MODELING HETEROGENEOUS INTERNET OF THINGS SYSTEMS USING CONNECTORS IN COMPONENT ORIENTED SOFTWARE ENGINEERING

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

SELIN ÜNAL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

JULY 2019

Approval of the thesis:

MODELING HETEROGENEOUS INTERNET OF THINGS SYSTEMS USING CONNECTORS IN COMPONENT ORIENTED SOFTWARE ENGINEERING

submitted by **SELIN ÜNAL** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar Dean, Graduate School of **Natural and Applied Sciences**	______________
Prof. Dr. Halit Oğuztüzün Head of Department, **Computer Engineering**	______________
Prof. Dr. Ali H. Doğru Supervisor, **Computer Engineering, METU**	______________

Examining Committee Members:

Assist. Prof. Dr. Gül Tokdemir Computer Engineering, Cankaya University	______________
Prof. Dr. Ali H. Dogru Computer Engineering, METU	______________
Assist. Prof. Dr. Pelin Angın Computer Engineering, METU	______________

Date:

ABSTRACT

MODELING HETEROGENEOUS INTERNET OF THINGS SYSTEMS USING CONNECTORS IN COMPONENT ORIENTED SOFTWARE ENGINEERING

Ünal, Selin
M.S., Department of Computer Engineering
Supervisor: Prof. Dr. Ali H. Doğru

July 2019, 72 pages

In this thesis a solution for modeling heterogeneous IoT applications in component oriented software engineering is provided by using software connectors. IoT is interconnected devices or humans in the means of internet which gains more importance day by day in different areas of the world. This kind of powerful and complex systems have challenges to overcome in nature. Each IoT system component has specific set of rules for communicating with the other components. In order to be able to communicate, components need to understand each other. If components are using different sets of rules for communication, these components can not understand each other, which causes the heterogeneity problem in IoT. Component oriented systems arose from the reuse paradigm. These systems include components which represent reusable building blocks. Connectors are used for connecting reusable components in component oriented systems. In this thesis, each component represents the "thing" in IoT and each connector represent a converter that connects components with different protocols for communication. By using COSECASE, we are showing that connectors offer a practical solution for the heterogeneity problem for modeling IoT systems.

Keywords: Internet Of Things, Heterogeneity, Connectors, Component Based Systems, Component Oriented Software Engineering, Component Oriented Software Engineering Modeling Language, Component Oriented Software Engineering Modelinag Tool

ÖZ

BİLEŞEN YÖNELİMLİ YAZILIM MÜHENDİSLİĞİNDE HETEROJEN NESNELERİN İNTERNETİ SİSTEMLERİNİN BAĞLAYICILAR KULLANILARAK MODELLENMESİ

Ünal, Selin
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Ali H. Doğru

Temmuz 2019 , 72 sayfa

Bu tezde bileşen yönelimli yazılım mühendisliği alanında, heterojenlik gösteren Nesnelerin İnterneti sistemlerini modellemek için bağlayıcılar kullanılarak bir çözüm sunulmaktadır. Nesnelerin İnterneti araçların veya insanların internet aracılığıyla birbirlerine bağlı oldukları ve günden güne dünyanın farklı alanlarında önem kazanan bir yaklaşımdır. Bu tarz güçlü ve karmaşık sistemler doğası gereği çözülmeyi bekleyen problemleri de beraberinde getirmektedir. Nesnelerin İnternetindeki her bir bileşen bir diğer bileşen ile iletişim kurabilmek için belirli kuralları işletmektedir. İletişimi gerçekleştirebilmek için bileşenlerin birbirlerini anlaması gerekmektedir. Eğer bir bileşen iletişim için bir diğer bileşenden farklı kuralları işletiyor ise, bu iki bileşen birbirlerini anlayamaz ve sonuç olarak iletişim kuramazlar. İki parçanın iletişim kuraması Nesnelerin İnterneti'nde heterojenlik problemi demektir. Bileşen tabanlı sistemler yeniden kullanma paradigması ile ortaya çıkmıştır. Bu sistemler yeniden kullanılabilir bileşen bloklarını içerir. Bağlayıcılar bileşen yönelimli sistemlerde iki bileşeni bağlamak için kullanılan yapılardır. Bu tezde her bir bileşen Nesnelerin İnternetindeki her bir "nesneye", her bir bağlayıcı da birbirinden farklı kuralları işleten bileşenleri bağlayan bir çeviriciye karşılık gelmektedir. Bileşen Tabanlı Yazılım Mühendisliği Modelleme aracı, COSECASE, kullanılarak, bağlayıcılar aracılığıyla heterojen olan IoT sistemlerini modelleyebilmek icin kullanışlı bir çözüm önerisi getirilmiştir.

Anahtar Kelimeler: Nesnelerin İnterneti, Heterojenlik, Bağlayıcılar, Bileşen Tabanlı Sistemler, Bileşen Yönelimli Yazılım Mühendisliği, Bileşen Yönelimli Yazılım Mühendisliği Modelleme Dili, Bileşen Yöenlimli Yazılım Mühendisliği Geliştirme Aracı

To my lovely family and my beloved one, Barış

ACKNOWLEDGMENTS

I would first like to thank my supervisor Professor Ali H. Doğru for his kindness, helpful, wise and powerful personality. Whenever I need support and have question or in trouble he was always there. Without his motivation and contribution this study would not have been completed. Thank you very much for being my supervisor.

I would like to thank Anıl Çetinkaya and Assistant Proffessor Doctor ERTAN ONUR for their valuable comments and contributions for this thesis.

I would like to thank my beloved one, Barış Ulu for his endless support and being always kind to me.

I would like to thank my dear friend Kübra Bedelci for her stimulating and enjoyable friendship. You always make me happy throughout this work.

I would like to thank my dear friend Kader Belli for her friendship. Whenever I need help and want to share something she is always with me.

I would like to special thank my friends Kezban Başıbüyük, Tuba Kesten and Hilal Yiğit for their endless support throughout this journey.

Last but not least, I would like to thank my father Sedat Ünal, my mother Nazile Ünal and my lovely sister Sena Ünal for their love and support throughout my life. With you, I am the person who I am. Thank you being my family.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

IOT	Internet Of Things
CBSE	Component Based Software Engineering
COSE	Component Oriented Software Engineering
COSEML	Component Oriented Software Engineering Modeling Language
COSECASE	Component Oriented Modeling Tool
XCOSEML	Extended Component Oriented Software Engineering Modeling Language
BLE	Bluetooth Low Energy
Bluetooth BR/EDR	Bluetooth Basic Rate/Enhanced Data Rate
6LoWPAN	Low Power Wireless Personal Area Networks
WiFi	Wireless Fidelity
OSI	Open Systems Interconnection
WAN	Wide Area Network
WPAN	Wide Personal Area Network
WLAN	Wide Local Area Network
MQTT	Message Queuing Telemetry Transport
AMQP	Advanced Message Queuing Protocol
RPL	Routing Protocol For Low-Power and Lossy Network
XMPP	Extensible Messaging and Presence Protocol
3G	Third Generation Standards
4G	Fourth Generation Standards
5G	Fifth Generation Standards
PHY	Physical
MAC	Media Access Control

CHAPTER 1

INTRODUCTION

1.1 IoT

Internet of Things is the world of devices or sensors that can connect to the Internet with various communication technologies. The thing refers to anything that can connect to the Internet in the IoT world [2]. IoT technology is being used in many different areas in the world to increase the quality of life. Some of the example domains for the IoT world can be listed as smart city, smart parking, smart health, smart traffic, smart home and smart building [3]. The core idea of the IoT is to connect each connectable thing to each other and make possible to share data between each other. Connection may occur between thing to thing, thing to human or human to human [4]. Sharing data process can include different steps for different things in the IoT world, since each of the things have different kind of hardware equipment. Two things need to connect to each other to manage a data sharing process. After connection is successfully established, communication starts between the two things.

1.2 CBSE and COSE

Component Base Software Engineering (CBSE) is the approach that is reusing software building blocks, namely components. The main idea behind CBSE is reusing software components instead of implementing them from the beginning [5]. In CBSE, with an appropriate interface definition, components can be integrated to the system easily [6]. For managing connections to encapsulate connection details in the components, there is a mechanism that is called connectors [5]. Component Oriented

Software Engineering (COSE) is also an approach that arose from reusing software components. COSE mainly focuses on reusing rather than implementing building blocks if they do not exists [7]. When compared to CBSE, it considers only the component notion, from abstractions to implementation, where as CBSE could utilize Object Orientation for example, for its process. IoT things can be modeled as components and connectors can be referred as connecting blocks of the two components in both CBSE and COSE approaches.

1.3 IoT Heterogeneity Problem

The Internet plays very important role in today's world. It is being used in almost all areas of life such as education, health, communication, government, transportation, social life, gaming and so on. The number of connected devices to the Internet increases day by day. In 2020, there will be 50 billion devices connected to the Internet [8]. There are different kinds of protocols, network connectivity options and communication methods for the devices on the Internet [9]. For example, they can use different kinds of protocols such as ZigBee, BLE, Z-Wave and 6LoWPAN [4]. The main purpose of IoT is to make devices to communicate and share information with each other. Devices need to understand each other to start communication. If two devices have different communication methods, they can not understand each other and they can not start to talk. Not being able to communicate because of using different communication methods is the heterogeneity problem in IoT. IoT world faces heterogeneity problem since things have a wide variety of network connectivity options, communication methods and protocols.

1.4 Approach

In CBSE, for connecting two components, connectors are used [5]. Modeling IoT devices as components and interconnections as connectors, we are proposing a connector based solution for modeling IoT systems considering heterogeneity problems through component-oriented development. Based on the reuse paradigm, component and connector definitions are provided using a component-oriented modeling tool, COSECASE . Firstly, IoT network protocols are classified and their details are specified. After that, pairs of communication protocols are selected if they can be converted to each other. In this thesis, protocols are considered according to the OSI model [10] and converted at the application level. After that new connectors and components are added to the COSECASE tool to demonstrate heterogeneity problem is managed in this component-oriented modeling tool.

1.5 Contributions and Novelties

Connectors have been classified and defined with XCOSEML which is a text-based domain specific language supporting variability in component-oriented development paradigm [5], [11]. However, connectors do not have implementation details for conversion while connecting specific components. In this thesis, we are proposing a connector based solution for modeling IoT systems considering IoT heterogeneity problem by defining interconnection details. COSECASE is selected as an implementation tool. IoT devices are modeled as components and their interconnections are modeled as connectors which provide adaptation for communication protocols. In COSECASE, connectors already have definitions, but variability management has not been completed yet. Connector definition is enhanced and implementation for variability management for IoT heterogeneity is added to the COSECASE. Modeling IoT systems in COSECASE by providing necessary conversion for specific protocols at the appplication level is provided in this thesis.

1.6 Outline of the Thesis

In Chapter 2, background information is provided about IoT, CBSE, COSE, COSEML, and software connectors. In Chapter 3, problem statement and an example problem are provided firstly and after that, related work about IoT heterogeneity is covered. In Chapter 4, connectors for IoT Heterogeneity in COSECASE is explained in detail. In Chapter 5, the example problem that is provided in chapter 3 is modeled in COSECASE by using provided solution. In chapter 6, conclusion and future work are provided.

CHAPTER 2

BACKGROUND

In this chapter some background information is provided in detail. The covered topics are IoT, IoT heterogeneity problem, component based software engineering (CBSE), component oriented software engineering (COSE), component oriented software engineering modeling tool (COSEML) and software connectors.

2.1 IoT

In this section, an IoT overview is provided, IoT architecture is covered and the protocols that are commonly used in IoT technologies are grouped. After that challenges in IoT are explained. Finally, IoT heterogeneity problem is explained.

2.1.1 IoT Overview

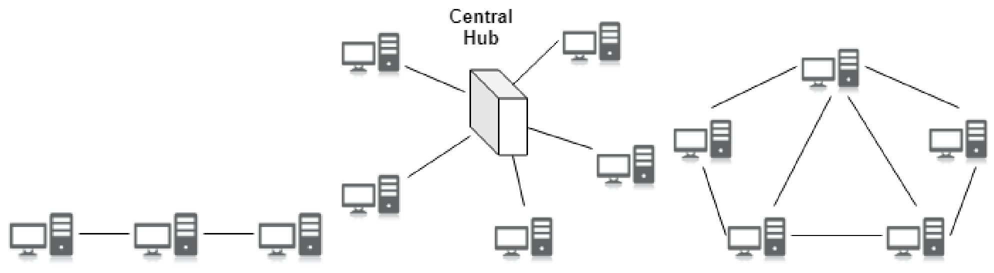
IoT is an infrastructure that connects things via wired or wireless networks and allows them to share information among each other [12]. Examples of IoT things include mobile phones, devices with sensors, computers with Wi-Fi, smart watches, smart door locks, smart lights and so on. Integrating these smart things to each other is challenging since IoT contains very complex heterogeneous networks [12]. Although the term Internet of Things is first proposed in the context of supply chain management domain, today IoT covers a lot of different kinds of application domains such as health, transport, utilities and education [13]. IoT mainly aims to interconnect the things to each other to allow them to communicate. In this way IoT systems increase the quality of life and help people by simplifying routine tasks. Nowadays, there are lots of IoT applications presented or being conducted. People can find their keys, unlock the doors, turn lights on and off, reserve park place from distance, send health conditions to their doctor only using a smart phone and even check trash fullness through a mobile application. As we can see from the examples, IoT solutions promise to ease people's lives in many ways.

2.1.2 IoT Architecture

IoT can consist of various subsystem architectures. In this section IoT architecture will be separated into management architectures and network architectures.

- *Management Architecture:* Generally IoT architecture is based on two main management types of architecture that are event-driven and time-based system architectures. In an event-driven architecture devices are triggered with an outside event and after being triggered, devices send data. For example a smoke sensor will be activated only when the smoke ratio exceeds a certain limit. In time-based architecture devices send data in certain periods of time. For example a temperature sensor will send data every two minutes [14].

- *Network Architecture:* IoT network architecture can be grouped under three types which are point-to-point connection, star and mesh. Point to point connection provides separate communication channels between two stations. Point to point connection's advantage is its simplicity. Disadvantage is depriving the chance to establish communication with device from outside the network. Star topology consists of one central hub and multiple terminal nodes. Each node can directly communicate only with the central hub. Connecting through the central hub all of the nodes can communicate with each other. Star topology's disadvantage is having a central hub. If this central hub goes down, system also goes down. Advantages are if one of the nodes goes down, system will stay awake. Mesh topology includes full mesh topology and partial mesh topology. In a full mesh topology, each node is connected to each other and can communicate with each other. In partial network topology only certain nodes are connected to each other and can communicate. Mesh network's advantage is that it can be constructed for wide areas and disadvantage is its complexity and high latency [14]. In Figure 2.1 point to point, star and mesh networks are shown together.



(a) Point to Point Network (b) Star Network (c) Mesh Network

Figure 2.1: IoT Network Architecture

2.1.3 IoT Protocols

Communication is the most important feature of the IoT systems. Thanks to networking technologies, IoT devices can communicate with each other as well as applications and services running on the cloud. Network protocols define sets of rules for connection and managing transmission of data across the network [15]. Foundations of network protocols are organized by the OSI model [16]. The OSI model represents communication systems in abstraction layers which are physical, data link, network, transport, session, presentation and application layers [17]. Table 2.1 shows the OSI model. An OSI layer generally communicates with three other OSI layers. An OSI layer can communicate with its upper layer, lower layer and peer layer in the network [18].

TCP/IP model is a simplified version of the OSI model. The TCP/IP layer does not have separate presentation and session layers, it has only the application layer. Additionally, TCP/IP has network access and physical layers together. Table 2.2 shows the TCP/IP model. We will use the TCP/IP model for abstraction of commonly used IoT protocols.

Brief description of TCP/IP layers is provided below [15]:

Table 2.1: OSI Model

Application Layer
Presentation Layer
Session Layer
Transport Layer
Network Layer
Data Link Layer
Physical Layer

Table 2.2: TCP/IP

Application Layer
Transport Layer
Internet Layer
Network Access & Physical Layer

- ***Network Access & Physical Layer :*** At the network layer, how each device is connected to the network with hardware is considered. A device can connect via an optic cable, wires or radio waves. At the link layer, devices are identified by their MAC address. Protocols are concerned with physical addressing at this layer such as how switches send frames to devices.

- ***Internet Layer:*** At this layer protocols define how routers deliver data packets between source and destination identified by IP addresses.

- ***Transport Layer:*** At this layer end-to-end communication is considered. Transport layer provides additional functionalities such as reliability, guaranteeing packets will be received in the order of they sent.

- ***Application Layer:*** This layer is responsible for application level messaging.

Commonly used IoT network protocols will be mapped into TCP/IP abstraction models.

- ***Network Access & Physical Layer IoT Network Technologies:*** IoT network access and physical layer protocols are commonly categorized into Wide Area Networks (WAN) and short range network [19]. WAN technology can be grouped into Low Power Wide Area Networks (LPWAN) and Cellular. In Figure 2.2 communication protocols are shown.

  LPWAN This technology is appropriate for low-power, long-range wireless communication. Examples can be given as SigFox, LoRa and NB-IOT.

  Cellular This technology provides low-power, low-cost IoT communication options by using existing cellular networks. Examples can be given as 3G and 4G.

  Short Range Networks This technology is operable in short distances. Examples can be given as BLE, Bluetooth BR/EDR, ZigBee, ZWave, NFC, RFID, WiFi and Ethernet.

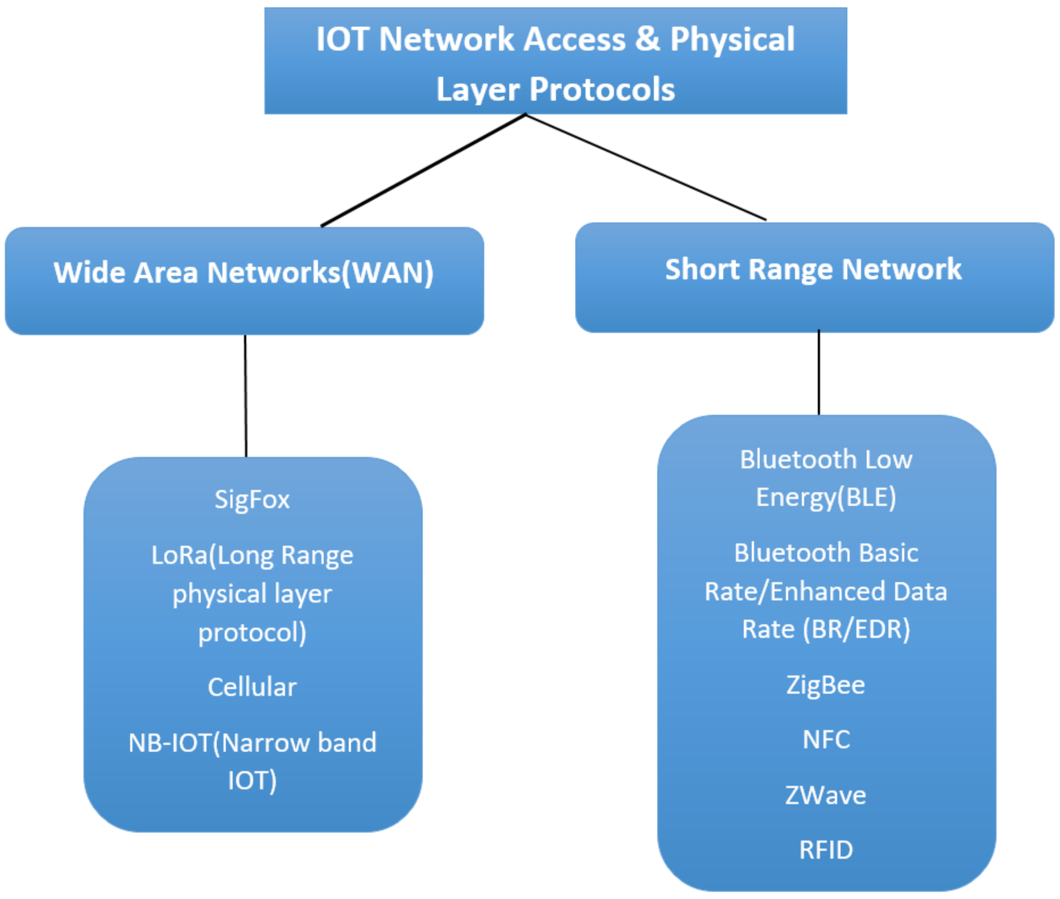

IOT Network Access & Physical Layer Protocols
Wide Area Networks(WAN)
Short Range Network
SigFox
LoRa(Long Range physical layer protocol)
Cellular
NB-IOT(Narrow band IOT)
Bluetooth Low Energy(BLE)
Bluetooth Basic Rate/Enhanced Data Rate (BR/EDR)
ZigBee
NFC
ZWave
RFID

Figure 2.2: Network Access & Physical Layer Protocols.

- ***Internet Layer Protocols:*** Generally used IoT Internet layer protocols are IPv6, 6LoWPAN and RPL.
- ***Application Layer Protocols:*** Generally used application layer protocols are MQTT, AMQP and XMPP.

2.1.4 IoT Challanges

According to [20], there are several challenges in the IoT world. These challenges can be classified as follows:

- ***Reliability:*** IoT systems should work according to their specifications. In IoT applications, systems should be highly reliable and data needs to be collected fast, and response needs to be given fast. Making wrong decision can be disastrous for some IoT applications for example emergency systems.
- ***Scalability:*** IoT applications need to be designed to enable extensible services and operations.
- ***Management:*** Providers should manage keeping track of failures, configuration and performance of the devices.
- ***Availability:*** IoT systems need to be available for subscribers of the system at any time and anywhere.
- ***Interoperability:*** Because of the large number of different kinds of devices and platforms it is challenging for IoT devices to work together.

2.1.5 IoT Heterogeneity Problem

In the IoT world, there are various kind of wired or wireless connected things as indicated in Section 2.1.3. These things include wide range of device types like low-power sensor devices and high-performance devices. Number of devices result in mixed network architectures in IoT systems [21]. IoT systems' main purpose is setting up smart environments. A smart environment means making devices connectable and communication between them starts automatically. Thank to data flow between entities, they can make decisions. According to the decisions taken, an entity can go different states or sends this information to another entity. IoT heterogeneity arose from these various kinds of sensors and devices.

2.2 Component Based and Component Oriented Software Engineering

2.2.1 Software Components

A software component is a reusable software building block that can be executed independently. To use a component, there is no need to know its implementation details. A component can be taught as a service provider. If a program needs a service, a component that provides needed service can be deployed into the program without worrying about where this component is being executed or without knowing the programming language in which this component is implemented. Components should be integrated to the system independently from other components. They are loosely coupled; when we change a component of a system, other components will not be affected from this change. A component is defined by its interfaces. Components can provide interfaces to other components or require interfaces from other components to be able to operate [22]. Since there can be components that provide or require similar services for operation, their interface definitions need to be done clearly and well documented. In this way user can select components properly for their needs. An example component visualisation is shown in Figure 2.3.

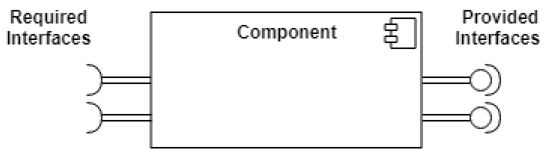


Figure 2.3: Component Interfaces.

2.2.2 Component Based Software Engineering

Component-based software engineering emerged in the late 1990s with the idea of reusing existing software components while building a software system rather than implementing each component from the scratch [5]. CBSE is an effective reuse based approach to define, implement and integrate loosely coupled and independent components into the system [22]. Although object oriented programming also arose from the idea of reusing, it does not provide extensive reuse since object classes are too detailed and specific which leads to the need to know source code details to understand class responsibilities. Software components are more abstract than objects and they are identified with interfaces therefore user does not need to know implementation details to use them. Therefore reusability capabilities of components are higher than objects. CBSE is an important approach since software systems are getting more complex and larger. Reusing existing components is important to develop better software systems more quickly and handle the complexity easier. There are several essential properties of CBSE. Firstly, independent components defined by its interfaces and their implementation details should be separated from their interface definitions. When an implementation of a component is changed the other parts of the system is not affected. The other essential property is component standards should define how components communicate and how their interfaces are used. In this way components written in a different language can be integrated to the system. The latter essential property is using middleware support while integrating components to each other to achieve component's communication. Middleware handles low level issues like resource allocation, transaction management and security. The last essential property is that available components should define their functionality clearly [22].

2.2.3 Component Oriented Software Engineering

Component Oriented Software Engineering is another approach for reusing existing software components rather than implementing them from scratch while building a software system. COSE was first introduced in 2003 as a new approach [7]. In CBSE while developing components, usually object oriented approaches are used. As opposed to CBSE, COSE highly depends on prebuilt components which increases the focus on reuse approach. The main difference between CBSE and COSE is the complete orientation towards the component concept. [7]. COSE transforms systems into two main groups of primitives that are a set of components and a set of connectors [1]. A set of connectors connect components for developing a target software system. In the COSE development process, firstly domain analysis is done and domain model is constructed. After considering system requirements and specifications, system is decomposed into components in an abstract level. After the decomposition step, abstract components are specified. Then, searching is conducted to find needed components for the system. After finding all defined components, integration step starts. By using software connectors, components are integrated and system model is obtained in COSE [23]. In Figure 2.4, life cycle of COSE development process is shown.

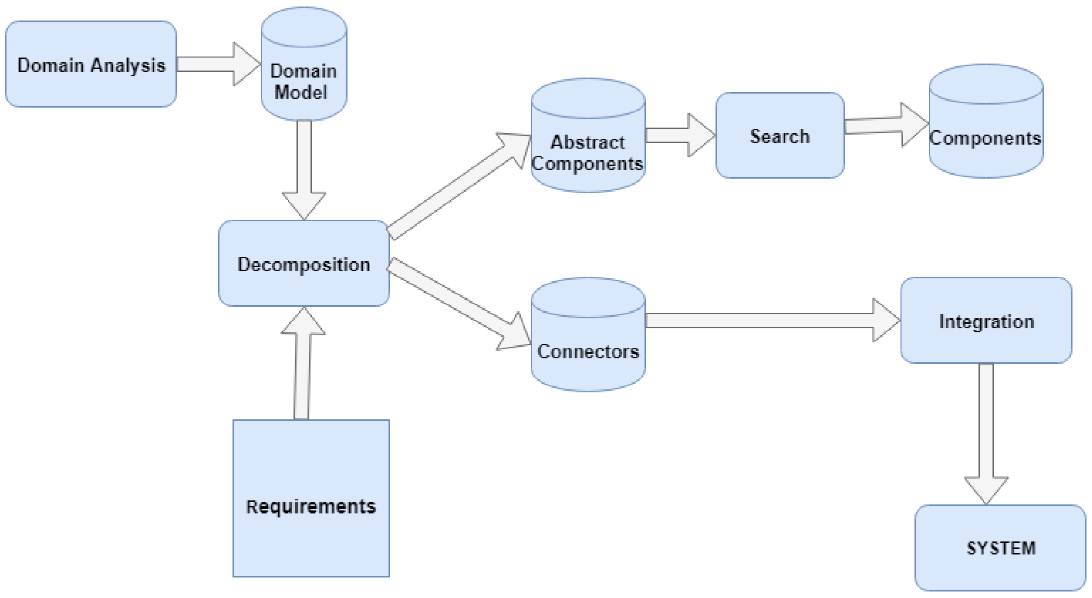
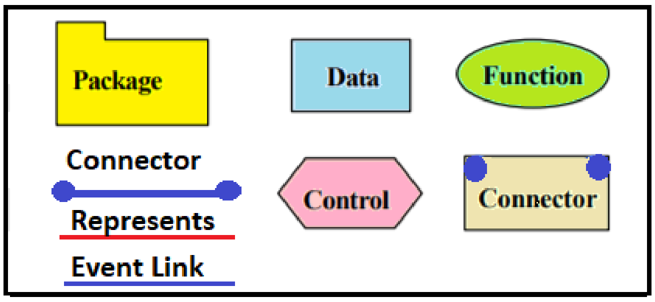


Figure 2.4: Component Oriented development process life cycle (Adapted from [23]).

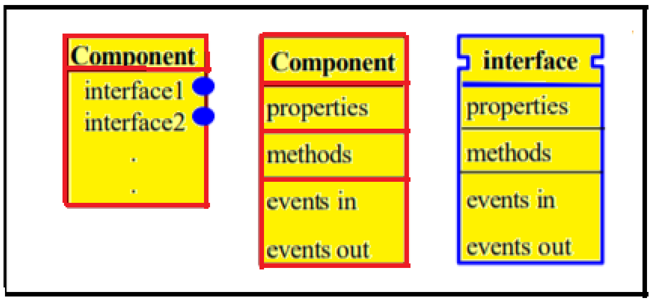
2.3 COSEML

A modeling language is any textual or graphical computer language that can be used to express information or systems in a structure via a set of consistent rules. Component Oriented Software Engineering Modeling Language (COSEML) is a modeling language that was developed to be used for COSE [1]. COSEML provides a way of developing software by composition with its own graphical representation of components with their connections. COSE development process starts with the abstract definition of the system parts as stated earlier. Later, for implementing the responsibilities of the abstract modules, physical components need to be introduced. Relations among the abstractions or physical components are modeled with connectors. There are abstract components, physical components and connectors in COSEML. Package, data, function and control abstractions are abstract components in COSEML. Components are the main units of physical components. Interfaces are provided for components. A component can have more than one interface. Connectors are also represented by a specific symbol. In Figure 2.5, abstract and physical components of COSEML are shown. One COSEML model is able to represent the complete model by using both abstract and physical components and connections [23]. Each COSEML entity is briefly described below.

- ***Package:*** It is used for organizing part-whole relations and it can contain further package, data, function or control elements.
- ***Function:*** It represents a system level function and it can contain further function and data.
- ***Data:*** It represents a system level entity and it can contain further function and data. It has internal operations.
- ***Control:*** It corresponds to a state machine in a package and it manages event traffic at the package boundary.
- ***Connector:*** It represents control and function flows across the system modules and it can be inserted between two modules.
- ***Component:*** It corresponds to an existing implemented components and it can contain one or more interfaces. It can contain other components.
- ***Interface:*** It is connection point of a component and services requested from the component have to be invoked through the interface.
- ***Represents:*** It indicates that a component will implement an abstraction.
- ***Event Link:*** It is used to link output event of one interface and input event of the other interface.
- ***Method Link:*** It connects a requester to the provider for a method.



(a) Abstract Components in COSEML



(b) Physical Components in COSEML

Figure 2.5: Graphical symbols in the COSEML (adapted from [1])

2.4 Software Connectors

Today's modern systems contain very complex components. It is important to properly integrate components in a system. It is also important to ensure that the communication between these components is properly maintained. The connection between components is performed by software connectors. Software connectors perform transfer of data and communication among components and they can also provide services such as messaging and transaction that are different from managing components' interconnection [24].

Software connectors are abstractions of the components interconnections in an architectural level. Connector abstractions can be symbolized as lines and boxes depending on the desired detail level. These lines can not fully represent the identity or properties of a connector. These connectors are also available for only managing interactions between components. Since systems are getting larger and more complex, connectors also need to evolve and adopt these changes. Connectors need to have their own identity, properties and their own executable code. They also need to gain capability of working with many different types of components. As systems are getting more complex and harder to manage, connectors have gained necessary properties. Connectors are usually defined independently from the application. However, there may be specific components and connectors for a specific domains. For example connectors can be used for converters of components of IoT.

2.4.1 Classification of Services Provided by Connectors

There are four main types of interaction services provided by connectors which are communication, coordination, conversion and facilitation [25].

- ***Communication:*** Data transmission between components. Components pass messages, exchange data and communicate results of computations.
- ***Coordination:*** Transfer of control among components. Components interact by passing thread of execution such as function calls and method invocations.
- ***Conversion:*** Interactions of heterogeneous components. For example if components have different types of data formats, conversion connectors can be used.
- ***Facilitation:*** Facilitates and optimizes interactions of the components. If a system needs optimization about using resources such as load balancing and concurrency control, facilitation connectors can be used.

2.4.2 Classification of Connectors

Interaction services can be used for the categorization of connectors in a broad way which does not explain details. To be able to build new kind of connectors, model and analyze them, connectors are classified into different types based on the way in which they realize interaction services which are procedure call, event, data access, linkage, stream, arbitrator, adaptor, and distributor [25].

- ***Procedure Call:*** These connectors model the flow of control through different invocation techniques and transfer data among the interacting components through the use of parameters. Examples of procedure call connectors can be given as fork and join in Unix like environments and operating system calls.

- ***Event:*** These connectors support transfer of control among components. Components interact by passing thread of execution such as function calls and method invocations.

- ***Data Access:*** These connectors allow access to the component which stores data. In case of required data and provided data is in different formats, data access connectors may transform formats of the data. Examples of data access connectors can be given as query mechanisms such as SQL for database access and accessing information in repositories.

- ***Linkage:*** These connectors are used to tie the components together and hold them in this state. In this way they provide a communication channel for other system connectors.

- ***Stream:*** These connectors are used to transform large amount of data among the components. They are also used in client-server systems with data transfer protocols. Examples of stream connectors can be given as UNIX pipes, TCP/UDP communication sockets and client-server protocols.

- ***Arbitrator:*** These connectors are used to resolve conflicts and streamline system operations when components do not know the other components' states and needs. Multi threaded systems can be given as an example area for the arbitrator connectors' usage area.

- ***Adaptor:*** These connectors support interaction of the components when communication of these components is not designed to inter operate. Heterogeneous environments such as different programming languages or computing platforms can be given as example areas for adaptor connectors' usage area.

- ***Distributor:*** These connectors identify interaction paths of components. Distributed systems exchange data using distributor connectors. Examples of distributed connectors can be given as domain name service, routing and switching.

CHAPTER 3

PROBLEM STATEMENT, AN EXAMPLE PROBLEM AND RELATED WORK

In this section, the smart city IoT domain will be explained in detail and IoT heterogeneity problem is shown by modeling the smart city components with the help of a feature model. After that, some provided solutions to the IoT heterogeneity problem will be covered.

3.1 Smart City and Heterogeneity Problem

3.1.1 Smart City

In today's world, millions of people live in big cities and this number is growing day by day. Citizens face various problems in the cities for example polluted air, heavy traffic, parking, finding available charging stations [3] and consuming redundant energy. There are a lot of connected devices to the Internet for example TV, Internet box, smart alarms, smart clocks, lights, cameras, connected cars and many other smart devices in the cities [26]. IoT reveals new solutions to citizens, companies and public administrations by using variety of data produced by these connected devices. This approach finds a lot of application areas such as medical aids, home automation, energy management systems, traffic management systems, industrial automation, automotive and so on [27]. Smart city is a solution of IoT for cities by collecting data from the connected devices in the city, interpreting collected data via services and returning results to the related recipient. Both citizens and city administration profit from provided solutions for smart city by increasing the quality of life of the citizens and providing economical advantage by decreasing operational cost [27].

3.1.2 Smart City Application Areas

The core application areas of a smart city can be listed as [27] [28]:

- Smart Homes and Smart Buildings
- Smart Healthcare
- Smart Parking
- Smart Transportation
- Smart Traffic
- Smart Security
- Smart Environment
- Noise Monitoring
- Smart Lighting
- City Energy Consumption
- Waste Management

In the following, some of the selected application areas that have more importance will be overviewed in terms of IoT solutions.

Smart Homes and Smart Buildings: Smart home technology enables to automate the ability to control items in the house or in the buildings. With smart home and smart building technologies, one can turn on or turn off the lights or the TV, control water heaters and room temperature, automate pet feeding, lock the doors, open the doors, open the curtain, activate fire detection system via a single button on the mobile phone or with a simple voice command. The items that are given in the examples need to be surrounded with necessary sensors to be able to communicate. Smart home technology provides to automate daily routines and promises to ease peoples lives in the home.

Smart Healthcare: Smart health technology enables to increase quality of health and gives chance to live for people. With smart health technology, doctors can check patients' health status from distance and can give advice according to results. For example with a smart wristband, patient's blood pressure or heartbeat can be measured and measured data can be sent to the doctor. In this way treatment can be done more secure and faster [3].

Smart Parking: Smart parking technology enables citizens to find available parking slots while parking their cars. Instead of searching an empty parking slot for a long time. With the help of a mobile application available parking slots can be found. A camera with sensor can identify whether a car is parked or not on a specific parking slot. After the gathered information one can know whether a park slot is available or not [3].

Smart Environment: Smart environment technology enables to measure the quality of the air in the parks, crowded areas, or fitness trails [27]. With the help of the sensors that measure oxygen level, one can find an appropriate outdoor activity place. Additionally, one can be informed about the weather condition simultaneously.

Smart Lighting: Smart lighting technology enables to automate street lights according to existence of the people, weather conditions or the times of the day [27]. With the help of smart lighting technology too much energy can be saved.

Waste Management: Waste management technology enables to find empty trash for citizens and for the garbage trucks. By integrating sensors in the garbage containers emptiness level of the container can be measured. After the collected data is sent to the data center, necessary information can be forwarded to the people or the garbage truck. In this way time, money and energy can be saved.

As we can see from the given examples of the smart city application areas, citizens and the government can gain lots of time, save energy and money, increase quality of life and safety and make more livable cities.

3.1.3 Smart City Challenges

IoT solutions give a chance to manage and monitor devices remotely, analyze and give respond to the information collected form devices. In this way IoT solutions make

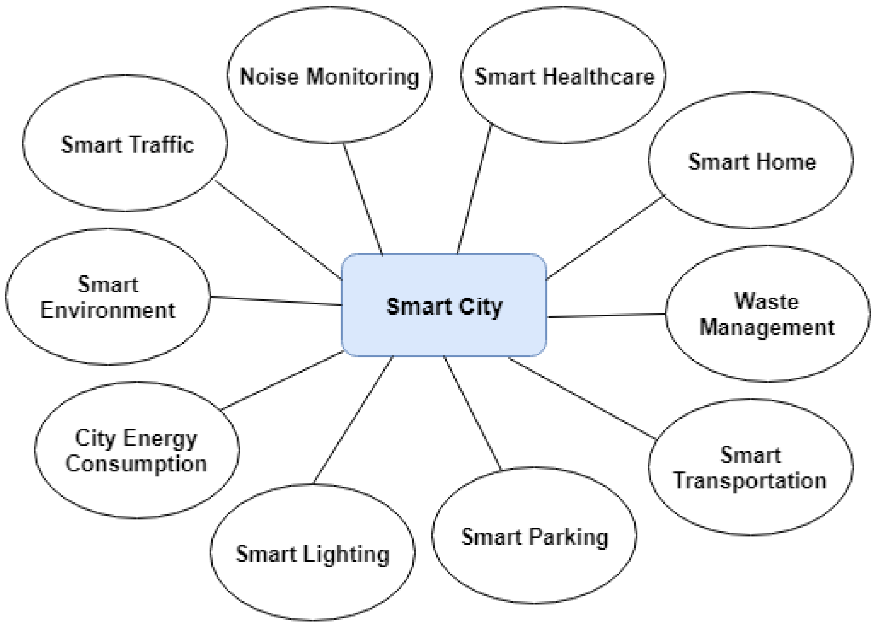


Figure 3.1: Smart City Components.

cities more comfortable and safer. There are challenges for IoT smart city technology to handle. Some of the challenges that are faced [26] are listed below:

- Reduce the cost and risks to produce IoT services.
- Connect various heterogeneous devices in the city.
- Decrease the time to integrate newly created IoT systems or services to the existing ones.
- Provide safe and secure systems to the city.

3.1.4 Modeling Smart City - Smart Parking Domain

In this section an example domain will be modeled using a feature diagram to identify smart city features in terms of IoT variability. Feature modeling helps to model domains as a set of features and identify the parts of the system [29]. In this work we do not aim to model the smart city domain completely. We only model a pilot study that considers a realistic domain part. Figure 3.1 shows the overview of the components of the smart city domain.

After specifying features of smart city, we develop a feature diagram to show the features of the smart city system. To show IoT variability smart parking system is selected and its features are provided in detail. Figure 3.2 shows the feature model for smart city and smart parking systems.

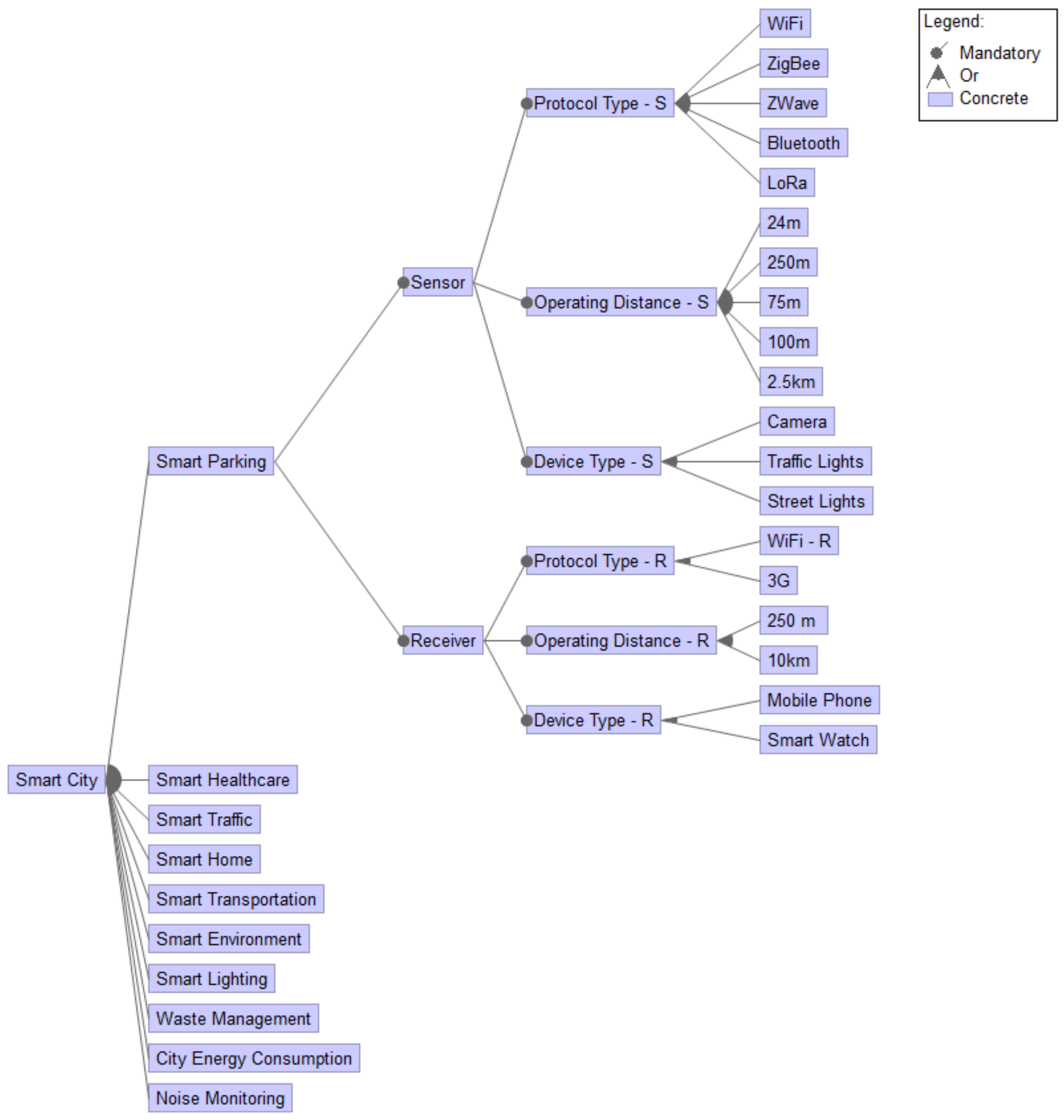


Figure 3.2: Smart City Feature Model.

Smart parking system has two main features. One of the main feature is Sensor which gathers data and the other one is Receiver which is informed via the gathered data. For the Sensor feature there are three different features defined that are Protocol Type - S, Operating Distance - S and Device Type - S (S for Sensor). Sensor has to have at least one feature from the Protocol Type - S, Operating Distance - S and Device Type

- S each. Protocol Type - S features are WiFi, ZigBee, Bluetooth, ZWave and LoRa. Operating Distance - S features are 24m, 250m, 75m, 100m and 2.5km. Device Type - S features are Camera, Traffic Lights and Street Lights. For Receiver feature there are three different features defined that are Protocol Type - R, Operating Distance - R and Device Type - R (R for Receiver). Receiver has to have at least one feature from the Protocol Type - R, Operating Distance - R and Device Type - R. Protocol Type - R features are WiFi - R (R for Receiver) and 3G. Operating Distance - R features are 250m and 10km. Device Type - R features are Mobile Phone and Smart Watch. There are constraints on the diagram that show that some of the features are only compatible with specific features. WiFi is operable in 250m, Bluetooth is operable in 100m, ZigBee is operable in 75m, ZWave is operable in 24m, LoRa is operable in 2.5km and 3G is operable in 10km [26].

3.1.5 Smart Parking Heterogeneity Problem

As indicated in Section 3.1.4, there are variable features in the smart city feature model. A valid example system from Figure 3.2 can be:

1. Smart City
2. Smart Parking
3. Sensor1 - Device Type - S - Camera, Protocol Type - S - ZigBee, Operating Distance - S - 75m
4. Receiver1 - Device Type - R - Traffic Lights, Protocol Type - R - Z-Wave, Operating Distance - R - 24m
5. Sensor2 - Device Type - S - Street Lights, Protocol Type - S - ZigBee, Operating Distance - S - 75m
6. Receiver2 - Device Type - R - Mobile Phone, Protocol Type - R - WiFi, Operating Distance - R - 250m

When we construct an example model from the given feature model diagram, we see that Sensor1 needs to talk with Receiver1 and Sensor2 needs to talk with Receiver2. However, Sensor1's protocol type is ZigBee, Receiver1's protocol type is

Z-Wave. Additionally, Sensor2's protocol type is ZigBee and Receiver2's protocol type is WiFi. These main features have different types of protocols, therefore they can not communicate with each other. This causes heterogeneity problem in the smart parking model.

3.2 Related Work

3.2.1 Ontology-Based Semantic Middleware Solution

According to [12], Wireless Network Sensors (WNS) exist for monitoring and following weather and drought changes. This WNSs consist of interconnected sensors which can sense and collect data and share information corresponding to weather and drought conditions. To measure drought condition changes these networks include different sensors. Even when these sensors measure the same property, they represent the sensed data differently. Additionally, communities use abstruse terms to represent and group events. These differences causes data heterogeneity which can be grouped as naming heterogeneity and cognitive heterogeneity. For example, water level is called 'Stav' in Check and 'Hoehoe' in German. These differences make it hard for seamless data sharing and full integration of interconnected heterogeneous devices. In this work an ontology-based semantic middleware solution is provided to eliminate data heterogeneity gathered from multiple sensors. An ontology is a formal description of the domain to share the artifacts that different applications can use [30]. Ontologies are expressed in a language that can be used by reasoning engines. They establish a formal vocabulary to share between applications. Semantic middleware maps data and send the mapped data to the necessary component. In short, provided semantic middleware solution will facilitate the integration of the heterogeneous sensor data.

3.2.2 Service Oriented Middleware for IoT

According to [31], there are number of challenges in IoT that can be enumerated as:

1. ***Scale:*** A lot of sensors and actuators exist on the network which make it hard

to manage and decide proper device.

2. ***Deep Heterogeneity:*** A lot of different kinds of hardware, protocols and data types exit on the network which cause heterogeneity problem.

3. ***Unknow Topology:*** Unknown and dynamic topology is another challenge for IoT. IoT networks can not know which things are located around their neighborhood that causes unknown topology problems in IoT.

4. ***Unknown Data Point Availability:*** If we want to collect some data from a specific place and this place does not have a sensor that measures desired data, this causes the unknown data point availability problem.

5. ***Incomplete or Inaccurate Metadata:*** Much of the metadata needs to be entered by a human. Since there exists a lot of things in the IoT world, incomplete or inaccurate metadata problem can occur.

In consideration of these problems, this survey provides a service oriented middleware which abstracts functionalities of things as services. With the representation of each thing as a service, interoperability and flexibility can be achieved. Additionally, since service oriented approach is used, this middleware provides loosely coupled and reusable services for IoT systems.

3.2.3 System Agnostic Ontology-Based Data Models

According to [32], IoT systems generate heterogeneous data streams which makes communication hard for devices. Corresponding heterogeneous data streams provide linked data technologies to provide interoperable data models that are based on existing ontologies. They present system agnostic ontology-based data models. The data models are used in a project called Virtualized programmable InTerfAces for innovvative cost-effective IoT depLoyments in smart cities (VITAL). VITAL is a system of systems. It can support any underlying IoT system. VITAL uses linked data standards for modeling and accessing data, JSON-LD as the data format and ontology for specifying the data. This work provides basis for the semantic data model for the VITAL project.

3.3 Difference of the Proposed Solution

In this thesis we are proposing a practical heterogeneity management solution for IoT system modeling in COSE without using middleware oriented or ontology based solutions. With the help of proposed IoT connectors, IoT system components can be integrated to a system easily. In the COSE world, this is the first provided heterogeneity management solution for modeling IoT systems in COSECASE. This research considers the future development of a framework where graphical modeling can lead to the composition of components and connectors as a working system. While solving the heterogeneity problem, a foundation for an integration platform is also proposed. The proposed connectors are easily configurable that makes them adaptable for variability management, that in turn facilitates a Software Product Line environment.

CHAPTER 4

IOT CONNECTORS IN COSEML

In this chapter implemented IoT connectors for COSEML are explained in detail. IoT connectors are used to achieve modelling IoT systems in COSE by providing a solution to IoT heterogeneity problem.

4.1 COSEML Connectors

COSEML connectors have visualization in the latest version, however they do not have implementation details in COSECASE. For modeling IoT systems in COSECASE, considering IoT heterogeneity problem, IoT connectors are added to the latest version of COSECASE. COSEML connector visual representation is shown in Figure 4.1.

Figure 4.1: COSEML Connector.

In this thesis, to be able to model IoT systems in COSE, COSEML IoT connectors are defined and implemented in COSECASE. With respect to proposed solution in [3], connectors connect two different components that have different communication protocols. Each component represents an IoT device. Connectors are assumed to have appropriate ports for each component that is connected to it. Connector ports handle network access and physical layer, internet layer, and transport layer conversions sequentially. Ports send core data to the connector software and connector handles received data accordingly. After connecting a component to the connector, it can receive related data packet from the first component. Since connector knows which components are connected to it, it can parse core data and sends prepared data to the second component appropriately. Since connector has second port that is connected to the second component, it can send related data packet to the second component. In this thesis, according to components' data packet structure, it is implemented that related data packets are prepared and sent to the second component properly. Figure 4.2 shows visual representation of proposed IoT connector for COSEML.

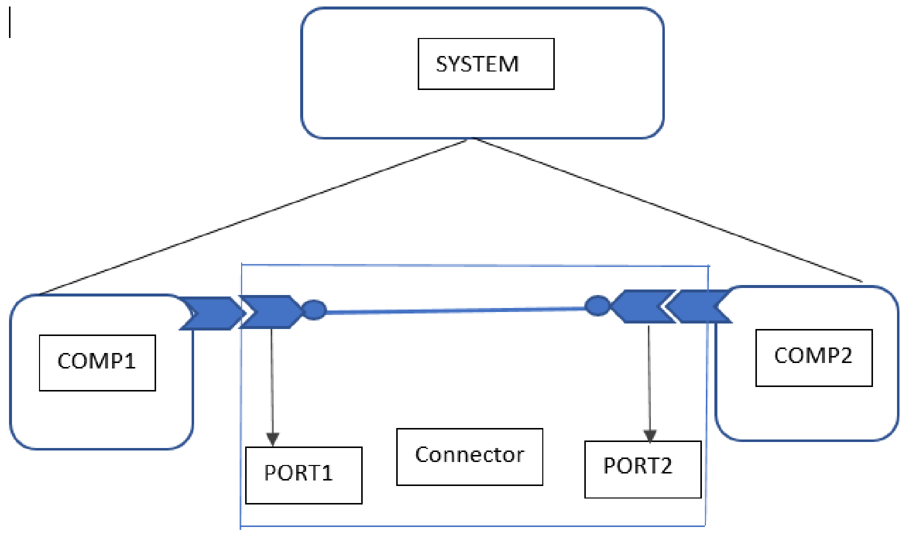


Figure 4.2: IoT Connector Visual Representation.

4.2 IoT Protocols Classification

There are various kind of IoT devices that are used in IoT systems. According to the systems' needs the most appropriate IoT device need to be selected. Main characteristics of IoT devices can be grouped as below [33]:

- Connectivity: How do IoT entities should be connected to each other? Are they close to each other or far away from each other? Are they connected to each other by using wireless or wired technologies?
- Power Management: How long should an IoT entity stay awake for actively

being operable?

- Data Processing and Storage: How does an IoT entity manage the data? How much data does it need to store?

Considering IoT device features, there are various kinds of protocols that are used in IoT systems. Some of the most appropriate and widely used IoT protocols are [34], [35], [36]:

- Bluetooth
- ZigBee
- Z-Wave
- IPv6 over Low Power Wireless Personal Area Net- work (6LoWPAN)
- Wireless Fidelity (Wi-Fi)
- Third-Generation (3G) and Fourth-Generation (4G) standards
- Fifth-Generation (5G)
- SigFox
- Thread
- WirelessHART
- Ethernet

Grouping protocols based on the TCP/IP model is done in section 2.1.3. In addition to grouping commonly used IoT protocols, they can also be grouped according to being wireless or wired, IP based or non-IP based and range properties. Tables 4.1 and 4.2 show grouped IoT protocols.

4.2.1 Wireless Communication Technologies

Wireless network protocols can be classified as WPAN, WLAN and WAN technologies.

Table 4.1: Wireless IoT Protocols

WIRELESS				
Short & Medium Range Protocols			Long Range Protocols(WAN)	
Non-IP Based WPAN	IP Based WPAN	WLAN	Cellular Connectivity	Others
Bluetooth ZigBee ZWave WirelessHART	6LoWPAN Thread	WiFi IEEE 802.11ac IEEE 802.11p IEEE 802.11ah	3G 4G 5G	LoRa SigFox NB-IOT

Table 4.2: Wired IoT Protocols

WIRED		
Short Range	Medium Range	Long Range
Serial Cable	IEEE 802.3 Ethernet	IEEE 802.3 Over Optical Fiber

- ***Wireless Personal Area Networks (WPAN):*** It is established for a user to exchange data in the 30 feet range using wireless technology [17]. Non-IP based technologies are more efficient than IP based solutions with respect to energy and cost.

- ***Wireless Local Area Network (WLAN):*** In limited geographical area this technology enables to communicate with radio wave technology [17].

- ***Wide Area Networks (WAN):*** It is a network technology that covers large geographical areas [17].

4.2.2 Wired Communication Technologies

In wired communication technologies data is transferred via cables between two devices. Routers and switches are also used to connect devices to each other.

4.3 Interoperability Of Protocols

Communication protocols have sets of rules that are different from each other. This makes connection of two different protocols impossible without a helper. For modeling IoT systems in COSE by using COSEML, connectors need to be implemented in such a way that they can be a solution to this heterogeneity problem. Before implementing connectors in COSEML, a set of protocol pairs are selected and their properties are identified. After that, their differences are specified and a possible solution is given to solve the adaptation problem between them. In this thesis, short and medium range protocols are selected because of their popularity and ease of access in IoT systems. According to the TCP/IP model, Network Access & Physical Layer protocols are selected since they are widely used IoT communication protocols.

In this thesis an IoT device will be represented as a COSEML component. While connecting two components in COSEML, a related COSEML connector will be used. For implementing a specific connector, firstly we select interoperable protocol pairs that are:

- Bluetooth BLE - Bluetooth BR/EDR
- ZigBee - ZWave
- ZigBee - WiFi

Overview of protocols is provided below [37], [16]:

- ***Bluetooth:*** Bluetooth is a low power wireless communication protocol that is used in various areas of technology for example mobile phone sensors, mouses and keyboards, health monitors and alarm systems. Bluetooth has three different modes in action that are:
  - ***Low Energy Mode(LE):*** It uses 2.4 GHz ISM band, operates at 1 Msym/s at a bit rate of 1 Mbps. It allows data rates of 125 Kbps, 500 Kbps, 1 Mbps and 2 Mbps.
  - ***Basic Rate/Enhanced Data Rate Mode(BR/EDR):*** It uses 2.4 GHz ISM band, operates at 1 Msym/s at a bit rate of 1 Mbps, 2 Mbps and 3 Mbps.

  - ***Alternative MAC/PHY (AMP):*** It uses 802.11 for high speed transport. It allows data rates up to 24 Mbps.

- ***ZigBee:*** It is based on IEEE 802.15.4 which uses 858 MHz, 915 MHz and 2.4 GHz ISM bands and allows data rates of 20 Kbps, 40 Kbps and 100 Kbps. ZigBee is targeted for commercial and residental IoT networking that is constrained by space, cost and power.

- ***Z-Wave:*** It uses 868 MHz, 908 MHz, 917 MHz and 919 MHz ISM bands. It allows data rates of 9.6 Kbps, 40 Kbps and 100 Kbps. It is mostly used for home automation technologies.

- ***WiFi:*** It uses 900 MHz ISM band. Allows data rates from 150 Kbps to 347 Mbps.

Selected protocol pairs are shown in Table 4.3 with their basic differences and possible solutions to these differences [16].

4.4 Connecting Components with IoT Connectors

After specifying protocol pairs that can be adapted through IoT connectors, data packet formats need to be specified for each protocol since an IoT connector will make conversion between components at the application level. Since IoT connectors will make conversion at application level from software perspective, detailed data packet information will be provided in this section. Ports on the connector will handle necessary physical layer connection. Selected protocol pairs will be explained in this section with their data packet formats in detail. First protocol pair is Bluetooth LE and Bluetooth BR/EDR. Second protocol pair is ZigBee and Z-Wave. The last protocol pair is ZigBee and WiFi.

4.4.1 Bluetooth LE Mode

The Bluetooth Low Energy (BLE) is commonly used in low power network and IoT applications that require low battery life devices [38]. BLE is widely used in portable

Table 4.3: Protocols Interoperability

Protocols	Different Features	Possible Solution
Bluetooth LE & Bluetooth BR/EDR	Data Rates Modulation Coding Data Packets	For difference in physical layer, there needs to be a dual-mode controller. Data formats should be converted appropriately by software, for different data packages.
ZigBee & Z-Wave	Frequency Addressing Data Rates Data Packets	For difference in physical layer, there needs to be a proper antenna to receive signal. Data formats should be converted appropriately by a software, for different data packages.
ZigBee & WiFi	Maximum Transmission Unit Size Fragmentation Data Packets	A bridge device can connect these two protocols. Data formats should be converted appropriately by software, for different data packages.

medical devices, smart phone accessories, remote controls, sports and fitness monitors. A single mode BLE is divided into three main parts that are controller, host and application. Each part has specific protocols to be able to operate properly. Application layer is the top-most layer responsible for containing user interface and data handling of actual use case that the application implements. Host layer is the middle layer containing protocols like Generic Access Profile (GAP), Generic Attribute Profile (GATT), Security Manager Protocol (SMP), Attribute Protocol (ATT) and Logical Link and Adaptation Protocol (L2CAP). Last layer is controller that includes Host Controller Interface (HCI), Link Layer (LL), and Physical Layer (PHY) [39]. There are two data packet formats for BLE. One of them is advertising packet that is used for initiating connections and finding available BLE devices. The other one is data packet that is sent after initiating a connection between two BLE devices. In this thesis we will mainly focus on BLE data packets. Detailed data packet for the BLE mode in bytes is shown below where numbers corresponds to bytes [40]:

Preamble: 1 - Access Address: 4 - PDU Header: 2 - L2 Header: 4 - Operation: 1 - Payload: 246 - MIC: 4 - CRC: 3

- ***Preamble:*** It is used for synchronization and timing estimation of the receiver.
- ***Access Address:*** It identifies a connection, makes sure the connection occurs uniquely.
- ***PDU Header2:*** It describes the packet type.
- ***L2 Header:*** It reassembly and fragmentation of packets that are longer in length than the allowed.
- ***Operation:*** It indicates the operation type such as write, notification, read.
- ***Payload:*** The meaningful data that is want to be sent.
- ***MIC:*** It is used for connection security, it is used with CRC.
- ***CRC:*** If packets pass CRC check they are expected to be reliably sent.

4.4.2 Bluetooth BR/EDR Mode

The Bluetooth BR/EDR mode known as classic bluetooth and it is commonly used to connect mobile phones to the headsets, mouse and keyboard. It is a connection-oriented protocol type. After connection is set up between two Bluetooth BR/EDR devices even if there is no data to sent the communication link is maintained. Before any data transmission is started, a device must be in its discoverable mode and when being found by a scanning device, it sends address and other necessary parameters [37]. Bluetooth BR/EDR protocol stack consists of upper layer an lower layer parts. Upper layer is called Host System and consists of RFCOMM/SDP and L2CAP components and Host Client Interface (HCI). Lower layer is called Bluetooth Controller and consists of Host Client Interface (HCI), Link Manager, Baseband and Radio components [39]. There are two data packet formats for Bluetooth BR/EDR. One of them is ACL, where packets hold user or control data used for asymmetric links. The other one is SCO packets that are used for speech transmission and transparent synchronous data. SCO packets are used for symmetric links. In this thesis, ACL packets will be considered. Detailed data packet for the BR/EDR mode in bytes is [41]:

Access Code: 9 - Packet Header: 7 - Payload Header: 2 - Payload: 339 - CRC: 2

- ***Access Code:*** Used for synchronization.
- ***Packet Header:*** Contains link control information.
- ***Payload Header:*** It specifies payload length.
- ***Payload:*** The meaningful data that is desired to be sent.
- ***CRC:*** Used for secure transferring.

4.4.3 Adaptation Through Bluetooth LE - Bluetooth BR/EDR IoT Connector

After specifying detailed information for Bluetooth LE and Bluetooth BR/EDR mode, a class diagram is provided for IoT components which are Bluetooth LE component and Bluetooth BR/EDR component and their related connector before starting implementation process. As stated in Table 4.3, there are differences both in network access & physical layer and application layer of Bluetooth LE and Bluetooth BR/EDR modes. Network access & physical layer differences will be handled by the ports that are connected on the IoT connector as shown in Figure 4.2. For experimentation, the tool is enhanced with the adaption simulation. The simulation shows the data flow from component to connector and connector to component as a verification of the connectivity. Newly developed IoT connector receives data packets from the first component. Since the developed IoT connector knows which components are connected to it, necessary data parsing and preparing process is started. According to the second component's needs, necessary data packet is prepared and sent to the second component that is a Bluetooth BR/EDR component. This process is managed by Simulation manager. In this thesis to simulate IoT components, user can set the data packet number. It is assumed that each time the payload data is filled completely and has a meaning for the other component. A class diagram for Bluetooth LE to Bluetooth BR/EDR IoT connector is shown as two separate parts in Figures 4.3 and 4.4 sequentially.

Bluetooth LE - Bluetooth BR/EDR Connector User Interface and Simulation In addition to existing component and connector definitions in the COSEML, new components that are Bluetooth LE and Bluetooth BR/EDR and their IoT connector is added to the COSEML. Each component is represented by a different color. When user hovers on the component, name of the component is shown on the screen. Bluetooth LE component holds necessary information related with Bluetooth LE protocol properties as specified in subsection 4.4.1. Bluetooth BR/EDR component holds necessary information related with Bluetooth BR/EDR protocol properties as specified in subsection 4.4.2. Bluetooth LE - Bluetooth BR/EDR connector is represented as a double-edged line. A play button is also added to COSEML user interface to start conversion between selected components. In Figure 4.5 and Figure 4.6 visual representations of Bluetooth LE component, Bluetooth BR/EDR component, connector and play button are shown sequentially.

General user interface of the COSEML corresponding to BLE component that represents IoT device talking with BLE protocol, Bluetooth BR/EDR component that represents IoT device talking with Bluetooth BR/EDR protocol and their corresponding IoT connector is shown in figure 4.7.

Specific software segments for the protocol properties are stored in the components. Connectors know which components are connected to them and manage data conversion according to component's protocol properties. When a data packet arrives to the connector, it starts to parse and identify necessary information to send the data to the second component in a proper way. After connecting the necessary elements, the tool displays data flow and conversion steps as a verification of the connectivity. A user should select desired components and connector to enable communication and should click the play button to see communication steps. Before starting the conversion process, user can specify the packet number to be sent to the second component as shown in Figure 4.8. When simulation starts, the tool opens two dialog boxes, one

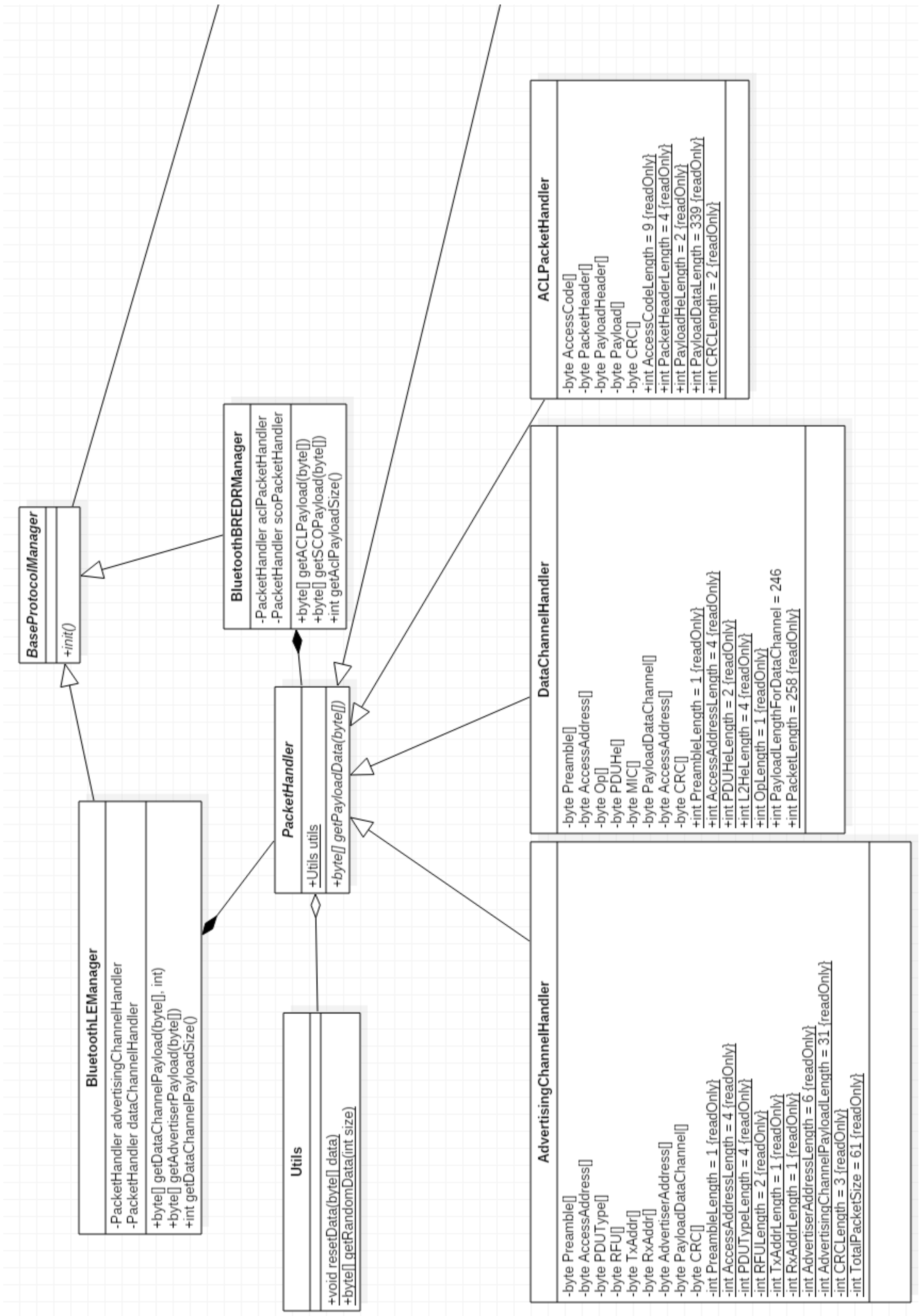

BaseProtocolManager
+init()
BluetoothBREDRManager
-PacketHandler aclPacketHandler
-PacketHandler scoPacketHandler
+byte[] getACLPayload(byte[])
+byte[] getSCOPayload(byte[])
+int getAclPayloadSize()
ACLPacketHandler
-byte AccessCode[]
-byte PacketHeader[]
-byte PayloadHeader[]
-byte Payload[]
-byte CRC[]
+int AccessCodeLength = 9 {readOnly}
+int PacketHeaderLength = 4 {readOnly}
+int PayloadHeLength = 2 {readOnly}
+int PayloadDataLength = 339 {readOnly}
+int CRCLength = 2 {readOnly}
PacketHandler
+Utils utils
+byte[] getPayloadData(byte[])
DataChannelHandler
-byte Preamble[]
-byte AccessAddress[]
-byte Op[]
-byte PDUHe[]
-byte MIC[]
-byte PayloadDataChannel[]
-byte AccessAddress[]
-byte CRC[]
+int PreambleLength = 1 {readOnly}
+int AccessAddressLength = 4 {readOnly}
+int PDUHeLength = 2 {readOnly}
+int L2HeLength = 4 {readOnly}
+int OpLength = 1 {readOnly}
+int PayloadLengthForDataChannel = 246
+int PacketLength = 258 {readOnly}
BluetoothLEManager
-PacketHandler advertisingChannelHandler
-PacketHandler dataChannelHandler
+byte[] getDataChannelPayload(byte[], int)
+byte[] getAdvertiserPayload(byte[])
+int getDataChannelPayloadSize()
Utils
+void resetData(byte[] data)
+byte[] getRandomData(int size)
AdvertisingChannelHandler
-byte Preamble[]
-byte AccessAddress[]
-byte PDUType[]
-byte RFU[]
-byte TxAddr[]
-byte RxAddr[]
-byte AdvertiserAddress[]
-byte PayloadDataChannel[]
-byte CRC[]
-int PreambleLength = 1 {readOnly}
-int AccessAddressLength = 4 {readOnly}
-int PDUTypeLength = 4 {readOnly}
-int RFULength = 2 {readOnly}
-int TxAddrLength = 1 {readOnly}
-int RxAddrLength = 1 {readOnly}
-int AdvertiserAddressLength = 6 {readOnly}
-int AdvertisingChannelPayloadLength = 31 {readOnly}
-int CRCLength = 3 {readOnly}
-int TotalPacketSize = 61 {readOnly}

Figure 4.3: BLE to Bluetooth BR/EDR IoT Connector Class Diagram - Part 1

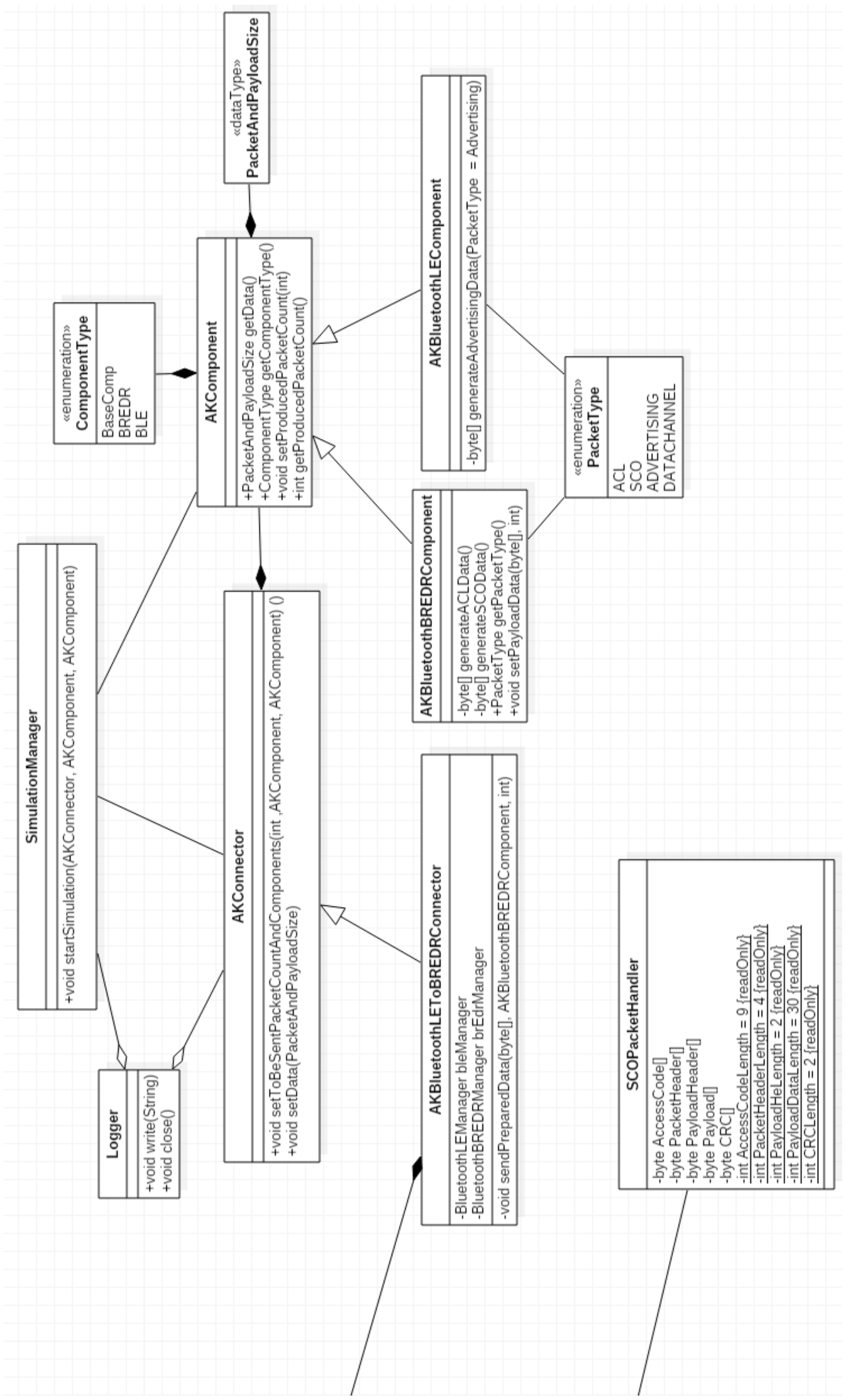
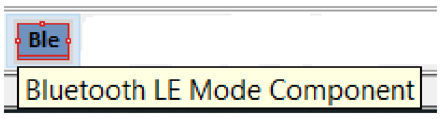
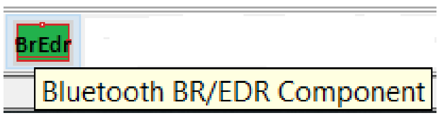

«dataType»
PacketAndPayloadSize
«enumeration»
ComponentType
BaseComp
BREDR
BLE
AKComponent
+PacketAndPayloadSize getData()
+ComponentType getComponentType()
+void setProducedPacketCount(int)
+int getProducedPacketCount()
AKBluetoothLEComponent
-byte[] generateAdvertisingData(PacketType = Advertising)
«enumeration»
PacketType
ACL
SCO
ADVERTISING
DATACHANNEL
AKBluetoothBREDRComponent
-byte[] generateACLData()
-byte[] generateSCOData()
+PacketType getPacketType()
+void setPayloadData(byte[], int)
SimulationManager
+void startSimulation(AKConnector, AKComponent, AKComponent)
AKConnector
+void setToBeSentPacketCountAndComponents(int ,AKComponent, AKComponent) ()
+void setData(PacketAndPayloadSize)
AKBluetoothLEToBREDRConnector
-BluetoothLEManager bleManager
-BluetoothBREDRManager brEdrManager
-void sendPreparedData(byte[], AKBluetoothBREDRComponent, int)
Logger
+void write(String)
+void close()
SCOPacketHandler
-byte AccessCode[]
-byte PacketHeader[]
-byte PayloadHeader[]
-byte Payload[]
-byte CRC[]
-int AccessCodeLength = 9 {readOnly}
-int PacketHeaderLength = 4 {readOnly}
-int PayloadHeLength = 2 {readOnly}
-int PayloadDataLength = 30 {readOnly}
-int CRCLength = 2 {readOnly}

Figure 4.4: BLE to Bluetooth BR/EDR IoT Connector Class Diagram - Part 2

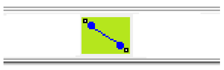


(a) Bluetooth LE Component



(b) Bluetooth BR/EDR Component

Figure 4.5: COSEML IoT Components


(a) Connector

(b) Play Button

Figure 4.6: COSEML IoT Connector and Play Button

for the connector and the other for the second component to print logging messages on the screen. In addition to these dialog boxes, the tool also logs the communication steps in a log file that is saved in a directory under the resources folder of the tool.

Considering the design details of the Bluetooth LE to Bluetooth BR/EDR connector, packet structure of the BLE is identified and parsed firstly. As stated in Subsection 4.4.1, BLE has 265 byte data packet size. The connector takes 246 byte payload data from the first port: BLE part of the connector. Bluetooth BR/EDR component has the ACL data packet and its packet structure is identified in Subsection 4.4.2. ACL data packet has 359 byte in total with 339 byte payload data. At this time, connector also knows packet size to send. If to be sent packet count is not reached yet, the connector waits for the next data packet to be received. As soon as payload data size equals the second component's payload size, the connector sends the prepared data packet to the second port: the BR/EDR part of the connector. If additional data remains in the connector and to be sent packet size is reached, connector sends all of the remaining data to the second component. In this way, the process has been completed. Output of an example scenario shown in Figure 4.9 will be explained. In this scenario, data packet size is set to 1. After modeling the example IoT application, the tool starts to simulate the conversion process. When the data packet arrives at the BLE port of the connector, it extracts 246 bytes of payload data. Since to be sent packet count is 1 for this scenario, the connector sends all of the payload data to the BR/EDR port of the

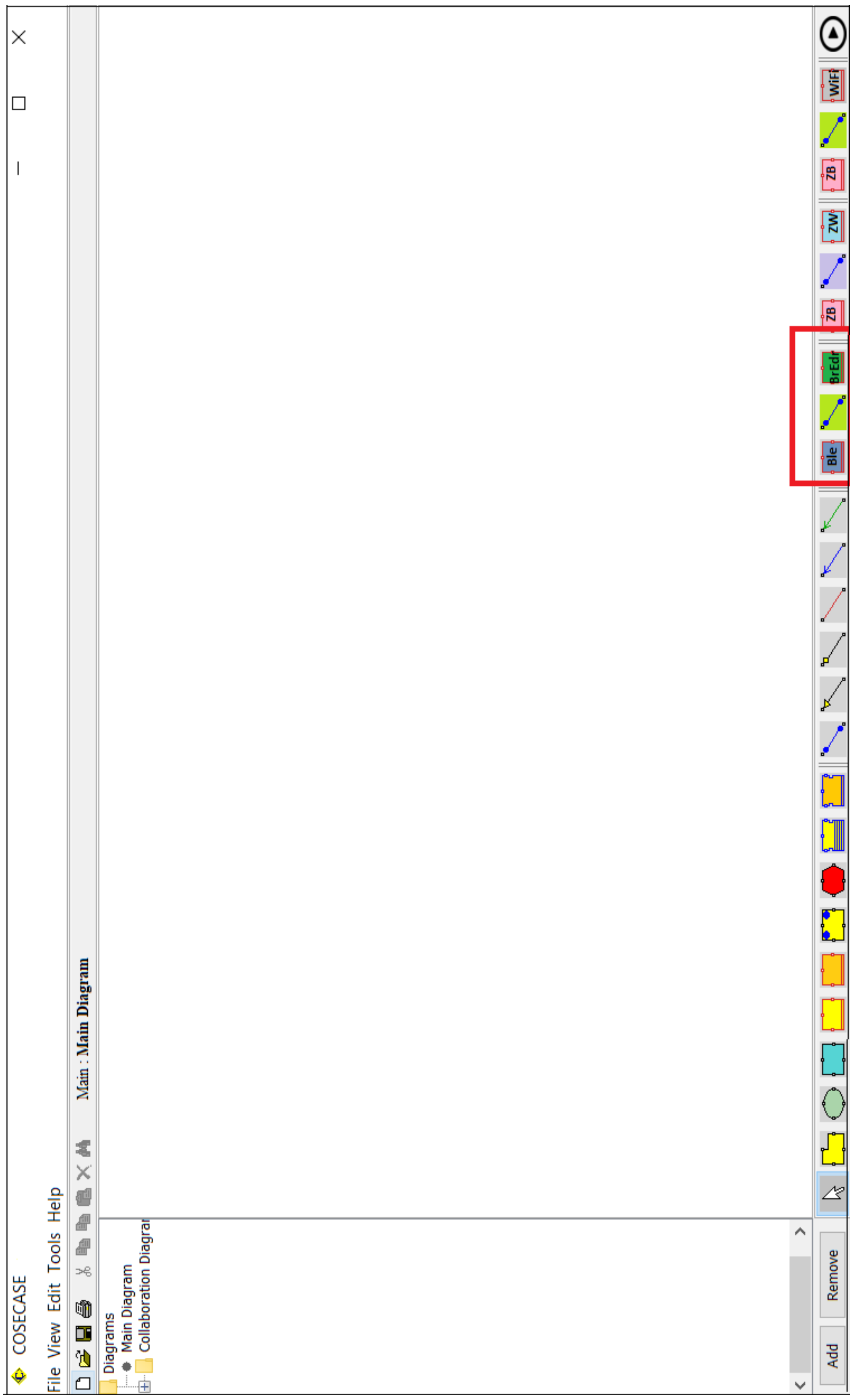
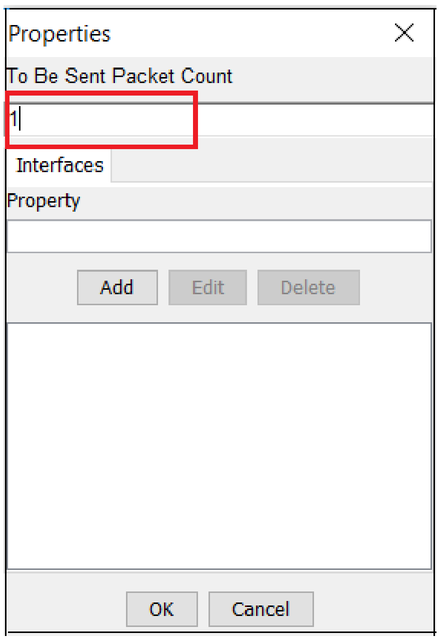


Figure 4.7: COSEML User Interface.


Properties

To Be Sent Packet Count

1

Figure 4.8: To Be Sent Packet Count Specification.

connector. In this way second component receives the payload data. Related console logs are shown in Figure 4.10. First console prints the received payload size and time information that is taken from the system's current time. Second console logs prints the second component's received data payload size and again time information that is received from the system. After conversion is completed a success message is prompted as shown in Figure 4.11.

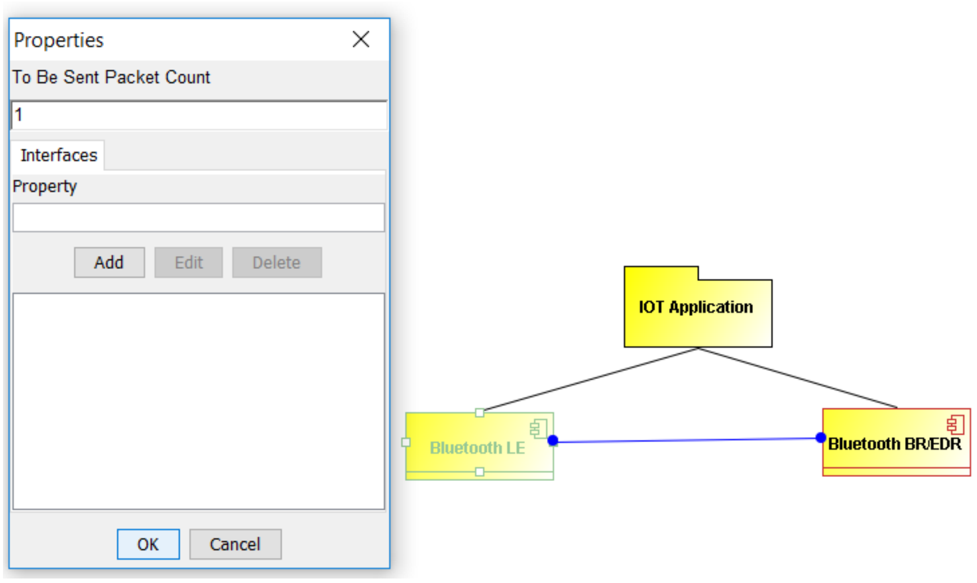
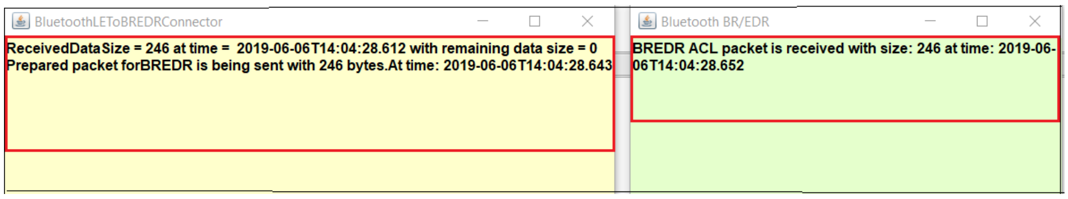


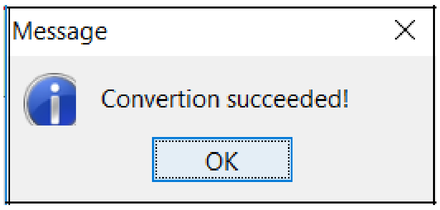
Figure 4.9: Example Scenario 1.



```
ReceivedDataSize = 246 at time =  2019-06-06T14:04:28.612 with remaining data size = 0
Prepared packet forBREDR is being sent with 246 bytes.At time: 2019-06-06T14:04:28.643
```

```
BREDR ACL packet is received with size: 246 at time: 2019-06-
06T14:04:28.652
```

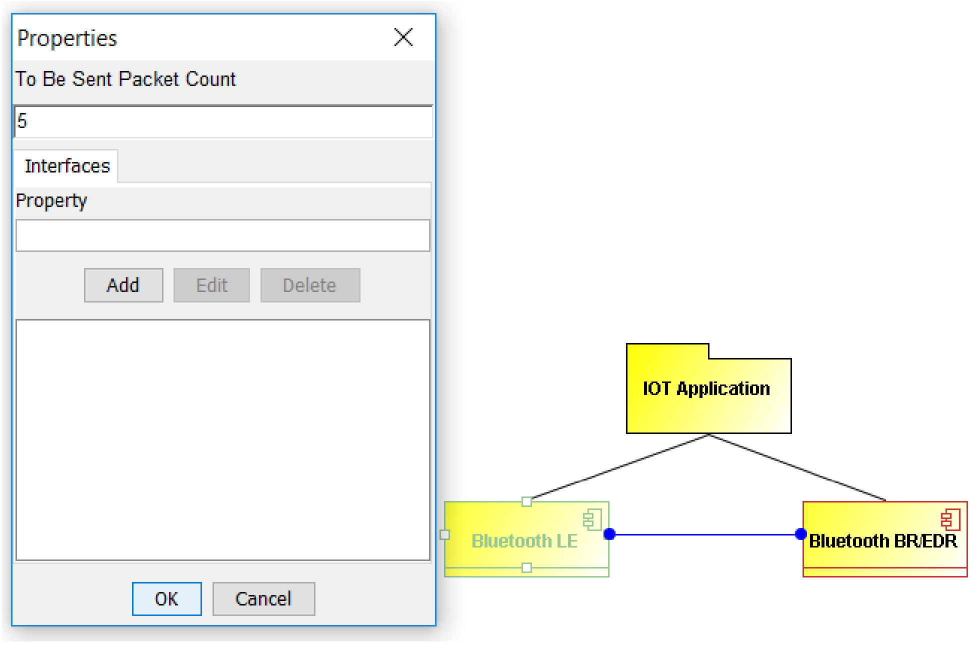
Figure 4.10: Output of Example Scenario 1.



Convertion succeeded!

Figure 4.11: Success Message.

As a second example scenario, to be sent packet count will be set to 5 as shown in


Properties

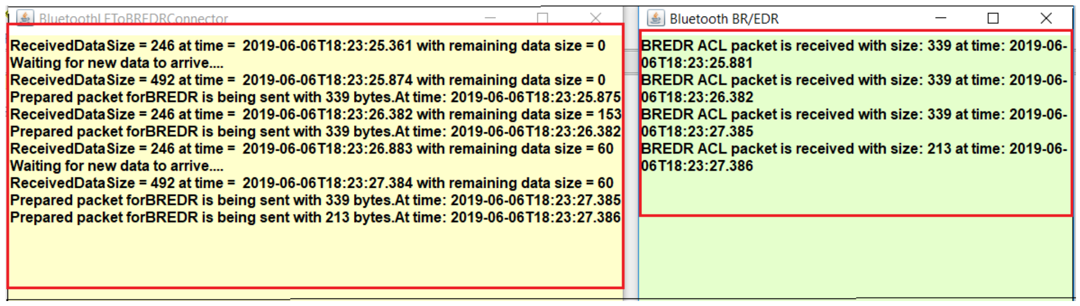
To Be Sent Packet Count

5

Figure 4.12: Example Scenario 2.

Figure 4.12. In this scenario, the connector knows to be sent data packet count is 5. Because of this when connector receives first data payload, instead of sending data packet to the second component with 246 bytes of payload data, it waits for the second data packet to complete the data size to 339, that is the second component's data payload size. As soon as the second data packet arrives to the component, it extracts 93 bytes of data and appends this data to the first one, completing 339 bytes and sends it to the BR/EDR port. In this way the second component has received its first data packet. In the connector there are 153 bytes remaining, but since to be sent data packet count has not been completed yet, it waits for new data payload to arrive. When new data payload arrives to the connector, it extract 186 bytes of data and appends this data to the remaining 153 bytes of data and sends 339 bytes data to the BR/EDR port. In this way the second component receives its second packet. There are still 60 bytes data in the connector and since to be sent data packet count has not completed yet, it waits for the fourth data packet to arrive. When fourth data packet is received, with the remaining 60 bytes of data totally 306 bytes data is held. Finally fifth data packet is received and there is 552 bytes of data and this data is sent in two packets of 339 bytes and 213 bytes. In Figure 4.13 flow of the scenario two is shown. As can be seen the adaption function includes buffer management. This is

because the sizes of received and sent data packets are of different sizes.



```
ReceivedDataSize = 246 at time =  2019-06-06T18:23:25.361 with remaining data size = 0
Waiting for new data to arrive....
ReceivedDataSize = 492 at time =  2019-06-06T18:23:25.874 with remaining data size = 0
Prepared packet forBREDR is being sent with 339 bytes.At time: 2019-06-06T18:23:25.875
ReceivedDataSize = 246 at time =  2019-06-06T18:23:26.382 with remaining data size = 153
Prepared packet forBREDR is being sent with 339 bytes.At time: 2019-06-06T18:23:26.382
ReceivedDataSize = 246 at time =  2019-06-06T18:23:26.883 with remaining data size = 60
Waiting for new data to arrive....
ReceivedDataSize = 492 at time =  2019-06-06T18:23:27.384 with remaining data size = 60
Prepared packet forBREDR is being sent with 339 bytes.At time: 2019-06-06T18:23:27.385
Prepared packet forBREDR is being sent with 213 bytes.At time: 2019-06-06T18:23:27.386
```

```
BREDR ACL packet is received with size: 339 at time: 2019-06-
06T18:23:25.881
BREDR ACL packet is received with size: 339 at time: 2019-06-
06T18:23:26.382
BREDR ACL packet is received with size: 339 at time: 2019-06-
06T18:23:27.385
BREDR ACL packet is received with size: 213 at time: 2019-06-
06T18:23:27.386
```

Figure 4.13: Output of Example Scenario Two.

4.4.4 ZigBee

ZigBee is a protocol that is based on the IEEE 802.15.4 foundation for low data rate short range wireless networking that is commonly used in home automation, in-home patient monitoring and commercial and residential IoT applications [37], [42]. ZigBee standards define only networking, application and security layers of the protocol. Physical and MAC layers are defined by IEEE 802.15.4 standards. There are four different frame types for ZigBee: Beacon frame, data frame, acknowledge frame, MAC command frame. Beacon frame is used by a coordinator to transmit beacons that are used to synchronize the clock of all the devices in the network. The data and acknowledge frames are used to transmit data and the response data about being received successfully or not. MAC command frames are used to transmit MAC command frames [42]. In this thesis, data frames will be considered. Detailed data packet information for the ZigBee data frames in bytes is explained in [37] and [42]:

Preamble: 4 - Start Packet Delimiter: 1 - PHY Header: 1 - Frame Control: 2 - Sequence Number: 1 - Address Information: 3 - Frame Control: 2 - Destination Address: 2 - Source Address: 2 - Radius: 1 - Sequence Number: 1 - Frame Control: 1 - Destination Endpoint: 1 - Cluster Id: 1 - Profile Id: 2 - Source Endpoint: 1 - Frame Payload: 106 - Frame Check Sequence: 2

- ***SHR:*** Preamble and start packet delimiter belongs to SHR. It enables the receiver to synchronize.

- ***PHY Header:*** Contains frame length information.
- ***Frame Control:*** Contains frame type.
- ***Destination Adress:*** Contains destination address.
- ***Destination Adress:*** Contains source address.
- ***Frame Payload:*** Contains core data.
- ***Frame Check Sequence:*** Used for data verification.

4.4.5 Z-Wave

Z-Wave is a protocol developed for low bandwidth data communication applications such as consumer and home automation, security sensors and alarms. Z-Wave is used in applications that require long battery life [43]. Z-Wave uses a structured protocol stack starting from physical layer to application layer. Z-Wave has four different types of frames that are singlecast, acknowledgment, multicast and broadcast frames. Siglecast frames are transmitted to specific Z-Wave node. Acknowledgment frames are used to make sure frames are sent successfully. Multicast frames are used to transmit more than one Z-Wave node. Broadcast frames are received by any Z-Wave nodes in the network. In this thesis single cast frames will be considered. Detailed data packet information for the Z-Wave single cast frames in bytes is shown below [43], [37]:

Preamble: 4 - Start of Frame: 1 - Home ID: 4 - Source ID: 1 - SingleCast Header: 1 - Data Length: 1 - Destination ID: 4 - Hop Count: 1 - Data Payload: 45 - Checksum: 1 - End of Frame: 1

- ***Preamble:*** Enables the receiver to synchronize.
- ***Home ID:*** Specifies unique network identifier.
- ***Source ID:*** Specifies unique identifier of a node.
- ***Data Length:*** Stores the length of the payload data.
- ***Destination ID:*** Used to address individual nodes.

- ***Data Payload:*** Contains payload data.
- ***Checksum:*** Used for checking correctness of the frame.

4.4.6 Adaptation Through ZigBee - Z-Wave IoT Connector

Before starting the implementation process, a class diagram is provided for IoT components which are ZigBee component and Z-Wave component and ZigBee to ZWave connector. As stated in Table 4.3, there are differences between both network access & physical layer and application layer of ZigBee and Z-Wave protocols. Network access & physical layer differences will be handled by the ports that are connected to the IoT connector as shown in Figure 4.2. For experimentation, the tool is enhanced with simulation. The simulation shows the data flow from component to connector and connector to component as a verification of the connectivity. Newly developed IoT connector receives data packets from the first component. Since developed IoT connector knows which components are connected to it, necessary data parsing and preparing process is started. According to second component's needs, necessary data packet is prepared and sent to the second component that is Z-Wave component. This process is managed by the Simulation manager in our implementation. In this thesis to simulate IoT connectors connect two IoT components providing a solution to heterogeneity problem at application level, user can set packet count that is to be sent. It is assumed that each time all of the payload data is filled totally and have a meaning for the other component. Class diagram for ZigBee to Z-Wave IoT connector is shown as two separate Figures in 4.14 and 4.15.

ZigBee - Z-Wave Connector User Interface and Simulation New components that are ZigBee, Z-Wave and also the ZigBee to Z-Wave connector are added to the COSE-CASE. Each component is represented by a different color. When user hovers on the component, name of the component is shown on the screen. ZigBee component holds necessary information related with ZigBee protocol properties as specified in subsection 4.4.4. Z-Wave component holds necessary information related with Z-Wave protocol properties as specified in subsection 4.4.5. ZigBee - Z-Wave connector is represented as a double-edged line. To start conversion between selected components, play button will be used. Visual representations of ZigBee component, Z-Wave component, and the connector are shown in Figure 4.16.

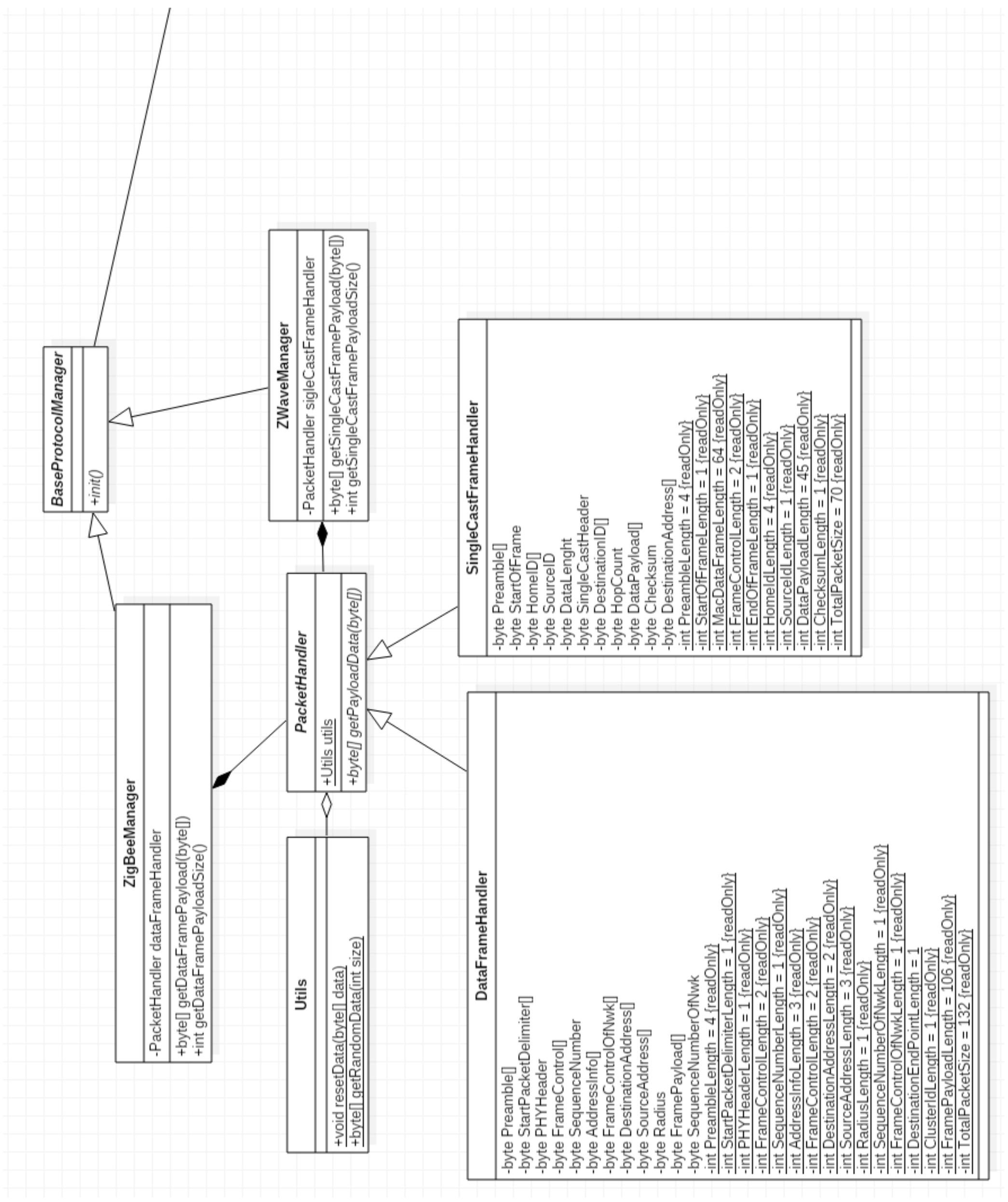

BaseProtocolManager
+init()
ZWaveManager
-PacketHandler sigleCastFrameHandler
+byte[] getSingleCastFramePayload(byte[])
+int getSingleCastFramePayloadSize()
ZigBeeManager
-PacketHandler dataFrameHandler
+byte[] getDataFramePayload(byte[])
+int getDataFramePayloadSize()
PacketHandler
+Utils utils
+byte[] getPayloadData(byte[])
Utils
+void resetData(byte[] data)
+byte[] getRandomData(int size)
SingleCastFrameHandler
-byte Preamble[]
-byte StartOfFrame
-byte HomeID[]
-byte SourceID
-byte DataLenght
-byte SingleCastHeader
-byte DestinationID[]
-byte HopCount
-byte DataPayload[]
-byte Checksum
-byte DestinationAddress[]
-int PreambleLength = 4 {readOnly}
-int StartOfFrameLength = 1 {readOnly}
-int MacDataFrameLength = 64 {readOnly}
-int FrameControlLength = 2 {readOnly}
-int EndOfFrameLength = 1 {readOnly}
-int HomeIdLength = 4 {readOnly}
-int SourceIdLength = 1 {readOnly}
-int DataPayloadLength = 45 {readOnly}
-int ChecksumLength = 1 {readOnly}
-int TotalPacketSize = 70 {readOnly}
DataFrameHandler
-byte Preamble[]
-byte StartPacketDelimiter[]
-byte PHYHeader
-byte FrameControl[]
-byte SequenceNumber
-byte AddressInfo[]
-byte FrameControlOfNwk[]
-byte DestinationAddress[]
-byte SourceAddress[]
-byte Radius
-byte FramePayload[]
-byte SequenceNumberOfNwk
-int PreambleLength = 4 {readOnly}
-int StartPacketDelimiterLength = 1 {readOnly}
-int PHYHeaderLength = 1 {readOnly}
-int FrameControlLength = 2 {readOnly}
-int SequenceNumberLength = 1 {readOnly}
-int AddressInfoLength = 3 {readOnly}
-int FrameControlLength = 2 {readOnly}
-int DestinationAddressLength = 2 {readOnly}
-int SourceAddressLength = 3 {readOnly}
-int RadiusLength = 1 {readOnly}
-int SequenceNumberOfNwkLength = 1 {readOnly}
-int FrameControlOfNwkLength = 1 {readOnly}
-int DestinationEndPointLength = 1
-int ClusterIdLength = 1 {readOnly}
-int FramePayloadLength = 106 {readOnly}
-int TotalPacketSize = 132 {readOnly}

Figure 4.14: ZigBee to Z-Wave IoT Connector Class Diagram Part1.

General user interface of the COSECASE corresponding to ZigBee that represents the IoT device talking with ZigBee protocol and Z-Wave component that represents the IoT device talking with Z-Wave protocol and their corresponding IoT connector are shown in figure 4.17. Selecting and starting simulation process is the same with the Bluetooth LE to Bluetooth BR/EDR connector. After user selects ZigBee and Z-Wave components from the tool, they need to be connected with the ZigBee to Z-Wave

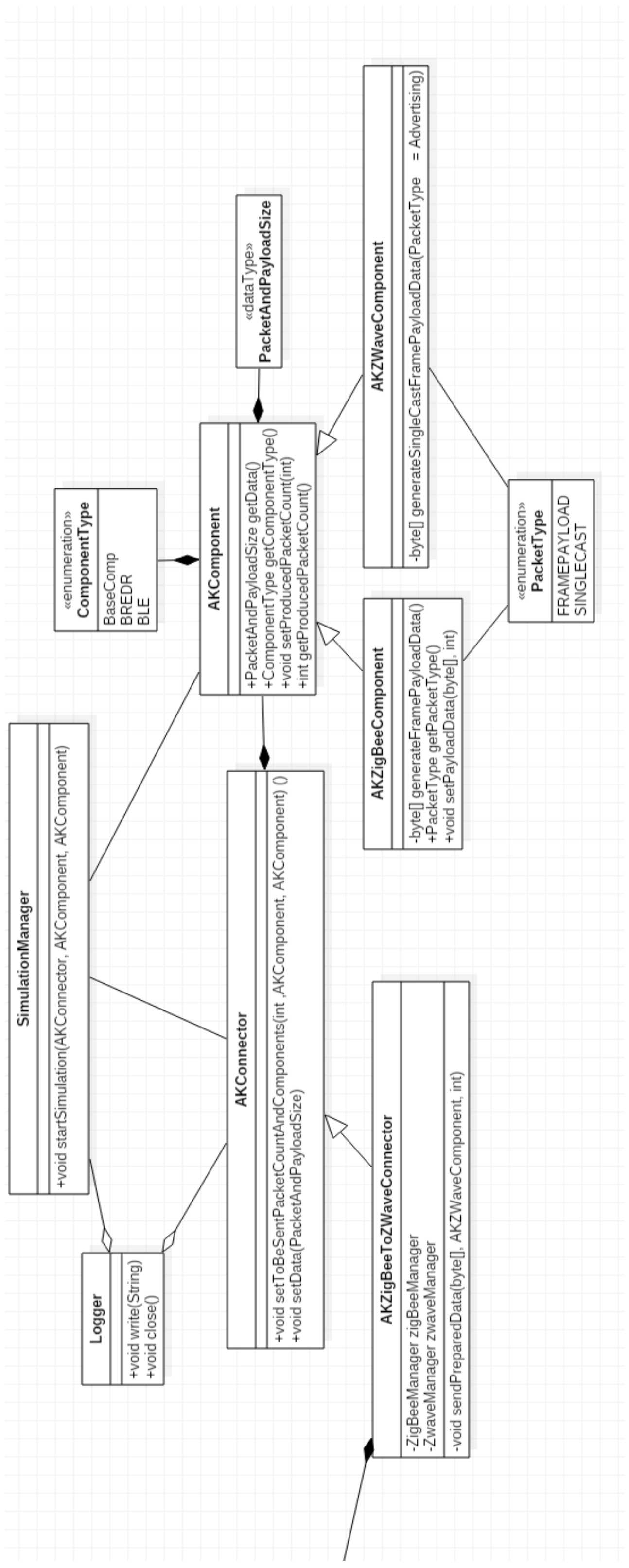
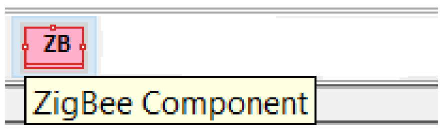
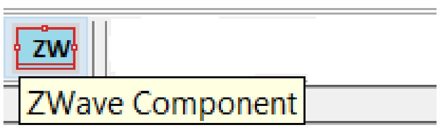

«dataType»
PacketAndPayloadSize
AKZWaveComponent
-byte[] generateSingleCastFramePayloadData(PacketType = Advertising)
«enumeration»
ComponentType
BaseComp
BREDR
BLE
AKComponent
+PacketAndPayloadSize getData()
+ComponentType getComponentType()
+void setProducedPacketCount(int)
+int getProducedPacketCount()
«enumeration»
PacketType
FRAMEPAYLOAD
SINGLECAST
AKZigBeeComponent
-byte[] generateFramePayloadData()
+PacketType getPacketType()
+void setPayloadData(byte[], int)
SimulationManager
+void startSimulation(AKConnector, AKComponent, AKComponent)
AKConnector
+void setToBeSentPacketCountAndComponents(int ,AKComponent, AKComponent) ()
+void setData(PacketAndPayloadSize)
AKZigBeeToZWaveConnector
-ZigBeeManager zigBeeManager
-ZwaveManager zwaveManager
-void sendPreparedData(byte[], AKZWaveComponent, int)
Logger
+void write(String)
+void close()

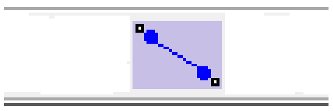
Figure 4.15: ZigBee to Z-Wave IoT Connector Class Diagram Part2.



(a) ZigBee Component



(b) Z-Wave Component



(c) IoT Connector

Figure 4.16: COSEML IoT Components

connector. User can set packet count at the ZigBee component to show simulation. After setting to be sent packet count user can start the simulation by clicking play button. Logs are recorded in the resources folder. Two dialog boxes are also opened for the connector and for the second component to display the outputs of the process.

Considering the design details of the ZigBee to Z-Wve connector, packet structure of the ZigBee is identified and parsed firstly. As stated in Subsection 4.4.4, ZigBee has a frame packet size of 132 bytes. The connector takes 106 bytes of payload data from the first port: ZigBee port of the connector. Z-Wave component has single cast frame payload and its packet structure is identified in Subsection 4.4.5. Single cast frame data packet has 58 bytes in total and 45 bytes of it is payload data. At this time, the connector also knows to be sent packet count. First component's data payload size is greater than the second one in this case. When connector receives the first packet, it divides received packets to the 45 bytes subparts and sends these 45 bytes packets to the Zi-Wave port of the connector. After that Z-Wave component receives its first data packet. Connector sends packets to the Z-Wave component until the data is finished. If to be sent packet count is not reached yet, connector waits for the next data packets from the ZigBee component. When to be sent packet count raeched, the process has been completed. For an example scenario shown in Figure 4.18, output of the process flow is explained. In this scenario, data packet size is set to 1. After modeling the example IoT application, the tool start to simulate conversion process. When the data packet arrives to the ZigBee port of the connector, it extracts 106 bytes payload data. Since to be sent packet count is 1 for this scenario, connectors sends

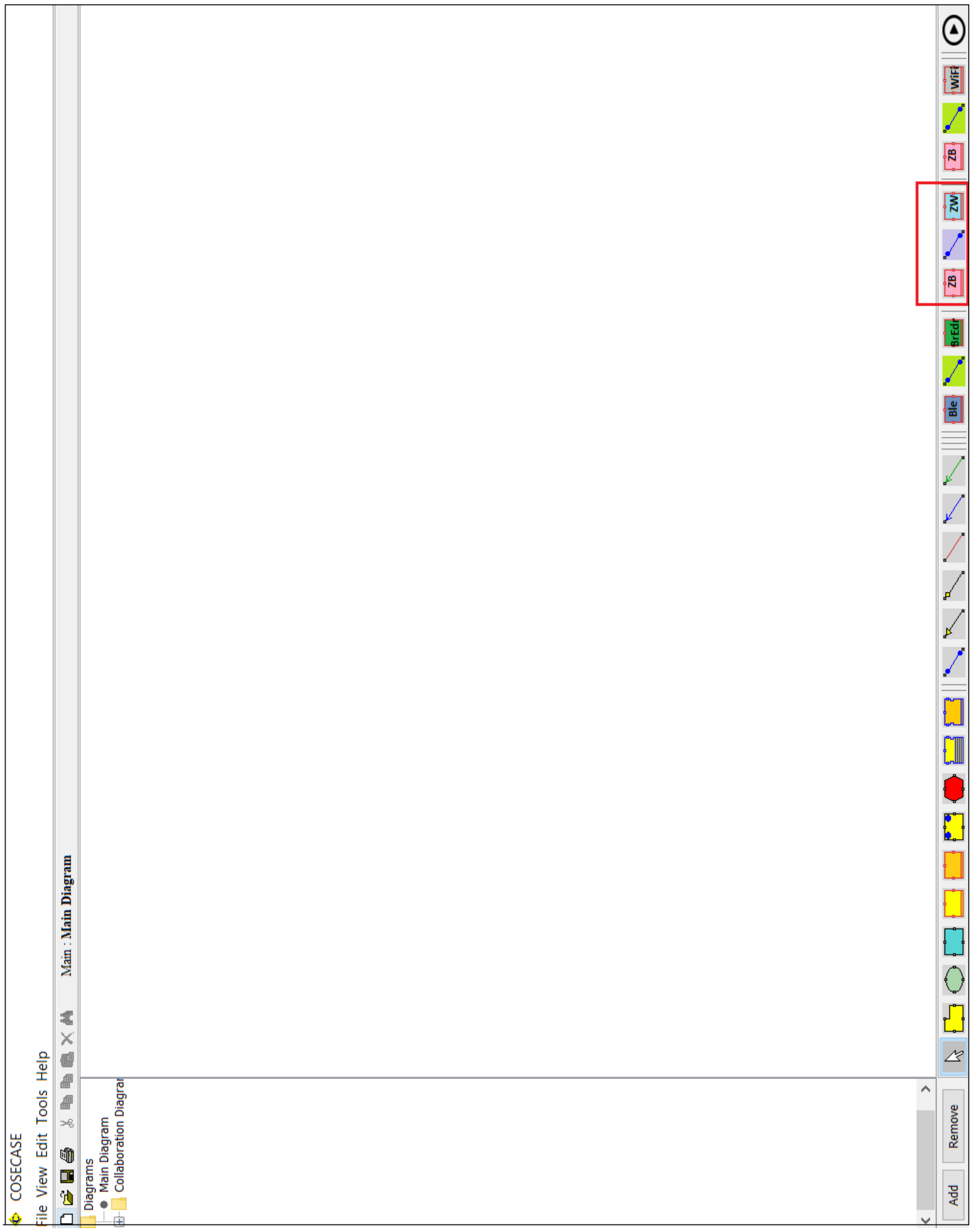


Figure 4.17: COSEML User Interface.

all of the payload data as a 45 bytes data packets to the Z-Wave port of the connector. In this way second component receives the payload data as a 45 bytes data packets. Since ZigBee payload size is 106, the final packet will be sent as 16 bytes of payload data packet. Related console logs are shown in Figure 4.19. First console records the received payload size and time information that is taken from system's current time. Second console logs the second component's received data payload size and again time information that is received from the system's current time information. After conversion is completed a success message is prompted as shown in Figure 4.11.

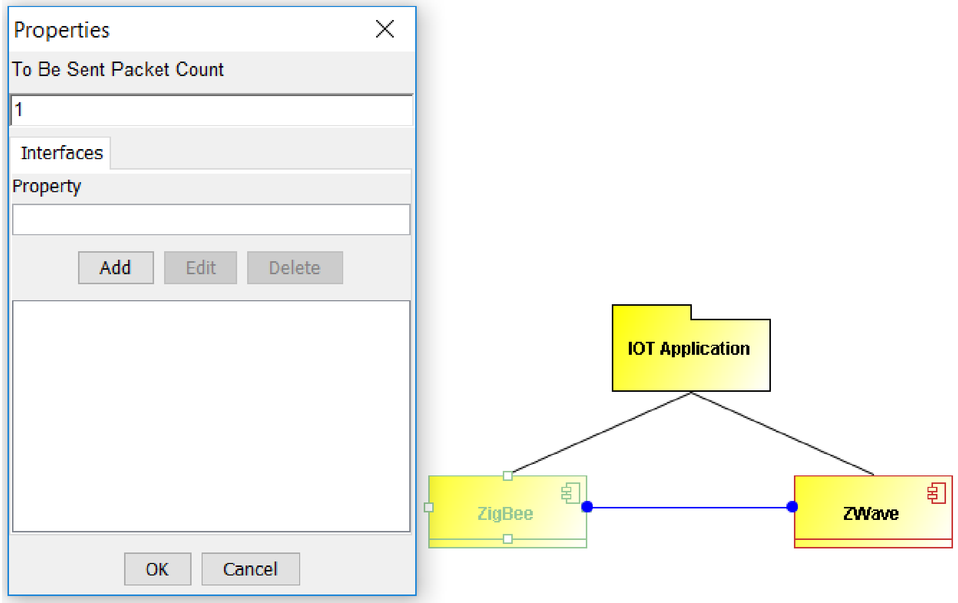
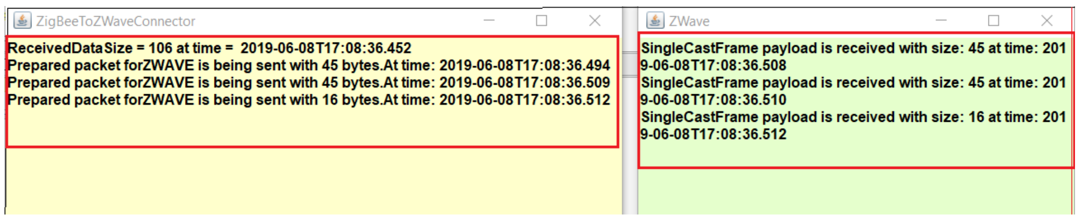


Figure 4.18: Example Scenario 1.



```
ReceivedDataSize = 106 at time =  2019-06-08T17:08:36.452
Prepared packet forZWAVE is being sent with 45 bytes.At time: 2019-06-08T17:08:36.494
Prepared packet forZWAVE is being sent with 45 bytes.At time: 2019-06-08T17:08:36.509
Prepared packet forZWAVE is being sent with 16 bytes.At time: 2019-06-08T17:08:36.512
```

```
SingleCastFrame payload is received with size: 45 at time: 201
9-06-08T17:08:36.508
SingleCastFrame payload is received with size: 45 at time: 201
9-06-08T17:08:36.510
SingleCastFrame payload is received with size: 16 at time: 201
9-06-08T17:08:36.512
```

Figure 4.19: Output Of Example Scenario 1.

As an example scenario two shown in Figure 4.20, to be sent packet count is set to 3. Connector will send packets in a 45 bytes data packets. In this case, since connector knows to be sent packet count, it waits for the next packets to arrive and with remaining bytes and as soon as the size is completed to 45 bytes, connector sends the data to the Z-Wave port. In this way, Z-Wave component receives the data packets. Outputs of this scenario is shown in Figure 4.21.

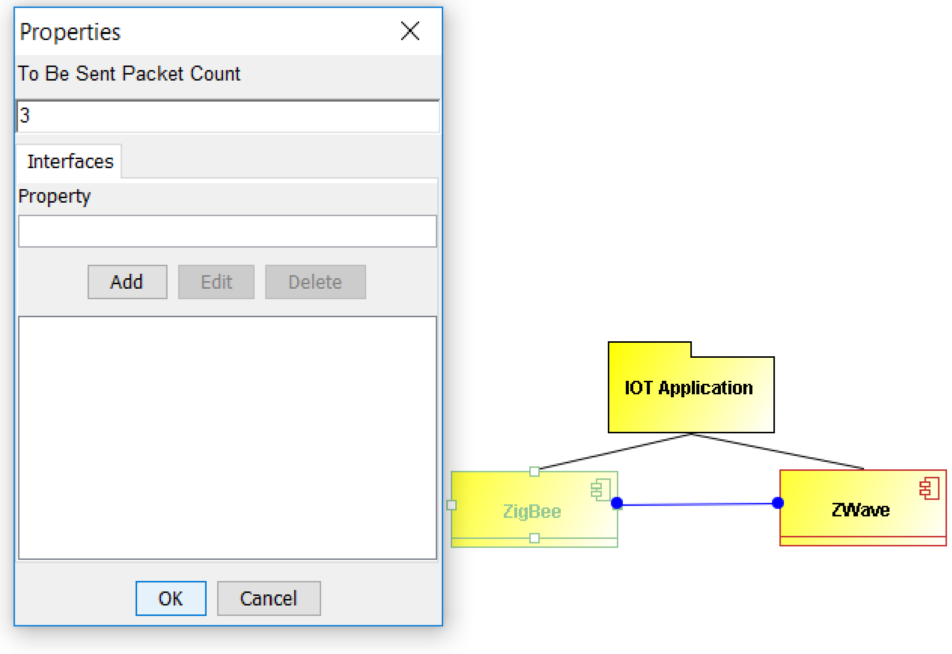
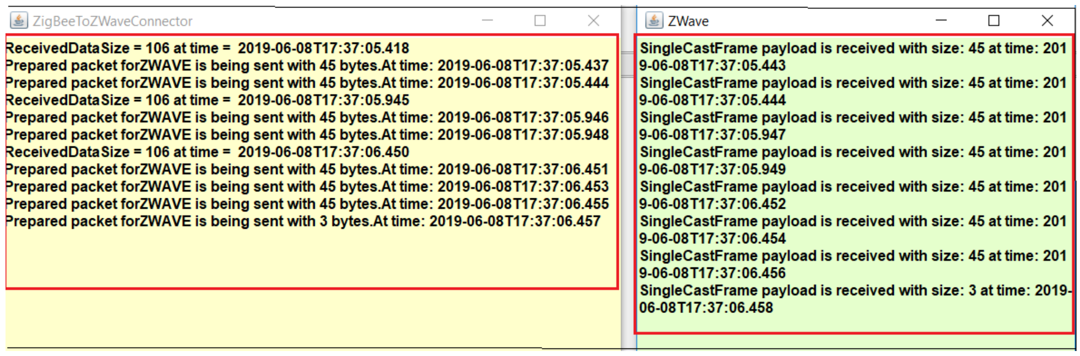


Figure 4.20: Example Scenario 2.



```
ReceivedDataSize = 106 at time =  2019-06-08T17:37:05.418
Prepared packet forZWAVE is being sent with 45 bytes.At time: 2019-06-08T17:37:05.437
Prepared packet forZWAVE is being sent with 45 bytes.At time: 2019-06-08T17:37:05.444
ReceivedDataSize = 106 at time =  2019-06-08T17:37:05.945
Prepared packet forZWAVE is being sent with 45 bytes.At time: 2019-06-08T17:37:05.946
Prepared packet forZWAVE is being sent with 45 bytes.At time: 2019-06-08T17:37:05.948
ReceivedDataSize = 106 at time =  2019-06-08T17:37:06.450
Prepared packet forZWAVE is being sent with 45 bytes.At time: 2019-06-08T17:37:06.451
Prepared packet forZWAVE is being sent with 45 bytes.At time: 2019-06-08T17:37:06.453
Prepared packet forZWAVE is being sent with 45 bytes.At time: 2019-06-08T17:37:06.455
Prepared packet forZWAVE is being sent with 3 bytes.At time: 2019-06-08T17:37:06.457
```

```
SingleCastFrame payload is received with size: 45 at time: 201
9-06-08T17:37:05.443
SingleCastFrame payload is received with size: 45 at time: 201
9-06-08T17:37:05.444
SingleCastFrame payload is received with size: 45 at time: 201
9-06-08T17:37:05.947
SingleCastFrame payload is received with size: 45 at time: 201
9-06-08T17:37:05.949
SingleCastFrame payload is received with size: 45 at time: 201
9-06-08T17:37:06.452
SingleCastFrame payload is received with size: 45 at time: 201
9-06-08T17:37:06.454
SingleCastFrame payload is received with size: 45 at time: 201
9-06-08T17:37:06.456
SingleCastFrame payload is received with size: 3 at time: 2019-
06-08T17:37:06.458
```

Figure 4.21: Output Of Example Scenario 2.

4.4.7 ZigBee

ZigBee protocol properties are explained in 4.4.4.

4.4.8 WiFi

WiFi is a protocol based on the 802.11 IEEE specification that transmits data over radio waves [44]. WiFi is commonly used at home, at work, airports, university campuses and schools. There are three different kinds of data frames for WiFi that are control frames, management frames and data frames [44]. Control frames are used for the acknowledgment of received data. Management frames are used to join or leave wireless networks. Data frames are used to carry data. In this thesis data frames will be considered. Detailed data packet information for the data frame in bytes as follows [44]:

Frame Control: 2 - Duration: 2 - Address1(Receiver): 6 - Address2(Sender): 6 - Address3(Filtering): 6 - Address4: 6 - Frame Body: 2312 - FCS: 4

- ***Frame Control:*** May affect the interpretation of MAC header fields .
- ***Duration:*** Carries the network allocation vector value.
- ***Addressing Bits:*** These fields control addressing issues.
- ***Frame Body:*** Contains payload data.

4.4.9 Adapting ZigBee & WiFi Components With IoT Connector

Before starting the implementation process, a class diagram is provided for the IoT components which are the ZigBee component and the WiFi component and the ZigBee to WiFi connector. As stated in Table 4.3, there are differences between both network access & physical layers and the application layer of ZigBee and WiFi protocols. Network access & physical layer differences will be handled by the ports that are on the IoT connector as shown in Figure 4.2. For experimentation, the tool is enhanced with simulation capabilities. The simulation shows the data flow from component to connector and connector to component as a verification of the connectivity. Since data packet sizes are similar to each other, for ZigBee to WiFi components connector will work as ZigBee to Z-Wave connector as stated in the subsection 4.4.6. For both of the cases, first component's data packet size is fewer than the second component data packet size. Therefore working mechanism will be similar and ZigBee to Z-Wave connector can be used by ZigBee to WiFi connector. In this way connector becomes reusable part of the COSECASE. Class diagram for ZigBee to WiFi IoT connector is shown as separate files in Figures 4.22 and 4.23.

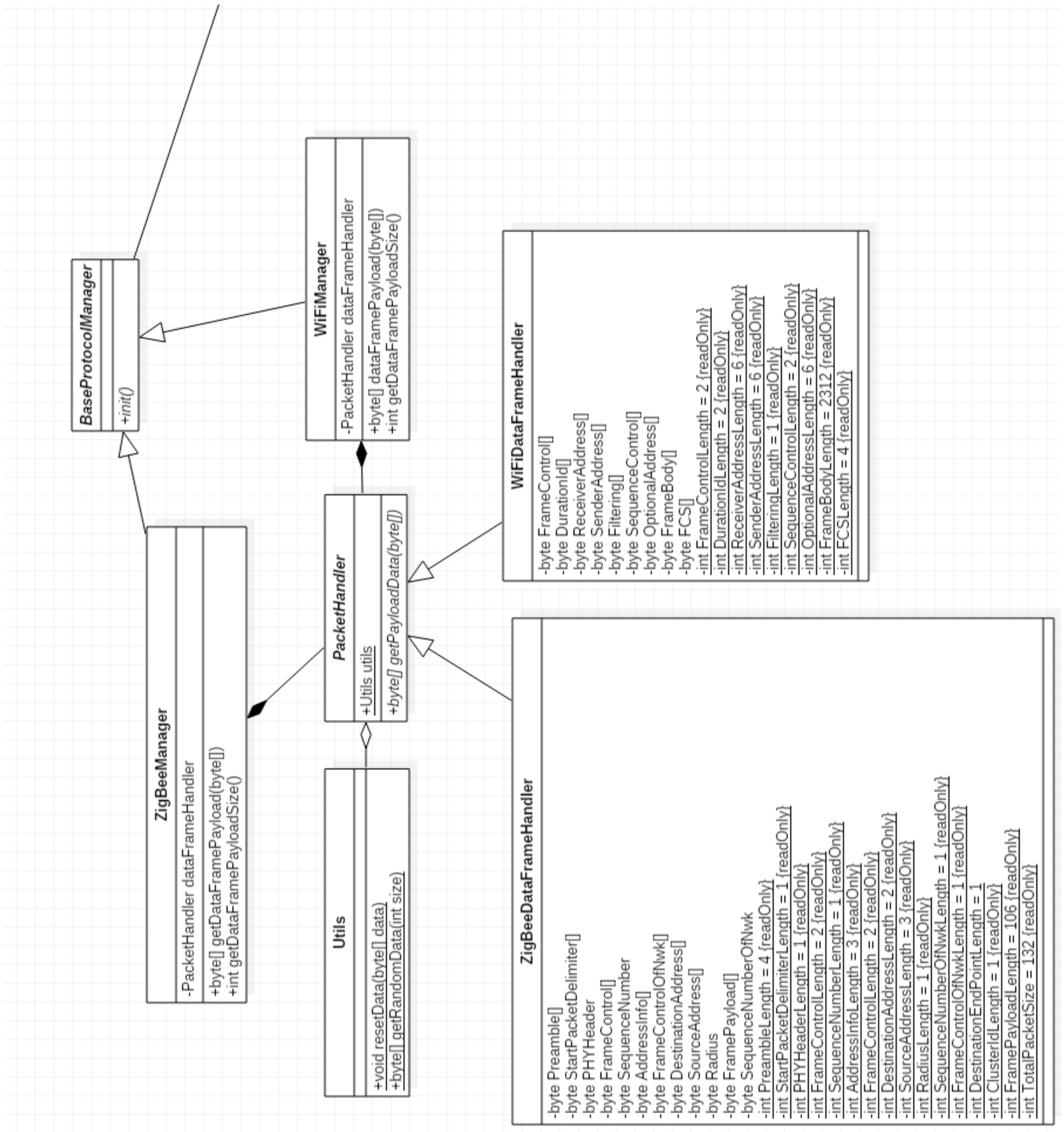

BaseProtocolManager
+init()
WiFiManager
-PacketHandler dataFrameHandler
+byte[] dataFramePayload(byte[])
+int getDataFramePayloadSize()
ZigBeeManager
-PacketHandler dataFrameHandler
+byte[] getDataFramePayload(byte[])
+int getDataFramePayloadSize()
PacketHandler
+Utils utils
+byte[] getPayloadData(byte[])
Utils
+void resetData(byte[] data)
+byte[] getRandomData(int size)
WiFiDataFrameHandler
-byte FrameControl[]
-byte DurationId[]
-byte ReceiverAddress[]
-byte SenderAddress[]
-byte Filtering[]
-byte SequenceControl[]
-byte OptionalAddress[]
-byte FrameBody[]
-byte FCS[]
-int FrameControlLength = 2 {readOnly}
-int DurationIdLength = 2 {readOnly}
-int ReceiverAddressLength = 6 {readOnly}
-int SenderAddressLength = 6 {readOnly}
-int FilteringLength = 1 {readOnly}
-int SequenceControlLength = 2 {readOnly}
-int OptionalAddressLength = 6 {readOnly}
-int FrameBodyLength = 2312 {readOnly}
-int FCSLength = 4 {readOnly}
ZigBeeDataFrameHandler
-byte Preamble[]
-byte StartPacketDelimiter[]
-byte PHYHeader
-byte FrameControl[]
-byte SequenceNumber
-byte AddressInfo[]
-byte FrameControlOfNwk[]
-byte DestinationAddress[]
-byte SourceAddress[]
-byte Radius
-byte FramePayload[]
-byte SequenceNumberOfNwk
-int PreambleLength = 4 {readOnly}
-int StartPacketDelimiterLength = 1 {readOnly}
-int PHYHeaderLength = 1 {readOnly}
-int FrameControlLength = 2 {readOnly}
-int SequenceNumberLength = 1 {readOnly}
-int AddressInfoLength = 3 {readOnly}
-int FrameControlLength = 2 {readOnly}
-int DestinationAddressLength = 2 {readOnly}
-int SourceAddressLength = 3 {readOnly}
-int RadiusLength = 1 {readOnly}
-int SequenceNumberOfNwkLength = 1 {readOnly}
-int FrameControlOfNwkLength = 1 {readOnly}
-int DestinationEndPointLength = 1
-int ClusterIdLength = 1 {readOnly}
-int FramePayloadLength = 106 {readOnly}
-int TotalPacketSize = 132 {readOnly}

Figure 4.22: ZigBee to WiFi IoT Connector Class Diagram - Part 1.

ZigBee - WiFi Connection, User Interface and Simulation Existing ZigBee component and ZigBee to Z-Wave connector are used for this case. WiFi component is added to the tool as a new component. This component is also represented with a different color and when user hovers on the component, name of the component is shown on the screen. WiFi component holds necessary information related with WiFi protocol properties as specified in subsection 4.4.8. After ZigBee and WiFi components are selected, ZigBee to Z-Wave connector is used for connect these components. To start conversion between the components, the play button will be used. General user interface of the COSECASE corresponding to ZigBee that represents IoT device talking with ZigBee protocol and WiFi component that represents IoT device talking with WiFi protocol and their corresponding IoT connector are shown in figure 4.24.

ZigBee component's packet structure is identified and parsed firstly by the connector. As stated in the subsection 4.4.4, ZigBee has 132 bytes of data frame packet. As stated in the subsection 4.4.8, WiFi has 2346 bytes of data frame packet. The connector receives 106 bytes of payload data from the first port: ZigBee port of the connector. WiFi component has data frame packet type that has 2312 bytes of payload data. Connector knows to be sent packet count. Since first component's packet size is smaller than the second one, until the buffer of 2312 is filled, the connector keeps receiving data packets from the first component. As soon as bytes the received data becomes 2312 bytes, the connector sends this prepared packet to the second port: WiFi port of the connector. In this way WiFi component receives the data. As an example scenario shown to be described using Figure 4.25, to be sent packet count is set to 1. From ZigBee port, 1 data packet will be received by the connector. When connector receives the data packet since to be sent packet count is 1, this packet is sent immediately to the WiFi port of the connector. In this way data payload packet is received by the WiFi component. Two dialog boxes are opened for the connector and WiFi components to print outputs of the scenario as shown in Figure 4.26. The connector sends 106 bytes of payload data to the WiFi component. Success message is prompted that shows message has been successfully sent as shown in Figure 4.11.

Figure 4.27 will be used for the explanation of scenario 2. To be sent packet count is set to 6. Connector will wait all of the 6 payload data packets to be received. Since 6 payload data size is fewer than the WiFi payload size, connector will send 636 bytes data packet at once. WiFi port receives the prepared data packet and sends this data packet to the WiFi component. As a result, WiFi component receives data. Outputs of this scenario is shown in Figure 4.28. After sending the data packet to the second component a success message will be prompted as shown in Figure 4.11.

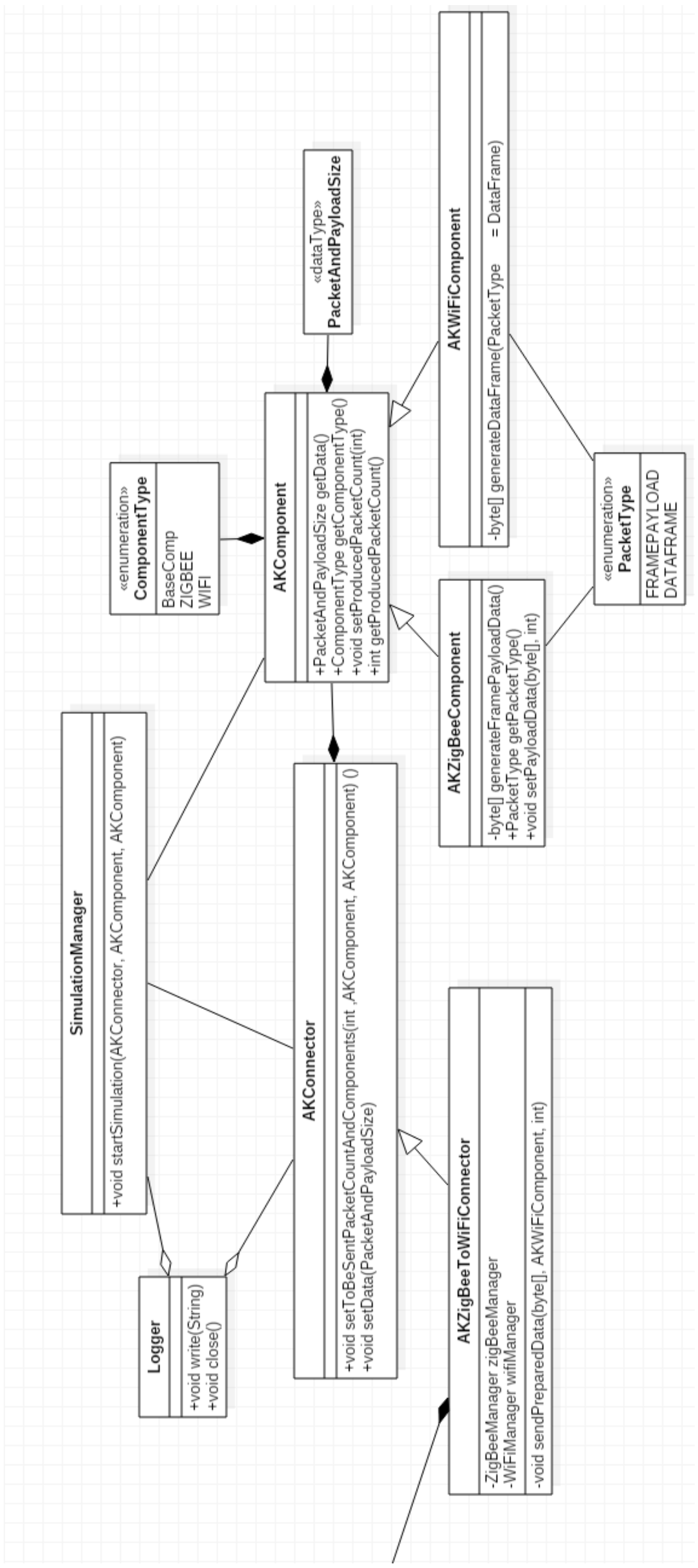

«dataType»
PacketAndPayloadSize
AKWiFiComponent
-byte[] generateDataFrame(PacketType = DataFrame)
«enumeration»
ComponentType
BaseComp
ZIGBEE
WIFI
AKComponent
+PacketAndPayloadSize getData()
+ComponentType getComponentType()
+void setProducedPacketCount(int)
+int getProducedPacketCount()
«enumeration»
PacketType
FRAMEPAYLOAD
DATAFRAME
AKZigBeeComponent
-byte[] generateFramePayloadData()
+PacketType getPacketType()
+void setPayloadData(byte[], int)
SimulationManager
+void startSimulation(AKConnector, AKComponent, AKComponent)
AKConnector
+void setToBeSentPacketCountAndComponents(int ,AKComponent, AKComponent) ()
+void setData(PacketAndPayloadSize)
AKZigBeeToWiFiConnector
-ZigBeeManager zigBeeManager
-WiFiManager wifiManager
-void sendPreparedData(byte[], AKWiFiComponent, int)
Logger
+void write(String)
+void close()

Figure 4.23: ZigBee to WiFi IoT Connector Class Diagram - Part 2.

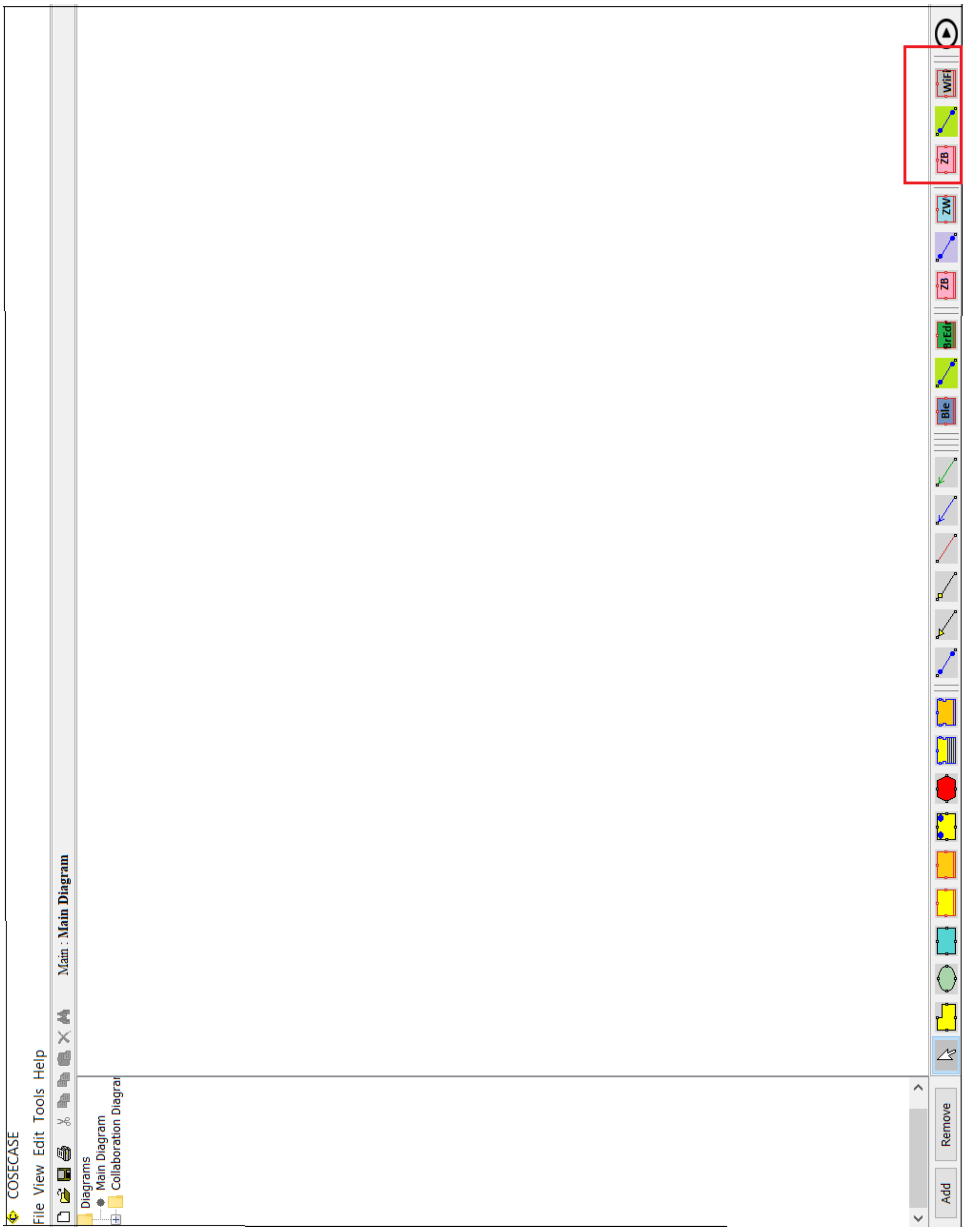


Figure 4.24: COSEML User Interface.

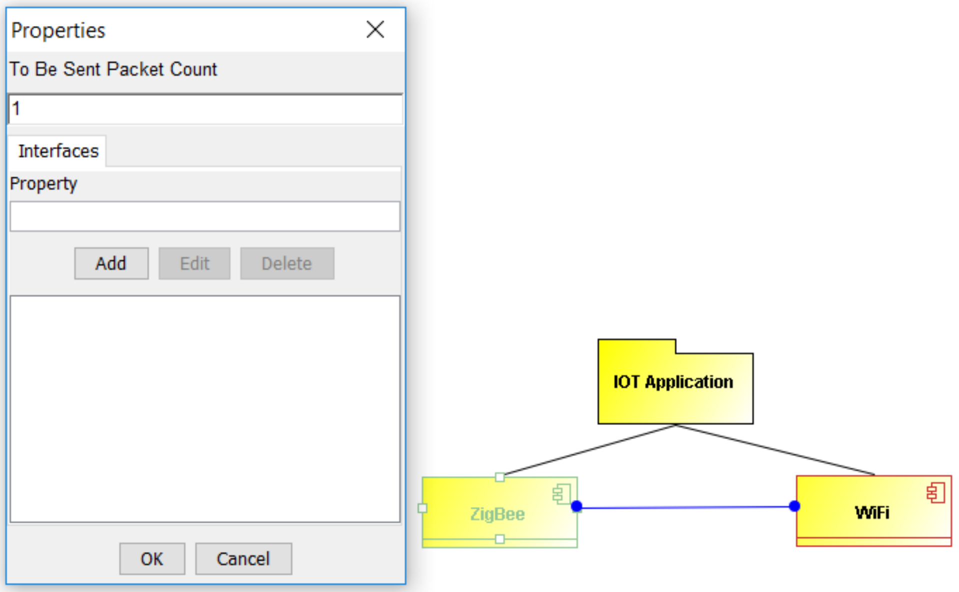
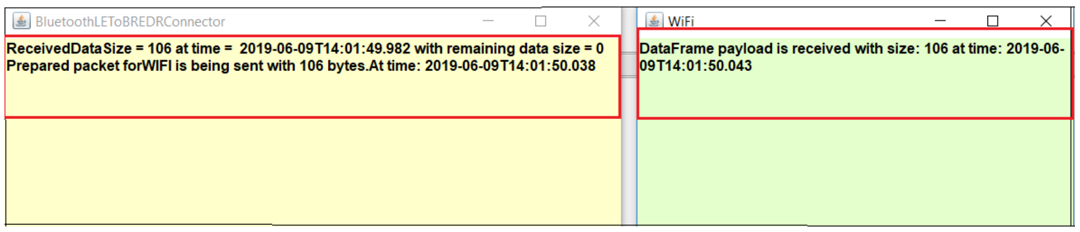


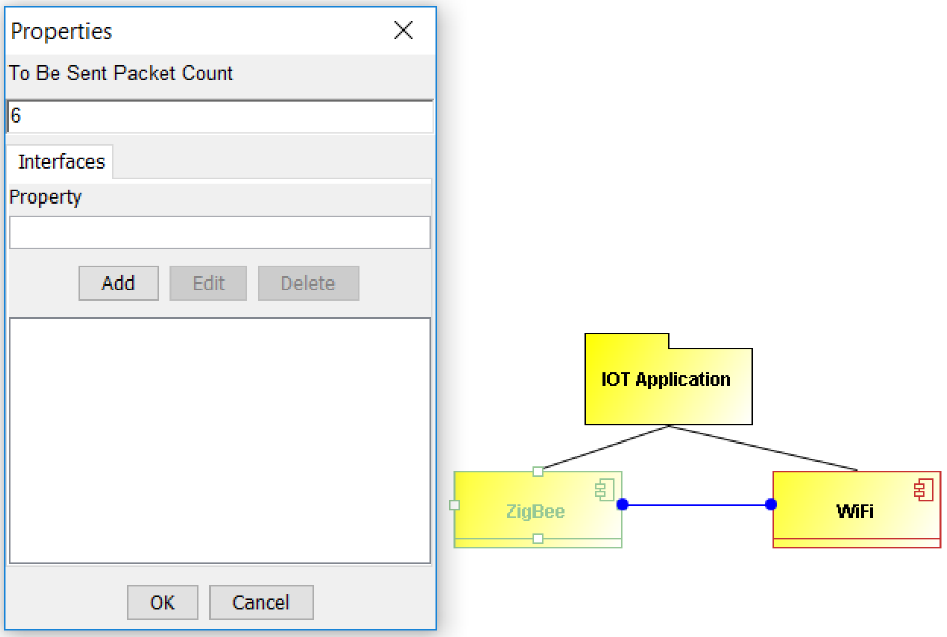
Figure 4.25: Example Scenario 1.



```
ReceivedDataSize = 106 at time =  2019-06-09T14:01:49.982 with remaining data size = 0
Prepared packet forWIFI is being sent with 106 bytes.At time: 2019-06-09T14:01:50.038
```

```
DataFrame payload is received with size: 106 at time: 2019-06-
09T14:01:50.043
```

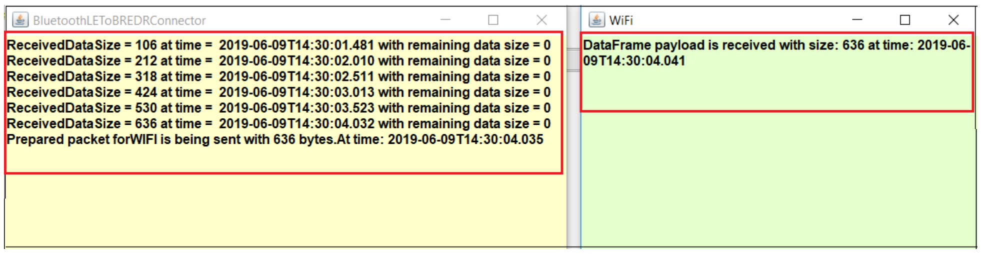
Figure 4.26: Output Of Example Scenario 1.



Properties

To Be Sent Packet Count: 6

Figure 4.27: Example Scenario 2.



```
ReceivedDataSize = 106 at time =  2019-06-09T14:30:01.481 with remaining data size = 0
ReceivedDataSize = 212 at time =  2019-06-09T14:30:02.010 with remaining data size = 0
ReceivedDataSize = 318 at time =  2019-06-09T14:30:02.511 with remaining data size = 0
ReceivedDataSize = 424 at time =  2019-06-09T14:30:03.013 with remaining data size = 0
ReceivedDataSize = 530 at time =  2019-06-09T14:30:03.523 with remaining data size = 0
ReceivedDataSize = 636 at time =  2019-06-09T14:30:04.032 with remaining data size = 0
Prepared packet forWIFI is being sent with 636 bytes.At time: 2019-06-09T14:30:04.035
```

```
DataFrame payload is received with size: 636 at time: 2019-06-
09T14:30:04.041
```

Figure 4.28: Output Of Example Scenario 2.

CHAPTER 5

MODELING SMART PARKING WITH COSECASE

In this chapter smart city domain that is explained in section 3.1 will be modeled in COSECASE with the help of newly developed components and connectors. As stated before smart city area gains more popularity in the world to make peoples lives easier. In section 3.1.4 smart city domain is modeled with a feature diagram to specify its features. Since smart city has a wide application area, to be more specific the smart parking domain is selected to be identified in detail. According to Figure 3.2, smart parking has main components that communicate using WiFi, ZigBee, Z-Wave and Bluetooth. Based on the example system stated in the Section 3.1.5, there four main components that are camera that talks using ZigBee, traffic lights that talks using Z-Wave, street lights that talks using ZigBee and second mobile phone that talks using WiFi. To model the example smart parking system in COSECASSE, we need to select related components first. To connect these components we need to select related connectors. To show connectivity we will use simulated data flow between components. In Figure 5.1 smart parking is modeled in COSECASE. Data flow of the modeled system is shown in Figure 5.2 and Figure 5.3 respectively.

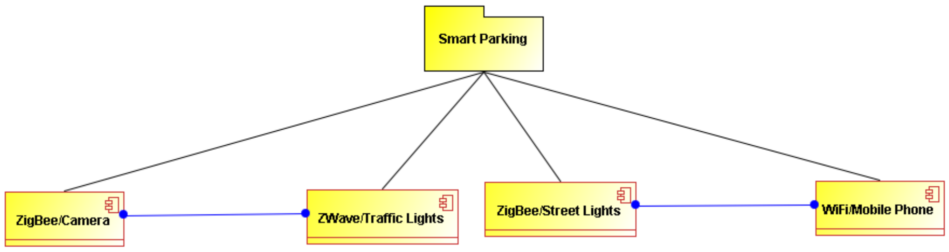
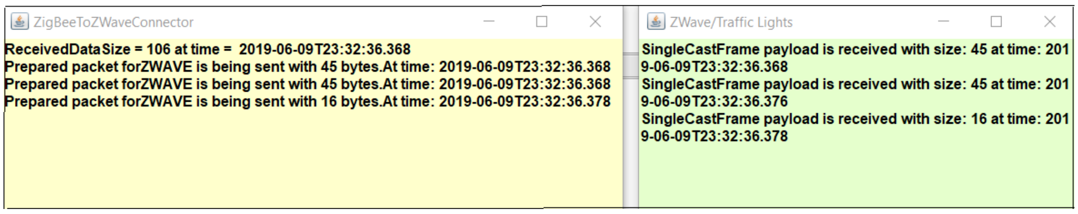


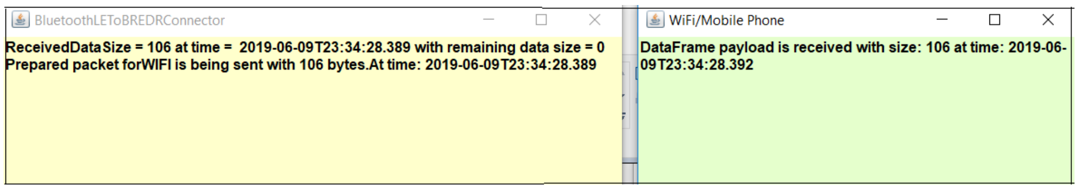
Figure 5.1: Smart Parking.



```
ReceivedDataSize = 106 at time =  2019-06-09T23:32:36.368
Prepared packet forZWAVE is being sent with 45 bytes.At time: 2019-06-09T23:32:36.368
Prepared packet forZWAVE is being sent with 45 bytes.At time: 2019-06-09T23:32:36.368
Prepared packet forZWAVE is being sent with 16 bytes.At time: 2019-06-09T23:32:36.378
```

```
SingleCastFrame payload is received with size: 45 at time: 201
9-06-09T23:32:36.368
SingleCastFrame payload is received with size: 45 at time: 201
9-06-09T23:32:36.376
SingleCastFrame payload is received with size: 16 at time: 201
9-06-09T23:32:36.378
```

Figure 5.2: ZigBee - Z-Wave Output.



```
ReceivedDataSize = 106 at time =  2019-06-09T23:34:28.389 with remaining data size = 0
Prepared packet forWIFI is being sent with 106 bytes.At time: 2019-06-09T23:34:28.389
```

```
DataFrame payload is received with size: 106 at time: 2019-06-
09T23:34:28.392
```

Figure 5.3: ZigBee - WiFi Output.

CHAPTER 6

CONCLUSION

6.1 Conclusion

In this thesis, connectors for IoT components are introduced for COSE. IoT is a very heterogeneous environment that makes hard for entities to communicate and send data to each other. We have provided a modelling environment in COSE considering heterogeneity in IoT world. The idea has been demonstrated by using our enhancements to the COSECASE tool.

There are protocols that changes according to application needs in the IoT world. We grouped protocols and specified their basic features and usage areas. Results of search shows us short range network protocol is widely used in IoT applications. Since short range IoT network protocols are commonly used in IoT applications, they are firstly selected to be implemented. Selected protocols are specified according to their features. Differences are listed and an appropriate solution is suggested for their heterogeneity. Solution could be at hardware level, at software level or both. In this thesis, an application level solution is provided for connectors in COSE development tool that is COSECASE. Connectors have ports compatible with connected component's hardware. Connectors are responsible for identifying and organizing packets of the protocols accordingly. Simulation capability is added to the tool to demonstrate the executability of the idea. A user can monitor the data flow across components via connectors. According to packet structures of protocols, connectors can be reused. If a new protocol with new features is needed to be included, a developer should implement a new connector. COSEML is enhanced with newly added connectors and components representing short range network protocols. Also a new connector structure is introduced. A new method is provided for connectors that are newly added to the tool. In this way, user can model IoT application in COSE by using COSEML.

IoT heterogeneity is demonstrated through an example project that is in the smart city domain. Smart city is a very hot topic subject in today's world. Since lots of solutions are being developed in the smart city domain, we selected this domain as an example. To be more specific we selected smart traffic as a sub topic from smart city. After providing a modelling environment for IoT applications, we modeled a smart traffic problem in the COSECASE tool and provided an example connection between related components.

6.2 Adding New IoT Connectors And Components

One can add new IoT components and connectors to the COSECASE tool. When user wants to add new components representing a new IoT protocol, he needs to identify the detailed packet structure of the protocol. After finding detailed packet structure information, user needs to specify payload data of this protocol at the application level. User can analyze the provided class diagrams for the IoT components that are provided in Chapter 4. In addition to adding new IoT components, a user can also add new IoT connectors to COSECASE. When user wants to add a new IoT connector to the tool, he needs to specify data packet properties of the two components in detail. After that, according to the second component's data packet structure, connector needs to adapt the received data packet to the destination format. User can also analyze IoT connectors' class diagrams that are provided in Chapter 4.

6.3 Suggestions And Future Work

As a future work, long range network layer protocols and application layer protocols will be specified and implemented as representative components in COSECASE. Their related connectors will also be added to the tool. Common connectors will be represented with a common name. Component and connectors will be grouped at specific toolbars. By defining and adding domain specific components and connectors, a user can model any kind of application by using COSECASE. For the purpose of this thesis, only uni-directional IoT connectors were implemented , meaning data flow is from the first component to the second one. As future work, bi-directional IoT connectors can be implemented. For this thesis IOT connectors are implemented with only two ports, as a future work one can add more than two ports and make multiport IoT connectors.

REFERENCES

[1] A. Dogru, *Component Oriented Software Engineering Modeling Language: COSEML*, pp. 4–7. Ankara: Middle East Technical University, 1999.

[2] T. Joseph, R. Jenu, A. K. Assis, S. K. V. A, S. P. M, and D. A. G, "Iot middleware for smart city," *IEEE International Symposium on Technologies for Smart Cities*, 2017.

[3] S. Ünal and A. Doğru, "A variability perspective for iot heterogeneity in smart cities," in *22nd International Conference: Emerging Trends and Technologies in Convergence Solutions (SDPS 2017)* (L. Jololian, D. E. Robbins, and S. L. Fernandes, eds.), pp. 38–44.

[4] S. Al-Sarawi, M. Anbar, K. Alieyan, and M. Alzubaidi, "Internet of things (iot) communication protocols: Review," in *ICIT 2017 Internet of Things* (IEEE, ed.), pp. 685–690.

[5] M. C. Kaya, M. S. Nikoo, S. Suloglu, B. Tekinerdogan, and A. H. Dogru, *Managing Heterogeneous Communication Challenges in the Internet of Things Using Connector Variability*, pp. 127–149. Cham: Springer International Publishing, 2017.

[6] A. H. Dogru, "Component oriented software engineering modeling language: Coseml."

[7] A. Dogru and M. Tanık, "A process model for component-oriented software engineering," *IEEE Software*, vol. 20, pp. 34–41, 2003.

[8] D. Evans, "The internet of things how the next evolution of the internet is changing everything."

[9] F. V. den Abeele, J. Hoebeke, I. Moerman, and P. Demeester, "Integration of heterogeneous devices and communication models via the cloud in the constrained internet of things," *International Journal of Distributed Sensor Networks*, 2015.

[10] C. M. Kozierok, *TCP / IP GUIDE.* No Starch Press, 2005.

[11] A. Çetinkaya, M. Çağrı Kaya, and A. Doğru, "Enhancing xcoseml with connector variability for component oriented development," in *21st International Conference: Emerging Trends and Technologies in Convergence Solutions (SDPS 2016)* (R. Juric and S. Güldal, eds.), pp. 120–125.

[12] A. K. Akanbi and M. Masinde, "Towards semantic integration of heterogeneous sensor data with indigenous knowledge for drought forecasting," in *Middleware Doctoral Symposium*, 2015.

[13] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, 07 2012.

[14] V. Aleksandrovics, E. Filicevs, and J. Kampars, "Internet of things: Structure, features and management," *Information Technology and Management Science*, vol. 19, pp. 78–84, 2017.

[15] A. Gerber, "Connecting all the things in the internet of things," *IBM developer works*, vol. 10, p. 1, 2017.

[16] S. Ünal and A. Doğru, "A solution to iot heterogeneity problem with connectors in component based systems," in *24th International Conference: Emerging Trends and Technologies in Convergence Solutions (SDPS 2019).*

[17] J. Edwards and R. Bramente, *Networking Self- Teaching Guide: OSI, TCP/IP, LANs, MANs, WANs, Implementation, Management and Maintanence*, pp. 343–367. Indianapolis: Wiley Publishing, 2009.

[18] I. Cisco Systems, *Internetworking Technologies Handbook*, pp. 5–30. Indianapolis: Cisco Press, 2003.

[19] S. Al-Sarawi, M. Anbar, K. Alieyan, and M. Alzubaidi, "Internet of things (iot) communication protocols: Review," *ICIT 2017 - 8th International Conference on Information Technology, Proceedings*, pp. 685–690, 2017.

[20] T. Salman and R. Jain, "Networking protocols and standards for internet of things," *Internet of Things and Data Analytics Handbook*, pp. 215–338, 2017.

[21] S. W. Kum and M. goo Kang, “Iot delegate: Smart home framework for heterogeneous iot service collaboration,” *KSII Transactions on Internet and Information Systems*, vol. 10, pp. 3958–3971, 2016.

[22] I. Sommerville, *Component Based Software Engineering*, pp. 453–477. Boston: Pearson, 2011.

[23] A. Dogru, *Component Oriented Software Engineering*, pp. 139–146. USA: The-ATLAS Publishing, 2006.

[24] E. M. Dashofy, N. Medvidovic, and R. N. Taylor, *Software Architecture: Foundations, Theory, and Practice*. New Jersey: Jonh Wiley & Sons, 2009.

[25] N. R. Mehta, N. Medvidovic, and S. Phadke, “Towards a taxonomy of software connectors,” *ACM*, pp. 178–187, 2000.

[26] B. Hammi, R. Khatoun, S. Zeadally, A. Fayad, and L. Khoukhi, “Future technologies for smart cities,” *Bulletin of the Moscow State Regional University (Economics)*, pp. 100–114, 2018.

[27] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, “Internet of things for smart cities,” *IEEE Internet of Things Journal*, vol. 1, pp. 22–32, 2014.

[28] A. G. M. Mohmmed and S. E. F. Osman, “Smart city & internet of things,” *International Research Journal of Computer Science (IRJCS)*, vol. 04, pp. 238–242, 2017.

[29] T. D. Vylder, “Feature modelling: A survey, a formalism and a transformation for analysis.”

[30] N. Noy, “Semantic integration: A survey of ontology-based approaches,” *Newsletter ACM SIGMOD Record*, vol. 33, pp. 65–70, 2004.

[31] T. Teixeira, S. Hachem, V. Issarny, and N. Georgantas, “Service oriented middleware for the internet of things : A perspective,” in *Proceedings of the 4th European conference on Towards a service- based internet*, pp. 220–229.

[32] A. Kazmi, Z. Jan, A. Zappa, and M. Serrano, “Overcoming the heterogeneity in the internet of things for smart cities,” *Springer International Publishing AG*, pp. 20–35, 2017.

[33] A. Gerber, “Choosing the best hardware for your next iot project use this hardware guide to determine which hardware to use when prototyping and developing iot projects,” *IBM developer works*, pp. 1–11, 2017.

[34] A. Triantafyllou, P. Sarigiannidis, and T. D.Lagkas, “Network protocols, schemes, and mechanisms for internet of things (iot): Features, open challenges, and trends,” *Wireless Communications and Mobile Computing*, vol. 2018, pp. 1–24, 08 2018.

[35] S. Madakam, R. Ramaswamy, and S. Tripathi, “Internet of things (iot): A literature review,” *Journal of Computer and Communications*, pp. 164–173, 05 2015.

[36] R. Joshi, S. Mellor, and P. Didier, “The industrial internet of things volume g5: Connectivity framework,” *Industrial Internet Consortium (IIC)*, pp. 1–127, 2017.

[37] P. Lea, *Internet Of Things For Architects*, pp. 110–305. Birmingham: Packt Publishing, 2018.

[38] Cypress, “Bluetooth low energy (ble).”

[39] R. Davidson, Akiba, C. Cufi, and K. Townsend, *Getting Started with Bluetooth Low Energy*, pp. 10–110. Boston: Artech House, 2014.

[40] J. Lindh, “Bluetooth low energy beacons.”

[41] R. . Schwarz, *R&S SMBV-K60/K117 Bluetooth Enhanced Data Rate, Bluetooth 5.0 User Manual*, pp. 13–93. München: Rohde & Schwarz GmbH & Co., 2017.

[42] S. Farahani, *ZigBee Wireless Networks and Tranceivers*, pp. 1–23. Burlington: Elsevier, 2008.

[43] J. Wright and J. Cache, *Hacking Exposed Wireless*, pp. 399–410. New York: McGraw-Hill Education, 2015.

[44] M. Gast, *802.11 Wireless Networks*, pp. 12–100. Sebastopol: O’Reilly Media, 2009.