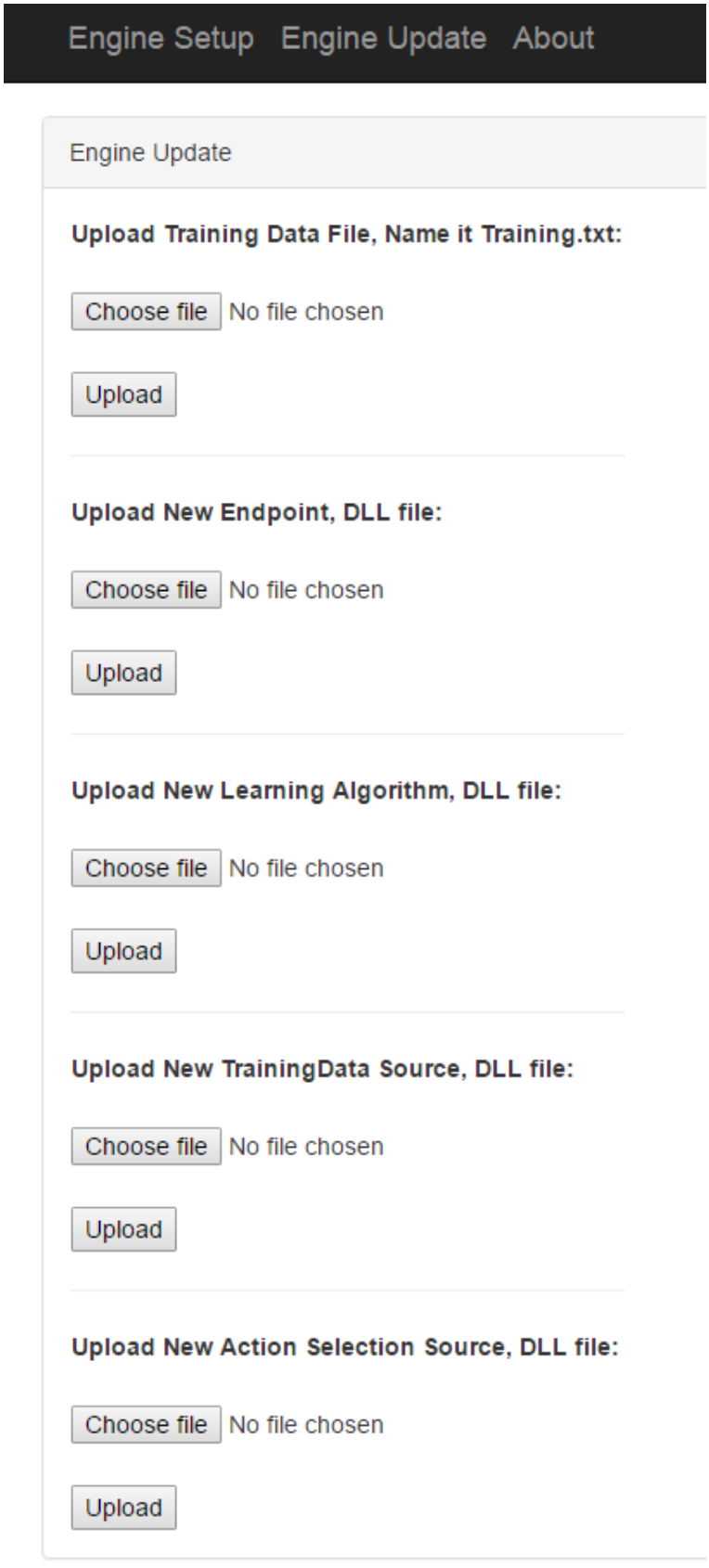


**Figure 5.4:** Engine Update

**Listing 5.2** ITrainingDataRead

```
public interface ITrainingDataRead
  {
      TrainingData TrainingDataSet { set; get; }

      void MakeTrainingData(int sourceNumber, string target, params string[] sourceList);

      void FillTrainingData();
  }
```

Practically, this data type is finally nothing but ADO.NET "datatable" type in .NET framework. However, the structure of the datatable i.e. the number of columns and the title of them is ruled by the number of sensors and their names which are in turn selected by user on the User Interface. Then, a base type called ITrainingDataRead is introduced as shown in the Listing 5.2 .

This C-Sharp interface type includes a TrainingData type which we already spoke about. As you can see, there is also a method called MakeTrainingData that accepts the properties of sensors. Finally, the last method called FillTrainingData that gives the possibility to receive training dataset from different resources.

The rest of this interface includes two more objects as shown in the Listing 5.3, an abstract base class and another interface type which is responsible for getting and setting meta-data for associated type. The reason we need this meta-data will be spoken later when we describe the Core mechanism. The abstract base class inheriting the FillTrainingData method from the interface ITrainingDataRead marks it as abstract to leave it to the writer of the instance of this interface. In fact, this is the part that makes difference between different extensions of this interface. The writer of new extension of Training Data interface fills the datatable from whatever resource possible, and attaches it to the Core of framework.

## 5.3 Machine Learning Algorithms Interface

Using the interface explained in the previous section the framework receives the training dataset and puts it in the language specific data structure explained. Now, using this interface, different Machine Learning algorithms can be attached to the Core to be trained by training dataset and react to unseen future stream incomings. It is important to mention that this framework is designed to accept only Supervised Machine Learning algorithms. That means, algorithms that are trained by training dataset which includes the input set together with the result for each input [KZP07]. The concept of learning

**Listing 5.3** Abstract Class TrainingDataRead

```
public abstract class TrainingDataRead : ITrainingDataRead
  {
     public int SourceNumber {set; get;}
     public string Target {set; get;}
     public string[] SourceList {set; get;}

     public TrainingData TrainingDataSet { get; set; }

     public void MakeTrainingData(int sourceNumber, string target, params string[]
         sourceList)
     {
        SourceNumber = sourceNumber;
        Target = target;
        SourceList = sourceList;
        TrainingDataSet = new TrainingData(sourceNumber, target, sourceList);
     }

     public abstract void FillTrainingData();

  }

  public interface ITrainingDataRead_Meta
  {
     string Name { get; }
  }
```

algorithms is described in section 3.2.3.1. In fact, the Knowledge Base (KB) of rule engine is the concrete knowledge of the user about the environment. The interface to attach algorithms can be used by next developers to write their own algorithms or use ready libraries written by other developers. We used Accord.Net library in which different classification algorithms are included [1]. The interface is shown in Listing 5.4.

In the highest level of abstraction, the interface is composed of three basic methods. The method Train which receives the training dataset from the interface explained in the previous section, then returns a generic type, "object". The object type is the generalization of a learning model, like a random decision tree or a neural network. Whatever it is, is passed to the next method called Classify. The Classify method responsibility is straightforward. Having the model collected from the Train method and a stream of sensor input as its input parameters, it delivers a string as result. In fact, this string is a description for the result of the data analysis. For example, it could be only true or false.

[1] http://accord-framework.net/

**Listing 5.4** Machine Learning Algorithms Interface

```
public interface IExtractModel
{
   object Train(DataTable mTrainingData);
   string Classify(object mModel, params string[] mInputStream);
   void StreamProperties(int sourceNumber, string target, params string[] sourceList);
}

public abstract class Classifier : IExtractModel
{
   protected int _sourceNumber;
   protected string _target;
   protected string[] _sourceList ;

   public void StreamProperties(int sourceNumber, string target, params string[]
       sourceList)
   {
      _sourceNumber = sourceNumber;
      _target = target;
      _sourceList = sourceList;
   }

   public abstract object Train(DataTable mTrainingData);
   public abstract string Classify(object mModel, params string[] mInputStream);
}

public interface IExtractModel_Meta
{
   string Name { get; }
}
```

The link between Train and Classify methods, is left to the Core of the framework. Basically, the link between Train and Classify is needed because the model obtained in Train will be used in Classify. To achieve this purpose, the Core itself passes the model obtained from Train to Classify, automatically. As you can see in Listing 5.4 the output parameter for Train method is the model which is input parameter for Classify method. The writer of the extension for this interface has to care only about the functionality of the two methods.

Finally, the StreamProperties method which receives the characteristics of input stream from User Interface. In reality, this method can be ignored when writing a new instance because this properties can be acquired by the passed datatable to Train method. However, having them explicitly in hand, ease the writing of sophisticated Machine Learning algorithms. In fact, it is good to mention that in all of our interfaces except the Action Selection interface we have a method to receive the properties of stream input. They

**Listing 5.5** Data Stream Endpoint Interface

```
public interface ISensorDataReader
  {
     void Start();
     void Stop();
  };

  public abstract class SensorDataReader : ISensorDataReader
  {
     protected InputAdaptor _inputAdaptor ;
     public delegate void DataRecievedHandler(params string[] data);
     public event DataRecievedHandler DataRecieved;
     public virtual void OnDataRecieved(params string[] data)
     {
       if (DataRecieved != null)
          DataRecieved(data);
     }

     public abstract void Stop();
     public abstract void Start();

     public void GetInputAdapter(InputAdaptor inputAdaptor)
     {
        _inputAdaptor = inputAdaptor;
     }
  }

  public interface ISensorDataReader_Meta
  {
     string Name { get; }
  }
```

all receive this properties from the Core of the system and as already mentioned the Core of the framework collects them through the interaction with User Interface. If you remember the start point in User Interface before any action is to select the sensors. As a result, at this point the Core realizes the input stream properties and communicates them to the corresponding interfaces.

This interface like other interfaces has its own special meta-data facility which the related usage will be explained in great detail later when we speak about the Core.

**Listing 5.6** Event Handler

```
public void DataRecievedEventHandler(params string[] data)
      {
         string result = _classifier.Classify(_model, data);
         _action.TakeAction(result);
      }
```

## 5.4 Data Stream Endpoint Interface

The two previous interfaces enable the framework to get the training dataset and extract learning model like a typical classifier to analyze the sensor input stream or any other event happening. While we have these tools in hand, the framework needs an interface to receive the input stream of data and deliver it to the learning model. We call this interface Data Stream Endpoint. the code for this interface comes in Listing 5.5 .

In the highest level of abstraction, there is a C-Sharp interface type, called ISensor-Datareader. Within the context of this interface type, we look at our Data Stream Endpoint interface as a container for a life-cycle. The method which is called Start, clearly starts this life-cycle and the method Stop puts an end to it. The Start and Stop methods here exactly correspond to the functionality of Start button and Stop button in the User Interface which we already saw. Then, we have an abstract base class, called SensorDataReader inheriting the Start and Stop methods from aforementioned interface type. The idea is, in the time between starting to listen to a channel of data stream and stopping it, an arrival of a data stream fires an event which must be handled in the Core of the system. In fact, the data arrival event occurring in the Data Stream Endpoint interface triggers its corresponding event-handler in the Core of the system. For that purpose, in this class there is an event type called DataRecieved based on the call-back function called DataRecievedHandler. Therefore, to react on the data arrival event in the Data Stream Endpoint extension, it is enough to call “OnDataRecieved (data)” and this event will be handled in the framework Core with a handler like in the Listing 5.6.

The class also includes a type called InputAdaptor. At endpoint, before passing the input stream to the “OnDataRecieved (data)” the data must be converted to an array of strings defined in the training dataset. That means the framework finally handles the data format in which for each sensor we have normalized string values like “low”, “mild”, and “high”. The training dataset must be in this format, as well. Therefore, if the sensor values from some resource were all double types we would have such a code in the Endpoint as shown in the Listing 5.7.

In Listing 5.8, you see the AdaptInputs method of the type InputAdaptor.

**Listing 5.7** Adaptor Mechanism

```
string[] data = _inputAdaptor.AdaptInputs(tempDoubleList.ToArray());
OnDataRecieved(data);
```

Finally, this interface also includes its own facility for handling its associated meta-data.

## 5.5 Action Selection Interface

So far, using the previous three interfaces, we have come to the point that we get data stream from some resource, analyze it using a Machine Learning model and get a result. Now, we are in the position that based on this result, an action must be taken. Defining the Action Selection mechanism all depends on the IoT solution we are working on. It could be turning off or turning on a switch, notifying the user of the system, or even controlling some other devices and sensors. In the implemented version of the architecture under discussion we kept the interface for Action Selection as simple as possible to give the possibility to the user of framework to decide about Action Selection part, completely from scratch. The Listing 5.9 shows the simple code for this interface.

It is worth mentioning at this point that in Artificial Intelligence and Expert Systems, the term Action Selection Mechanisms (ASM) refers to the techniques used to select an action among other possible actions. Literally, it answers to the question: "What to do next?" [BB06]. However, in our framework the term is used somehow differently. In Action Selection interface, actually the action is already selected by an Artificial Intelligence (AI) Action Selection Mechanism like a Bayesian network or a decision tree. Here, we want to utilize that selected action (true or false), for example, to turn on/off a light or send an Email.

## 5.6 Framework Core

Up to this point, we have discussed enough about the components attached to the Core of the system but we never went deep into the Core implementation. To bring up those components again, they are nothing but the four interfaces together with a User Interface. Basically, the system is supposed to function in such a way that the user can select the preferred extensions of the four interfaces through User Interface and even switch among them at run-time. The configuration of the software should be in such a way that the integration of new extensions of the interfaces takes minimum effort.

**Listing 5.8** AdaptInputs Method

```
public string[] AdaptInputs(params double[] inData)
     {
        List<string> outData = new List<string>();
        int i = 0;
        foreach(double d in inData)
        {
           if(_sourceList[i] == "temperature")
           {
              if (d < 30.0)
                 outData.Add("low");
              else if (d >= 30.0 && d < 40.0)
                 outData.Add("mild");
              else
                 outData.Add("high");
           }

           else if (_sourceList[i] == "pressure")
           {
              if (d < 700.0)
                 outData.Add("low");
              else if (d >= 700.0 && d < 1000.0)
                 outData.Add("mild");
              else
                 outData.Add("high");
           }

           else if (_sourceList[i] == "humidity")
           {
              if (d < 20.0)
                 outData.Add("low");
              else if (d >= 20.0 && d < 40.0)
                 outData.Add("mild");
              else
                 outData.Add("high");
           }
           i++;
        }
        return (outData.ToArray());
     }
```

**Listing 5.9** Action Selection

```
public interface IActionSelection
   {
      void TakeAction(string result);
   }
   public abstract class Actionselection: IActionSelection
   {
      public abstract void TakeAction(string result);
   }
   public interface IActionSelection_Meta
   {
     string Name { get; }
   }
```

That means to add or remove modules without any recompilation. To achieve this goal, Component Based Software Engineering (CBSE) meets our requirements [HC01]. This method in .NET Framework 4 and higher versions is introduced as Managed Extensibility Framework (MEF).

### 5.6.1 Why Managed Extensibility Framework (MEF)

The Managed Extensibility Framework is a composition layer for .NET that improves the flexibility, maintainability and testability of large applications. MEF can be used for third-party plugin extensibility, or it can bring the benefits of a loosely-coupled plugin-like architecture to regular applications. It is a library for creating lightweight, extensible applications. It allows application developers to discover and use extensions with no configuration required. It also lets extension developers easily encapsulate code and avoid fragile hard dependencies. MEF not only allows extensions to be reused within applications, but across applications as well. Using MEF is as easy as, export it, import it and compose it[2]. MEF presents a simple solution for the run-time extensibility problem. Until now, any application that wanted to support a plugin model needed to create its own infrastructure from scratch. Those plugins would often be application-specific and could not be reused across multiple implementations. MEF provides a standard way for the host application to expose itself and consume external extensions. Extensions, by their nature, can be reused amongst different applications. However, an extension could still be implemented in such a way that is application-specific. Extensions themselves can depend on one another and MEF will make sure they are wired together in the correct order (another thing you won't have to worry about). MEF offers a set of discovery

[2] https://msdn.microsoft.com/en-us/library/dd460648(v=vs.110).aspx

approaches for your application to locate and load available extensions. MEF allows tagging extensions with additional meta-data which facilitates rich querying and filtering. One of the key design goals of our IoT framework is, it should be extensible and this is the reason we decided to use MEF. With MEF we can use different algorithms (as and when it becomes available) for sensor data analytics: e.g. drop an analytics assembly into a folder and it instantly becomes available to the application.

### 5.6.2 Implementation of Core Using MEF

The Core is implemented on the code behind for the HTML page Engine Setup in User Interface. The application in its root directory has four folders to accommodate corresponding DLL files. That is, initially we compile the four interfaces together with their linked libraries and put them in the four aforementioned folders in the root directory of the application. Now, the user of the system should write extensions of these interfaces then compile them, make DLL files and upload them to the server. The extensions will automatically go in the related folders to sit beside their interface DLL files. The extensions must have proper meta-data in the format of a meaningful string which will go to the corresponding drop-down lists in the page Engine Setup, automatically without any need for a configuration file. Now, the user can select preferred extensions at run-time and start the engine. The extensions will be automatically wired up together to make an IoT solution work. The role that MEF plays here is clear. It should discover all extensions and present them to user on the User Interface and when the user picks his/her own preferred extensions, it hast to wire up them together. This is all possible with code like the one showed in Listing 5.10 .This is the code behind for the HTML page Engine Setup discussed previously.

The first field for this class is a “CompositionContainer” type object. This is the start point on bringing in the MEF. Instead of this explicit registration of available components, MEF provides a way to discover them implicitly, via composition. A MEF component, called a part, declaratively specifies both its dependencies (known as imports) and what capabilities (known as exports) it makes available. When a part is created, the MEF composition engine satisfies its imports with what is available from other parts. The core of the MEF composition model is the composition container, which contains all the parts available and performs composition. (That is, the matching up of imports to exports.)

In the next field, we have an object of a type called Driver. Driver is a type which we will be speaking about, later in detail. Shortly, it has been developed for two purposes. First, to extract extensions that user selects, second extract all meta-data related to extensions and present them to user. We could have even several extensions of driver as you can see in the Listing 5.10 it is marked with import attribute. Since in this project we need

**Listing 5.10** Core Implementation

```
public partial class _Default : Page
  {
     private CompositionContainer _container;

     [Import(typeof(Driver))]
     public Driver driver;

     private object _model;
     private IExtractModel _classifier = null;
     private SensorDataReader _reader = null;
     private IActionSelection _action = null;

     public _Default();
     protected void Page_Load(object sender, EventArgs e);
     public void DataRecievedEventHandler(params string[] data);
     protected void Startbtn_Click(object sender, EventArgs e);
     protected void Stopbtn_Click(object sender, EventArgs e);

  }
```

only this driver we put it exactly beside -Default class in the same executing assembly. The other fields are of our interfaces types which will get their objects in the method Startbtn-Click, that means when Start button is clicked. The field that has "object" type, is the model extracted by Train method of classifier and will be used in Classify method of the classifier.

The Listing 5.11 shows the constructor of this class.

The composition container makes use of a catalog. A catalog is an object that makes available parts discovered from some source. MEF provides catalogs to discover parts from a provided type, an assembly, or a directory. Here, the paths for the DLL files in the root directory of application are added. Also, the path for current executing assembly is added to import Driver object. Therefore, in the constructor of page i.e. the -Default method we import a driver and from driver all other imports i.e. interface extensions happens.

The next method is Page-Load method and you see the code in the Listing 5.12 . When the page loads the driver as discussed is already imported. In Page-Load function, we call the DriverGetAllMetaData method of driver to discover all meta-data and assign them to drop-down lists.

To show that how the meta-data for extensions is discovered with this method let's take Input Stream drop-down list as example. As you see in the Figure 5.5 it has two items. The two items are, in fact the meta-data string in each extension of Data Stream

**Listing 5.11** Core Constructor

```
public _Default()
      {
         var catalog = new AggregateCatalog();
         catalog.Catalogs.Add(new AssemblyCatalog(typeof(_Default).Assembly));
         catalog.Catalogs.Add(new DirectoryCatalog(Server.MapPath("~/AlgorithimsDLL")));
         catalog.Catalogs.Add(new DirectoryCatalog(Server.MapPath("~/EndpointDLL")));
         catalog.Catalogs.Add(new DirectoryCatalog(Server.MapPath("~/ActionDLL")));
         catalog.Catalogs.Add(new DirectoryCatalog(Server.MapPath("~/TrainingDataDLL")));

         _container = new CompositionContainer(catalog);

         try
         {
            this._container.ComposeParts(this);
         }
         catch (CompositionException compositionException)
         {
            Console.WriteLine(compositionException.ToString());
         }
      }
```

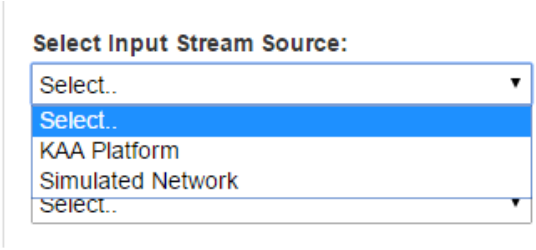


**Figure 5.5:** DropDownList

Endpoint interface. That meta-data exists in export attribute of each extension. For example the KAA Platform extension starts with code listing like in the listing 5.13. The Listing does not show the whole code for KAALogServer extension of SensorDatareader. However, the string "KAA Platform" in export attribute of this extension is discovered by MEF and presented in drop-down list as you can see in Figure 5.5.

The next method DataRecievedEventHandler corresponds to the delegate introduced in the Data Stream Endpoint interface. The idea is the user in client side, extracts the reader, model, classifier, and action in Startbtn-Click, and delivers them to the server which will be busy with this event-handler until again the user on client side, with Stopbtn-Click click, commands the server to stop. This event-handler is invoked when a stream of data arrives. You can see it in the Listing 5.6. In this method for the first time we see the usage of -model, -classifier, -action fields of the class under discussion.

**Listing 5.12** Page Load

```
protected void Page_Load(object sender, EventArgs e)
     {
        List<string> modelsList = new List<string>();
        List<string> endpointsList = new List<string>();
        List<string> readersList = new List<string>();
        List<string> actionsList = new List<string>();

        driver.DriverGetAllMetaDatas(out modelsList,out endpointsList,out
            readersList,out actionsList);

        if (!IsPostBack)
        {
           Stopbtn.Enabled = false;

           foreach (string st in modelsList)
           {
              LearningAlgorithmddl.Items.Add(new ListItem(st));
           }
           foreach (string st in readersList)
           {
              TrainingDataSourceddl.Items.Add(new ListItem(st)); }

           foreach(string st in endpointsList)
           {
              InputStreamSourceddl.Items.Add(new ListItem(st));
           }
           foreach(string st in actionsList)
           {
              Actuatorddl.Items.Add(new ListItem(st));
           }

        }
     }
```

In the next method Startbtn-Click, we will see that they are user's preferred interface extensions.

The Listing 5.14 shows the code for Startbtn-Click method. What happens here is straightforward. In this function, reader, model, endpoint and act are four strings which will get their values from drop-down lists in User Interface. Then we will pass them together with another parameter called inputs to DriverRunning method of Driver to select user's preferred extensions of interfaces. Keep in mind that inputs is a list of strings getting values from user's preferred sensors in the related check-box list in the User Interface page Engine Setup. The latter determines the number and type of sensors for all interfaces.

**Listing 5.13** An Example of Data Stream Endpoint Extension

```
[Export(typeof(SensorDataReader))]
[ExportMetadata("Name", "KAA Platform")]
public class KAALogServer : SensorDataReader
{
   public HttpListener listener;
   public override void Start()
   {
      listener = new HttpListener();
      listener.Prefixes.Add("http://+:10000/");

      listener.Start();
      listener.BeginGetContext(new AsyncCallback(ListenerCallback), listener);
   }
   public override void Stop()
   {
      listener.Stop();
   }
```

By invoking DriverRunning method of Driver the four fields, -model, -classifier, -reader and -action get their objects. By calling -reader.Start () the engine starts running. Also, pay attention that as you can see in the code listing here in the framework Core, we subscribe the event-handler to the event in the Data Stream Endpoint interface through this line shown in the Listing 5.15 .

Finally, the last method in this class is Stopbtn-Click which is invoked when the Stop button in User Interface is clicked. The code for this part is shown in Listing 5.16. When the user-preferred Data Stream Endpoint extension on method Startbtn-Click is assigned to the field called -reader, we keep it in an application level (not session level) variable. We do this, to be able to use -reader once again in Stopbtn-Click method. The reason to do so is, the User Interface and the engine Core talk to each other on top of HTTP protocol and HTTP is basically a stateless protocol and when client and server talk to each other, until next communication they forget about one another. Eventually, as you see in the listing the Stop method of -reader is called to finish listening. But, before using the application level variable, an explicit type-cast is necessary as shown in the Listing 5.16 .

So far, we have been speaking about the -Default class and its methods, which is responsible for setting up the engine. One of the fields inside this class is of type Driver. The object of type Driver within this class must be imported by MEF. The Driver object is responsible for two objectives. First, it imports all interface extensions from the DLL repositories and based on user selections, picks the preferred ones. Second, when the Engine Setup page is about to load on the client machine, it discovers all the meta-data available in interface extensions and present them to the user in drop-down lists of the

**Listing 5.14** Start-Button Method

```
protected void Startbtn_Click(object sender, EventArgs e)
    {
       string reader = string.Empty;
       string model = string.Empty;
       string endpoint = string.Empty;
       string act = string.Empty;
       List<string> inputs;
       if(TrainingDataSourceddl.Text == "Select.." || LearningAlgorithmddl.Text ==
           "Select.." || InputStreamSourceddl.Text == "Select.." || Actuatorddl.Text ==
           "Select.." || SensorsCheckbl.SelectedIndex == -1)
       {
          Label1.Text = "Status: Engine not running!! (Please select all fields..)";
          return;
       }
       else
       {
          Label1.Text = string.Empty;
       }
       reader = TrainingDataSourceddl.Text;
       model = LearningAlgorithmddl.Text;
       endpoint = InputStreamSourceddl.Text;
       act = Actuatorddl.Text;
       inputs = new List<string>();

       foreach (ListItem li in SensorsCheckbl.Items)
       {
          if(li.Selected == true)
          {
             inputs.Add(li.Text);
          }
       }
       InputAdaptor inputAdaptor = new InputAdaptor();
       inputAdaptor.StreamProperties(inputs.Count, inputs.ToArray()); // input adaptor
           is set here
       Startbtn.Enabled = false;
       Stopbtn.Enabled = true;
       LearningAlgorithmddl.Enabled = false;
       TrainingDataSourceddl.Enabled = false;
       InputStreamSourceddl.Enabled = false;
       Actuatorddl.Enabled = false;
       SensorsCheckbl.Enabled = false;

       _model = driver.DriverRunning(out _classifier, out _reader, out _action ,reader,
           model,endpoint,act,inputs);
       _reader.GetInputAdapter(inputAdaptor); //input adaptor is passed to endpoint here
       Application["reader"] = _reader;
       _reader.DataRecieved += DataRecievedEventHandler;
       _reader.Start();

       Label1.Text = "Status: Engine is running successfully..";
    }
```

**Listing 5.15** Event Handler Subscription

```
_reader.DataRecieved += DataRecievedEventHandler;
```

**Listing 5.16** Stop-Button Method

```
protected void Stopbtn_Click(object sender, EventArgs e)
{
   Startbtn.Enabled = true;
   Stopbtn.Enabled = false;

   LearningAlgorithmddl.Enabled = true;
   TrainingDataSourceddl.Enabled = true;
   InputStreamSourceddl.Enabled = true;
   Actuatorddl.Enabled = true;
   SensorsCheckbl.Enabled = true;

   if (Application["reader"] != null)
   {
      SensorDataReader reader =(SensorDataReader) Application["reader"];
      reader.Stop();
      reader = null;
      Application["reader"] = null;
   }
   Label1.Text = "Status: User stopped the engine..";
}
```

page. Once again, keep in mind that the two classes -Default and Driver reside under the same name-space and together make the code behind for the HTML code of page Engine Setup in the front-end of the application. The Listing 5.17 shows the code for Driver class.

The class includes four C-Sharp IEnumerable type variables. They are readers, models, actions and endpoints. The idea is simple, these variables finally import all interface extensions from DLL files. Then the method DriverRunning as already discussed selects and delivers the ones that user prefers on the User Interface. The method DriverRunning has four string variables in its input arguments which come from drop-down lists in the page Engine Setup, and based on them within four foreach loops, picks the selected extensions. Moreover, it has some output types that finally serve as user preferred extensions. The Listing 5.18 shows the corresponding foreach loops in the DriverRunning method.

The method DriverGetAllMetaDatas extracts all the meta-data in interface extensions and delivers them to the User Interface to be shown in the corresponding drop-down lists in the page Engine Setup. This is done by looping through the four IEnumerable

**Listing 5.17** Driver Class

```
[Export(typeof(Driver))]
 public class Driver
 {
    [ImportMany]
    IEnumerable<Lazy<ITrainingDataRead, ITrainingDataRead_Meta>> readers = null;
    [ImportMany]
    IEnumerable<Lazy<IExtractModel, IExtractModel_Meta>> models = null;
    [ImportMany]
    IEnumerable<Lazy<IActionSelection, IActionSelection_Meta>> actions = null;
    [ImportMany]
    IEnumerable<Lazy<SensorDataReader, ISensorDataReader_Meta>> endPoints = null;

    public object DriverRunning(out IExtractModel classifier, out SensorDataReader
        sdReader, out IActionSelection action, string Reader, string Model, string
        Endpoint, string act, List<string> inputs);

    public void DriverGetAllMetaDatas(out List<string> ModelsList, out List<string>
        EndpointsList, out List<string> ReadersList, out List<string> ActionsList);
 }
```

variables called models, endpoints, readers and actions. The Listing 5.19 shows how it is done this method.

**Listing 5.18** DriverRunning Method Mechanism

```
foreach (Lazy<ITrainingDataRead, ITrainingDataRead_Meta> i in readers)
{
   if (i.Metadata.Name.Equals(reader))
   {
      i.Value.MakeTrainingData(number, "decision", inputNames);
      i.Value.FillTrainingData();
      dt = i.Value.TrainingDataSet.TrainingDataTable;
   }
}
foreach (Lazy<IExtractModel, IExtractModel_Meta> i in models)
{
   if (i.Metadata.Name.Equals(model))
   {
      i.Value.StreamProperties(number, "decision", inputNames);
      MDL = i.Value;
      ob = i.Value.Train(dt);
   }

}
foreach (Lazy<IActionSelection, IActionSelection_Meta> i in actions)
{
   if (i.Metadata.Name.Equals(actuator))
      ACT = i.Value;
}
foreach (Lazy<SensorDataReader, ISensorDataReader_Meta> i in endPoints)
{
   if (i.Metadata.Name.Equals(endpoint))
      SDR = i.Value;
}
```

**Listing 5.19** DriverGetAllMetaDatas Method Mechanism

```
List<string> modelsList = new List<string>();
         List<string> endpointsList = new List<string>();
         List<string> readersList = new List<string>();
         List<string> actionsList = new List<string>();

         foreach (Lazy<IExtractModel, IExtractModel_Meta> i in models)
         {
            modelsList.Add(i.Metadata.Name);
         }
         foreach (Lazy<SensorDataReader, ISensorDataReader_Meta> i in endPoints)
         {
            endpointsList.Add(i.Metadata.Name);
         }
         foreach (Lazy<ITrainingDataRead, ITrainingDataRead_Meta> i in readers)
         {
            readersList.Add(i.Metadata.Name);
         }
        foreach (Lazy<IActionSelection, IActionSelection_Meta> i in actions)
         {
            actionsList.Add(i.Metadata.Name);
         }
```

# 6 Summary and Future Work

In this thesis, we discussed the position of rule-based engines in IoT applications. Essentially, any Iot platform needs a mechanism to process and utilized the incoming data streams from sensor networks. In the first chapter, we discussed that IoT is still in its early years of progress which is why the main trend in IoT researches is around device connectivity and hardware management. In this regard, many crutial topics are involved. For example, how physical objects can be integrated into the Internet body easily, how they can remain connected persistently, how they can transmit data safely and securely, and so on. IoT platforms, apart from device connectivity and device management, need a software module to process and react on the received data from connected devices in real-time fashion. To give enough flexibility to this software module a software framework is the solution. In the heart of IoT platforms, a software framework makes developing a variety of IoT applications possible. The framework itself is usually host for a rule-based system mechanism to process and react on data.

In chapter 2, we discussed the essential background to understand IoT applications. In chapter 3, we narrowed down our discussion to IoT platforms and their building blocks. Among the blocks, the processing and action management block is discussed in more detail which is the place to host a software framework as rule-based engine. furthermore in this chapter, we discussed the current rule engines in IoT. In chapter 4, we discussed the architecture and functionality of our offered framework and finally, in chapter 5 we described the related implementation.

## Future Work

In future, to add more features and capabilities to our offered framework, several issues are involved. In this section, we briefly bring up those challenges, offering possible approaches to cope with.

The first challenge for such a framework is of course the problem of training dataset. In the implemented version of architecture we get the user to upload the training dataset through a text file following a special format. User can enter the wrong format which could in turn result in unexpected exception. This part is still open to work on. The user

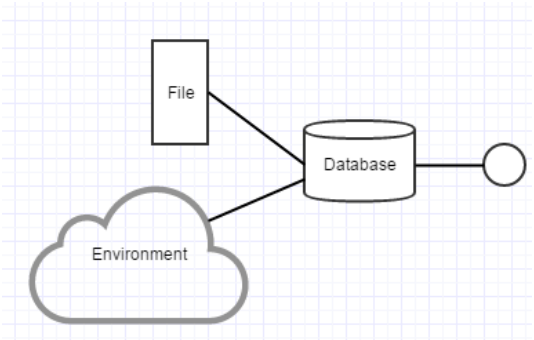


**Figure 6.1:** Improved Learning Algorithms Training Interface

can either enter wrong format or even if the format is correct, inconsistent data. To cope with this situation, one can think of controlling user through proper exception handler until they enter proper training dataset. However, a better approach could be parsing the training dataset and correcting it within the context of software.

Moreover, we can also improve the mechanism of Training Data interface. In fact, having the previously described interface allows to get the training dataset either from text file or database which work independent from one another. It is good to mention that an improvement could be to make a link between file and database. That is, when the file is uploaded to the server, the system first adds the content of the file to database then trains the Learning algorithm. this way, the database is the only endpoint to get training dataset. This approach allows to have a Knowledge Base (KB) for Learners which is filled by the user knowledge of environment(e.g. text file) and real environment observations, as well. Basically, training a Learner best happens by monitoring real environment events, which for this project is open field to work on. Schematically, you can see it in the Figure 6.1.

The second issue is the security of system. In this implementation we excluded any kind of security issues. As it is obvious, security of data flow, like confidentiality and integrity of data coming and going out of the application are very important which can be solved with Cryptographic and other security mechanisms.

The third improvement is all about Visualization. It is very beneficial to enable users to see the happenings in the skin of the application. For example, the incoming sensor data can be visualized effectively. Even, it is possible to give statistical outcomes to the user, based on their specific requirements.

The last but not the least improvement to our framework could be enabling the Action Selection interface to be a Web service requester. That is, when the result for data analysis is ready, it can be utilized to invoke a proper Web service based on the requirements of the user. How to request and bind to a Web service is a matter of challenge in Action

Selection interface. In Figure 4.1 showing the overall architecture of the system, this improvement is predicted. Finding a Web service can be achieved using UDDI (Universal Description, Discovery, and Integration) standard [ACKM04]. However, in section 4.2, when describing our test use-case we almost achieved a similar goal by subscribing to some particular Service on IFTTT i.e. Email service.

# Bibliography


[AA12] N. A. Ali, M. Abu-Elkheir. “Data management for the internet of things: Green directions.” In: *Globecom Workshops (GC Wkshps), 2012 IEEE*. IEEE. 2012, pp. 386–390 (cit. on p. 11).

[ACKM04] G. Alonso, F. Casati, H. Kuno, V. Machiraju. *Web services*. Springer, 2004 (cit. on p. 81).

[AHA13] M. Abu-Elkheir, M. Hayajneh, N. A. Ali. “Data management for the internet of things: Design primitives and solution.” In: *Sensors* 13.11 (2013), pp. 15582–15612 (cit. on pp. 22, 23).

[AKAH14] M. Aazam, I. Khan, A. A. Alsaffar, E.-N. Huh. “Cloud of Things: Integrating Internet of Things and cloud computing and the issues involved.” In: *Applied Sciences and Technology (IBCAST), 2014 11th International Bhurban Conference on*. IEEE. 2014, pp. 414–419 (cit. on pp. 18, 21, 25).

[AMN15] M. H. Asghar, N. Mohammadzadeh, A. Negi. “Principle application and vision in Internet of Things (IoT).” In: *Computing, Communication & Automation (ICCCA), 2015 International Conference on*. IEEE. 2015, pp. 427–431 (cit. on pp. 11, 15).

[BB06] C. Brom, J. Bryson. “Action selection for intelligent systems.” In: *European Network for the Advancement of Artificial Cognitive Systems* (2006) (cit. on p. 66).

[BL12] M. Blackstock, R. Lea. “IoT mashups with the WoTKit.” In: *Internet of Things (IOT), 2012 3rd International Conference on the*. IEEE. 2012, pp. 159–166 (cit. on p. 40).

[BL14] M. Blackstock, R. Lea. “Toward a Distributed Data Flow Platform for the Web of Things.” In: *Web of Things (WoT), 2014 5th International Workshop on the*. 2014 (cit. on p. 39).

[CDBZ12] A. P. Castellani, M. Dissegna, N. Bui, M. Zorzi. “WebIoT: A web application framework for the internet of things.” In: *Wireless Communications and Networking Conference Workshops (WCNCW), 2012 IEEE*. IEEE. 2012, pp. 202–207 (cit. on p. 12).

[CFS+14] C. Y. Chen, J. H. Fu, T.-L. Sung, P.-F. Wang, E. Jou, M.-W. Feng. "Complex event processing for the internet of things and its applications." In: *Automation Science and Engineering (CASE), 2014 IEEE International Conference on*. IEEE. 2014, pp. 1144–1149 (cit. on p. 43).

[Giu10] D. Giusto. *A. lera, G. Morabito, l. Atzori (Eds.) The Internet of Things*. 2010 (cit. on p. 11).

[GKN+11] A. Gluhak, S. Krco, M. Nati, D. Pfisterer, N. Mitton, T. Razafindralambo. "A survey on facilities for experimental internet of things research." In: *IEEE Communications Magazine* 49.11 (2011), pp. 58–67 (cit. on p. 11).

[Haq04] U. Haque. "Pachube project." In: *Pachube project* (2004) (cit. on p. 12).

[HC01] G. T. Heineman, W. T. Councill. "Component-based software engineering." In: *Putting the pieces together, addison-westley* (2001), p. 5 (cit. on p. 68).

[HL10a] Y. Huang, G. Li. "A semantic analysis for internet of things." In: *Intelligent Computation Technology and Automation (ICICTA), 2010 International Conference on*. Vol. 1. IEEE. 2010, pp. 336–339 (cit. on p. 16).

[HL10b] Y. Huang, G. Li. "Descriptive models for Internet of Things." In: *Intelligent Control and Information Processing (ICICIP), 2010 International Conference on*. IEEE. 2010, pp. 483–486 (cit. on p. 16).

[HTM+14] J. Holler, V. Tsiatsis, C. Mulligan, S. Avesand, S. Karnouskos, D. Boyle. *From Machine-to-machine to the Internet of Things: Introduction to a New Age of Intelligence*. Academic Press, 2014 (cit. on pp. 17, 18, 41).

[iot15] iot-analytics.com. *IOT PLATFORMS, The central backbone for the Internet of Things(White Paper)*. Tech. rep. Jan. 2015 (cit. on pp. 29–32).

[JC14] C. Jun, C. Chi. "Design of complex event-processing IDS in internet of things." In: *Measuring Technology and Mechatronics Automation (ICMTMA), 2014 Sixth International Conference on*. IEEE. 2014, pp. 226–229 (cit. on p. 43).

[KGB] K. Krishnakumar, J. Gubbi, R. Buyya. "A Framework for IoT Sensor Data Analytics and Visualisation in Cloud Computing Environments." In: () (cit. on pp. 12, 13).

[KRBA14] M. Kiran, P. Rajalakshmi, K. Bharadwaj, A. Acharyya. "Adaptive rule engine based IoT enabled remote health care data acquisition and smart transmission system." In: *Internet of Things (WF-IoT), 2014 IEEE World Forum on*. IEEE. 2014, pp. 253–258 (cit. on p. 36).

[KZP07] S. B. Kotsiantis, I. Zaharakis, P. Pintelas. *Supervised machine learning: A review of classification techniques*. 2007 (cit. on p. 61).

[MF10] F. Mattern, C. Floerkemeier. “From the Internet of Computers to the Internet of Things.” In: *From active data management to event-based systems and more*. Springer, 2010, pp. 242–259 (cit. on pp. 18, 19).

[MLL+10] W. Miao, T. LU, F. LING, et al. “Research on the Architecture of Internet of Things [C].” In: *Proceedings of the 3rd International Conference on Advanced Computer Theory and Engineering: August*. 2010, pp. 20–22 (cit. on p. 20).

[NLM+06] S. Nath, J. Liu, J. Miller, F. Zhao, A. Santanche. “Sensormap: a web site for sensors world-wide.” In: *Proceedings of the 4th international conference on Embedded networked sensor systems*. ACM. 2006, pp. 373–374 (cit. on p. 12).

[OBE07] S. OBEROI. “Introduction to complex event processing & data streams.” In: *SOA World Magazine, S* (2007), pp. 20–24 (cit. on p. 42).

[PBR15] T. Padiya, M. Bhise, P. Rajkotiya. “Data Management for Internet of Things.” In: *Region 10 Symposium (TENSYMP), 2015 IEEE*. IEEE. 2015, pp. 62–65 (cit. on p. 24).

[PCZ09] W. Peng, J. Chen, H. Zhou. “An Implementation of ID3—Decision Tree Learning Algorithm.” In: *From web. arch. usyd. edu. au/wpeng/DecisionTree2. pdf Retrieved date: May* 13 (2009) (cit. on p. 38).

[Qui86] J. R. Quinlan. “Induction of decision trees.” In: *Machine learning* 1.1 (1986), pp. 81–106 (cit. on p. 37).

[RNI95] S. Russell, P. Norvig, A. Intelligence. “A modern approach.” In: *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs* 25 (1995), p. 27 (cit. on pp. 34, 36, 37).

[RSS+12] B. Rao, P. Saluia, N. Sharma, A. Mittal, S. Sharma. “Cloud computing for Internet of Things & sensing based applications.” In: *Sensing Technology (ICST), 2012 Sixth International Conference on*. IEEE. 2012, pp. 374–380 (cit. on p. 25).

[Sou12] T. B. Sousa. “Dataflow programming concept, languages and applications.” In: *Doctoral Symposium on Informatics Engineering*. Vol. 7. 2012, p. 13 (cit. on pp. 39, 40).

[STJ14] D. Singh, G. Tripathi, A. J. Jara. “A survey of internet-of-things: future vision, architecture, challenges and services.” In: *Internet of Things (WF-IoT), 2014 IEEE World Forum on*. IEEE. 2014, pp. 287–292 (cit. on p. 15).

[TW10] L. Tan, N. Wang. “Future internet: The internet of things.” In: *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on*. Vol. 5. IEEE. 2010, pp. V5–376 (cit. on p. 11).

[VF13] O. Vermesan, P. Friess. *Internet of things: converging technologies for smart environments and integrated ecosystems*. River Publishers, 2013 (cit. on p. 11).

[VFG+11] O. Vermesan, P. Friess, P. Guillemin, S. Gusmeroli, H. Sundmaeker, A. Bassi, I. S. Jubert, M. Mazura, M. Harrison, M. Eisenhauer, et al. "Internet of things strategic research roadmap." In: *Internet of Things-Global Technological and Societal Trends* (2011), pp. 9–52 (cit. on p. 22).

[Wei12] C. M. Wei. "Complex Event Processing Mechanism in Internet of Things and its Application in Logistics." In: *Applied Mechanics and Materials*. Vol. 235. Trans Tech Publ. 2012, pp. 309–313 (cit. on p. 43).

[XJH13] X. Xingmei, Z. Jing, W. He. "Research on the basic characteristics, the key technologies, the network architecture and security problems of the Internet of things." In: *Computer Science and Network Technology (ICCSNT), 2013 3rd International Conference on*. IEEE. 2013, pp. 825–828 (cit. on p. 15).

[YCL11] W. Yao, C.-H. Chu, Z. Li. "Leveraging complex event processing for smart hospitals using RFID." In: *Journal of Network and Computer Applications* 34.3 (2011), pp. 799–810 (cit. on p. 41).

[ZBC+14] A. Zanella, N. Bui, A. Castellani, L. Vangelista, M. Zorzi. "Internet of things for smart cities." In: *Internet of Things Journal, IEEE* 1.1 (2014), pp. 22–32 (cit. on p. 15).

All links were last followed on May 24, 2016.

**Declaration**

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

---

place, date, signature