A Platform as a Service for Concurrency-based Applications Provisioning in Internet of Things

Deepak Kumar Aggarwal

A thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements for the

Degree of Master of Applied Science at

Concordia University

Montréal, Québec, Canada

October 2017

CONCORDIA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: **Deepak Kumar Aggarwal**

Entitled: **A Platform as a Service for Concurrency-based Applications Provisioning in Internet of Things**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science

Complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Chair
Dr. Rabin Raut

_____ Examiner
Dr. Yann-Gaël Guéhéneuc

_____Examiner
Dr. Ferhat Khendek

_____Supervisor
Dr. Roch Glitho

Approved by: _____

Chair of Department or Graduate Program Director

_____2017 _____

Dean of Faculty

ABSTRACT

A Platform as a Service for Concurrency-based Applications Provisioning in Internet of Things

Deepak Kumar Aggarwal

The Internet of Things (IoT) is becoming ubiquitous with sensor nodes getting more intelligent and capable of transmitting their processed data to a cloud. Concurrency-based applications play a vital role in the rise of the IoT. A concurrency-based application has multiple processes running independently and interacting with each other. An example of the concurrency-based application in IoT is wildfire management application. In a forest, various sensor devices are deployed in the different areas. The processes of the wildfire management application are running on the sensor devices of different areas independently. The process of a particular area monitors the temperature and interacts with the processes in the neighboring areas. This interaction is based on the fire contour algorithm, which allows the application to provide the real-time direction and evolution of the fire in case of the fire incident.

Cloud computing is a paradigm for swiftly provisioning a shared pool of configurable resources (e.g., services, applications, network, and storage) on demand. Cloud computing can help to tackle the challenges of IoT-based applications provisioning by offering the Platform as a Service (PaaS). Therefore, developers of such concurrency-based applications can use cloud's PaaS for faster development as well as cost efficiency. However, the PaaS faces several challenges at the time of provisioning of concurrency-based application (i.e., development, deployment, and management). For the concurrency-based application development phase, PaaS should support the various API and frameworks for the development of multiple processes of the concurrency-based application, which helps to start development quickly. For the deployment of

the concurrency-based application, the PaaS must offer the facility of hosting of different processes in an isolated environment and enable process's edges to allow inter-process communication. In management phase of the concurrency-based application, PaaS should be able to orchestrate the chain of processes defined during the development phase.

The main intent of this dissertation is to provide a PaaS solution for concurrency-based applications provisioning in IoT to solve the challenges as mentioned above. The major contributions of the thesis are in twofold. First, we propose a PaaS architecture for concurrency-based applications provisioning in IoT. Second, we provide a proof of concept in which a prototype is built using as basis Cloud Foundry, an existing PaaS platform, and TelosB as the IoT Infrastructure devices. The performance measurements have also been made to validate the results claimed.

Acknowledgements

First of all, I express my gratitude to the Almighty, who blessed me with the zeal and enthusiasm to complete this work successfully.

I would like to thank my supervisor, Dr. Roch Glitho of Concordia Institute for Information Systems Engineering (CIISE) for his suggestions and constant support during this research. I am grateful to him for motivating and inspiring me to go deeply into the field of Cloud Platform and Internet of Things Systems and supporting me throughout the lifecycle of MASc. studies. His wide knowledge and logical way of thinking have been of great value for me. I am also grateful for the financial support provided by Dr. Glitho, Concordia University, for funding my studies.

I want to thank all my lab mates for their constant feedback during our various discussions and meetings. They have been very generous for their help during my research. I would also like to thank my friends Suryaveer Chauhan, Nada Khabouchi, and Parul Khanna for supporting me during my stressful and challenging moments.

I want to thank my sister Dr. Rajni Aron for providing me with the unfailing backing and my entire family for continuous moral support throughout my years of study. They have made this endeavor even more worthwhile.

Table of Contents

Acronyms and Abbreviations

API	Application Programming Interface
AWS	Amazon Web Service
CaaS	Communication-as-a-Service
CM	Cloud Manager
CNI	Container Networking Interface
CPI	Cloud Provider Interface
CPU	Central Processing Unit
DBMS	Database Management Service
DEA	Droplet Execution Agent
DNS	Domain Name Server
FCA	Fire Contour Algorithm
GUI	Graphical User Interface
HTTP	Hyper Text Transfer Protocol
IaaS	Infrastructure as a Service
IDE	Integrated Development Environment
IoT	Internet of Things
IP	Internet Protocol
J2EE	Java 2 Enterprise Edition
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
KVM	Kernel-based Virtual Machine
LAN	Local Area Network

LRP	Long Running Process
M2M	Machine-to-Machine
MaaS	Monitoring-as-a-Service
MQTT	Message Queueing Telemetry Transport
MVC	Model View Controller
NESC	Network Embedded System C
NIST	National Institute of Standards and Technology
ORM	Object-relational Mapping
OS	Operating System
PaaS	Platform as a Service
PC	Personal Computer
REST	REpresentational State Transfer
RFID	Radio-Frequency Identification
SaaS	Software as a Service
SLA	Service Level Agreement
TCP	Transfer Control Protocol
UAA	User Account and Authentication
VIM	Virtual Infrastructure Manager
VM	Virtual Machine
VMM	Virtual Machine Manager
VXLAN	Virtual Extensible Local Area Network
WORA	Write Once, Run Anywhere
WSN	Wireless Sensor Network
XML	Extensible Markup Language

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, we first provide an overview of the key concepts related to our research such as provisioning of concurrency-based application in the Internet of Things (IoT) with a focus on Platform as a Service (PaaS). Then the motivation and problem statement are discussed. A summary of thesis contributions is also presented. The chapter concludes with an outline of how this thesis is organized.

1.1 Definitions

In the following subsections, few key concepts are discussed which are relevant to our research.

1.1.1 Cloud Computing

According to National Institute of Standards and Technology (NIST), "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction." These resources can be a network, server, storage, applications, and services [1]. Another definition of the cloud is, "a large pool of easily usable and accessible virtualized resources that can be dynamically reconfigured to adjust to a variable load, allowing for an optimum resources utilization" [2].

Cloud computing came with a creative intention of providing its users with services at different possible levels. Based on the kind of resource usage [18], cloud architecture can be modelled as a set of three layers which consists of Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS).

1.2 Platform as a Service

PaaS is a platform to develop and deploy the application onto the cloud without worrying about underlying infrastructure [1]. The platform helps consumer to deploy applications, created using programming languages, libraries, services, and tools supported by the provider. The consumer does not need to handle any task related to managing or controlling the underlying cloud infrastructure including network, operating systems, servers, or storage, but has control over the deployed applications and configuration settings for the application-hosting environment. Examples include Google App Engine [3] and Cloud Foundry [4].

1.1.2 Internet of Things

The Internet of Things (IoT) is a novel paradigm that is rapidly becoming popular in the current modern wireless telecommunication. The basic idea of the concept is the pervasive presence around us of a variety of things and objects such as Radio-Frequency Identification (RFID) tags, sensors, actuators and mobile phones. These objects through unique addressing schemes can interact with each other and cooperate with their neighbors to reach common goals [5].

1.1.3 Concurrency-based Application

A concurrency-based application is one in which two or more processes that cooperate in performing a task where each process is a sequential program that executes a sequence of instructions. The processes cooperate by synchronization and communication [6].

According to Van Roy [7], concurrency-based application is a program having "several independent activities, each of which executes at its own pace." Also, the activities may perform some interaction among them. A concurrency-based application is conceptually harder to undertake and understand compared to a sequential application, as coexistence and coordination of multiple concurrent activities required to be manage. For example, the home automation system is a concurrency-based application. The system contains three different processes A, B, and C which monitors the smart devices of room, kitchen, and restroom respectively and the owner's location. Based on the movement of the owner, the processes communicate with each other. If the owner leaves the kitchen and approaches towards the room, the process B interacts with process A. The process A activates the smart devices of the room such as air conditioner, smart bulb, and smart door. As the owner reaches to the room, the process A notifies back to the process B, which then starts turning off the appliances in the kitchen.

1.4 Motivation and Problem Statement

Concurrency-based applications play an indispensable asset for large-scale IoT domain, such as home automation system, disaster management. The main reasons behind its importance is the execution of multiple processes independently and an inter-communication between multiple processes to achieve the desired goal. For example, in disaster management like a wildfire, traditionally whenever fire incident happened, there was no earlier stage information to save the neighborhood area, so there is a need for concurrency-based application provisioning in IoT domain. The different processes of concurrency-based application, can monitor different areas independently and interact with each other to provide the real-time situation, which helps to save the neighboring area.

For cost efficiency and faster development of concurrency-based applications, cloud PaaS services can be used. However, the PaaS provider faces several challenges concerning concurrency-based application provisioning including (a) development, (b) deployment, and (c) management. A mechanism for concurrency-based application provisioning in IoT is required to address these challenges. The first challenge for the development of the concurrency-based application, PaaS should support the various APIs' and frameworks for the development of multiple processes of the concurrency-based application. The second challenge is during the deployment phase of the concurrency-based application that PaaS should enable the edges of different processes in such a way that they allow communicate directly. It should also provide an isolated environment to each process for its independent execution. The last challenge is the management phase; the PaaS should provide orchestration services for the chain of execution specified during the development phase. Besides, it should also provide service discovery, so that processes can search each other to make a direct route between different processes of concurrency-based application in IoT. The developer of the concurrency-based application can be benefited from a PaaS that deal with

these challenges. To realize such PaaS, we need a sound architecture that enables easy concurrency-based application provisioning in IoT.

1.5 Thesis Contribution

The contributions of this thesis are stated below:

- A set of requirements on the PaaS for concurrency-based application provisioning in IoT is identified.
- A review of the state-of-the-art solutions for PaaS with respect to the defined requirements is presented.
- High-level architecture of Platform as a Service has been proposed for concurrency-based application provisioning in IoT.
- The Implementation architecture for PaaS is also presented for concurrency-based application provisioning.
- The prototype and measurements are also done on the extended PaaS architecture for concurrency-based provisioning.

1.6 Thesis Organization

This thesis is organized as following.

Chapter 2 discusses the key concepts related to the research domain in more detail.

Chapter 3 explains the motivating scenario and set of requirements for PaaS derived it. The state-of-art is also evaluated against the requirements.

Chapter 4 proposes PaaS architecture (i.e., Extended Cloud Foundry) and discusses its design and communication interfaces. It also describes the application lifecycle for the concurrency-based application provisioning.

Chapter 5 describes the implementation architecture and technologies used for the proof-of-concept prototype.

Chapter 6 concludes the thesis with summary of the overall contributions and possible future research directions.

Chapter 2

Background

This chapter presents the background concepts relevant to research domain of this thesis. The following concepts are explained: Internet of Things (IoT), Cloud Computing, Platform as a Service (PaaS), Concurrency-based Application.

2.1 Internet of Things (IoT)

This section describes the basic overview of the Internet of Things by defining a brief introduction of IoT, its benefits, and applications.

2.1.1 Definition of IoT

The term “Internet of Things” was first coined by Kevin Ashton in 1999 to describe an automated process of transferring the data to and fro from things using technology like RFID [8]. The RFID group defines the Internet of Things as “the worldwide network of interconnected objects uniquely addressable based on standard communication protocols” [9]. The basic idea of this concept is the pervasive presence around us of a variety of things or

objects such as sensors, RFID tags, actuators, mobile phones. These objects through unique addressing schemes can interact with each other and cooperate with their neighbors to reach common goals [5].

According to Gubbi, J., and Buyya, R. [9], Internet of Things can define as "interconnection of sensing and actuating devices providing the ability to share information across platforms through a unified framework and developing a common operating picture for enabling innovative applications. This interconnection is achieved by seamless, ubiquitous sensing, data analytics and information representation with cloud computing as the unifying framework."

2.1.2 Benefits of IoT

1. ***Decreased Costs:*** By using IoT devices, the cost for running the application is decreased. For example, various wearable equipment is available for diagnostic monitoring of the patients, which collect real-time data [10]. This real-time data helps doctors to analyze the patient's health report remotely and provide needful treatment subsequently. This prompt treatment reduces the visits to doctors unnecessarily. To be specific, these home care facilities are guaranteed to reduce the hospital visits.

2. ***Improved Asset Utilization:*** The connectivity of assets (such as machinery, equipment or tools) through IoT has improved the tracking. The business can benefit with the real-time information and visibility of supply chains and assets. For instance, IoT devices can help the user to locate the assets efficiently and improve

the preventive maintenance of business resources and machinery which increase efficiency [11].

3. ***Improved Productivity:*** The critical parameter of any industry is the productivity. Through IoT, industries can improve the productivity by giving employees just-in-time training, overcoming the mismatch of available vs. required skills and improving labor efficiency [12].

2.1.4 Applications of Internet of Things

The emergence of IoT has influenced various application domains. These IoT-based applications can be classified into different types based on coverage, user involvement, repeatability, network availability, coverage, heterogeneity, and impact [13]. These applications have been categorized into mainly three application domains: Personal and Home, Utilities, and Enterprise. A few typical applications in each domain are discussed below:

1. ***Personal and Home Application Domain:*** In Personal and Home, the sensing information collected from smart devices is used only by the individual who directly own the network. Usually, Wi-Fi is used for enabling higher bandwidth data (video) transfer as well as higher sampling rates (sound) [9]. The control of various home equipment such as washing machine, electric bulb, refrigerator, air conditioner allows the better energy management. Social networking is another example of transformation with billions of interconnected objects [14].

2. ***Utilities-based Applications:*** Utilities-based applications in the Internet of Things domain provide service optimization to the consumer by the service provider. The main role of the service provider is to keep track of the consumption by using a smart meter which is provided by hydro supply companies, electricity supply companies etc. These utility services usually are managed by the large organizations on the national level for tracking the efficient resource management and critical utilities.

   Smart grid and Smart metering are other potential IoT applications which are being implemented around the world [15]. Energy efficiency is one of the essential parameters need to be considered for utility-based applications. Efficient power consumption can be achieved by continuously monitoring every electricity point within a house and using this information to modify the way electricity is consumed. This information at the city scale is used for maintaining the load balance within the grid ensuring uninterrupted service.

3. ***Enterprise-based Applications:*** In factory setup, sensors have always been an integral part of automation, tracking resources, climate control, security and so on. Various test-beds have been installed for its implementation. This smart environment includes several subsystems like smart retail, smart city, smart agriculture, smart transportation and more. The main point should be noted that these subsystems cover more focused groups and share the data. For example, the smart city applications can share data with the public.

2.2 Cloud Computing

In this section, the basic overview of cloud computing is presented. We start with the brief definition of cloud computing, followed by the different service models offered by the cloud computing. Finally, we also discuss its key benefits.

2.2.1 Definition of Cloud Computing

According to NIST definition of cloud computing "A model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [1]. It provides us easy access to the virtualized resources that can be scaled up and down based on the resource utilization. Cloud computing makes optimum resource utilization feasible.

2.2.2 Characteristics of Cloud Computing

NIST definition covers all essential characteristics of cloud computing [1]. As listed below there are certain basic features of cloud:

i. **On-demand self-service:** Cloud computing allows its consumers with on-demand provisioning capabilities.

ii. **Broad network access:** Cloud capabilities can be accessed from anywhere over the network using heterogeneous client platforms (e.g., PDAs, mobile phones, laptops, and desktop systems).

iii. **Resource pooling:** These resources being either actual physical resources or the virtual resources are dynamically pooled to fulfill the consumers' needs in a location independent way.

iv. **Rapid elasticity:** Resources are rapidly provisioned to meet the consumers' needs. Cloud computing users need not plan ahead for provisioning.

v. **Pay as per use**: Cloud services are metered services. Users pay only for the resources used as per the duration of use.

2.2.3 Benefits of Cloud Computing

Cloud computing offers several substantial advantages. They are:

1. **Efficiency:** Cloud provider allows the user to access cloud-based applications and data virtually with any internet connected device.

2. **Scalability:** Virtually unlimited scalability is possible because of the massive capacity offered by the cloud providers [16].

3. **Reliability:** Services running on the cloud should meet several desired requirements such as reliability, availability, performance, fault tolerance and more.

4. **Easy access:** Customers can easily access provisioned resources over the network

through various types of devices.

5. **Flexibility:** The consumer can access the cloud services from anywhere in the world where connection is available [17].

2.2.4 Service Models of Cloud Computing

Cloud computing came with an original intention of providing its users with services at different possible levels. Based on the kind of resource usage, cloud architecture can be modelled as a set of three layers consisting of Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS) as shown in Figure 2.1. However, other services like Communication-as-a-Service (CaaS) and Monitoring-as-a-Service (MaaS) also come under cloud service models. Cloud computing has three main service models [1] – Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). Figure 2-1 illustrates the service models:

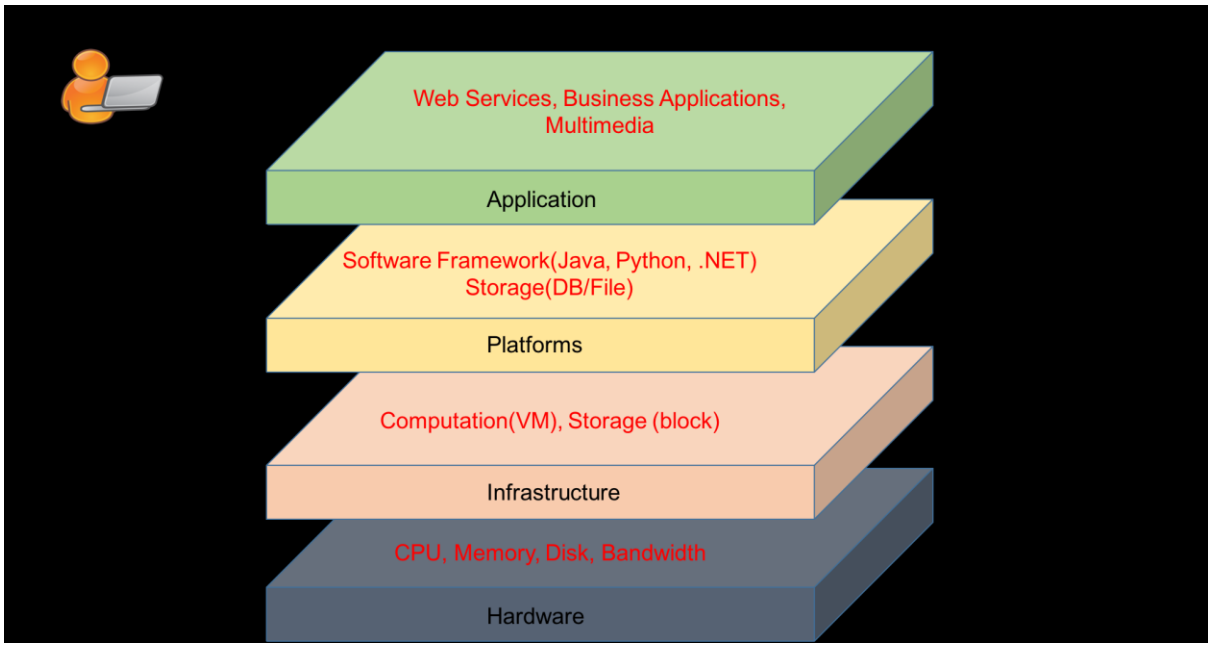

Figure 2.1: Service Models of Cloud Computing [18]

2.2.4.1 Infrastructure as a Service (IaaS)

IaaS provides the fundamental building blocks of computing resources. It allows cloud users to build virtual infrastructure but which can be created, reconfigured, resized, and removed in less time as when a task requires it [1]. The best-known examples for IaaS are

Amazon EC2, Amazon Simple DB, and Amazon S3, Rackspace cloud and Eucalyptus.

2.2.4.2 Platform as a Service (PaaS)

PaaS provides a cloud service that abstracts away the infrastructure but rather provides a software development platform. It is possible to code and run an application on a PaaS, and the system makes sure that the application has the necessary infrastructure to execute it.

Platform as a Service is discussed in detail in the next section 2.3.

2.2.4.3 Software as a Service (SaaS)

SaaS provides end users with the use of cloud applications along with proper abstraction at the level of underlying infrastructure [19]. Examples of SaaS services include Hadoop, Picasa, Salesforce.com, Gmail, Google docs and more.

2.3 Platform as a Service (PaaS)

In this section, we first present the available definition of Platform as a Service. Then we describe its various advantages, benefits and existing PaaS examples.

2.3.1 Definition of PaaS

Platform as a service operates at the layer above raw computing hardware, whether physical or virtual. NIST defines Platform as a Service as a "The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages and tools supported by the provider" [1]. PaaS provides the application provisioning lifecycle, i.e., application development, deployment, and management [20]. In application development, PaaS provides various APIs' and frameworks to develop applications and services. During the deployment of a cloud application, PaaS provides the hosting environment to deploy in the particular service container and various interface for the interaction with the other components of PaaS. Based on the Service level agreements (SLA) service, PaaS provider provides different management services for the orchestration of application execution such as scalability, database migration and more.

Some examples of PaaS solutions are the Google App Engine [3] system, Cloud Foundry [4] which works on top of the Amazon Web Services IaaS system, and "Force.com" built as part of the SalesForce.com software as a service offering.

2.3.2 Major characteristics of PaaS

- **Rapid deployment mechanisms:** It is typical of many PaaS offerings to provide developers and operators with a "push and run" mechanism for deploying and running applications. It helps to provide dynamic allocation of resources when the application code is passed to the PaaS via an API [21].

- **Support for custom applications:** Support for the development, deployment, and operation of custom applications. PaaS offers typically support “born in the cloud” or “cloud native” applications that can take full advantage of the scalable, elastic and distributed capabilities of cloud infrastructure [22].

- **Provision of runtime environment:** Typically, each runtime environment supports small set of programming languages and frameworks – for example, Node.js, Ruby, and PHP runtimes.

- **Provision of services:** The service is an essential concept in reducing the effort and complexity in building software systems. PaaS provides the facility for the provisioning of different services. Various services offer the different capability, typically invoked via an API. The services are installed and run by the service provider, removing responsibility of the cloud user. For example, in a database server, the responsibility for ensuring availability and reliability, for having replicas and backups of the database data, for securing the data and so on, all falls on the service provider [22].

- **Security capabilities:** Security is one of the most critical aspects of any business solution. It is not surprising to find that PaaS offerings provide built-in security capabilities, reducing the load on developers and operators to provide and manage these capabilities such as firewall, endpoint management, secure protocol handling, access and authorization, encryption of data in motion and at rest, integrity checking.

2.3.3 Benefits of PaaS

- PaaS is an excellent choice for developers and programmers as it comes ready to use, provides flexibility for mounting environmental architecture as user alike. It also allows the cloning of environments for testing, approval, and production on the same server and software configurations, monitoring real-time memory consumption, CPU, and bandwidth, and also saves time.

- It is easy to work on a pre-configured platform thus saving time, effort and is much more efficient than starting one from scratch. It provides very well for most web projects while adding convenience in managing this environment.

- When multiple developers are working on the development or when external parties are involved in the development process, PaaS is a great option to bring in the speed and flexibility to the development process.

- For organizations following agile methodology for software development, PaaS eases the difficulties associated with rapid development and iteration of application.

- When the user wishes to spread capital investment – by providing the underlying solid infrastructure, PaaS model reduces the organization's overhead costs.

- Large organizations who want to customize applications

2.3.4 Example of Existing PaaS

This section briefly introduces the existing PaaS – Cloud Foundry. It is an open PaaS, which provides the choice of application services, development framework, and deployment clouds. Cloud foundry's container-based architecture can run cloud-based applications in a various programming languages (e.g., Java, Ruby, Go, Python, .NET, PHP, JavaScript). It also supports several frameworks such as Rails, Java Spring, Rack, Node.js [4] and provides services during application execution (e.g., PostgreSQL, MySQL, Redis, RabbitMQ, MongoDB). To support a new programming language, the mechanism called buildpacks are provided by the cloud foundry. Buildpacks examine the user-provided artifacts to determine the application's dependencies and download it for the application execution. Cloud Foundry allows for the choice of independent underlying infrastructure (e.g., OpenStack, AWS, Azure, vSphere). It provides an interface called Cloud Provider Interface (CPI), which has a set of APIs' for the deployment of cloud foundry on the targeted infrastructure using a BOSH deployment tool [23].

2.4 Concurrency-based Application

This section provides an overview of the concurrency-based application by defining a brief definition of concurrency-based application, its examples, provisioning, and benefits.

2.4.1 Definitions of Concurrency-based Application

Concurrency is a fundamentally complex and expensive concept where several independent activities are being executed at their own pace. A concurrency-based application is defined as two or more processes that cooperate in performing a task where each process is a sequential

program that executes a sequence of instructions. The processes of concurrency-based applications cooperate with each other through synchronization and communication [6].

According to Van Roy [7], the concurrency-based application is a program having several independent activities, each of which executes at its own pace. In addition, the activities may perform some kind of interaction among themselves. A concurrency-based application is conceptually harder to undertake and to understand than sequential application, as in it is required to manage the coexistence and coordination of multiple concurrent activities.

2.4.2 Examples of Concurrency-based Applications

Concurrency-based applications play a major role in IoT domain to manage the applications efficiently. Three notable examples of concurrency-based applications are illustrated as follows:

1. ***Multiplayer Pervasive Game:*** A pervasive multiplayer game is IoT-based technology game in which multiple players can play in the same game environment at the same time. It is an example of the concurrency-based application in the Internet of Things. In this game, the vast area is equipped with tokens. The players can play the game by logging into their smartphone. The main goal of the game is that the player has to scan the opponent's tokens. Suppose at an instance two players log-in to play the game. The game region is divided into two areas with equal

distribution of tokens. Now, the game has two processes, one for each player. Each process monitors its player location and tokens. As the game executes, the processes interact with each other to share the information of the players (e.g., location). The player has to chase opponent player's tokens with the smart device and stop the opponent to reach to his tokens. Whenever the players get a strike with each other, there is a virtual fight between them. Whoever collects it's opponent's tokens first is the winner.

2. ***Wildfire Management System:*** Wildfire management application is an example of a concurrency-based application in IoT. In the forest, different areas are equipped with the sensor devices. The processes of wildfire management application are running on the sensor devices of different areas independently. The main task of the processes is to monitor the temperature in a specified area and interact with the processes running in the neighboring areas. The interaction is done on the basis of fire contour algorithm that provides the real-time direction, speed, and intensity of the fire in case of the fire.

3. ***Home Automation System:*** An example of concurrency-based application is a home automation system. It is made up of interactive processes. The different processes of concurrency-based application like A, B, and C are monitoring the smart devices of the room, kitchen, and restroom respectively and the location of the person. The processes communicate with each other based on the movement of the individual. If the individual leaves the kitchen and approaches towards the room, the process B interacts with process A. The process A activates smart devices like air conditioner,

lights, music system and similar. After completing the task, processes go into the idle state and smart devices automatically turn on as the person reaches the room.

2.4.3 Concurrency-based Application Provisioning in IoT

The provisioning of a concurrency-based application comprises development, deployment and management of multiple processes to run simultaneously and may include interaction between these processes on PaaS. PaaS providers can provision and offer concurrency services which concurrency application developers can use in their applications. Concurrency-based application provisioning entails the whole lifecycle of the application, which consists of application development, deployment, and management. Using IoT-based PaaS, cloud PaaS providers provision concurrency-based applications and offer them as SaaS.

2.4.4 Benefits of Concurrency-based Application

1. **Reduce Complexity:** The main motivation for integrating parallelism for designing a prototype is to provide means for explicitly modeling concurrency-based applications [6]. For example, concurrent systems such as pervasive multiplayer system and home automation system, which must respond to many external stimuli, which are therefore inherently parallel and often distributed. To deal with non-determinism and to reduce their complexity, such applications are preferably structured as independent concurrent processes.

2. **Reduce Execution Time:** The main benefit of concurrency-based application is to reduce execution time [6]. To achieve speedup through parallelism, play an important motivation role for executing the concurrency-based application on distributed system such as cloud and IoT.

2.5 Chapter Summary

This chapter has discussed the background concepts which are related to this thesis. First, we introduced the concept of the Internet of Things, by giving its definition, benefits, and applications. Then we followed by discussing the concept of cloud computing, its different definitions, characteristics, benefits and three main service models. Finally, we discussed Platform as a Service (PaaS) and concurrency-based application.

Chapter 3

Scenario, Requirements, and State-of-the-Art Evaluation

This chapter is divided into three sections. The first section presents a motivating scenario. From motivational scenario, a set of requirements have been derived in the second section.

The last section is focused on reviewing the state-of-the-art solutions and evaluated them based on the set of requirements followed by the summary.

3.1 Motivating Scenario

This section presents a motivating scenario in the concurrency-based application in IoT. The motivating scenario helps to derive the requirements for the platform as a service for the concurrency-based application provisioning in IoT.

3.1.4 Wildfire Management Application: Use Case Motivating Scenario

The wildfire management application is a concurrency-based application that has multiple processes (such as fire monitoring process) running on the sensor nodes of different areas independently. In this wildfire management scenario, different sensor nodes (commonly known as "motes") have been deployed in a different area of forest. The primary function of the nodes is to capture and report real-time data about the wildfire. These nodes can monitor environmental parameters such as temperature, pressure, and humidity and can communicate over the wireless radio. The primary intent of this use-case is to help the city administrator in decisions making to extinguish the wildfire at an early stage as shown in Figure 3.1. The wildfire management application provides the direction, speed, and intensity of the fire, using the Fire Contour algorithm[24]. During the regular operation, each node is able to monitor the local temperature periodically and detects the fire when the sensed value goes beyond the predefined threshold value (T_{detect}). The process running on the sensor node joins the fire contour services using the circle based model [24] approach for fire front detection. We assume that each process is aware of their node location, and the node coverage area after the

fire incident. The process starts broadcasting their node position to the processes running on the entire neighborhood. Additionally, node burn at a specific temperature (T_{burn}), such that ($T_{burn} > T_{detect}$). The process can transmit their node's position before burning (T_{burn}). The processes running on the nodes maintain the hierarchy and network topology to maintain local approximation of the whole forest fire.

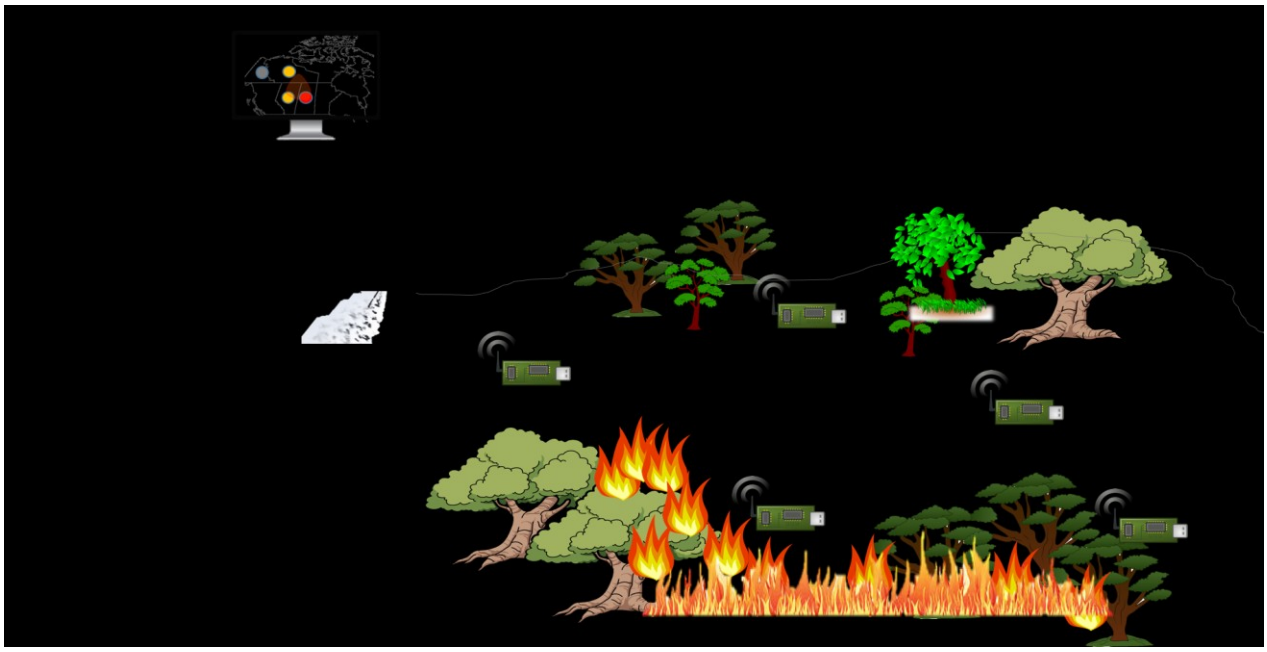

Figure 3.1: Wildfire management system architecture

3.2 Requirements

This section presents the basic requirements on the PaaS for the concurrency-based application lifecycle provisioning in IoT.

3.2.1 Requirements on the PaaS

Platform as a service (PaaS) is a cloud computing model that provides applications over the Internet. In a PaaS model, a cloud provider delivers software and hardware tools to a user as a service for application development. A PaaS cloud provider hosts the hardware and software on its infrastructure. The requirements on PaaS for the concurrency-based application provisioning in are mentioned below:

- **Developing the application:** For the development phase, PaaS provider should enable the development of the application that could be hosted and executed in PaaS. It should provide various APIs', framework and tools in support to build a concurrency-based application in IoT. These are the fundamental requirements required when a developer is building and testing their applications on the platform. For example, in the aforementioned use-case scenario, the interaction between multiple processes of wildfire management application will be tested on the same platform as of production.

- **Deploying the application:** For the deployment phase, PaaS architecture should enable edges of different processes of the concurrency-based application. The enabling edges of different processes allow making a direct connection between different service containers. In the introduced use-case, for instance, it is cleared that different fire monitoring processes of wildfire management application are running independently. These processes should be deployed in the different container to provide an isolated environment. In case of the fire, different processes need to interact with each other directly to share their region status.

- **Managing the application:** For the management phase, we focus on the execution step of the concurrency-based application. During execution, cloud provider needs to manage its service levels and govern usage of the platform itself. The PaaS should be able to orchestrate the chain of execution specified during the development phase. It should also be able to provide the built-in service discovery that may require discovering the different processes of concurrency-based application to interact with each other. In addition, it should be able to ensure that the Service Level Agreements are met.

3.3 State of the Art (Review and Evaluation)

Cloud Computing and Internet of Things are emerging computing paradigms featuring distinctly different computing resources and system architecture. As both paradigms are growingly being adopted in more and more application areas, researchers and practitioners have started to investigate the convergence of cloud and IoT to exploit their intrinsic complementary.

As per our knowledge, no research has proposed a Platform as a Service architecture for concurrency-based application provisioning in the Internet of Things. They are less likely to use the existing architecture for provisioning of critical concurrency-based applications. Related work has been categorized into two sub-sections. The first subsection reviews the existing PaaS solutions related to concurrency-based applications provisioning. The second subsection discusses research work which has proposed the IoT-Cloud based architecture.

3.3.1 PaaS Solutions

PaaS offers development, deployment, and management of application by decreasing the cost and complexity of purchasing and maintaining software and hardware and provisioning hosting capabilities. There are various types of PaaS vendors which offer application hosting and a deployment environment along with various integrated services. We discuss few PaaS providers in detail as follows:

1. Cloud Foundry

Cloud Foundry [25] is a layer of cloud services that provides a platform allowing one to deploy, execute and manage applications without the need for building and maintaining the infrastructure typically associated during execution of an app. Cloud Foundry is a portable, open source PaaS and it has a notable influence in the cloud computing aspect as Linux, OpenStack, and the Open Compute Project because of its governance structure. Figure 3.2 shows the block diagram of the cloud foundry architecture. The entry gate of cloud foundry is the router, which routes all request to the cloud controller after authenticating users with their cloud foundry credentials as shown in Figure 3.2. Cloud controller, which maintains the application life-cycle and redirect to the correspondent Droplet Execution Agent (DEA) that runs the appropriate service. DEAs provide service containers and other necessary buildpacks that are necessary to run the deployed application. However, cloud foundry cannot support the deployment and management phase to provision the concurrency-based application. It does not provide any IDE and/or APIs for the development of a concurrency-based application.

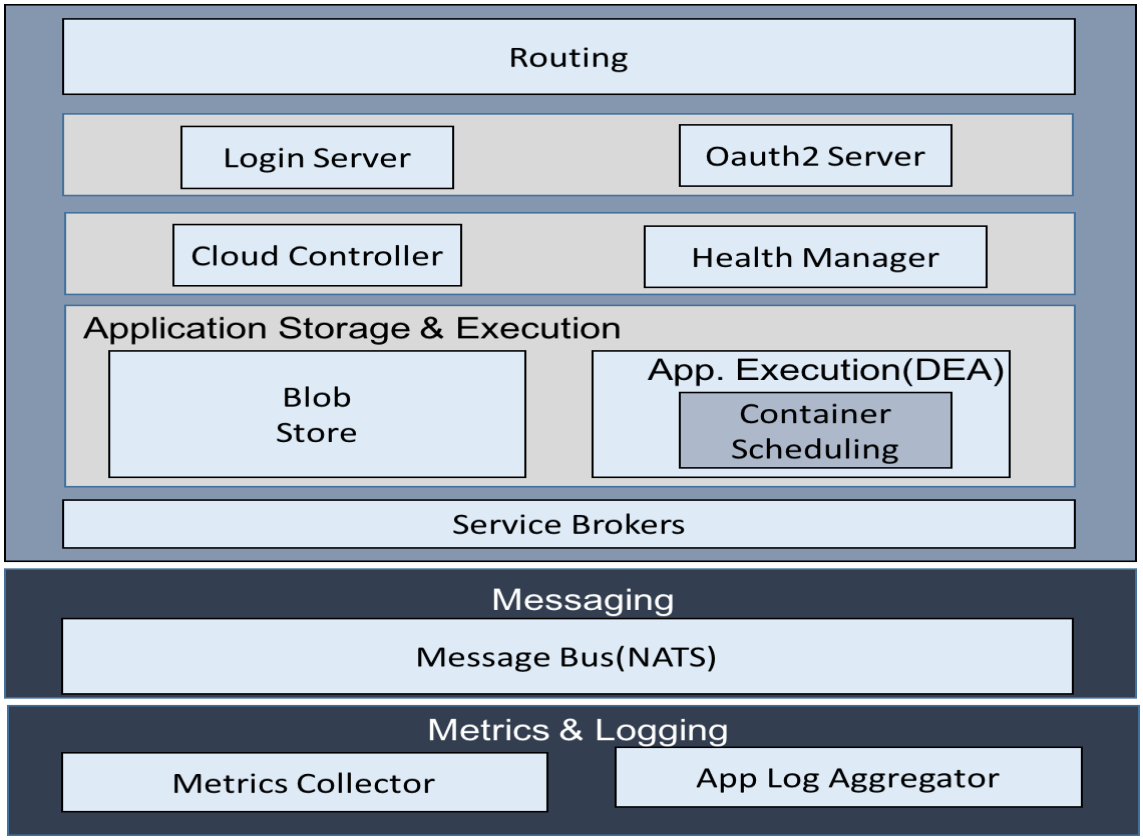

Figure 3.2: Block diagram of Cloud Foundry architecture [25]

2. Cloudify

Cloudify [26] is an open source cloud orchestration framework. Cloudify enables the model applications and services and automates their entire life-cycle, including deployment on any cloud or data center environment. Figure 3.3 shows the Cloudify stack that helps to monitor all phases of a deployed application, identifying issues and abortion, manually or automatically re-mediating such issues and performing continuous maintenance tasks. Cloudify provides few features such as application modeling, pluggability, and security. With the help of application modeling, a user can describe application with all its resource requirements in a precise manner. Pluggability is one of the cores, unique features of Cloudify. It provides reusable components abstraction for the system. Secured communication is obtained using SSL, which allows clients to verify the authenticity of the

Cloudify manager, and to secure that the data sent to and from it is encrypted. However, regarding concurrency-based application provisioning, the Cloudify does not fulfill the essential requirement such as development phase. However, it supports the partial deployment and management of the concurrency-based application.

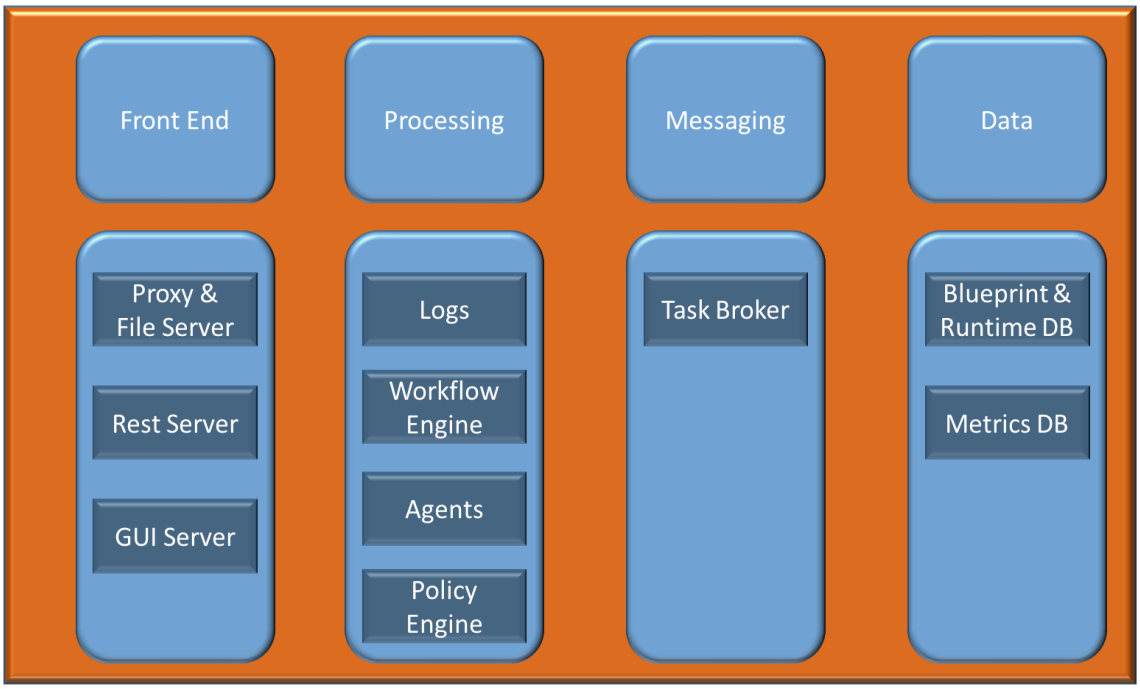

Figure 3.3: Cloudify Stack [26]

3. OpenShift

OpenShift [27] enables to create, deploy and manage applications in the cloud. Two basic functional units of the OpenShift are the Broker and Node servers as shown in Figure 3.4. The communication between the Broker and Nodes is done through a message queuing service. The broker is the single point of contact for all application management activities. It is responsible for controlling user logins, DNS, application state, and general orchestration of the applications. Nodes are the systems that host user applications. The OpenShift supports the application lifecycle and mainly consider for application deployment phase. However, there is missing of some components related to development and management phase to support concurrency-based application provisioning.

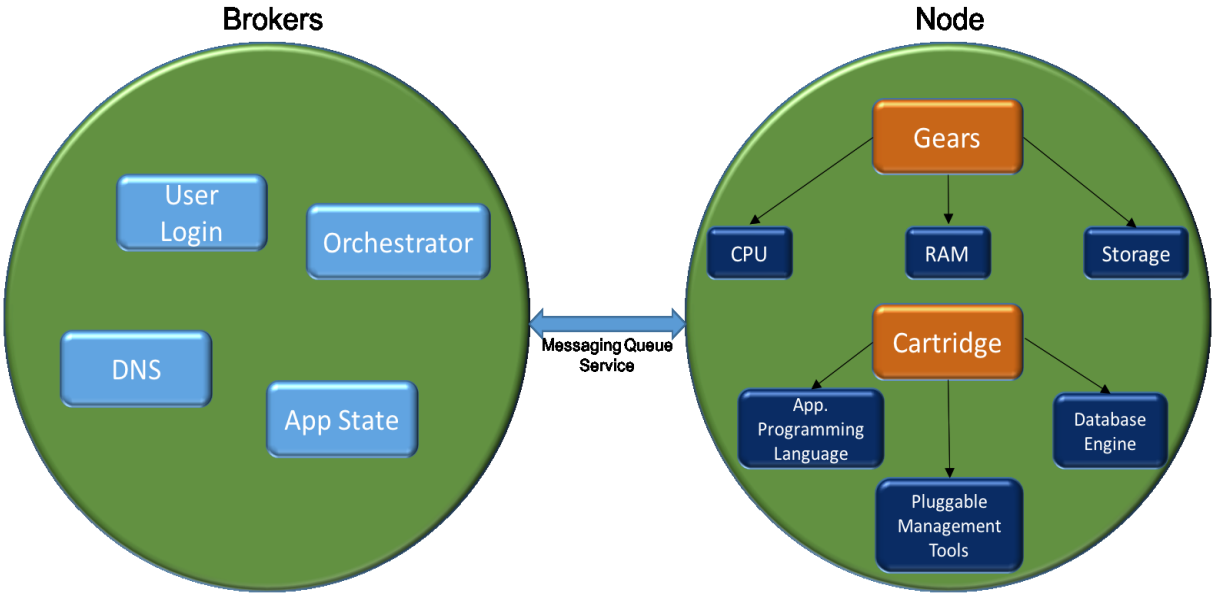

Figure 3.4: OpenShift Components [27]

4. AppScale

AppScale [28] is a scalable, distributed, and fault-tolerant cloud runtime system that executes over cluster resources. In particular, AppScale is a cloud platform, i.e., a platform as a service (PaaS) cloud framework, which executes over cluster resources. It can be deployed on Xen, Kernel-based Virtual Machine (KVM), Amazon EC2 or Eucalyptus. The AppScale structural design uses the standard three-tier web deployment paradigm in the design as shown in Figure 3.5. In the following design cycles, more components are added to the AppScale. The cluster resources carrying AppScale can be operated with or without virtualization (e.g., KVM, Xen) or via popular cloud infrastructures including Eucalyptus and Amazon EC2. The AppScale platform abstracts virtualize, and multiplexes cloud and system services across multiple applications, enabling write-once, run-anywhere program development for the cloud. The AppScale platform exports services and APIs in addition to those given by Google App Engine. These technologies are essential for application domains beyond those of web

services, e.g., data analytics and data and computationally intensive applications. However, it does not support the application life-cycle phases: development, deployment, and management phase for concurrency-based application. Some features exist in AppScale for deployment of compute-intensive application but not for concurrency based applications.

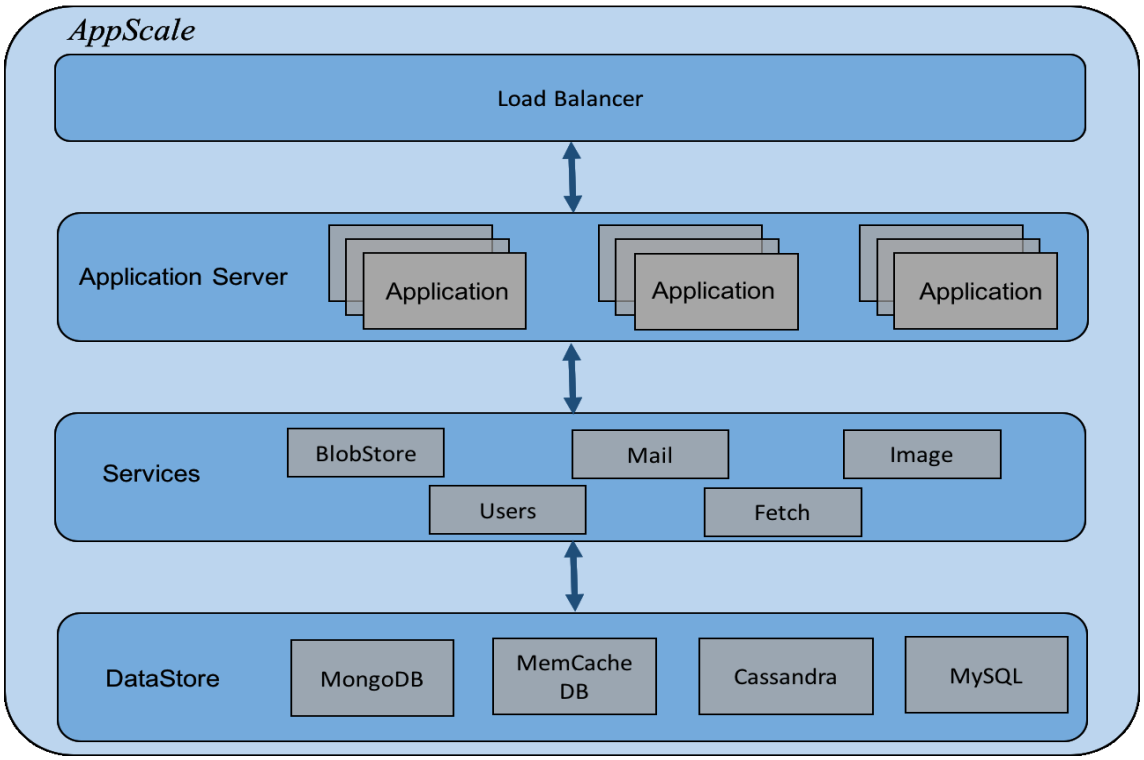

Figure 3.5: Design of the AppScale cloud platform [28]

5. Aneka

Aneka [29] is a platform and a framework for developing distributed applications on the cloud. It provides the spare CPU cycles of a heterogeneous network of desktop PCs and servers on demand. Aneka provides a rich set of APIs for transparently utilizing such resources and expressing the business logic of applications by applying the preferred programming abstractions to the developers. The system administrators can leverage on a collection of tools to monitor and control the deployed infrastructure. The Aneka based computing cloud is a set of physical and virtualized resources connected through a network,

which is either the Internet or a private intranet. One of the critical features of Aneka is the ability to provide different ways for expressing distributed applications by offering different programming models. Aneka can elastically scale applications and provides few API for deployment of concurrency based applications. However, it does not provide the APIs' specific to the development and management of concurrency-based applications in IoT.

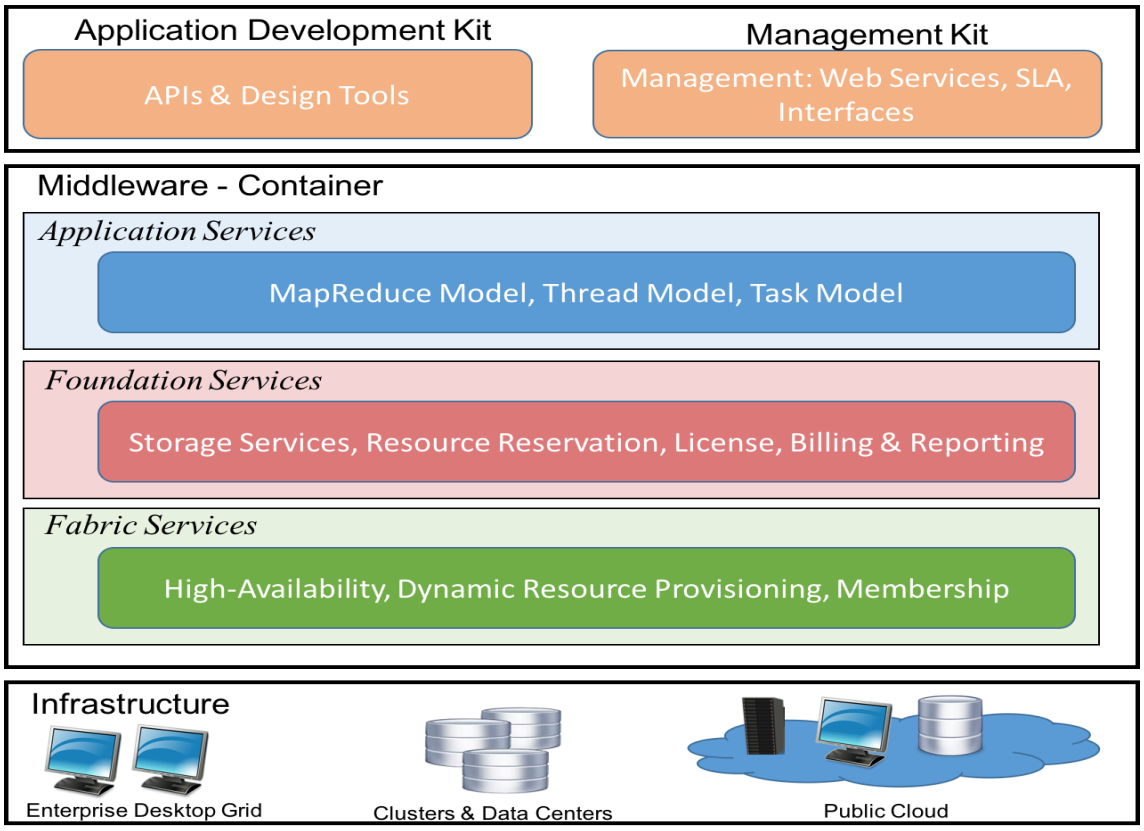

Figure 3.6: Basic Architecture of Aneka [29]

3.3.2 IoT-Cloud based Architecture

IoT has gained much momentum lately. Provisioning of concurrency-based application is a tedious task. If the IoT devices are integrated with PaaS for provisioning of concurrency-

based application, the possibilities are endless. In this section, the existing work related to the provisioning of concurrency-based applications in IoT has been discussed.

Fei li et al. [30] introduced the IoT PaaS architecture, upon which IoT solutions can be delivered as virtual verticals by leveraging middleware services and computing resources on the cloud. On IoT PaaS, two types of services such as event processing and persisted data services have been considered. Event processing is to process and analyze real-time events generated by sensory devices. The service can produce data flows and detect new patterns or events according to data users' specifications. The primary target of this work controls the metering services of a commercialized application as shown in Figure 3.7. They have mainly focused on the development, deployment, and management of the domain-specific control applications such as industrial applications. However, the implementation of the application development, deployment and management are not addressed.

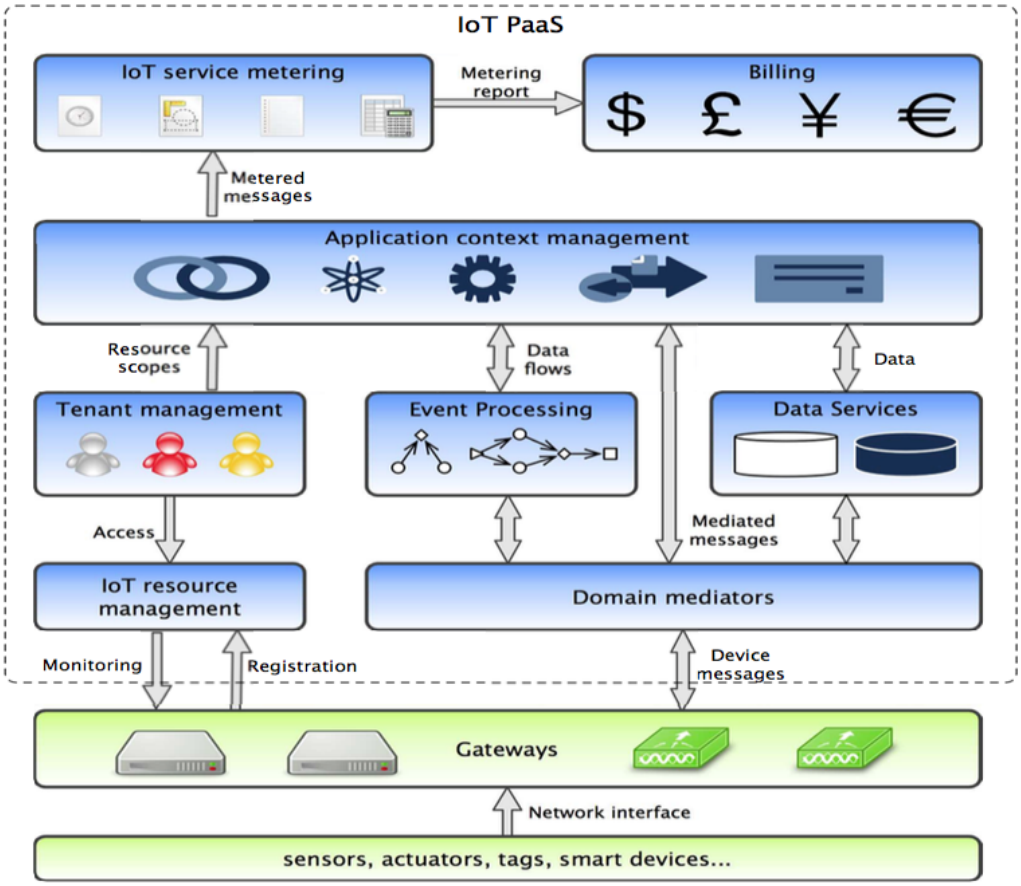

Figure 3.7: The IoT PaaS Platform [30]

Luis M. Vaquero et al. [31] focused load balancing feature for provisioning of applications. They have addressed issues both at container and database level to achieve scalability. At container level, a better scalability can be achieved by enabling multi-tenant containers (having the ability to run components belonging to different users) as shown in Figure 3.8. This view imposes strong isolation requirements which all platforms may not be able to achieve by default. They have tried to maintain the scalability by adding DBMS instances. By using this architecture, the functional requirements for the provisioning of a concurrency-based application cannot be attained. In this architecture, management phase is discussed in detail. They have not emphasized on deployment and development phase.

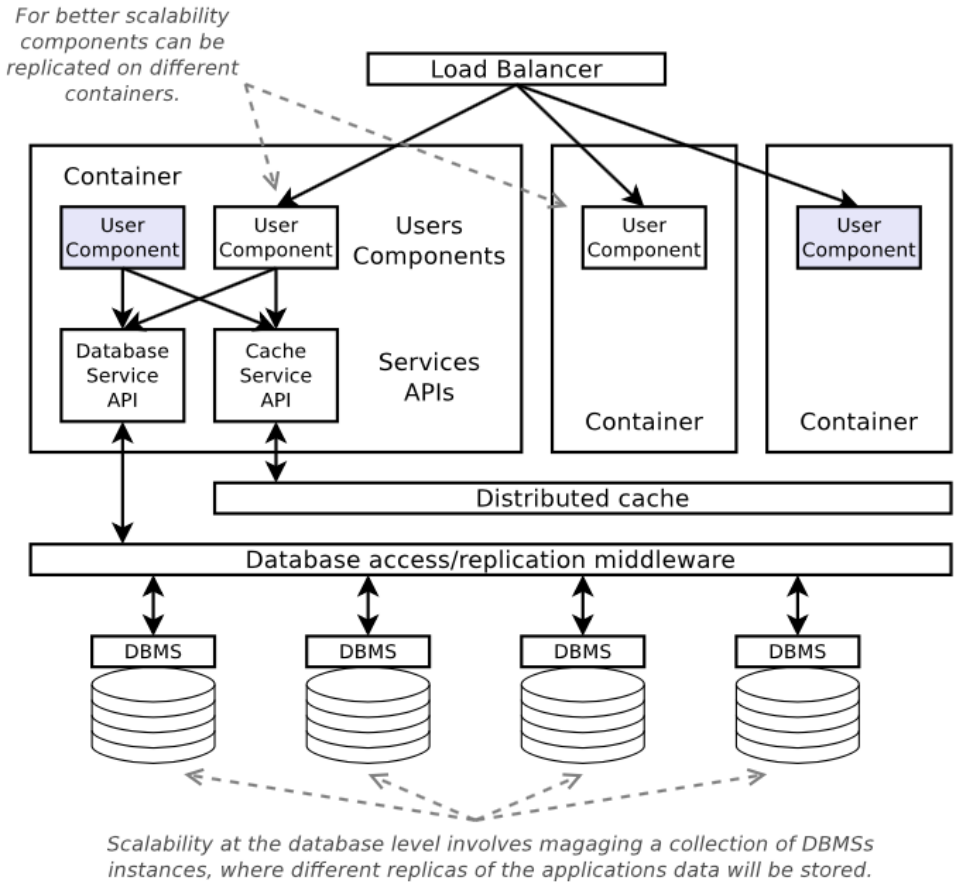

Figure 3.8: Schematic View of a PaaS System [31]

Roberto Girau et al. [32] have proposed Lysis, a cloud-based platform for the deployment of Internet of Things (IoT) applications. The Lysis platform is designed by considering four functional levels: real-world level, virtualization level, aggregation level and application level as shown in Figure 3.9. On the base level, all the physical devices can be abstracted in the cloud through the hardware abstraction layer. The virtualization level is made up of virtual social objects and help to communicate real-world objects. The main concern of aggregation is responsible for composing different social virtual objects to set up entities with augmented functionalities called micro-engines, and the last level is the application level in which user-oriented macro services are deployed. At the base level they have not covered main points such as applications require device-to-device communications as some decisions have to be

taken cooperatively, and again this prevents from completely exploiting the virtual cloud counterparts. For some devices, the vendors do not disclose all the details to have complete access to some capabilities. For this reason, the abstraction allows access to a controlled view of the device sensors. Moreover, they have considered development and deployment requirements at virtualization level and application level. However, management phase level requirements are not discussed.

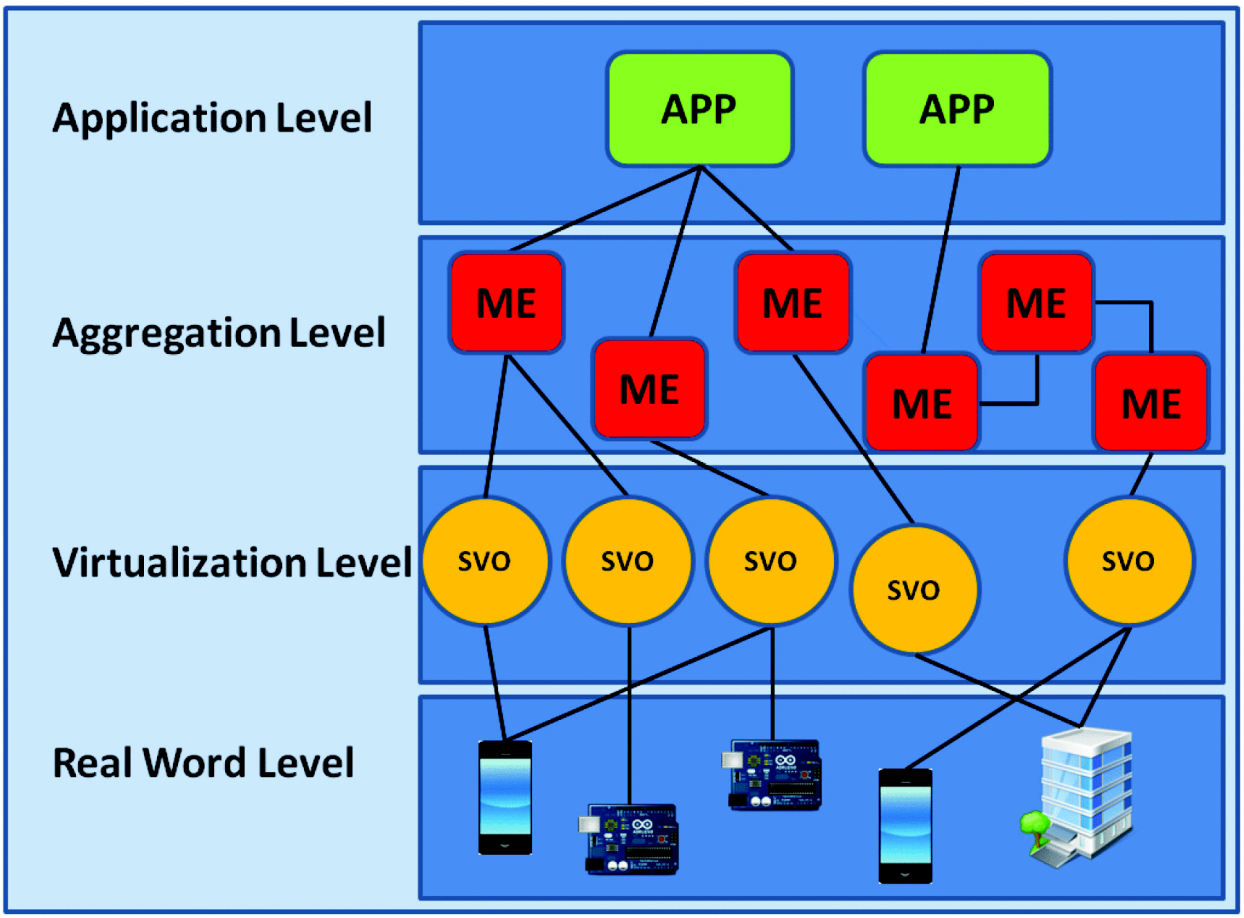

Figure 3.9: Four Layer of the Lysis Architecture [32]

M. Fazio et al. [33] has discussed two strategies for managing sensing resources. The first strategy follows the data-centric model, where the cloud offers environmental data to its clients as a service, without any knowledge of how data are measured and processed. The second strategy adopts the device-centric model, which enables the cloud clients to use a

virtual sensing infrastructure that they can customize for specific purposes. As shown in Figure 3.10, a client can access sensed data and customize one. Authors have adapted to interconnect heterogeneous monitoring environments and the cloud. They, however, have no solutions to handle them. Hence, this work does not meet the requirements of concurrency application development and deployment phase.

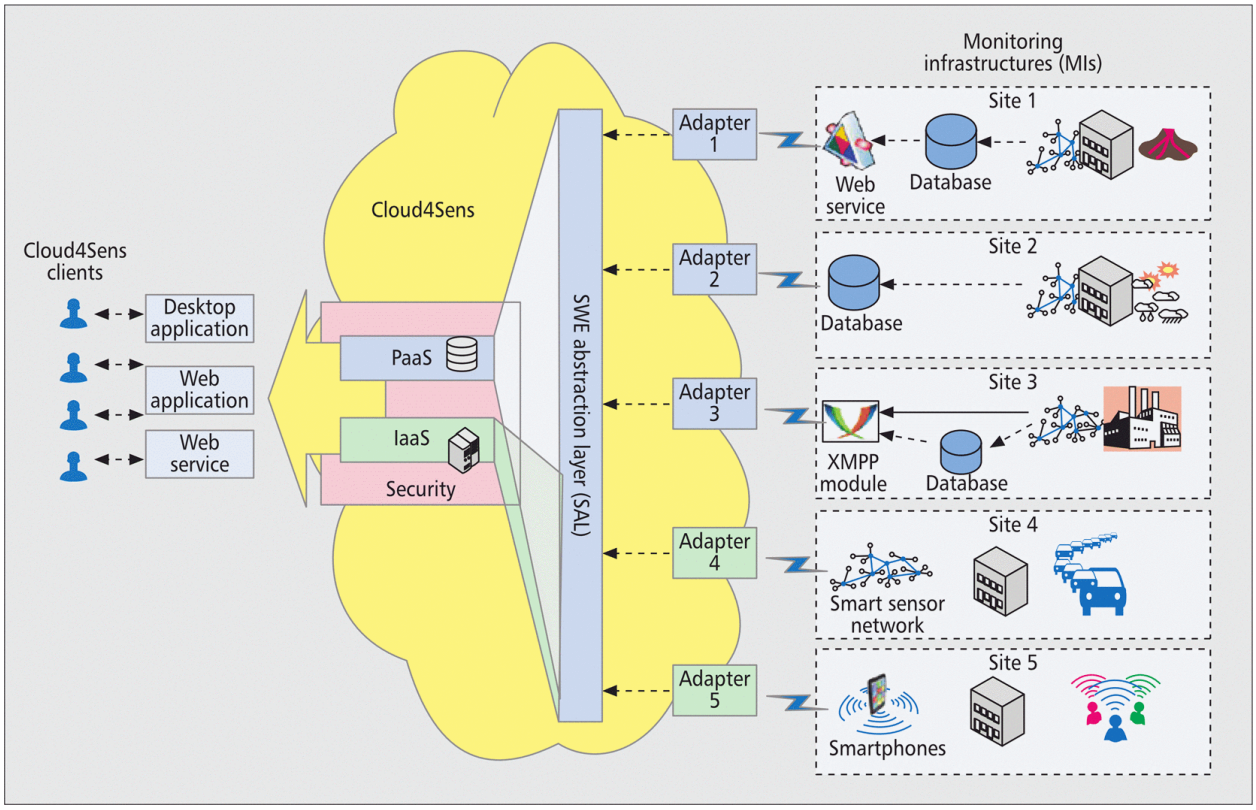

Figure 3.10: Cloud4Sens Architecture [33]

Antonio Celesti et al. [34] introduced a new development model for PaaS named "the Distributed Resilient Adaptable Cloud Oriented (DRACO)" as shown in Figure 3.11. It consists of three layers: Virtual Machine Manager (VMM), Virtual Infrastructure Manager (VIM), and Cloud Manager (CM). The base layer VMM layer can be associated to the Hypervisor running on top of the Operating System (OS). The middle layer, VIM provides an abstraction layer for the VMM. It interacts with the Hypervisor running on top of the OS of each physical server included in the cloud datacenter, and enables the capability of deploying

virtual machines on the physical hardware where the VMM is running. Most of the cloud middleware mainly exploit this layer adding some new features. The main purpose of the VIM layer is essentially being its ability to set up VMs regardless of the underlying VMM. The final upper layer CM layer is built on top of VIM layer. They have focused on how DRACO PaaS can be adopted for the development of any specialized PaaS. The authors have provided a platform for the development of complex algorithms in the cloud. They have analyzed a distributed file system which is able to optimize the concurrent writing operations. However, before doing all optimized operation, we strongly need deployment and management phase. In this work, authors have ignored the central aspect of the application life-cycle.

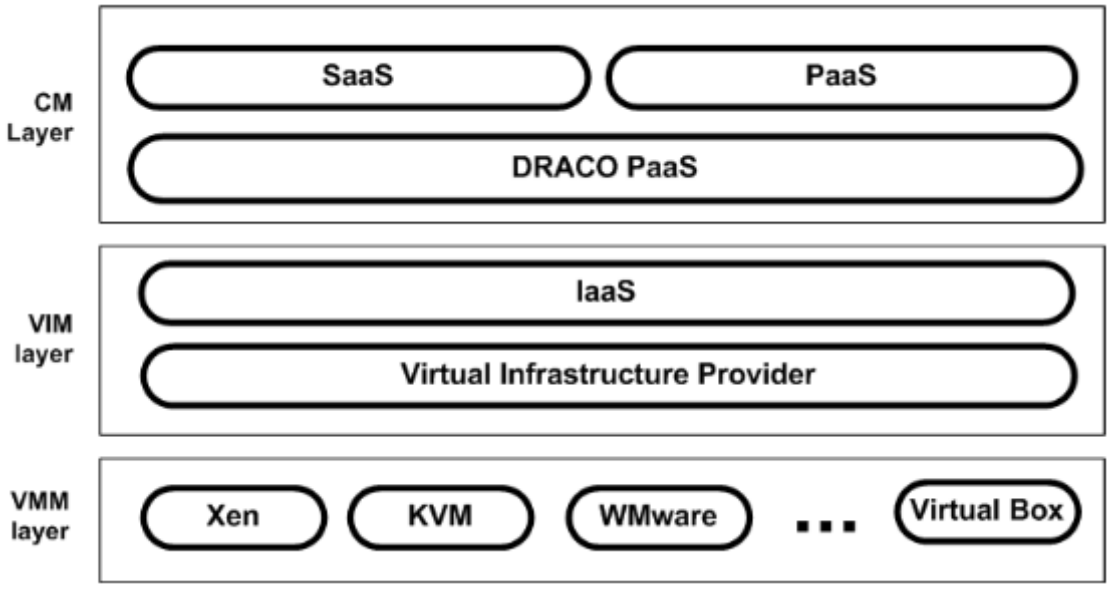

Figure 3.11: Virtual DARCO Network in three-layer Cloud Computing Stack [34]

Vögler, Michael et al. [35] presented LEONORE that supports push-based as well as pull-based deployments. To improve scalability and reduce generated network traffic between cloud and edge infrastructure, they have emphasized on distributed provisioning approach on resource-constrained, different edge devices in large-scale IoT deployments. Their

architecture helps to improve provisioning time while drastically reducing bandwidth consumption.

Sherif Abdelwahab et al. [36] presented a multilayer architecture of Cloud-Assisted Remote Sensing (CARS). The main aim of the CARS is to sense and collect data from anywhere in a distributed way by describing each layer's functionalities and responsibilities, as well as its interactions and interfaces with its upper and lower layers. The architecture, which has four main layers: 1) fog layer; 2) stratus layer; 3) alto-cumulus layer; and 4) cirrus layer. Fog layer mainly provides an abstraction for all physical objects. It also provides a unique identification of all things through IPv6. In Figure 3.12, stratus layer is a mid, tier-2 layer that consists of thousands of clouds whose main resources are sensory devices and sensing and actuating networks. Alto-cumulus is a middle layer that works as a point of liaison between stratus and cirrus layers. It facilitates discussions related to pricing, policy and regulations, and SLAs between stratus and cirrus layers. Cirrus layer is the highest layer in the CARS architecture, and its primary role is to cooperate with CARS service customers and satisfy their requests with the aid of lower layers. CARS provides a set of APIs to develop applications that abstract the physical SANs. It then meets the requirement concerning the development phase. On the other hand, several architectural considerations are also mentioned such as the interface design, the interactions between CARS entities, sensor availability and mobility and cross-vendor resource sharing. The proposed work does not meet the requirements of deployment and management phase for concurrency-based application provisioning in IoT domain.

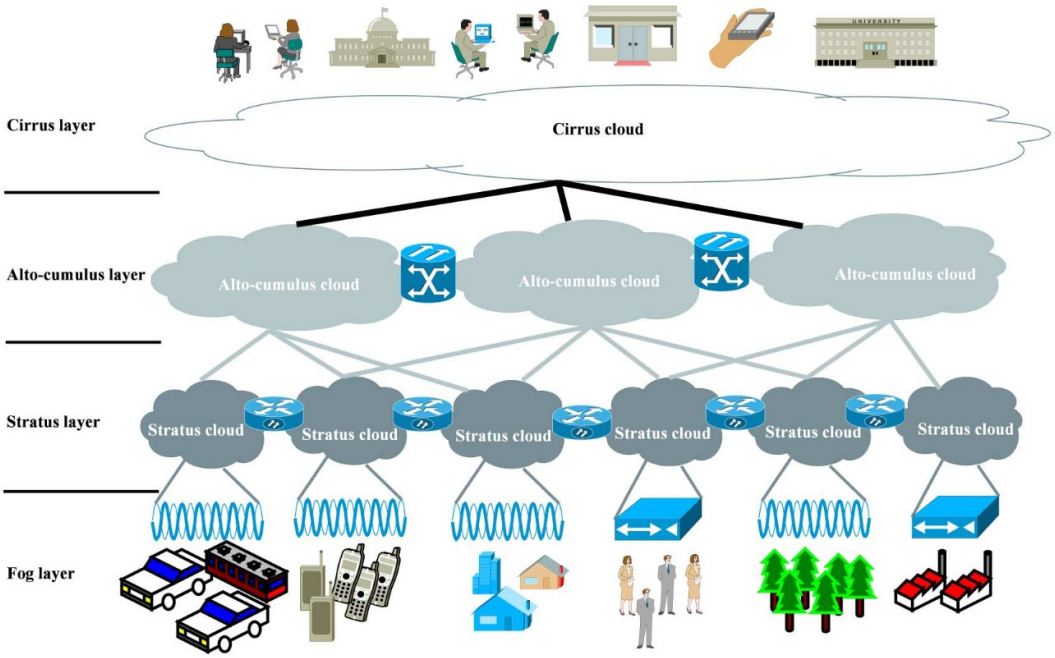

Figure 3.12: Proposed CARS architecture [36]

John K. Zao et al. [37] have proposed EEG Tractor Beam that is based on augmented brain-computer interfaces. They have developed a ubiquitous brain state monitoring and data sharing system using wireless EEG headsets, smart-phones, and ubiquitous lightweight servers. In order to support real-time interactions between the wireless EEG headsets and the ubiquitous servers, the system must provide reliable event-based communication. As shown in in Figure 3.13, publish/subscribe protocols have become the preferred data transport mechanism for Machine-to-Machine (M2M) communication and the Internet of Things. They have conducted real-time synchronous data streaming and launch a multi-player on-line BCI game EEG Tractor Beam among multiple sites in the United States and Taiwan. EEG Tractor beam does not provide any tools and/or APIs for development. Hence, it does not meet the requirement concerning the development phase. They are using standard M2M publish/subscribe protocol and Message Queueing Telemetry Transport (MQTT) for deployment and execution. This work does not meet the requirement concerning the management phase.

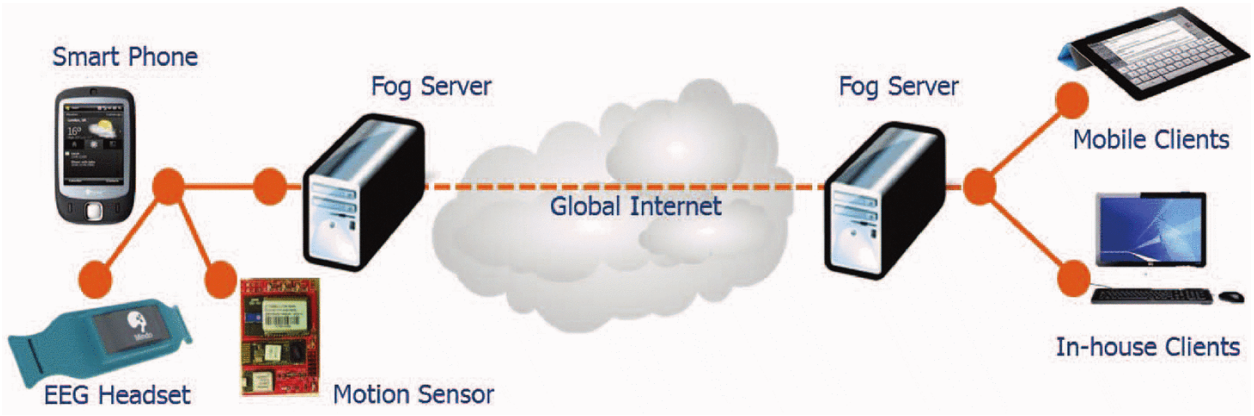

Figure 3.13: Architecture of Neuroimaging System [37]

From the above work, we can see that there are many PaaS architecture for IoT application. To the best of our understanding, there are no solutions available in literature that can offer both the user-centric and provider-centric services at the same time. Today's cloud platforms are home to a plethora of different types of applications, all having unique characteristics. The Table 3-1 below summarizes the evaluation of the state-of-the-art. The column value "Partial Satisfied" means that the related work deals with and fulfills only parts of the requirement. "Not discussed" means the related work does not deal with the requirement whereas "Not Satisfied" means the related work deals with the requirement but does not meet it. "Satisfied" means the requirement is fully satisfied by the related work.

Related Work	Requirements	Development Phase	Deployment Phase	Management Phase
PaaS Solutions	[25]	Not Satisfied	Partial Satisfied	Partial Satisfied
	[26]	Not Satisfied	Partial Satisfied	Partial Satisfied
	[27]	Not Satisfied	Satisfied	Partial Satisfied
	[28]	Not Satisfied	Partial Satisfied	Partial Satisfied
	[29]	Not Satisfied	Partial Satisfied	Partial Satisfied
IoT Cloud based Architecture	[30]	Not Addressed	Not Addressed	Partial Satisfied
	[31]	Not Satisfied	Not Satisfied	Partial Satisfied
	[32]	Partial Satisfied	Partial Satisfied	Not Satisfied
	[33]	Not Satisfied	Not Satisfied	Partial Satisfied
	[34]	Partial Satisfied	Not Satisfied	Not Addressed
	[35]	Partial Satisfied	Partial Satisfied	Partial Satisfied
	[36]	Partial Satisfied	Not Satisfied	Not Satisfied
	[37]	Not Satisfied	Partial Satisfied	Not Satisfied

Table 3.1 Summary of evaluation of the state-of-the-art solutions regarding Provisioning of Concurrency-based Application in IoT

3.4 Chapter Summary

The chapter presents a motivating scenario for concurrency-based application in IoT. After that, a set of requirements are derived based on the motivating scenario and demonstrated the need of concurrency-based application in IoT. A classification of the requirements on the PaaS for provisioning of Concurrency-based Application. The state-of-the-art solution related to this research domain is evaluated based on the requirements described in section 3.3. We found that none of the state-of-the-art solutions satisfy our requirements fully.

Chapter 4

Proposed Architecture

The previous chapter discussed the research work accomplished in the area of IoT-based PaaS architecture for concurrency-based application provisioning. The study depicted that concurrency-based application provisioning challenges in IoT have not been addressed to a large extent. To provide a solution to concurrency-based application provisioning, architecture has been proposed and designed in this chapter. The first section discusses the proposed business model. The second section discusses the various interfaces and provides the overall architecture. The overall architecture is designed by considering the identified requirements in the previous chapter. Few principles are set for the proposed architecture. The third section illustrates the detailed scenario of development, deployment, and execution with the use-case. Finally, it discusses how the proposed architecture meets the requirements and summarizes this chapter.

4.1 A Business Model for Concurrency-based application Provisioning

This section presents a business model in the concurrency-based application in IoT. Firstly, the initial assumptions which are taken are stated. The actors of motivating scenario are identified in the next subsection. Finally, we apply the scenario in a wildfire management use-case in IoT.

4.1.1 Assumptions

Let us consider the case of an IoT-based cloud environment where there is a concurrency-

based application provisioning provided by a PaaS provider. We assume that there are application developers who can develop and provide concurrency-based applications by using the infrastructure according to their requirements. The main assumption of the proposed work is that all the concurrency-based applications are offered as cloud services. The key benefit is to allows developing and easy access to the application from anywhere and anytime.

4.1.2 Actors

In this motivating scenario, four types of actors are identified: the end-user, IoT SaaS provider, IoT PaaS provider, and IoT IaaS provider. Each actor may play several roles through interactions. The brief description of each actor is as following:

1) ***The End-User:*** The main role of end-user is to discover and use the applications that are provided by IoT SaaS providers.

2) ***IoT SaaS Provider:*** The IoT SaaS provider is the application developer, which provides the facility to the end-users for the consumption of concurrency-based applications in IoT.

3) ***IoT PaaS Provider:*** The PaaS provider enables the development, deployment, and execution of a concurrency-based application by offering necessary development frameworks, various service APIs' and tools.

4) ***IoT IaaS Provider:*** The IoT IaaS provider is responsible for installing, and managing IoT sensor devices in different locations. It also provides the computation, networking, and storage to the IoT Cloud PaaS.

4.1.3 Interactions among Actors

Figure 4.1 shows the interactions between the different actors, i.e., end-users, the IoT SaaS provider, the IoT PaaS providers, and the IoT IaaS providers.

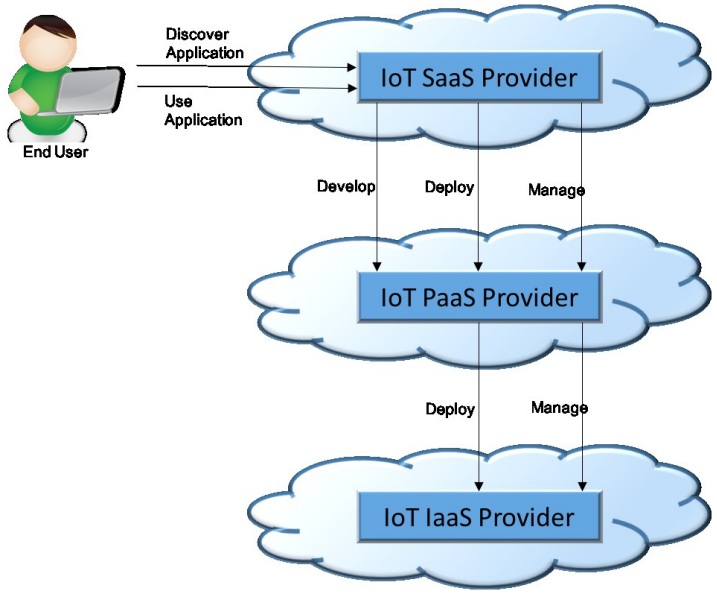

Figure 4.1: Different roles of the actors

A. End-user Interaction

In the proposed scenario, the end user interactions can be defined into two types that happen between the end-user and the IoT SaaS provider.

- **The Discover Concurrency-based Application Interaction**

  This interaction allows the end-user to find the concurrency-based application that suits its need. The discovery may involve an intermediate broker that plays the role to bridge the gap between the end-user and the concurrency-based application provider.

- **The Use Concurrency-based Application Interaction**

The end-user uses the concurrency-based application through this interaction. The use of the concurrency-based application involves usage, monitoring, detecting events, and taking appropriate actions.

B. IoT SaaS Interactions

This section presents interactions between the IoT SaaS and the IoT PaaS. Three types of interaction are defined as following:

- **Develop Concurrency-based Application Interaction**

  The IoT SaaS provider or the application developer is responsible for developing the applications. The IoT SaaS provider interacts with the IoT PaaS provider for analyzing the requirements, designing, implementation, and testing of the concurrency-based application in IoT.

- **Deploy Concurrency-based Application Interaction**

  After developing the concurrency-based application, IoT SaaS provider deploys it on the cloud platform. This interaction involves the installation, configuration, and activation of the concurrency-based application.

- **Manage Concurrency-based Application Interaction**

  This interaction is used to describe the execution, utilization and management of the concurrency-based application by IoT SaaS Provider.

C. IoT PaaS Interactions

These interactions occur between the IoT PaaS provider and the IoT IaaS provider.

- **Deploy Concurrency-based Application Interaction**

  This interaction describes the introduction of the different processes of concurrency-based application in the different Infrastructure IoT Cloud. On the basis of the requirements and guidelines on how to deploy in the physical system, further actions are taken.

- **Manage Concurrency-based Application Interaction**

  This interaction describes the actual use of different processes of a concurrency-based application on the different IoT IaaS. This different IoT IaaS executes the sensing tasks by sensing different events, generating sensor measurements, and sending sensor data to the multiple processes of concurrency-based applications existing in the IoT PaaS provider.

4.1.4 The Business Model Applied to the Wildfire Management Scenario

We now apply the described interactions in the wildfire management use case. The actors are the IoT SaaS provider, IoT PaaS provider, IoT IaaS provider, and the end-user as a city administrator. Figure 4.2 shows the sequence diagram of the end-to-end execution of the scenario.

To offer the wildfire management application as a concurrency-based application, the IoT SaaS provider must request the development API from the PaaS provider. The PaaS provider and IaaS provider are involved in the development, deployment, and the management of the fire monitoring processes of the wildfire management application.

For the development of the concurrency-based application, the PaaS provider provides the framework, libraries, and tools related to the fire monitoring process (process). The PaaS

provider composes these multiple processes to develop the wildfire management application. When the PaaS provider deploys the wildfire management application, it also requests the installation of the different processes from the interacting IoT PaaS provider. When the city admin discovers and uses the wildfire management application, it requests the execution of the multiple processes on different IoT Infrastructure cloud.

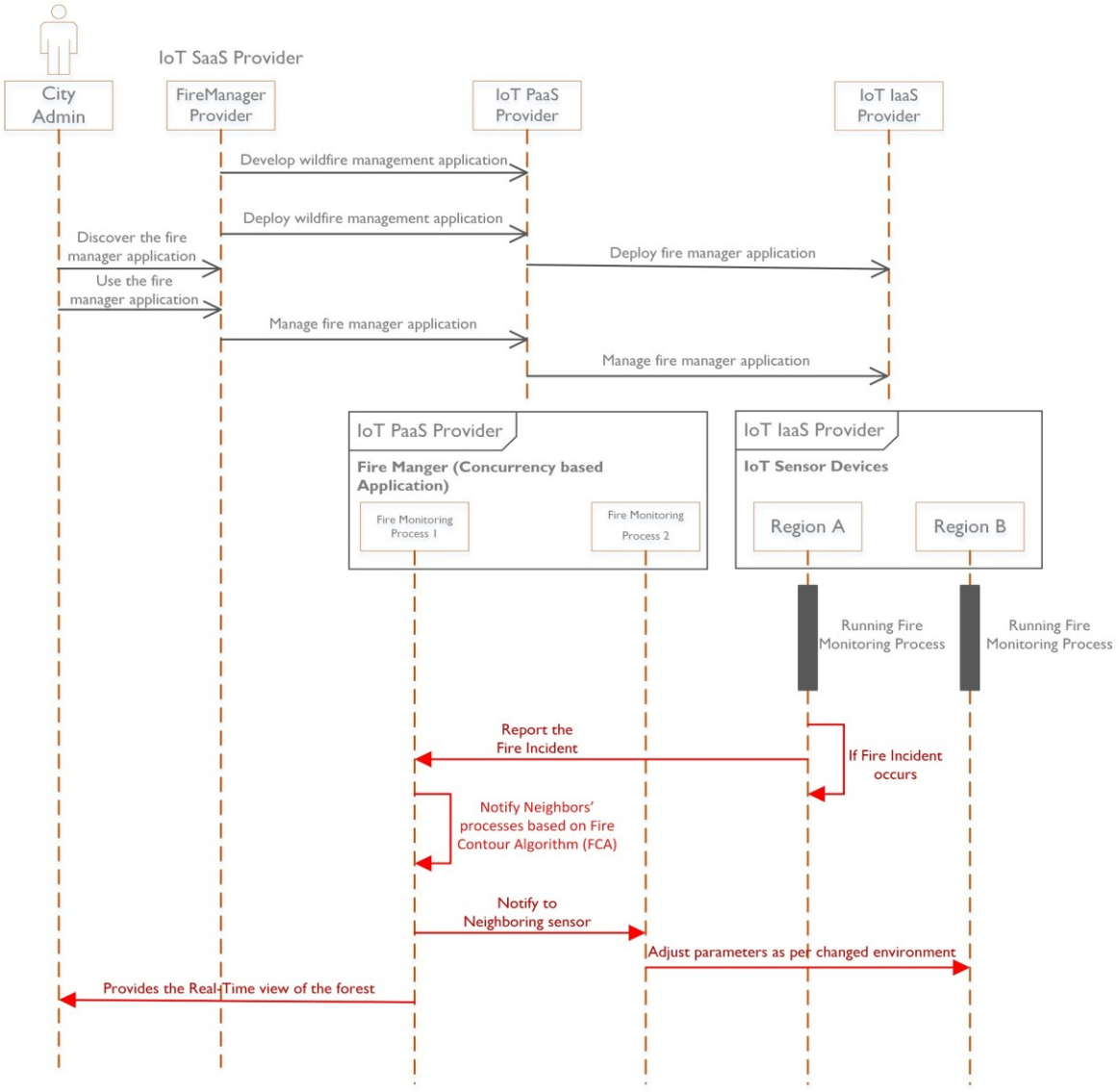

Figure 4.2: Interaction among the actors and end-to-end execution of Wildfire Management Scenario

4.2 Overall Architecture

In this section, the architectural principles are discussed that is followed to design the proposed architecture for a concurrency-based application provisioning. Then the architectural components are described.

4.2.1 Principles

Our architecture is based on a set of following principles. The following principles are laid down for designing the proposed architecture for provisioning of concurrency-based applications.

- **Use of Cloud Foundry open source:** The architecture is designed as an extension to the high-level architecture of cloud foundry. This architecture allows us to reuse the existing PaaS as much as possible for the concurrency-based application provisioning in an Internet of Things.

- **Use of RESTful interface**: REpresentational State Transfer (REST) paradigm is used for the interactions between the different processes of the concurrency-based application. These processes are running in different containers within the PaaS. The interaction between different processes share the content information based on the event happened such as a process monitoring an area can report fire incident to the processes monitoring neighboring areas.

4.2.2 Proposed Architecture

In the proposed architecture, the novel modules of PaaS to support concurrency-based application provisioning in IoT are introduced. While designing these modules, the existing PaaS architecture is considered as a basis.

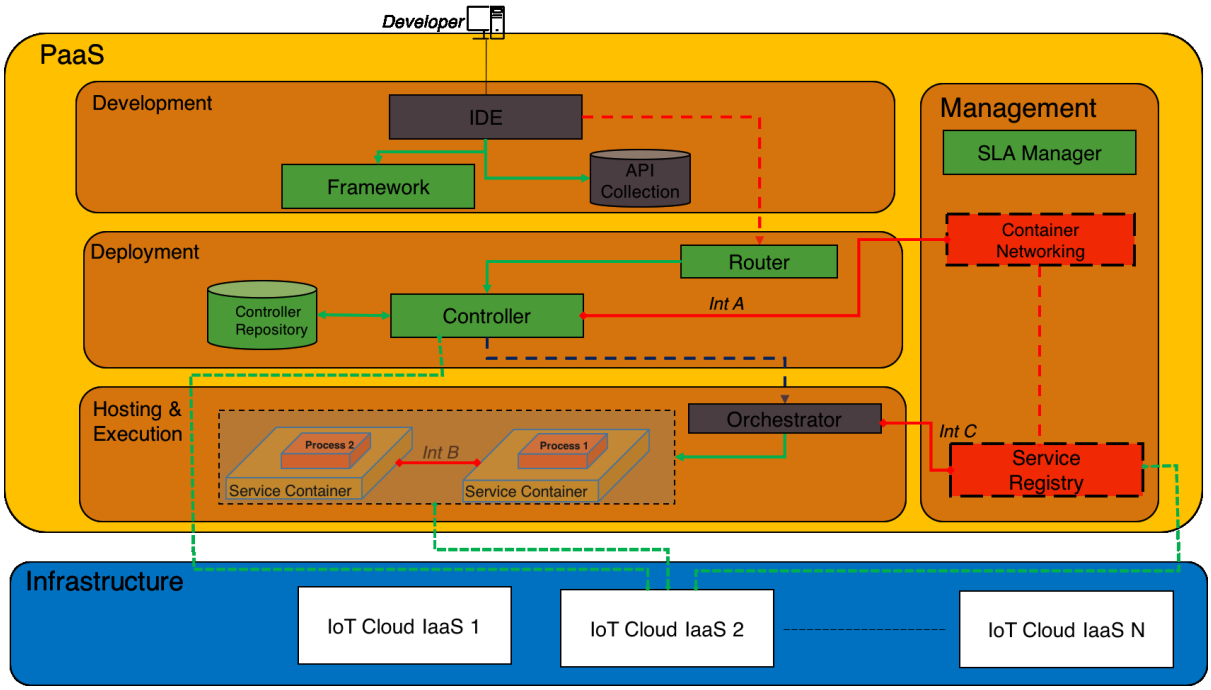

Figure 4.3: Proposed Architecture

Figure 4.3 shows the proposed architecture for provisioning of concurrency-based application. Green color boxes such as framework, router, controller, and controller repository represent the components which do not require any change before being able to handle concurrency-based applications provisioning. The router routes incoming traffic to the appropriate component of the platform. The controller is a brain of PaaS which manages the entire application lifecycle. Controller repository stores the metadata of application, and provides with all supported libraries and framework during application deployment. Blue color boxes such as IDE, API collections, and orchestrator represent the components which

have been modified. To provide the facility of developing and composing, different APIs and novel modules are added to the IDE while taking account of all the properties and the capabilities of the target placement domain. Red color boxes represent the new components that are introduced in the proposed architecture. Novel components such as container networking and service registry are added to use with the existing components of cloud foundry. The container networking is a novel module which provides the network interface to the different processes of the concurrency-based application. The primary use of container networking provides the direct interaction between the different application processes running into a different container. The main intent of the service registry is to provide the facility of discovering the requested process. Additionally, service registry is also a novel module where all the processes of concurrency-based application register themselves. This registry must decide which IP/address to return based on who is asking to make the communication between two different processes during the execution. In existing cloud foundry PaaS architecture, there was no direct interaction with development and deployment layer. To make it possible for the execution of an application in an efficient manner, new interaction flow is added. In the next section, we describe the interaction between existing modules and new modules as layer-wise. Each layer has explained the working of components and how these modules help in the provisioning of concurrency-based applications.

1) Architectural modules

In the proposed architecture, *four main layers* can be distinguished: development, deployment, hosting & execution and management. All components of different layers are explained as follows:

Concurrency-based application development layer: The main intent of the first layer is to provide the Integrated Development Environment (IDE) to the developers. This development environment facilitates the developers with various tools such as APIs', libraries, frameworks and so on. This layer is mainly used for the development, testing, and deployment of different processes of the concurrency-based application. The current PaaS does not come with the existing IDE, so we have to re-use the third party IDE to support the development phase. Moreover, the existing IDEs' such as Eclipse, NetBeans does not support such development of a concurrency-based application. Therefore, to provide the facility of developing and composing, different APIs and novel modules such as Eureka API are added to the IDE while taking account of all the properties and the capabilities of the target placement domain. These new modules enable the interaction with various modules in PaaS and allow for easy deployment to the PaaS. Developers will try to access the frameworks and API collections as per the requirements of concurrency-based application related to the Internet of Things smart devices firmware. The main use of framework for concurrency-based applications is to provide the various libraries to facilitate base to build different processes in a concurrency-based application. The API provides the interface for the easy access of exposed function of libraries. The IDE also provides an inbuilt server for the testing of different processes of the concurrency-based application.

Concurrency-based application deployment layer: After developing concurrency-based applications, the task of deployment is started. The main module of this layer is the controller. When developer pushes the application to the platform, a router will direct the application deployment request to the controller through IDE. The

controller will communicate with controller repository to read the manifest which contains all the information related to the application. During the deployment process, the controller interacts with container networking module with the appropriate interface to provide the separate identity to the application container. The container networking enables the edge of the different processes of the concurrency-based application. Additionally, container networking module will interact with service registry to take information about processes of the concurrency-based application. A service registry is a module where all the processes of concurrency-based application register themselves. Service registry decides which IP/address to return based on who is asking to make the communication between two different processes during the execution as shown in Figure 4.4. Simultaneously, management layer module SLA manager will communicate with the controller to discuss Service Level Agreement. Then based on the SLA, it discovers and instantiates the elastic runtime of PaaS required for hosting and executing the application's processes (e.g., service containers, DBMS). It also operates the deployment of the concurrency-based application's processes over these resources. The processes of concurrency-based application are deployed in a dedicated service container. The deployed processes interact and exchange messages through appropriate interfaces based on the condition or event happened.

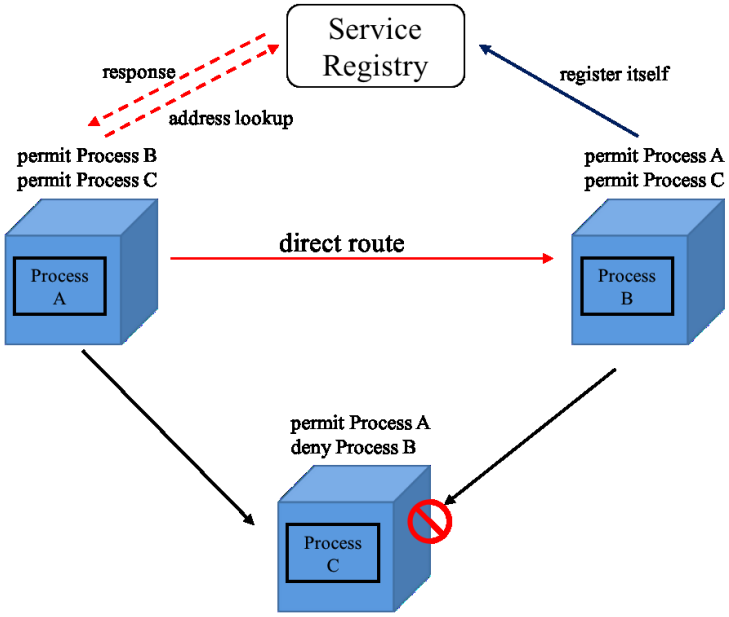

Figure 4.4: Service Registry during the execution of Concurrency-based application

***Concurrency-based application hosting & execution layer*:**

The orchestrator modified in this layer provides the execution flow between the different service containers. We have modified the orchestration plan based on our requirements. The different processes of concurrency-based applications are executing in the different service container to get isolated environment. The orchestrator will maintain the messages exchanged between the different service containers based on the specified condition.

***Concurrency-based application management layer*:**

This layer plays a vital role as that has to interacts with the three above discussed layers simultaneously. In this layer, the SLA manager will handle the new modules and the extended module from the regular PaaS. Based on the SLA agreement, the SLA manager will enable the different modules which are required during the applications life-cycle. For the provisioning of concurrency-based application in IoT,

it is required to use the specific framework for the development phase, container networking, and service registry during the deployment and operational phases.

2) Interfaces

Interfaces are designed between different service containers of PaaS as shown in Figure 4.3. There are no changes in the PaaS-IaaS interfaces.

- Int. A is a deployment interface that allows the controller to read the application's manifest and allocate the necessary resources to the multiple processes of the concurrency-based application. It provides the unique identity to each process of a concurrency-based application using the container networking module. Interface A REST API as defined in Table 4.1 below will list, create and delete the network policy to allow direct network traffic from one process to other processes of the concurrency-based application.

Method	Operation	Path	Arguments	Request Body Parameters	Response Body
GET	List of all network policies	/networking/v1/external/policies	Id: policy_group_id		Source ID, Destination ID, Protocol(TCP, UDP), Ports
POST	Create Policies	/networking/v1/external/policies		Source ID, Destination ID, Protocol(TCP, UDP), Ports	
DELETE	Delete Policies	/networking/v1/external/policies/delete		Source ID, Destination ID, Protocol(TCP, UDP), Ports	

Table 4.1: Interface A REST API

- Int. B is a signaling, controlling and data interface which allows concurrency-based application's processes (within the containers) interact with each other during execution of the whole application.

Method	Operation	Path	Description
POST	Notify the neighboring process (Current status)	/process/**procesID**	Input: JSON/XML payload HTTPCode: 204 on success

Table 4.2: Interface B REST API

- Int. C is a management interface that allows the PaaS management module to monitor (and eventually act on) the PaaS resources such as service registry. Table 4.3 provides the list of REST operations available for Interface C.

Method	Operation	Path	Description
POST	Register new application instance	/eureka/v2/apps/appID	Input: JSON/XML payload HTTPCode: 204 on success
DELETE	De-register application instance	/eureka/v2/apps/appID/instanceID	HTTP Code: 200 on success
GET	Query for all instances	/eureka/v2/apps	HTTP Code: 200 on success Output: JSON/XML
GET	Query for all appID instances	/eureka/v2/apps/appID	HTTP Code: 200 on success Output: JSON/XML

Table 4-3: Interface C REST API

Int. B and Int. C are designed according to the REST principle. However, they are used to meet the new requirements. Int. B, for instance, is used to support communication between

different service containers based on the end-to-end principle. Int. C is used to provide management of service discovery, which decides what IP to return based on who is making the request.

Int. A interface is designed according to the REST principle. It uses to provide the unique identity to the application processes over the available resources. The unique addressing of each processes of a concurrency-based application allows the direct networking during the execution of the application.

4.3 Illustrative Scenario: Wildfire Management Use-Case Scenario

In this section, we describe the wildfire management use-case scenario based on the proposed architecture. The illustrative scenario consists of several actors. The first actor is a wildfire management application which is a concurrency-based application is hosted on the IoT PaaS. The second actor is the IoT IaaS providers who deploy various sensor nodes (e.g., temperature & humidity sensors) in different forest regions including forest, nearby public streets, and private homes. The IoT IaaS provider runs the multiple fire monitoring processes of wildfire management application in a different region to watch the whole forest.

We divide the scenario into two parts. The first part concerns development and deployment of wildfire management application. It illustrates how a developer can develop a concurrency-based application in IoT efficiently in the development phase. The second part relates to the execution of all fire monitoring processes in the different IoT IaaS. It shows how the PaaS manages the interaction between fire monitoring processes of wildfire management application in IoT.

Both parts of the scenario demonstrate the relevant interactions among the different architectural components of the proposed PaaS architecture. Although the scenario

constitutes a subset of all the functionalities of the proposed PaaS, it helps understand how the proposed architecture works and facilitates wildfire management application provisioning.

4.3.1 Concurrency-based Application development and deployment

In this scenario, we assume that the developer or IoT SaaS provider wants to develop a concurrency-based application in IoT. The targeted IoT smart devices are capable of handling Message Queue Telemetry Transport (MQTT) protocol, a lightweight messaging protocol and can execute the sensing tasks by sensing different measurements specified by ASTM standard. We also assume that deluge protocol offered by IoT IaaSs to support selective re-tasking in case of emergency. This deluge protocol is reliable data dissemination protocol, which reprograms the sensor nodes in the network.

The developer develops the concurrency-based application using IDE which incorporates various libraries in the code. In order to handle the temperature readings from the IoT sensor devices, we have used the fire monitoring libraries. These libraries help to convert the raw data to the end-user understandable form. When fire incident is reported, the city admin has to get a real-time view of the forest from the IoT SaaS provider. Moreover, we assume that the developer also makes REST API request as referred in Table 4.2 to notify the fire monitoring process running on the neighboring regions. So, when the fire incident notification is received, the neighbor's fire monitoring process uses a deluge protocol to do re-tasking of the neighboring IoT sensor devices.

When the IoT SaaS provider finishes developing the wildfire management application, an IoT PaaS IDE, is used to deploy the multiple processes in the IoT PaaS. The IDE has an IoT PaaS provider related plugin to interact with the capabilities of target placement domain. The IDE

conveys the command to push application, which deploys the application's manifest file on the controller using the router gateway. The controller of the PaaS parses the manifest and lists the required hosting & execution components. The interactions of components are illustrated in Figure 4.5. After the processes are deployed, the IoT SaaS provider, using the same PaaS GUI, starts the wildfire management application.

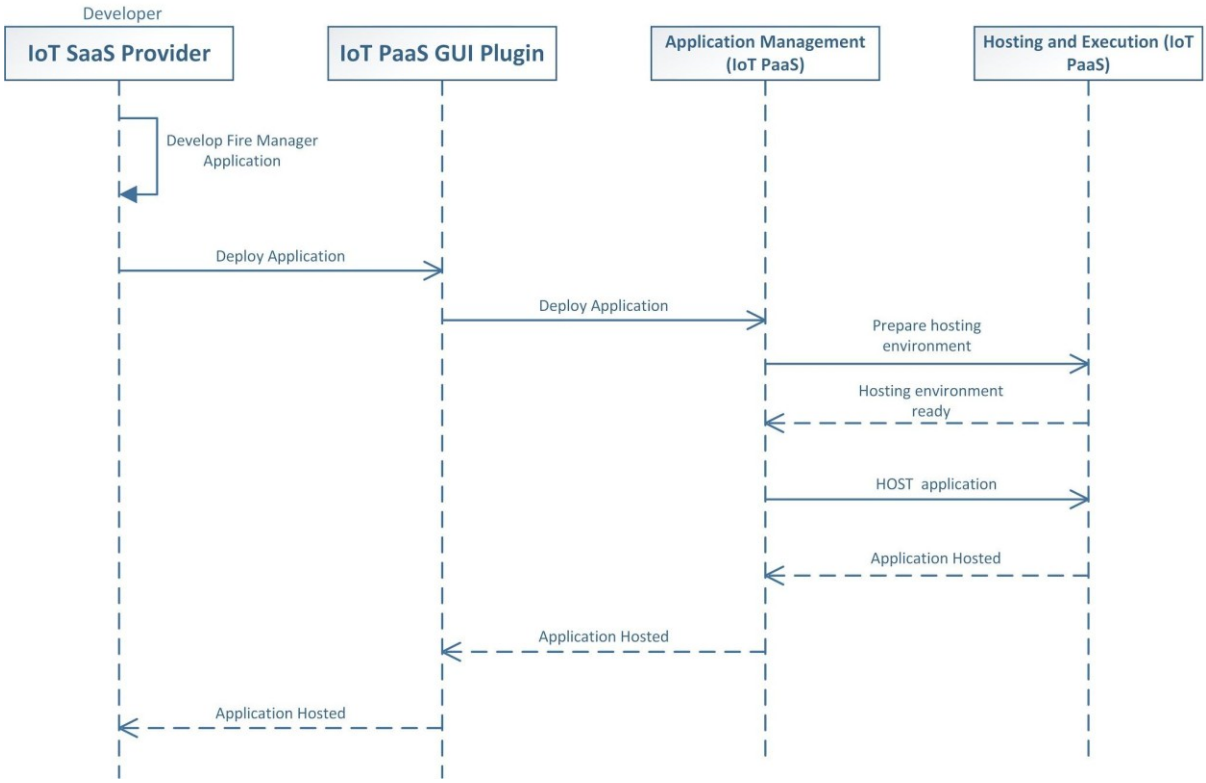

Figure 4.5: Illustrative diagram for Application development and deployment

4.3.2 Concurrency-based Application execution

Once the application is started, multiple fire monitoring processes start receiving temperature information of the different region, i.e., IoT IaaS A, IoT IaaS B. Figure 4.6 depicts the interactions when the fire monitoring process receives an observed accident happened in any region. The process first invokes the real-time map drawer API to notify to the city admin.

After that, it uses the service discovery component in the code to handle neighbor region notification. The city admin interaction is not shown in the Figure 4.6.

Orchestration and management component is used for handling of API invocation. The main aim of this component is to determine necessary libraries and selects appropriate IaaSs. It is assumed that it selects IoT IaaS B as a neighbor region and IoT IaaS A, where the incident has happened. Next, it requests IoT IaaS, via deluge protocol, to selective re-tasking of IoT IaaS B. The peer-to-peer connection between the processes of a neighbor region is not shown in the Figure 4.6. After activation, orchestration and management component orchestrates an operation using service discovery and then executes it. A full-fledged concurrency-based application execution in an Internet of Things is represented. A real-time view of accident happened is returned to the city-admin.

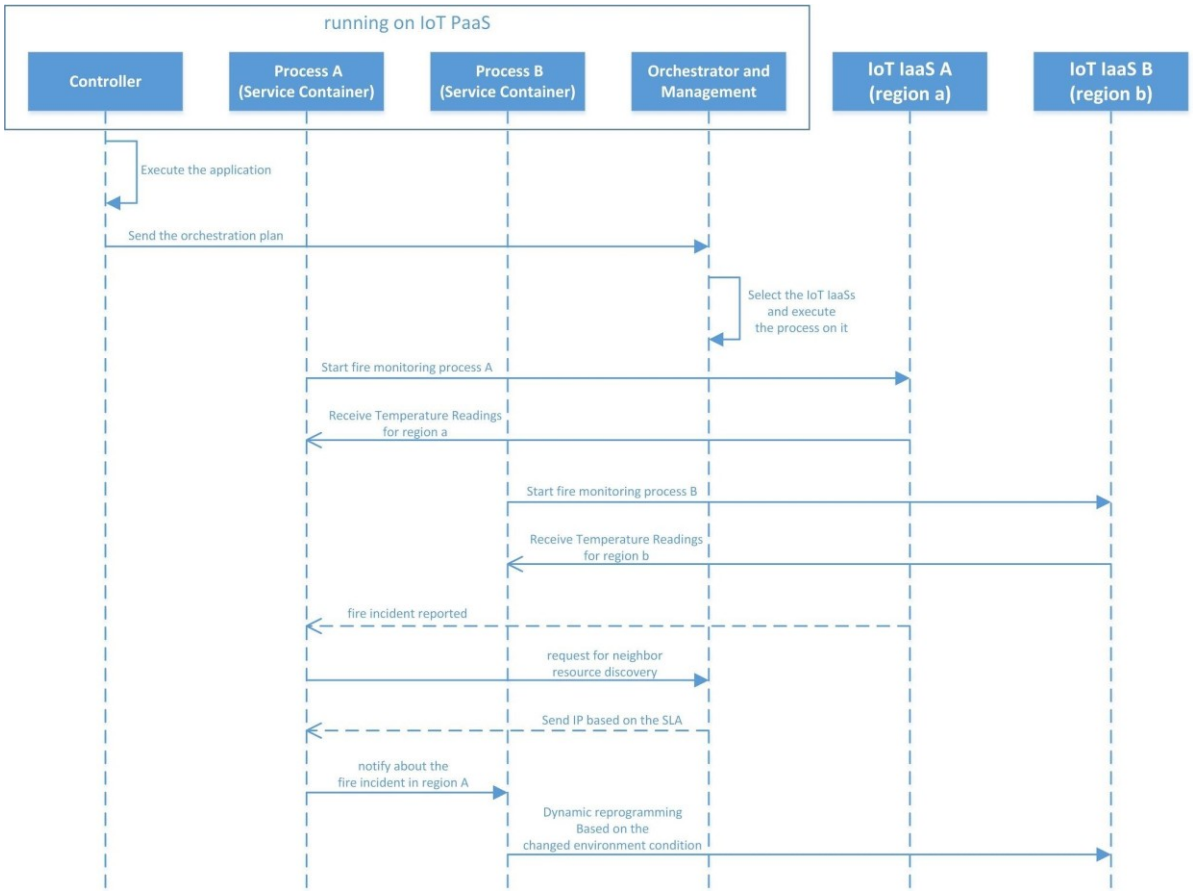

Figure 4.6: Illustrative diagram for the Concurrency-based application execution

4.4 How the Proposed Architecture Meets the Requirement

The proposed architecture of PaaS satisfies all the requirements mentioned in chapter 3. For concurrency-based application development, it provides necessary IDE for that abstract complexity of concurrency. Moreover, The IDE is used to build the concurrency-based application in IoT using various supported APIs, libraries, and framework. This IDE is integrated with the IoT PaaS provider related plugin to deploy the different processes autonomously. During the deployment, it discovers the hosting resources and instantiation of resources allocation based on the requirement. For the concurrency-based deployment, it enables the edges of the application's processes using the container networking. After finishing the deployment process, it sends the acknowledgment for an application hosted. Moreover, it also provides the IDE for starting an application. For the management, it uses the service registry to allow different processes of concurrency-based application to discover each other. It also uses the SLA module to interact simultaneously with the development and deployment process.

4.5 Chapter Summary

In this chapter, firstly we described the business model with the assumptions which are used in our design model. We presented the business actors: the end-user, the IoT SaaS provider, the PaaS provider and the IaaS provider. Then, we explained the interactions of the business model. Secondly, we presented the overall architecture for a PaaS for concurrency-based application provisioning in IoT. We discussed architectural principles that we followed in our design, the main architectural modules and then the interfaces between PaaS-IaaS. Next, we provided an illustrative scenario, showing how different components of the proposed

architecture communicate with each other and how the proposed APIs can help the IoT SaaS providers easily develop the concurrency-based application.

Chapter 5

Validation: Prototype and Performance Results

In chapter 4, we have proposed a general architecture of PaaS for concurrency-based application provisioning in the Internet of Things. In this chapter, we discuss the design of software architecture of PaaS, a prototype to validate the software architecture and the performance measurements of the prototype. Lastly, the chapter summary is presented.

5.1 Software architecture

Cloud Foundry is a cloud application platform, which supports the deployment and management of the web applications. Figure 5.1 shows the extended architecture of cloud foundry which supports the development, deployment, and management of concurrency-based application in IoT. The development phase is realized with components such as IDE, libraries, framework and more which provides the development environment to the developer of concurrency-based applications. The deployment phase is responsible for the installation of a concurrency-based application on the cloud environment. It is realized with components such as a router, cloud controller, Diego and more to allocate resources to the different processes of concurrency-based application in IoT. The management phase maintains the interaction with the development and deployment phases based on the SLA and orchestrates the application execution by providing sufficient resources. In the next section, we have

shown the flow that how components work in each phase of concurrency based application life-cycle.

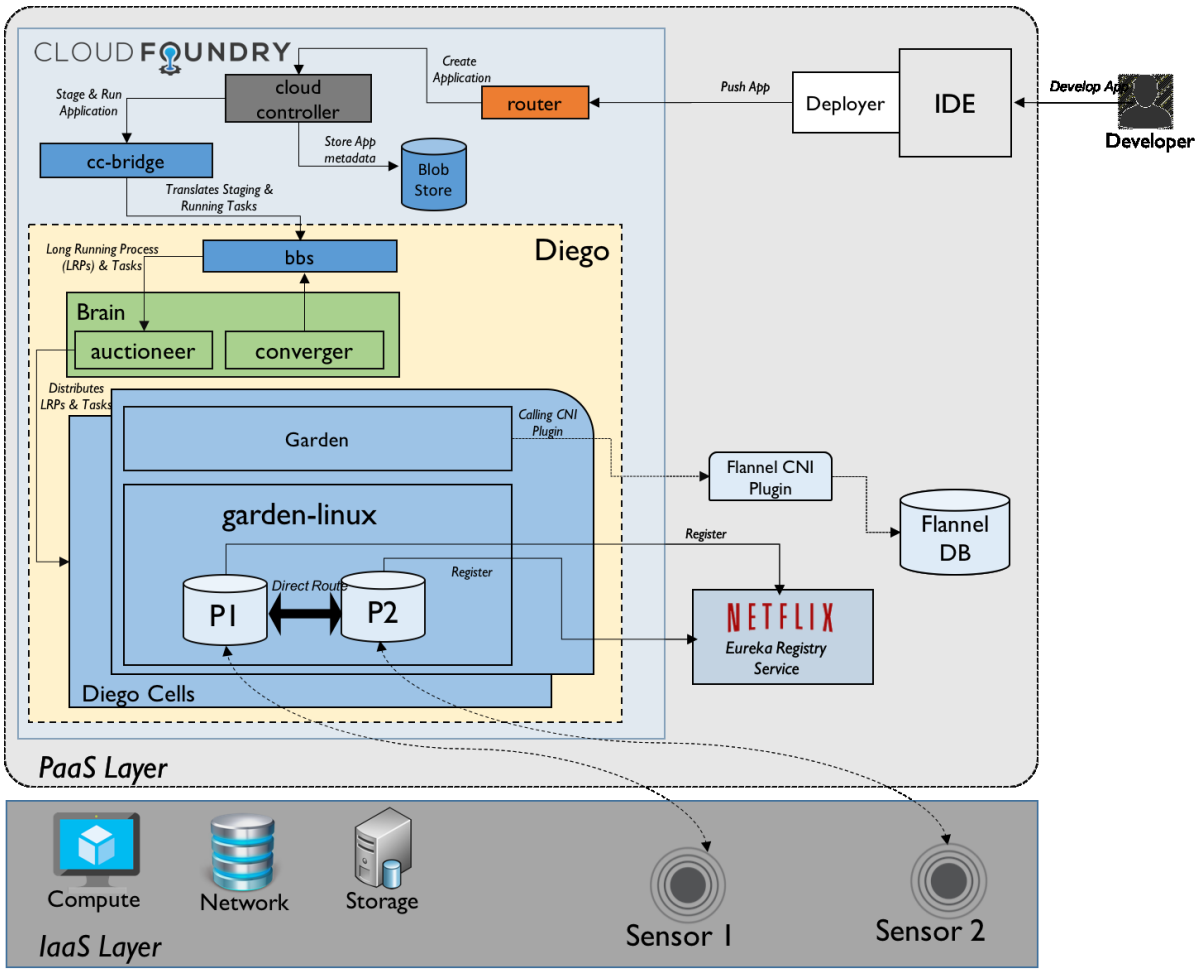

Figure 5.1: Software Architecture

5.1.1 Software modules for the Development and the Deployment phases

Figure 5.1 shows the overall software architecture that we propose for the concurrency-based application provisioning in IoT. This software architecture has been designed by breaking down the modules of proposed architecture (discussed in Chapter 4). In this section, we discuss the main modules used during the development and deployment phase and the interaction between the modules as shown in Figure 5.2.

1. **Integrated development environment (IDE):** The developer develops the concurrency-based application in IoT using the IDE tool such as Eclipse. Moreover, using various libraries such as Eureka and TinyOS based on IoT.

2. **Deployer:** The deployer able to make the connection between development and deployment phase. After the development of an application, the application processes will be pushed to the cloud environment using deployer.

3. **Router:** The router is the entry point of the cloud foundry which maintains the dynamic route table and handles all external traffic. Once the router receives the application push request, it forwards to the cloud controller for the creation of the new application.

4. **Cloud Controller:** The cloud controller maintains the application life-cycle, and direct the application to the Diego during the deployment stage as shown in Figure 5-1. The cloud controller sends the application staging request to the cloud controller bridge (cc-bridge). The cc-bridge translates the staging task to the tasks and long running process (LRP) and directs to Diego brain via bulletin board system (bbs).

5. **Blobstore:** The blobstore is a repository for the application metadata such as code packages, buildpacks, resource files, and droplet image. The blobstore is used by the cloud controller to store its state.

6. **Diego:** Diego monitors the tasks during the deployment by allocating temporary resources. During the deployment phase, the Diego processes the tasks for the finite duration and provide the temporary container to build the application droplet. The droplet is the result of staging and taking all application packages to create an executable binary artifact and store in the blobstore. The container gets destroyed, once the droplet is completed.

7. **Flannel:** Flannel is a CNI plugin which provides the private network interface to each container during the deployment phase.

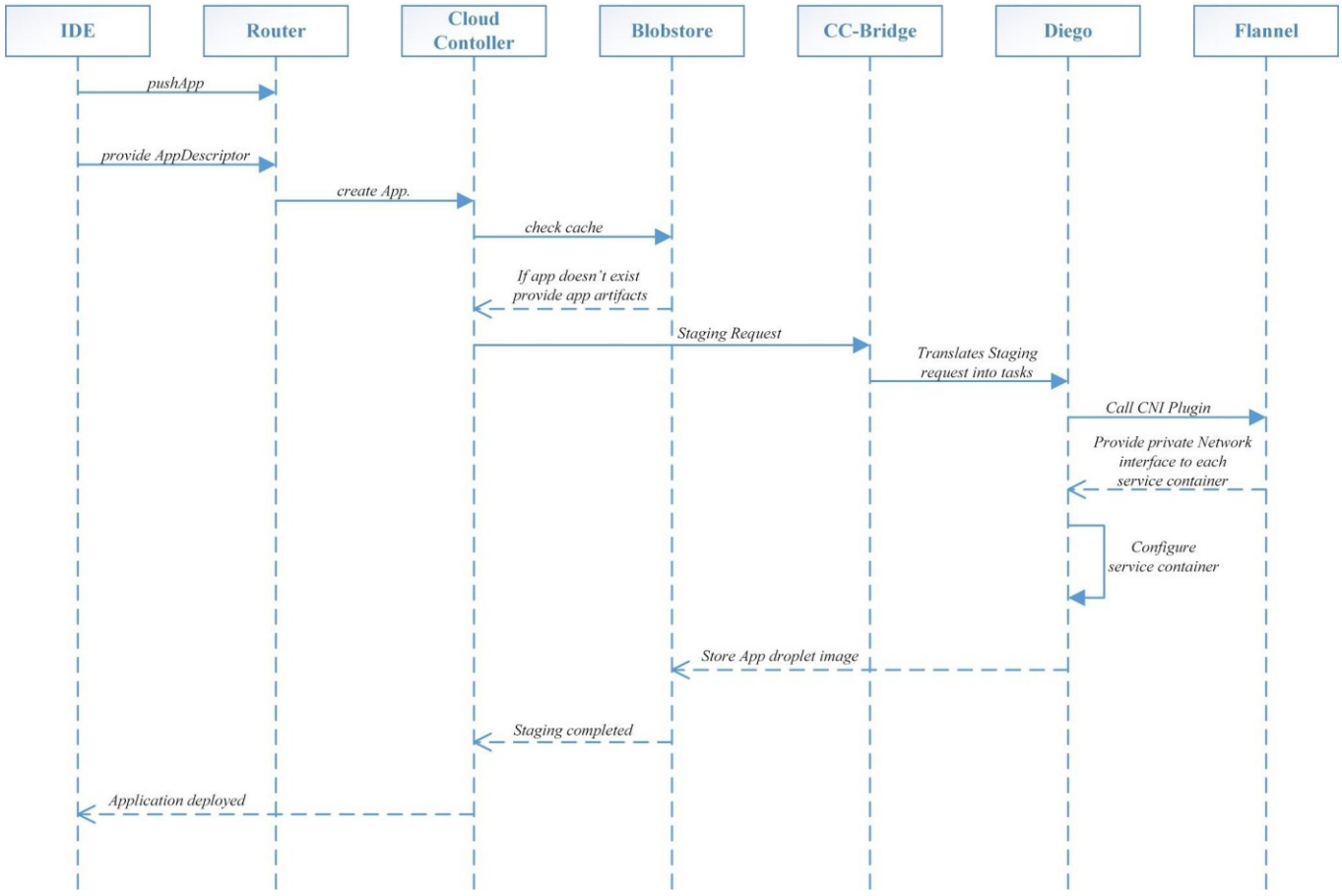

Figure 5.2: Interaction of components during development and deployment phase

5.1.2 Software modules for the Management Phase

The primary intent of the Diego is to maintain the application execution life-cycle' tasks such as managing the scheduling, orchestration and running of workload on the containers. It is realized with the following main modules used for the execution and management of the concurrency-based application. After that, the interaction between the modules as shown in Figure 5.3.

1. **BBS (bulletin board system):** The main task of bbs is to submit the LRPs to the auctioneer (Diego brain) and monitors the desiredLRP and actualLRP using the convergence process during the application execution.

2. **Garden API:** It is a platform independent API for the creation and management of container. It runs the different container image such as Windows, Linux, and runC. However, this garden API also calls the flannel CNI plugin to provide the private network to each container for the processes of the concurrency-based application.

3. **Service Container:** Cloud foundry runs all deployed applications in the service containers. The resulting droplet image created during the deployment phase is running inside the container to provide the isolated environment.

4. **Service Registry:** The different processes of concurrency-based applications depend on the services to bind themselves together for the direct communication. The service registry stores all the key-value pair of the running application.

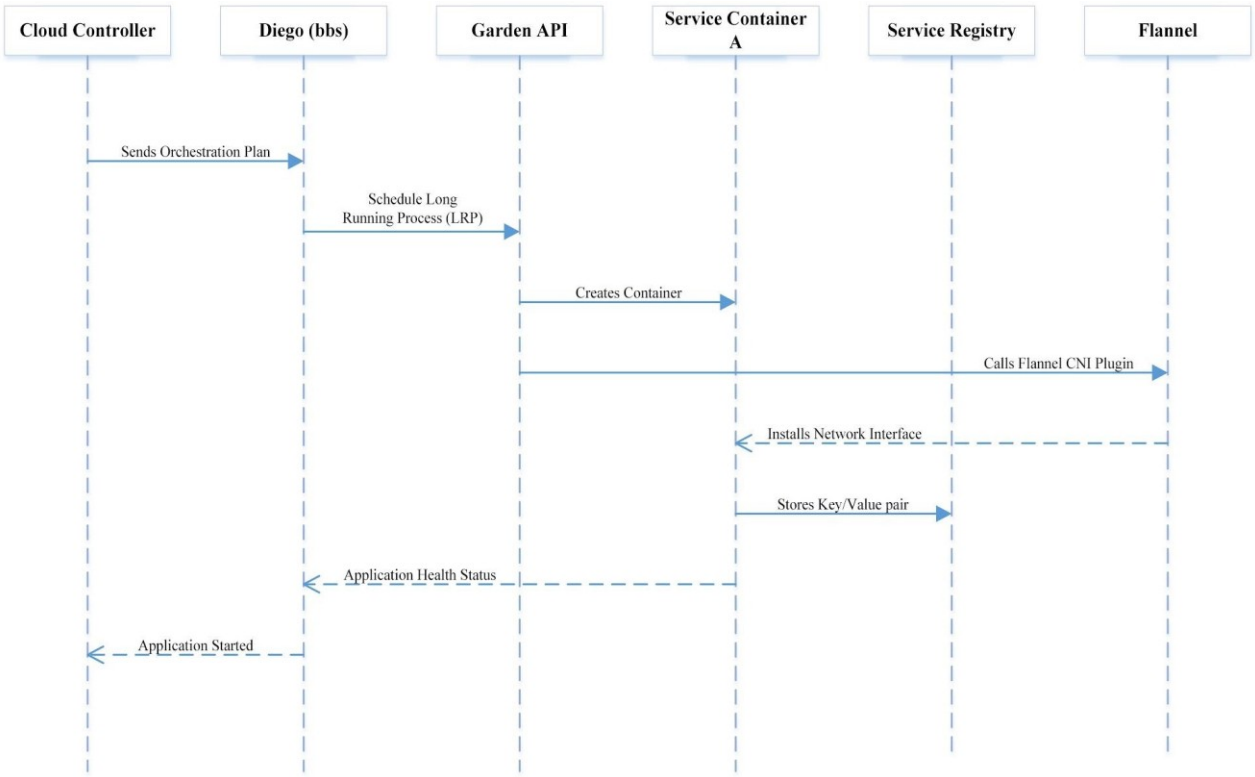

Figure 5.3 Interaction of components during Management Phase

5.2 Proof of Concept Prototype and Performance Results

In this section, a prototype of the proposed architecture for wildfire monitoring use-case scenario discussed in Chapter 3 has been implemented. The wildfire management application is a concurrency-based application and has been provisioned on the proposed cloud foundry architecture. To show the effectiveness of the proposed architecture, the performance measurements have been taken. In this section, an overview is given, and then, a detailed

discussion follows. The section ends with the experimentations made by considering different scenarios of wildfire management application.

5.2.1. Overview of the proof of concept prototype

Figure 5.4 shows wildfire management application interface offered to the city admin. For simulating the fire notification, we have considered main highlighted locations of Montreal city map. These three locations represent processes where fire monitoring processes are running. When the city admin clicks on these locations, it shows the current environmental condition, i.e., the current temperature and the location coordinates of the sensor node. In this scenario, the city admin is interested in the early detection and real-time view of the fire, i.e., location and direction of the fire.

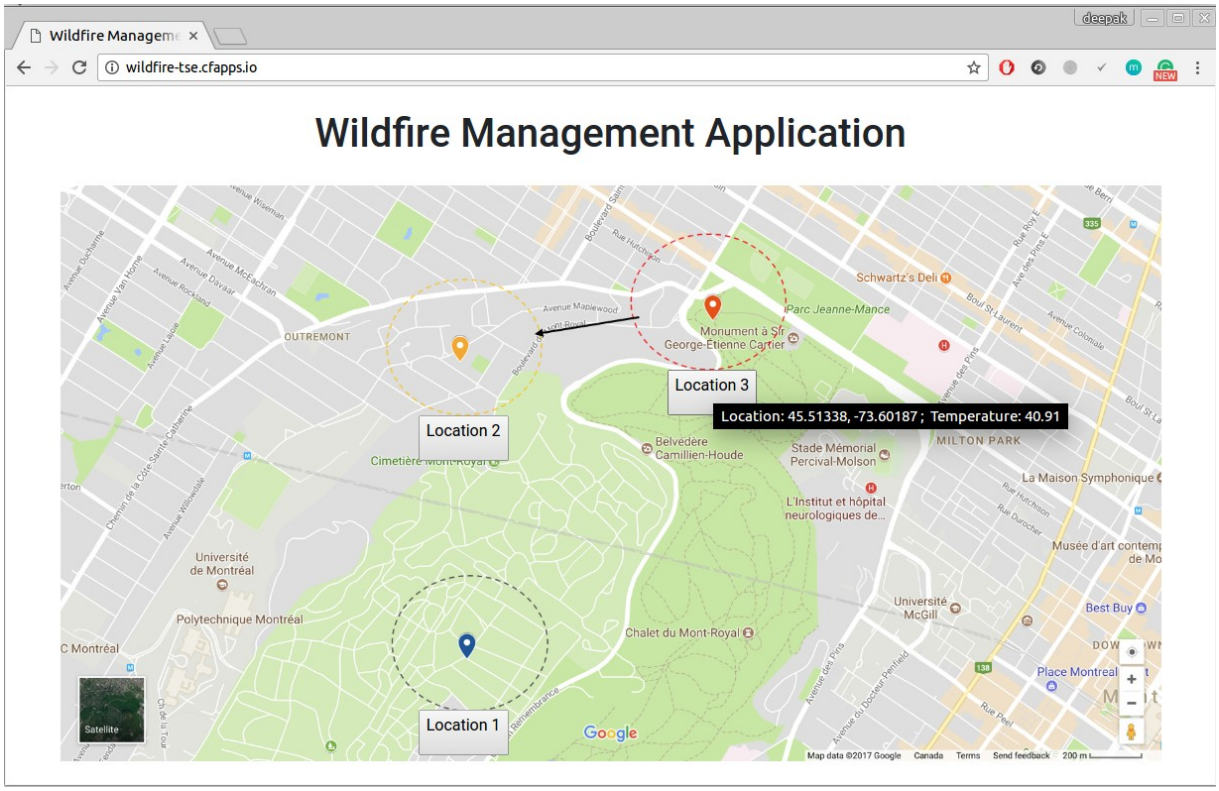

Figure 5.4: Wildfire Management application end user interface

The sensors have been deployed in the different areas and the fire monitoring process of wildfire management application running on the of each sensor node to monitor the environment condition. The wildfire management application uses the circle-based model of the fire contour algorithm. As a subset of Fire contour algorithm, we have developed the interaction part between the processes running on the sensor nodes. To provide the real-time view, we have taken following assumptions. First, we assume that the fire monitoring process is already deployed in the individual sensor node. The second assumption is that the sensor nodes are aware of their location. Third, we are assuming the coverage of sensor node in case of fire. We used the three advanticsys sensor nodes, and each node is monitoring three different areas using the fire monitoring process concurrently. All the application processes are coded in the spring framework MVC application.

The processes running on different sensor nodes use the RESTful web service to notify about its current situation (URI: HTTP::///WildfireManagementApplication/event/fire). Each sensor node builds and maintains its approximation to the wildfire by using the information it listens from the network as a fire spreads.

In this scenario, we are considering the circular area around each sensor node. In case of the wildfire, the composition of a set of circles presents the shape of entire shape by mean of fire. For instance, if there is fire at location 2, the process monitoring location 2 disseminates the information about the fire event and sensor coordinates to the all other processes in the network. Based on the shortest Euclidean distance, it makes the connection to its direct neighbor. The neighbor can change the behavior of its sensor nodes based on the interaction with the affected sensor nodes. In the simplest scenario, it updates the local approximation to the wildfire accordingly. In this way, the processes running on the sensor nodes able to provide the real-time view of the fire.

5.2.2 Software Tools

i. BOSH: Bosh [23] creates and deploys virtual machines for the cloud applications and provides the facility of services on the top of the targeted computing infrastructure using the YAML deployment manifest. It is an open source for release engineering, deployment, and life-cycle management. It covers the entire set of principles for the release engineering, i.e., identifiability, consistency, reproducibility, and agility. It makes the structured platform to the cloud foundry. It also provides simple flexibility for the operators for the deployment and management of cloud foundry. Bosh plays a vital role by making a base for the cloud foundry

platform. This base enables the monitoring, software updates and failure recovery with zero-to-minimal downtime for the platform deployment. These features of release engineering reduce the burden from the operators and make developers away from unnecessary risks. Bosh does the configuration of infrastructure (IaaS, Physical Servers) through the code. Bosh makes the abstraction of infrastructure into the generalized platform for the deployment and provides the benefits of physical server being rationalize as far as possible.

i.

ii. TinyOS (IoT Sensor Gateway): TinyOS [38] is an open source operating system designed for low-power IoT devices, such as those used in wireless sensor networks, ubiquitous computing, personal area networks, smart buildings, and smart meters. The operating systems running on these nodes help in network establishment and data communication. TinyOS has a component-based, event-driven programming model. These components are the core building blocks of OS. nesC programming language is used to write these components. Two major types of components are module and configuration. The module is used for implementation of component interfaces, while the configuration is used to accumulate all elements together and connect all the interfaces of the components. A component consists of a set of tasks, events and command handlers. Commands are used to invoke the requests, and events are the notification messages sent to the caller.

iii. Advanticsys Telosb: Advanticsys [39] provides a broad range of wireless sensor devices to support the Internet of Things based on IEEE 802.15.4 standard. This advanticsys telosb is also a kind of sensor node that is used in the wireless sensor network to design various applications. It uses an open source domain such as TinyOS, Contiki that have been made to enable the experimentation for the research community. Telosb sensor node includes many fundamentals features for research such as USB programming capabilities, IEEE 802.15.4

radio with integrated antenna, a low powered MCU with elaborate memory and the sensor suite. It offers many leading features such as Zigbee RF transceiver, low current usage micro-controller unit, incorporated on-board antenna, as well as sensor suite including incorporated temperature, light and humidity sensor.

iv. Open-Source Cloud Foundry Software: Cloud Foundry [4] offers a portable, open source platform as a service software, developed by VMWare. The aim to promote data backup and recovery and accelerate the journey to cloud computing. Cloud Foundry, primarily written in ruby, a dynamic, general-purpose object-oriented language, provides many frameworks, languages, and ready to use services to the end users for implementing their web applications. It focuses on its distributed architecture, similarly to the OpenStack, to grant the broadest possible scalability, reliability, and elasticity. Cloud foundry follows some clear design guidelines, namely, loose coupling with event-driven and non-blocking interactions. In particular, Cloud foundry consists of five main components: the cloud controller, the router, the Diego, the garden API and a set of services. Other elements include the User Account and Authentication (UAA) server for PaaS customer authentication, the garden container to isolate applications developed on the top the PaaS, and stacks to provide a standard set of development tools and libraries.

v. Flannel: Flannel [40] is the open source Container Network Interface (CNI) plugin designed by Kubernetes. It mainly provides the configuration to the Layer 3 network and makes a VXLAN network (also known as overlay network) over the running container's environment. It consists of libraries and specification to configure network interfaces in garden containers. Flannel involves itself only when the resources have been allocated to the application processes and removes allocated network configuration when the container gets

deleted. This reason attracts that flannel CNI has a broad range of support and the specification is simple to implement.

vi. Netflix Eureka: Eureka [41] is a REST (Representational State Transfer) based service that provides the service registry and discovery server. Eureka service is initially used in the AWS cloud for locating services for load balancing and failover of middle-tier servers. An application can use service registry to discover the registered service dynamically. When an application client registers itself with the Service Registry by providing its meta-data, such as its host and port. The registry tracks on the regular heartbeat message from each service instance of registered application service. If an instance begins to fail to send the pulse consistently, the service registry removes the service instance from its registry.

vii: Spring Framework: Spring framework [42] is an open-source Java platform, released under the Apache 2.0 license in June 2003. The Spring framework provides a configuration and comprehensive programming model for Java-based applications on a different kind of deployment platform. A primary element of spring framework is infrastructural support at the application level. Spring mainly focuses on the examining of enterprise applications so that developers can focus on business logic at application-level, without unnecessary ties to specific deployment environments. Spring helps development teams everywhere to build simple, portable, fast and flexible JVM-based systems and applications. It is one of the most popular application development frameworks for to create high performing, easily testable, and reusable code. It is lightweight when it comes to size and transparency and targets to make J2EE development easier to use. Spring provides ready-made components for different layers in a java application, like Spring MVC, Spring ORM, Spring. So, it is straightforward to use and helps in rapid application development.

viii: Maven: Maven [43] is a repository to build the application artifact and various types of dependencies. It mainly includes two types: local and remote. Maven local repository keeps the all dependencies (library jars, plugin) of the project. When the maven builds run, then maven downloads all the dependency jars into the local repository automatically. It helps to avoid references to dependencies stored on the remote machine every time a project is built. Remote repositories refer to the type of repository that accessed by a variety of protocols such as HTTP:// and file://. These repositories usually set up by a third party to provide artifacts for downloading (for example, uk.maven.org house maven's central repository and repo.maven.apache.org).

ix: RESTful: REST [44] stands for REpresentational State Transfer, firstly introduced by Roy Fielding in the year 2000. REST is a web-based architecture standard and uses HTTP protocol for data communication. It revolves around resources where each component is a resource, and a resource is accessed by a common interface using HTTP standard methods. In REST, it uses the REST client-server architecture, where REST server provides the resource access, and the REST client uses and presents the resources. The resource is distinguished by URIs/Global IDs. REST uses various representations to represent a resource like text, JSON, and XML. JSON is now the most popular format being used in Web Services.

x: Deluge: Deluge [45] is a nesC module provided by TinyOS to remotely reprogramming of the network. It is a reliable and robust data dissemination protocol for propagating large binary objects to all nodes of Wireless Sensor Network (WSN). It is limited to the network-wide broadcasting of application binaries. Every sensor node in the WSN network run the same application image.

5.2.3 Prototype Architecture

Figure 5.5 shows the prototype architecture. There are major three domains. The first domain is the cloud domain. It can be accessed via a local area network. It includes the PaaS for a cloud environment and the IaaS that is needed to execute the components located in the cloud. The extended native cloud foundry based on the open source cloud foundry is used for the PaaS. It is a private instance that runs on top of a classic IaaS. The other details of the PaaS not shown in the Figure are discussed here. For the development phase, the used IDE is an eclipse. Eclipse provides an editor for developing the event-based coordination policies between processes of wildfire management application. Besides, the eureka library is added for the interaction between the processes of concurrency-based application, and it is based on Java client-server components. In addition, the eclipse cloud foundry plugin is used, which interacts with the controller API for application processes' deployment on the cloud.

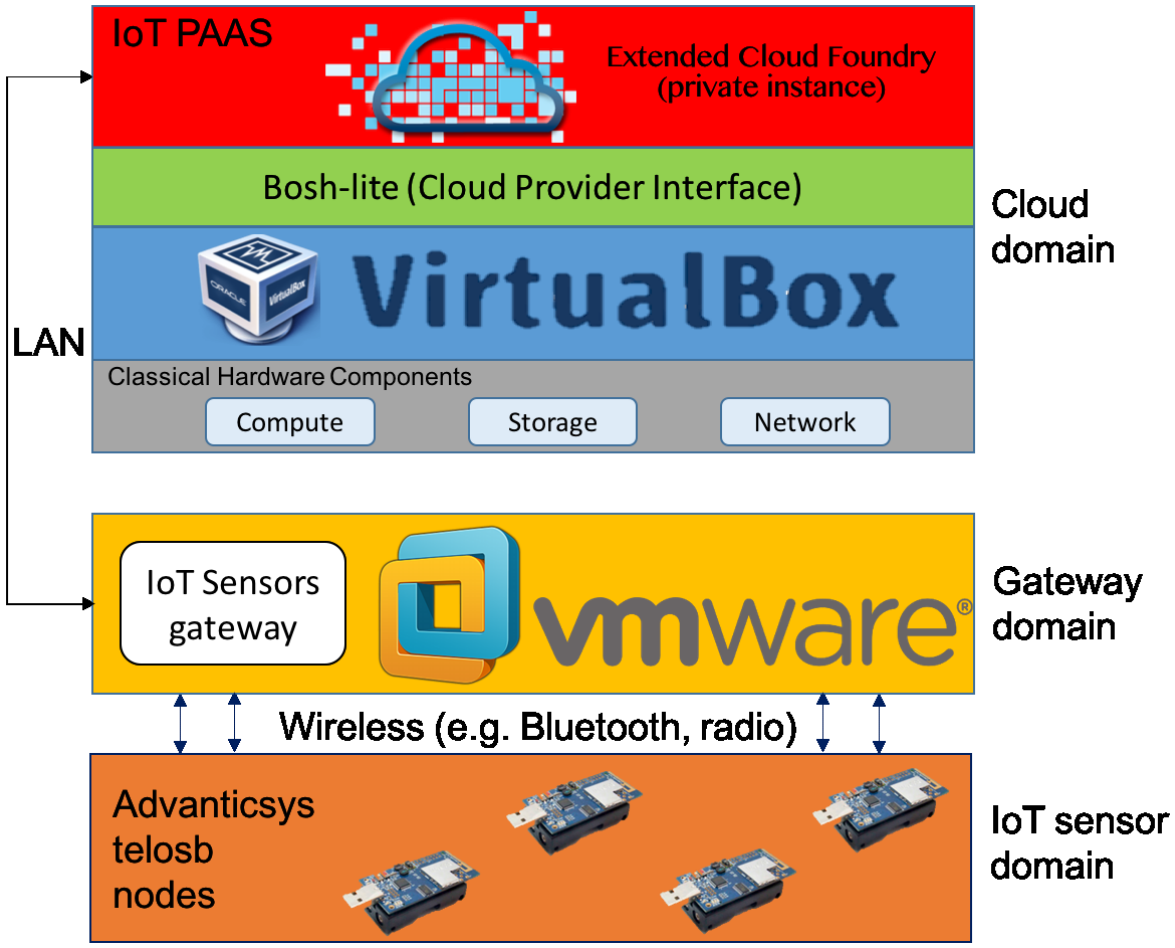

Figure 5.5: Overview of Prototype

For the deployment phase, eclipse cloud foundry plugin is used as a deployer, which interacts with the controller API for application processes' deployment on the cloud. The deployer provides the packaging of the components within the war file and deploys them in the extended cloud foundry system as droplets. The controller checks for the existing application in the cache and stores the metadata for the new application. The controller builds the corresponding orchestration plan to the Diego (see Figure 5.2). For the management phase, the Diego engine executes the orchestration plan to orchestrate the applications' components execution. The processes of wildfire management application are implemented as a Java tool and running on the isolated container environment. During the execution, the processes of wildfire management application monitor their area and share the status with each other.

The second domain is the gateway domain which acts as a middleware between cloud domain and IoT domain, as sensor nodes do not have an Internet compatibility. It is made up of the IoT gateway required for protocol and information model conversion environment needed for the execution of the process located in the cloud domain. The IoT gateway is deployed on the VMware virtual machine as shown in the Figure 5-5. It interacts with the IoT PaaS via LAN connection.

The third and last domain is the IoT domain that consists of the various wireless sensors used for sensing temperature (i.e., 4 Advanticsys wireless sensors). These sensor nodes able to provide to monitor various environmental magnitudes such as temperature, pressure, and humidity. It also consists of base-station node which gathers the all environmental data from all sensor nodes wirelessly. This base-station node is at the edge of the gateway domain, connected via USB. All the domains are located in TSE Concordia lab in Montreal.

5.3 Validation and Performance Evaluation

In this section, firstly it explains the performance metrics used for the experiments, and then it provides the experimental measurement results and analysis.

5.3.1 Performance Metrics

We evaluated the prototype performance using four metrics: temperature receiving delay, response time, re-tasking delay and the end-to-end delay.

I. **Temperature receiving delay:** Temperature receiving delay is the time taken by each sensor node to send the temperature readings to its respective fire monitoring process of wildfire management application. It is a time difference when sensor sends the HTTP post request and the time it receives a “200OK” response.

II. **Response time:** The response time is the communication delay between two processes monitoring different areas. It is the time taken to transmit a message from a process running on one sensor node to the process running on neighbor sensor node and to get a response "200OK". We have noted the response time for the wildfire incident scenario. During fire incident, one process running on a particular sensor node needs to share information with the processes running on the neighbor's sensor nodes.

III. **Re-tasking delay:** The re-tasking delay is the time taken by the neighboring sensor nodes to change its behavior of sensor node based on the interaction with the process monitoring affected location.

IV. **End-to-End delay:** The end-to-end is the time taken to present the real-time view of the fire to the city admin. We have noted the end-to-end delay with re-tasking and without re-tasking.

5.3.2 Performance Results

In this section, we have presented the experiment results performed using the wildfire management scenario.

5.3.2.1 Temperature Receiving Delay (TRD)

The temperature receiving delay is the time difference between when sensor node sends the temperature reading to its particular process and receive a success code (200OK). As the

sensor node has a constant sampling delay of 5 seconds, that it senses the temperature, so we are not considering the sampling delay in this graph. We are using three sensors, and the TRD is noted for all three sensor nodes. We have taken only 21 measurements for each sensor. We can see that delay for the first one would be higher than the various subsequent readings as shown in Figure 5.6. This high delay is because of the three-way handshaking of the TCP during the first HTTP post request (201 created). The average temperature receiving delay is 28 milliseconds (ms).

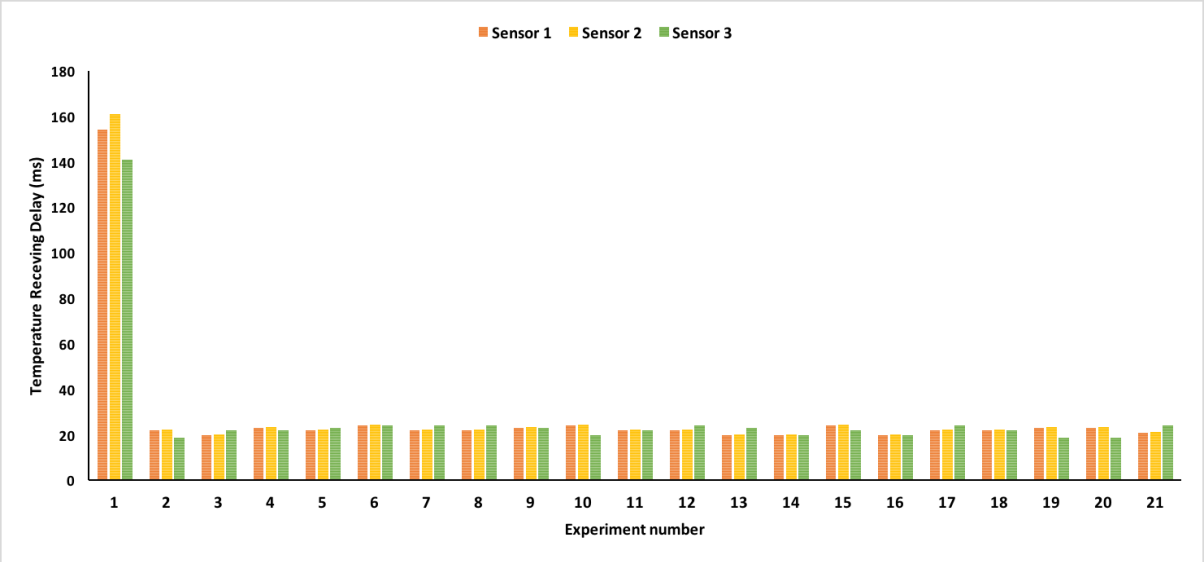

Figure 5.6 Temperature Receiving Delay (TRD)

5.3.2.1 Response Time

Figure 5.7 shows the response time proposed architecture. In total fifty measurements are taken, we observed that the average response time is 2.08 milliseconds. The main reason to get the lower response time is that the processes are communicating directly via peer to peer connection.

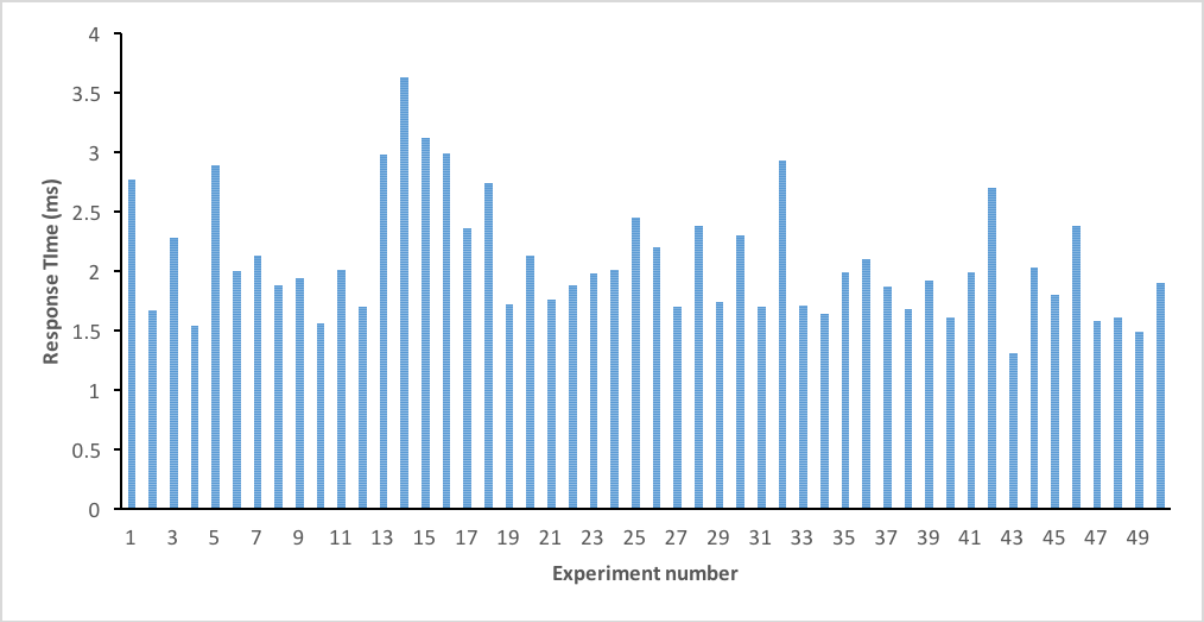

Figure 5.7: Response Time

5.3.2.3 Sensor Re-Tasking Delay

The sensor re-tasking delay is the time it takes to deploy the new program image on the neighbor sensor node. In the above use-case scenario, if fire event occurred at one location, it shares the information with its direct neighbor and the neighbor process can re-program its sensor node by reducing the sampling delay. Figure 5.8 shows the re-tasking delay of sensors. The average re-tasking delay for the process is 2928 milliseconds.

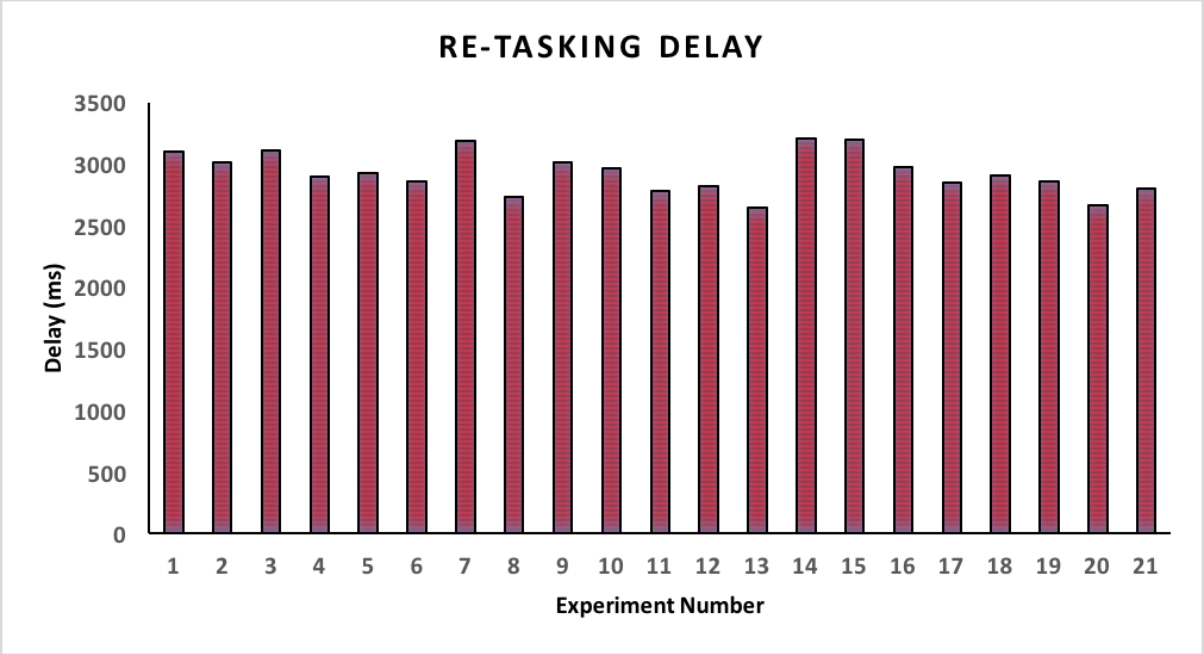

Figure 5.8: Sensor Re-Tasking Delay (SRD)

5.3.2.4 End-to-End Delay

We have two types of end-to-end delay, the first is related to the re-tasking of the neighbor sensor node and the second is related to without re-tasking of the neighbor sensor node. Figure 5.9 shows the end-to-end delay for receiving the temperature while communicating. For the re-tasking of the neighbor node, we measured the temperature receiving delay, response time, re-tasking delay and temperature receiving delay after re-tasking to provide the real-time view to the city admin. For the without re-tasking node, we measured the temperature receiving delay and response time. In without re-tasking scenario, the sampling rate remains same which leads to high end-to-end delay. However, with a re-tasking scenario, we are reducing the sampling rate to 0.5 seconds, which provide the better performance in end-to-end delay for wildfire management scenario as shown in Figure 5.9.

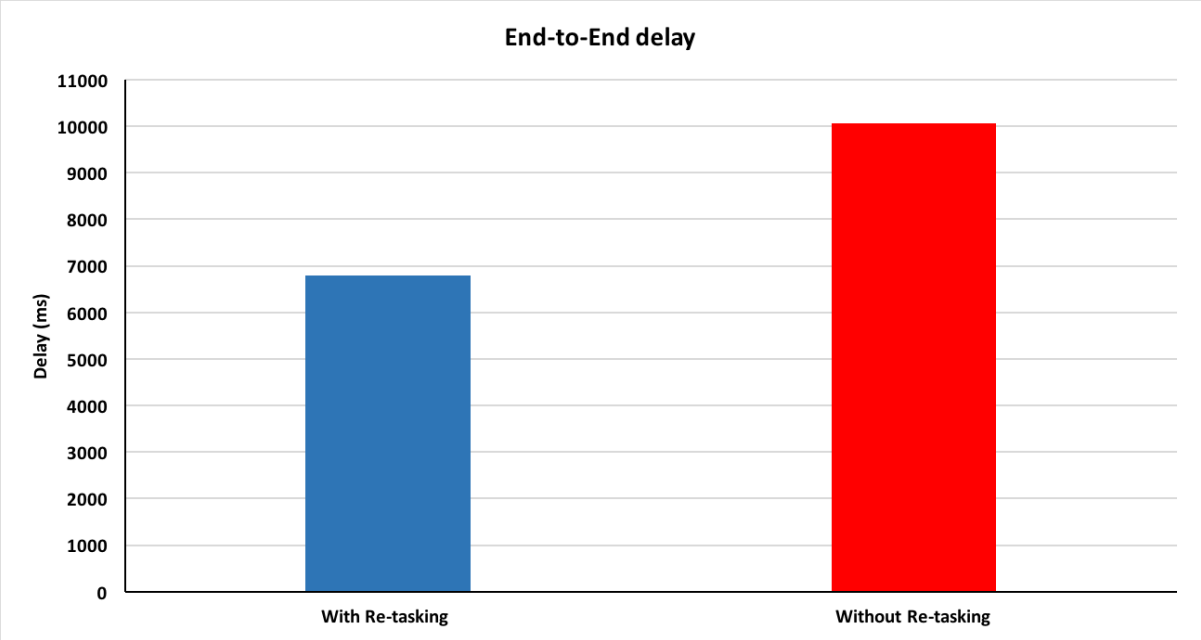

Figure 5.9: End-to-End Delay

5.4 Summary

This chapter first presented the software architecture of platform as a service for the provisioning of concurrency-based application in IoT. Then, two scenarios are presented showing the interaction between the software components. After that, a brief introduction of the software tools used during implementation is given, and the prototype architecture is defined. Finally, performance measurement results are given, and results are compared and analyzed for the proposed architecture.

Chapter 6

Conclusion and Future work

In this chapter, the overall contributions are summarized and research directions for future work are discussed.

6.1 Summary of Contribution

The concurrency-based application is an integral part of a wide range of IoT-based applications such as Pervasive Multi-player games and Wildfire Management application. However, the provisioning of concurrency-based application consists of phases such as development, deployment, and management which remain very difficult for the PaaS providers. One challenge is that PaaS providers need to know the intricate low-level details of technologies, protocols and their interactions. Another challenge is to enable inter-process communication to achieve optimal response.

This thesis proposed a novel business model, consists of the following roles: End-user, IoT SaaS provider, IoT PaaS provider, and IoT IaaS provider. The primary focus of this thesis is on the IoT PaaS provider role. It is assumed that IoT IaaS provider plays the role of IoT sensor devices and cloud infrastructure. From a motivating scenario for concurrency-based application provisioning in IoT, we have derived a set of requirements on the IoT PaaS and reviewed state of the art and evaluated them against the requirements. The findings suggest that none of them meet all our requirements.

To offer a solution to concurrency-based application provisioning issues, architecture has been proposed and designed by considering the previously discussed business model. This

thesis proposed a novel IoT PaaS architecture that facilitates solutions for concurrency-based application provisioning. The proposed architecture presents the various interfaces to provide the interactions between PaaS, and IaaS level. The overall architecture is designed by considering the identified requirements and few principles are set for the proposed architecture.

As part of this architecture, a high-level development phase for the application developers is proposed. Thus, the proposed architecture tackles the challenge to provide hosting of different processes in an isolated environment and enable process's edges to allow inter-process communication. Concurrency-based application developers can use newly added development phase in the proposed architecture to speed up development and to save cost. For another challenge to orchestrate the chain of processes defined during the development phase, we have provided the service discovery facility too.

A software architecture based on the proposed architecture of IoT PaaS is also designed. A prototype is implemented to validate the architecture. The implemented scenario includes an IoT PaaS provider offering concurrency-based application provisioning, a wildfire management application has been deployed on PaaS, and different processes of wildfire management application are running on different IoT IaaS. For performance evaluation, one critical scenario of wildfire of the concurrency-based application has been considered. Performance metrics such as temperature receiving delay, response time, re-tasking delay, and end-to-end delay are considered for the evaluation of the proposed architecture.

6.2 Future Work

The proposed IoT PaaS architecture explains the provisioning of various processes of the concurrency-based application. All processes of the concurrency-based application have

statically deployed on the IoT PaaS. There is a need to focus on dynamic PaaS resource allocation algorithms for efficient resource utilization and for meeting the growing demand for applications, as the load of these applications changes over time dynamically. These algorithms may enable vertical or horizontal scaling. For instance, when the re-tasking is done, the load can increase up to a certain limit. Based on the threshold, the processes of concurrency-based application should be scaled accordingly. Future work is to investigate the issues involved in designing suitable prediction models for different types of workloads.

Moreover, in future, the issues involved in the placement of different processes of the concurrency-based application on a cloud or at the edge based on a type of the application should be considered. These placement algorithms should consider the Quality of Service (QoS) parameters such as latency, and resource usage. For example, if the application is time critical, then some processes should be placed at the edge of the network to provide faster processing and less latency for better performance.

In the prototype architecture work, we have used HTTP protocol to receive the temperature readings from the IoT devices. Further, the work can be extended by using QUIC protocol. The main ability of QUIC protocol is to used as a platform for wide-scale experimentation with transport mechanisms, both at the server and at the client. QUIC combines the cryptographic and transport handshake into one round trip when setting up a secure transport connection.

Energy consumption is also one of the vital parameter to be considered for provisioning concurrency-based application in IoT domain. IoT-based cloud services can be used for real-time and dynamic collection of energy consumption-related data. Load balancing is an essential part of the development of concurrency-based applications. There is a need to design efficient load balancing algorithms for the provisioning of concurrency-based

applications. Future work is to investigate existing cloud resource allocation algorithms that can be used for the execution of concurrency-based application.

Bibliography

[1]. Cloud Computing "S. NIST, 800-145: The NIST definition of cloud computing" 2011.

[2]. Vaquero, Luis M., et al. "A break in the clouds: towards a cloud definition." *ACM SIGCOMM Computer Communication Review*39.1 (2008): 50-55.

[3]. Google, Inc.: Google App Engine. http://code.google.com/appengine/

[4]. Cloud Foundry Platform. http://www.cloudfoundry.org/

[5]. Atzori, Luigi, Antonio Iera, and Giacomo Morabito. "The internet of things: A survey." Computer networks 54, no. 15 (2010): 2787-2805.

[6]. Hasselbring, Wilhelm. "Programming languages and systems for prototyping concurrent applications." ACM Computing Surveys (CSUR) 32.1 (2000): 43-79.

[7]. Van-Roy, Peter, and Seif Haridi. *Concepts, techniques, and models of computer programming*. MIT press, 2004.

[8]. Ashton, Kevin. "That Internet of Things Thing. RFID Journal (2009)." *URL: http://www. rfidjournal. com/articles/view* 4986.

[9]. Gubbi, Jayavardhana, et al. "Internet of Things (IoT): A vision, architectural elements, and future directions." *Future generation computer systems* 29.7 (2013): 1645-1660.

[10]. Hiremath, Shivayogi, Geng Yang, and Kunal Mankodiya. "Wearable Internet of Things: Concept, architectural components and promises for person-centered

healthcare." *Wireless Mobile Communication and Healthcare (Mobihealth), 2014 EAI 4th International Conference on*. IEEE, 2014.

[11]. "The Internet of Things," McKinsey & Co., March 2010. http://www.mckinsey.com/insights/high_tech_telecoms_internet/the_internet_of_things

[12]. "Can the Internet of Everything Bring Back the High-Growth Economy?" Progressive Policy Institute, September 2013. http://www.progressivepolicy.org/wp-content/uploads/2013/09/09.2013-Mandel_Can-the-Internet-of-Everything-Bring-Back-the-High-Growth-Economy-1.pdf

[13]. A. Gluhak, S. Krco, M. Nati, D. Pfisterer, N. Mitton, T. Razafindralambo, A Survey on Facilities for Experimental Internet of Things Research, IEEE Communications Magazine 49 (2011) 58–67.

[14]. M. Darianian, M.P. Michael, Smart home mobile RFID-based Internet-of-Things systems and services, in: 2008 International Conference on Advanced Computer Theory and Engineering, 2008, pp. 116–120.

[15]. M. Yun, B. Yuxin, Research on the architecture and key technology of Internet of Things (IoT) applied on smart grid, Advances in Energy Engineering (ICAEE). (2010) 69–72.

[16]. E. Mancini, M. Rak, U. Villano, Perf Cloud: GRID services for performance-oriented development of cloud computing applications'', in Proceedings 18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE 09). (Groningen, The Netherlands, 2009), 201–206 (2009)

[17]. Buyya, Rajkumar, et al. "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility." Future Generation computer systems 25.6 (2009): 599-616.

[18]. Zhang, Qi, Lu Cheng, and Raouf Boutaba. "Cloud computing: state-of-the-art and research challenges." Journal of internet services and applications 1.1 (2010): 7-18.

[19]. Liang-Jie Zhang, Qun Zhou, "CCOA: Cloud Computing Open Architecture Web Services", ICWS 2009, pp. 607-616, 2009.

[20]. Cloud Computing Reference Architecture https://pdfs.semanticscholar.org/f273/edb973b7b20baa3b69608c067f585556e97e.pdf, 2014

[21]. Practical Guide to Platform as a Service by Claude Baudoin, Michael Behrendt "http://www.cloud-council.org/CSCC-Practical-Guide-to-PaaS.pdf" [September 2015]

[22]. Marinos, Alexandros, and Gerard Briscoe. "Community cloud computing." IEEE International Conference on Cloud Computing. Springer Berlin Heidelberg, 2009

[23]. "BOSH." [Online]. Available: https://bosh.io/docs/about.html.

[24]. Serna, M. Ángeles, Aurelio Bermudez, and Rafael Casado. "Circle-based approximation to forest fires with distributed wireless sensor networks." Wireless communications and networking conference (WCNC), 2013 IEEE. IEEE, 2013.

[25]. D. Bernstein, "Cloud Foundry Aims to Become the OpenStack of PaaS" in IEEE Cloud Computing, vol. 1, no. 2, pp. 57-60, July 2014. doi: 10.1109/MCC.2014.32

[26]. S. T. Graham and X. Liu, "Critical Evaluation on jClouds and Cloudify Abstract APIs against EC2, Azure and HP-Cloud" 2014 IEEE 38th International Computer

Software and Applications Conference Workshops, Vasteras, 2014, pp. 510-515. doi: 10.1109/COMPSACW.2014.85

[27]. A. Lomov, (2014) “OpenShift and Cloud Foundry PaaS: High-level Overview of Features and Architectures”, Available at www.altoros.com/openshift_and_cloud_foundry_paas.html

[28]. Bunch, Chris, Navraj Chohan & Chandra Krintz, (2011) “Appscale: open-source platform-as- a- service”, UCSB Technical Report

[29]. C. Vecchiola, X. Chu, and R. Buyya, “Aneka: a software platform for .NET-based cloud computing,” High Speed Large Scale Sci. Comput., vol. 18, pp. 267–295, 2009.

[30]. Li, Fei, Michael Vögler, Markus Claeßens, and Schahram Dustdar. “Efficient and scalable IoT service delivery on cloud” In Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on, pp. 740-747. IEEE, 2013.

[31]. Vaquero, Luis M., Luis Rodero-Merino, and Rajkumar Buyya. “Dynamically scaling applications in the cloud” ACM SIGCOMM Computer Communication Review 41, no. 1(2011): 45-52.

[32]. R. Girau, S. Martis and L. Atzori, “Lysis: A Platform for IoT Distributed Applications Over Socially Connected Objects” in IEEE Internet of Things Journal, vol. 4, no. 1, pp. 40-51, Feb. 2017. doi: 10.1109/JIOT.2016.2616022

[33]. M. Fazio and A. Puliafito, “Cloud4sens: a cloud-based architecture for sensor controlling and monitoring” in IEEE Communications Magazine, vol. 53, no. 3, pp. 41-47, March 2015. doi: 10.1109/MCOM.2015.7060517

[34]. Celesti, Antonio, Nicola Peditto, Fabio Verboso, Massimo Villari, and Antonio Puliafito. “Draco paas: a distributed resilient adaptable cloud oriented platform”

In Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International, pp. 1490-1497. IEEE, 2013.

[35]. Vögler, Michael, Johannes M. Schleicher, Christian Inzinger, and Schahram Dustdar. “A scalable framework for provisioning large-scale IoT deployments” ACM Transactions on Internet Technology (TOIT) 16, no. 2 (2016)

[36]. S. Abdelwahab, B. Hamdaoui, M. Guizani and A. Rayes, “Enabling Smart Cloud Services Through Remote Sensing: An Internet of Everything Enabler” in IEEE Internet of Things Journal, vol. 1, no. 3, pp. 276-288, June 2014. doi: 10.1109/JIOT.2014.2325071

[37]. J. K. Zao et al., “Augmented Brain Computer Interaction Based on Fog Computing and Linked Data” 2014 International Conference on Intelligent Environments, Shanghai, 2014, pp. 374-377. doi: 10.1109/IE.2014.54

[38]. Levis, Philip, et al. "TinyOS: An operating system for sensor networks." Ambient intelligence 35 (2005): 115-148.

[39]. “Advanticsys Telosb” [Online] Available: https://telosbsensors.wordpress.com [Accessed: 15-Sept.-2017]

[40]. Flannel [Online] Available: https://github.com/coreos/flannel [Accessed: 15-Sept.-2017]

[41]. Netflix Eureka [Online] Available: https://github.com/Netflix/eureka/wiki [Accessed: 15-Sept.-2017]

[42]. Spring Framework [Online] Available: https://spring.io [Accessed: 15-Sept.-2017]

[43]. Maven [Online] Available: https://maven.apache.org [Accessed: 15-Sept.-2017]

[44]. Li, Hongjun. "RESTful Web service frameworks in Java." signal processing, communications and computing (ICSPCC), 2011 IEEE International Conference on. IEEE, 2011.

[45]. Deluge [Online] Available: http://tinyos.stanford.edu/tinyos-wiki/index.php/Deluge_T2 [Accessed: 15-Sept.-2017]