Architecture Design

* *IoT anyware* -



Team number 1

June 2015

Document History

|  |  |  |
| --- | --- | --- |
| **Rev.** | **Description** | **Date** |
| 0.0 | Creation | 07/06/2015 |
| 0.1 | Section 1, 2 | 10/06/2015 |
| 0.9 | Refine all sections | 17/06/2015 |
|  |  |  |

**TABLE OF CONTENTS**

[1. DOCUMENT DESCRIPTION 3](#_Toc422355519)

[1.1 The Purpose of Document 3](#_Toc422355520)

[1.2 The Document Organization 3](#_Toc422355521)

[1.3 Terminology and Definitions 4](#_Toc422355522)

[1.4 References and Relevant Documents 4](#_Toc422355523)

[2. PROJECT OVERVIEW 4](#_Toc422355524)

[2.1 The Purpose of Project 4](#_Toc422355525)

[2.2 The Scope of Project 4](#_Toc422355526)

[2.3 The Stakeholders 5](#_Toc422355527)

[3. ARCHITECTURAL DRIVERS 5](#_Toc422355528)

[3.1 High Level Functional Requirements 5](#_Toc422355529)

[3.2 Quality Attributes 7](#_Toc422355530)

[3.3 Constraints 9](#_Toc422355531)

[4. SYSTEM CONTEXT 9](#_Toc422355532)

[5. 1st DECOMPOSITION OF IOT SYSTEM 11](#_Toc422355533)

[5.1 Dynamic Perspective 11](#_Toc422355534)

[6. DECOMPOSITION OF IOT SERVER 13](#_Toc422355535)

[6.1 Dynamic Perspective 13](#_Toc422355536)

[6.1.1 1st Decomposition of IoT Server 13](#_Toc422355537)

[6.1.2 2st Decomposition of IoT Server 15](#_Toc422355538)

[6.1.3 3st Decomposition of IoT Server 17](#_Toc422355539)

[6.2 Static Perspective 20](#_Toc422355540)

[6.3 Physical Perspective 24](#_Toc422355541)

[7. DECOMPOSITION OF USER APP 26](#_Toc422355542)

[7.1 Dynamic Perspective 26](#_Toc422355543)

[7.2 Static Perspective 28](#_Toc422355544)

[8. DECOMPOSITION OF SA NODE 30](#_Toc422355545)

[8.1 1st Decomposition of SA node (Static Perspective) 31](#_Toc422355546)

[8.2 2nd Decomposition of SA Network Manager (Dynamic Perspective) 33](#_Toc422355547)

[9. SYSTEM DESIGN SUPPLEMENT 36](#_Toc422355548)

1. DOCUMENT DESCRIPTION

## 1.1 The Purpose of Document

The purpose of this document is to describe the software architecture design of the “Internet of Things” infrastructure. This project is the part of the course work of the LG Architecture Training Program at Carnegie Melon University. This document contains the architectural drivers, and architecture design of the system, and design decisions driven by the architectural drivers. It also provides the interfaces and protocols between system components in detail for the implementers to work in parallel.

# 1.2 The Document Organization

The order of presentation is chosen to convey the design of the system in top down approach. First, section 2 introduces the project context to give a general understanding of the system. The summary of the architectural drivers are presented in section 3 to elaborate what are important factors that define the design choices. In section 4, the system context is described; explaining the boundary of the system, and system entities interacts with the environment. Each entity is decomposed to the level where all significant functional requirements and quality attributes are applied to the design through section 5 to 8. In section 9 to 11, the behavior of the system is specified.

This document is organized as follows.

* Section 2 - Project Overview

-

* Section 3 - Architectural Drivers

-

* Section 4 - System Context

-

* Section 5 - 1st Decomposition

-

* Section 6 - Decomposition of IoT Server

-

* Section 7 - Decomposition of SA node

-

* Section 8 - Decomposition of User App

-

* Section 9 - System design supplement

-

## 1.3 Terminology and Definitions

This section defines acronyms and terms that used in this document.

|  |  |
| --- | --- |
| Terminology | Definition |
| IoT | Internet of things |
| SA | Sensor/actuator |
| SA node | A device consist of sensors and/or actuators, and a network module communicates with the IoT system. |
| QA | Quality attribute |
| Sensor/actuator profile | A characteristic and/or capability description of the sensor or actuator. |
| SA node description | A description of the SA node; including names of sensors and actuators, and their profiles. |

## 1.4 References and Relevant Documents

[1] <http://en.wikipedia.org/wiki/Internet_of_Things>

[2] Team1\_01\_Architecture\_Driver\_Specifications\_v1.0.doc

[3] Team1\_03\_Event Bus Interface Specification\_v1.0.doc

[4] Team1\_04\_RESTful Web API Specification\_v1.0.doc

2. PROJECT OVERVIEW

## 2.1 The Purpose of Project

The Internet of things refers to the network of devices equipped with electronics, software, sensors, and connectivity to provide the greater values or services by exchanging data with the manufacturer, service provider, and/or other connected devices[1]. The purpose of this project is to build the IoT infrastructure which meets the following key goals.

* Create an IoT infrastructure to support accessing sensors and actuators installed in the home or business.
* Create an infrastructure to provide an ecosystem to develop cost competitive home or business IoT products for value-added-resellers and other 3rd party hardware and software application developers, service providers, and installers and maintainers.
* Build a basic data centric infrastructure to provide IoT data sets for developers to create future data mining, analytic operations, and services.

## 2.2 The Scope of Project

The scope of the project is to design and build the IoT infrastructure. The IoT infrastructure is consist of three major parts; the IoT server, the SA node, and the user application. The IoT server provides services - event publication and subscription, and secure access to the connected devices or the data collection for SA nodes and user applications. The SA node can post sensor states or notifications, and/or retrieve actuator control messages or any events from other devices, systems, or users for extended services through the IoT server. In the other hand, the user application can post actuator control event, retrieve sensor states or notifications from the SA node, and/or access the collection of data through the IoT server.

## 2.3 The Stakeholders

The stakeholders of the project are customers, and IoT business interest group such as sensor/actuator producers, home builders, third-party service providers, application developers, etc, and the project management. Functional requirements are extracted from the use cases of customers or the IoT business interest group, and the quality attributes are derived from the requirements. Business and/or technical constrains, the project planning and the priority of the QAs are discussed and confirmed with the project managements.

3. ARCHITECTURAL DRIVERS

The aim of this section is to describe the architectural drivers of the IoT project: high level functionality in the form of use cases, the summary of quality attributes, and constraints. Refer to Team1\_ADS.pdf[2] for more detail on the architectural drivers of the project.

## 3.1 High Level Functional Requirements

The key functional requirements of the system are as follows.

* The system should provide secured IoT services.
* The system enables the user to control actuators, and to determine sensor states.
* The system should collect user command and sensor states, and present them to the rightful customers.

Below are the functional requirements of the system in more detail.

|  |  |
| --- | --- |
| **Access secured services** | FR01 |
| **Description**:  User accesses the system in secured environment. User must login to the system for services. Unauthorized persons are not allowed to control sensors installed in home, register SA nodes, or access any data gathered in the system. | |

|  |  |
| --- | --- |
| **Discover SA nodes** | FR02 |
| **Description**:  User queries home to find out how many nodes are installed and what sensors/actuators are installed on each node. | |

|  |  |
| --- | --- |
| **Determine sensors and control actuators** | FR03 |
| **Description**:  User can determine the temperature/humidity, turn on and off lights, open and close the door, turn on the alarm, and determine if anyone is home. However, user must set the alarm off prior to opening the door. | |

|  |  |
| --- | --- |
| **Log user commands and sensor values** | FR04 |
| **Description**:  User commands and sensor values are stored in IoT infrastructure for some period of time. This data set can be utilized by developers to create future data mining, analytic operations, and services. | |

|  |  |
| --- | --- |
| **Send emergency message** | FR05 |
| **Description**:  An emergency message is sent to the user when door is opened manually or the house is suddenly occupied while alarm is set. | |

|  |  |
| --- | --- |
| **Lock house automatically** | FR06 |
| **Description**:  User is informed upon the vacancy of house and asked to lock the house. If the home is vacancy for 30 seconds and alarm is not set, SA node notifies user. If the user failed to respond to the message within 5 minutes, the door is closed, and the alarm is set automatically. | |

|  |  |
| --- | --- |
| **Turn off light automatically** | FR07 |
| **Description**:  When no one is home for 10 minutes, the light is turned off automatically. | |

|  |  |
| --- | --- |
| **Register SA node** | FR08 |
| **Description**:  Authorized user adds nodes to the system. Equipped sensors and actuators are recognized. | |

|  |  |
| --- | --- |
| **Unregister SA node** | FR09 |
| **Description**:  Authorized user removes nodes from the system. | |

|  |  |
| --- | --- |
| **Grant SA node access permission** | FR10 |
| **Description:**  User who registered a SA node is the owner of the node. He/she grants an access permission of the node to others. However, only owner of the node can unregister the node. | |

|  |  |
| --- | --- |
| **Transfer SA node ownership** | FR11 |
| **Description:**  User who registered a SA node is the owner of the node. He/she can give up the ownership of the node, and transfer the ownership to other user. The user who gives up the ownership still has the access permission of the node. | |

## 3.2 Quality Attributes

Quality attributes that the system must promote in the architecture design are:

* The system must identify unauthorized accesses, and protect data and services from it.
* The system should not allow users to register SA nodes that are not owned by them.
* The system should identify the failure of SA nodes.
* The system should make it easy to add emerging protocols.
* The system should make it easy for application developers (private persons, VARs, or other 3rd parties) to build custom apps, services, and/or make mashups from existing available services.

One of the most important quality attribute of the IoT system is the security, since the security breach of the system can lead to physical damages of people. The system must provide best efforts to prevent the malicious use of the system. Only authorized users should be able to control or register the SA nodes of which they have permission.

The essential quality attribute of the IoT system is the availability. The failure of the SA node can cripple the quality of entire system services. The result of this drawback can be lost productivity, lost revenue, damaged customer relationships, bad publicity, and lawsuits. If a mission-critical sensor or actuator becomes unavailable, the user could be placed in jeopardy.

The IoT fragmentation will grow until one standard, ecosystem or technical model dominates the market. For this reason, our IoT system has to be flexible enough to adapt emerging technologies or protocols. In addition, the system should provide an easy to use development environment for 3rd party service providers or application developers in order to build the solid IoT ecosystem to become a leader in the industry.

Five out of nine quality attributes are prioritized as high as shown below.

|  |  |  |  |
| --- | --- | --- | --- |
| **Total** | **High Priority** | **Medium Priority** | **Low Priority** |
| 9 | 5 | 4 | 0 |

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Priority** | **Quality Attribute** | **Descriptions** |
| QA01 | H | Security | Hackers or malicious people try to break into the system. When unauthorized user attempts to login to the system, the system maintains the audit trail. If the attempt is repeated more than 5 times, the account is locked, and the source of tempering is identified. |
| QA02 | H | Security | Hackers or malicious people try to register the SA node that is not owned by them. When unauthorized user attempts to register the SA node that he/she doesn’t own, the system maintains the audit trail, and cancel the registration in 10 minutes. |
| QA03 | H | Availability | SA node can crash, hang, or be disconnected from the network for various reasons. If SA node is inoperable or out of reach, the system should be aware of such events, and notify user within 2 minutes. |
| QA04 | M | Availability | SA node can be disconnected from the network for various reasons. If SA node is not able to reach the system due to network failure, it should store recent event at least for one day. When the network is restored, SA node should send the logs to the system. |
| QA05 | M | Scalability | The number of SA node user can be more than one. The system should be able to serve 10 user controls to the same SA node. (Concurrent access and control is not considered in this scenario) |
| QA06 | M | Scalability | More than one SA node can be installed at home. The system should be able to support at least 100 nodes concurrently. |
| QA07 | H | Modifiability | The system should make it easy to add emerging protocols (eg. Bluetooth 802.15, ZigBee 802.15.4) to the system. Average skilled developers should be able to implement it within two months. |
| QA08 | M | Usability | The system should make it easy for users to register or unregister SA nodes. Ordinary user should be able to register or unregister the node within 5 minutes by following the provided manual. |
| QA09 | H | Extensibility | The system should make it easy for application developers (private persons, VARs, or other 3rd parties) to build custom apps, services, and/or make mashups from existing available services. Average skilled developers should be able to build the application in six months. |

## 3.3 Constraints

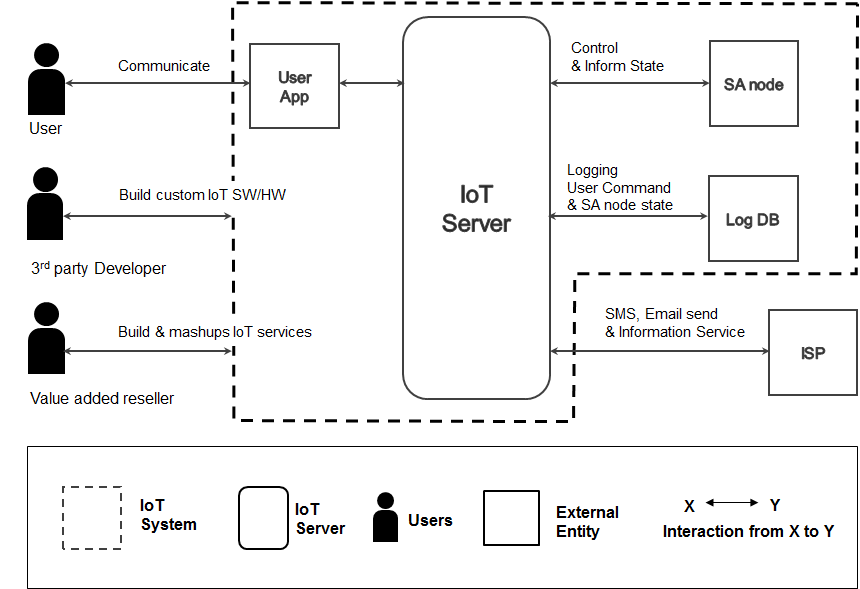
Major risk that the project team confronts is the budget of the project. The team is consist of five developers; very new to the development environment such as JAVA, Python, and Arduino. However, the project has to be completed in 35 days to outpace competitors.

The business and technical constrains of the project are organized in the table below.

|  |  |  |
| --- | --- | --- |
| **ID** | **Type** | **Description** |
| TC01 | Technical | JAVA compiler, Arduino 1.0.6 is preferred. |
| TC02 | Technical | Permissible languages for this system (excluding the SA nodes) are JAVA and Python. |
| TC03 | Technical | 802.11 is only supported in the system. |
| TB01 | Business | Development period: 5 weeks (3 hours/day) |
| TB02 | Business | Development team: 5 developers. |

4. SYSTEM CONTEXT

Figure 4.1 System context shows the boundary of the system, and system entities interacts with the external entities. End users communicate with sensors and actuators installed in the home or business via any mobile device (laptop, phone, and tablet) or desktop system connected to the Internet. The system supports an ecosystem of developers, value-added-resellers (VARs), and other 3rd party hardware and software application developers, service providers, and installers and maintainers. A basic data centric infrastructure supports the collection of data that would enable developers to create future data mining, analytic operations, and services. The *IoT Server* interacts with external information-service-providers (ISP) to send messages, or retrieve valuable information such as weather casts or local traffic information.

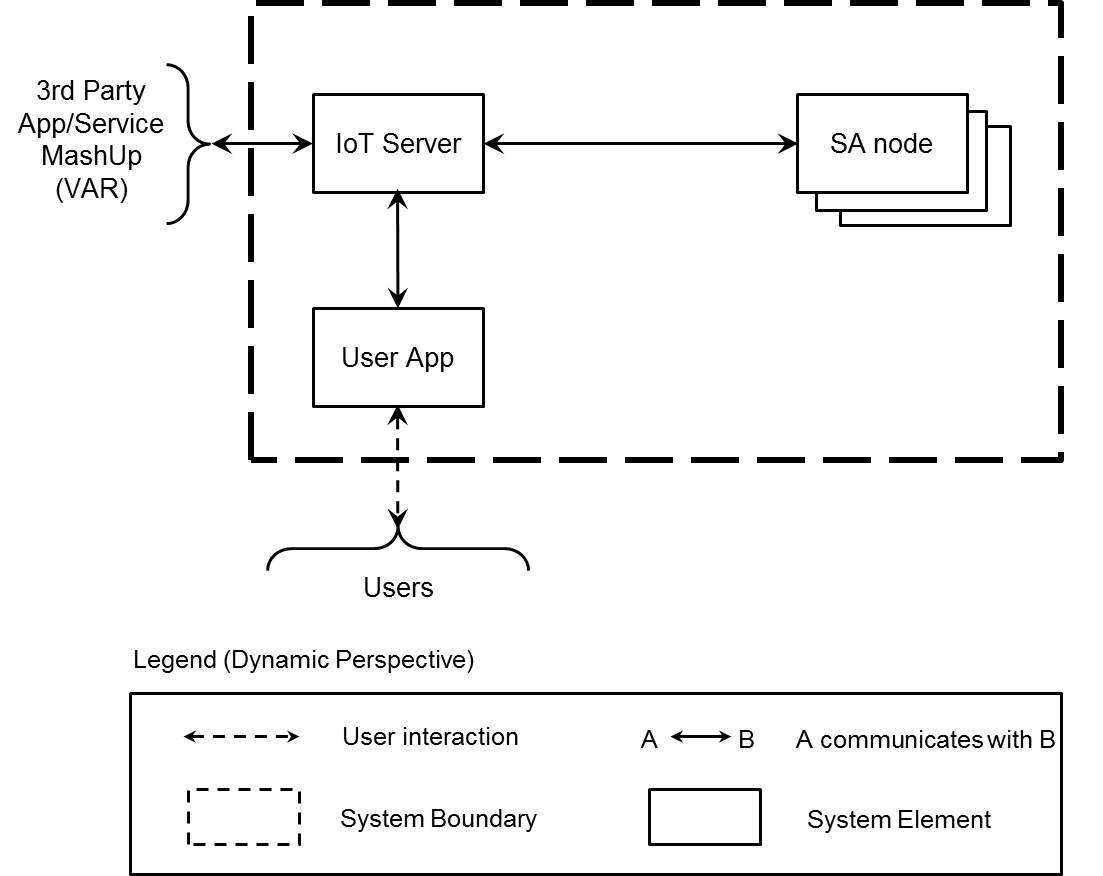


**Figure 4.1 System context**

5. 1st DECOMPOSITION OF IOT SYSTEM

## 5.1 Dynamic Perspective

In this section, we will closely look into *IoT System* that was described in the system context chapter 4 from the context diagram (Figure 4.1). Figure 5.1 shows decomposition of the *IoT System* from system context. The system decomposition in detail is from dynamic perspective. Dynamic perspective shows elements and relationships, it can just show the elements and relationship of the system at a single instant in time, or all elements and relationships at any time during execution. The *IoT System* is compromised with *IoT Server*, *SA nodes*, and *User App***.** The responsibilities and relationship of each element are described in Table 5.1 and 5.2.



**Figure 5.1 First level of decomposition - Dynamic perspective of IoT system**

**Table 5.1 Element Responsibility Catalog for First-Level Decomposition**

|  |  |
| --- | --- |
| **Perspective:** Dynamic  **Associated Drawings:** Figure 5.1 | |
| **Element** | **Responsibilities** |
| *IoT Server* | * This element (*IoT Server*) is a composition of elements providing IoT service to *User App* and *SA nodes*. These elements are described in each section 6. * *User App, SA nodes*, and 3rd party app/service interacts with the *IoT server* in secured environment. The *IoT Server* authenticates and authorizes them to access services. |
| *User App* | * This element is an application enables end users to access *SA nodes* via IoT system, and/or provide IoT services to the users. * *User App* can be installed on various devices (mobile, laptop, PC) * *User App* may have GUI interface to interact with user. |
| *SA node* | * This element is installed at home to control SA (sensors and actuators). * *SA node* has to have an ability to access the Internet. * *SA node* might be one or more to support various services. |

**Table 5.2 Relationship Responsibility Catalog for First-Level Decomposition**

|  |  |
| --- | --- |
| **Perspective:** Dynamic  **Associated Drawings:** Figure 5.1 | |
| **Relationship** | **Responsibilities** |
| B  A | * This relationship is used to connect *SA node*, *User App* and 3rd party App/Service to the *IoT System*. This symbol indicates that an element A connects to an element B over a network connection. |
|  | * This relationship indicates interaction between end user and *User App* |

**Table 5.3 First Level of Server Decomposition Rationale**

|  |
| --- |
| **Perspective:** Dynamic  **Associated Drawings:** Figure 5.1  **Associated Responsibilities:** Tables 5.1 and 5.2 |
| This decomposition focuses on the IoT system. The IoT system is decomposed into 3 major parts; *IoT server, SA node, and User App*. The system provides IoT services to end users, and communicates with *SA node* to control sensors and actuators located at home. *User App* offers the opportunity for users to remotely control their devices. |

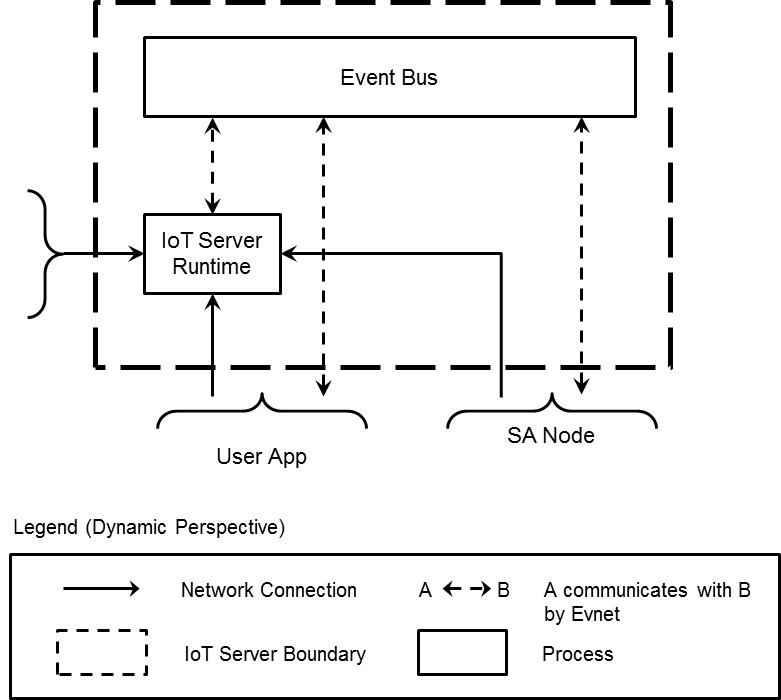
6. DECOMPOSITION OF IOT SERVER

In this section, we will closely look into *IoT Server* that was described in the system context chapter 4 from the context diagram (Figure 4.1).

## 6.1 Dynamic Perspective

## 6.1.1 1st Decomposition of IoT Server

The *IoT Server* is consist of two elements, such as IoT Server Runtime and Event Bus. The Event Bus provide common data channel between *User App* and *SA Node*. However, commonly Event Bus has limited authentication features which depend on id/password. In order to reinforce basic security functionality of event bus, the IoT Server Runtime provide more advanced authorization and authentication. The IoT Server Runtime manage user account with access control policy. And generate time-limited session, which required to access Event Bus.



**Figure 6.1 First level of decomposition - focusing on the decomposition of the Server**

**Table 6.1 Element Responsibility Catalog for First-Level Decomposition**

|  |  |
| --- | --- |
| **Perspective:** Dynamic  **Associated Drawings:** Figure 6.1 | |
| **Element** | **Responsibilities** |
| IoT Server Runtime | * This element is a process or collection of processes. * Authorization and authentication to access the event bus element. * Managing user configurations and register, unregister and transfer SA node. * Managing rules which related with external services. Determine and send action event when rule is triggered. * Sending push notification with SMS / Email. * Request query to external services. * Logging sensor status and all request within IoT Server Runtime. * Providing logged data which belongs to user. |
| Event bus | * This element is a process or collection of processes. * Coordinate message passing between SA node and User App element. |

**Table 6.2 Relationship Responsibility Catalog for First-Level Decomposition**

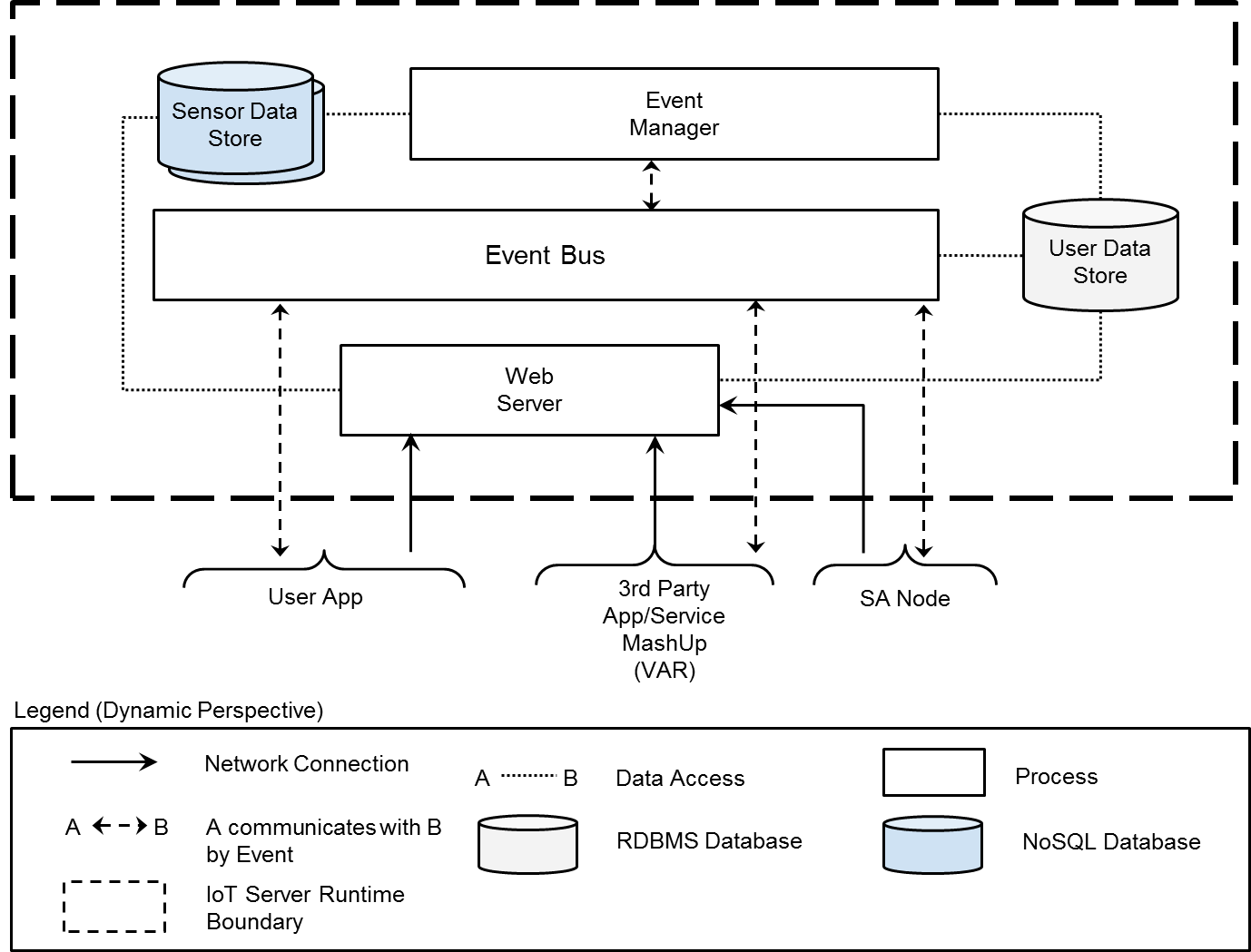
|  |  |
| --- | --- |
| **Perspective:** Dynamic  **Associated Drawings:** Figure 6.1 | |
| **Relationship** | **Responsibilities** |
| B  A | * This relationship will be used to connect 3rd Party App/Service and User App and SA node to a *IoT Anyware System* deployed in the field. This symbol indicates that element A connects to element B over a network connection. The connector envisioned for *the IoT Anyware System* will be a TCP/IP, socket- oriented protocol for authentication and authorization. The physical network is not specified or implied. |
| B  A | * This relationship indicates that element A communicates with B by message event at runtime. |

**Table 6.3 First Level of Server Decomposition Rationale**

|  |
| --- |
| **Perspective:** Dynamic  **Associated Drawings:** Figure 6.1  **Associated Responsibilities:** Tables 6.1 and 6.2 |
| This decomposition focuses only on the server process and omits a decomposition of the others. This server was decomposed into an *IoT Server* Runtime software and an Event Bus. This partitioning was chosen to maintain separation of concerns between authorization and authentication the system and event message passing. (*QA01 Security*) This eases runtime add new SA node and allows for the easy addition of new kinds of SA node. (*QA09 Extensibility*) |

## 6.1.2 2nd Decomposition of IoT Server

WebServer는 security를 위한 사용자 인증 이외에, SA node 등록, 삭제, 공유와 같은 사용자에 대한 SA node 관리 기능을 담당 합니다. 해당 정보는 SA node 혹은 User App 에서 저장할 경우에, 초기화 혹은 경우에 정보를 유실할 수 있으며, 도난이나 유실, 변조에 대한 위험이 있기 때문에 서버에서 처리 합니다. WebServer와의 통신은 channel에 대한 안전성을 확보하기 위해서 HTTPS로 이루어 집니다. 등록된 사용자와 SA node의 정보는 관계형 Databse에 저장되며, Event Bus는 해당 정보를 참조해서 Event Bus로의 연결 요청에 대한 인증을 처리합니다. Event Bus에 대한 의존성을 느슨하게 하기 위해서 Event Bus에서 발생하는 Event에 대한 처리는 외부의 Event Manager에서 담당 합니다. Event Manager는 Event Bus에서 발생하는 Event 들에 대한 모니터링을 해서, Event에 대한 logging 등의 적절한 처리를 합니다. 발생되는 Event의 양이 폭발적으로 증가될 수 있기 때문에, scalability를 확보하기 위해서 NoSQL Database를 사용합니다. 해당 정보는 이후에 사용자가 log 조회 요청을 하는 경우에, Web Server를 통해서 전달이 됩니다.



**Figure 6.2 Second level of decomposition - focusing on the decomposition of the IoT Server Runtime.**

**Table 6.4 Element Responsibility Catalog for Second-Level Decomposition**

|  |  |
| --- | --- |
| **Perspective:** Dynamic  **Associated Drawings: Figure 6.2** | |
| **Element** | **Responsibilities** |
| Web  Server | * This element is a single process. * Authorization and authentication to access the event bus element. * Managing user configurations and register, unregister and transfer SA node. * Provide logged data which belongs to user. |
| Event Manager | * This element is a process or collection of processes. * Managing rules which related with external services. Determine and send action event when rule is triggered. * Sending push notification with SMS / Email. * Request query to external services. * Logging sensor status and all request within IoT Server Runtime. |
| Event Bus | * This element is a process or collection of processes. * Provide message passing within connected clients. (pub/sub) * Restrict client connection only to who has valid session. The session is generated by the Web Server and valid for limited time period. * Restrict sub/pub topics only to starts with client’s ID. In order to prevent malicious user could not pub/sub other topic. |
| Sensor Data Store | * The data store is a collection of files stored on a medium remote to the deployed IoT Anyware System. The data store is the repository for periodic generated sensor data and all kinds of listened event which include your control event to *SA node*. |
| User Data Store | * The data store is a collection of files stored on a medium remote to the deployed IoT Anyware System. The data store is the repository for the user information which include email address, mobile phone number to provide push notification. |

**Table 6.5 Relationship Responsibility Catalog for Second-Level Decomposition**

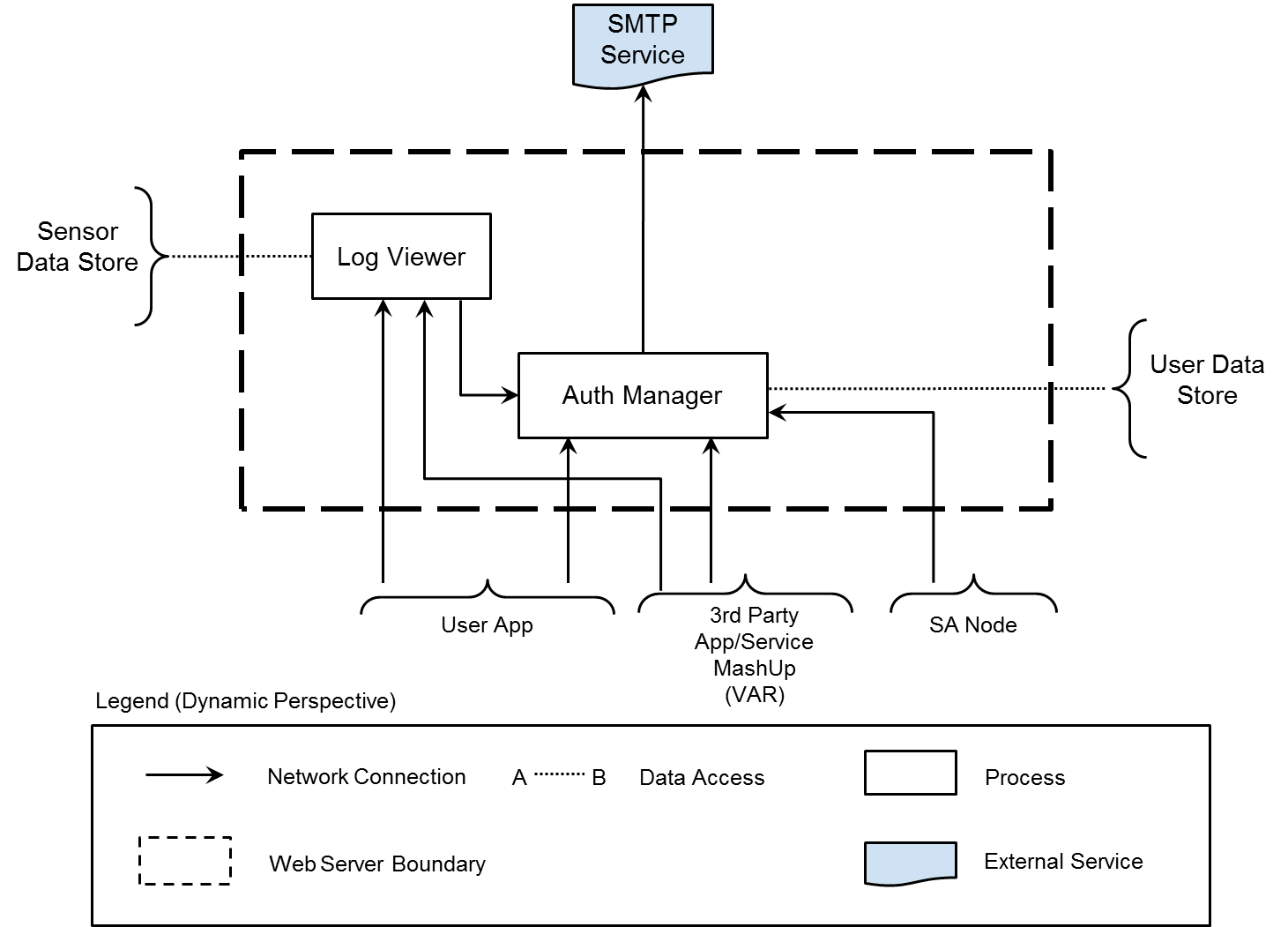
|  |  |
| --- | --- |
| **Perspective:** Dynamic  **Associated Drawings:** Figure 6.2 | |
| **Relationship** | **Responsibilities** |
| B  A | * This relationship will be used to connect 3rd Party App/Service and User App and SA node to a *IoT Anyware System* deployed in the field. This symbol indicates that element A connects to element B over a network connection. The connector envisioned for *the Web Server* will be a HTTPS. |
| B  A | * This relationship indicates that element A communicates with B by message event at runtime. |
| B  A | * This relationship indicates that element A CURD (Create / Update / Read / Delete) B. The connector will be a SQL query. |

**Table 6.6 Second Level of IoT Server Runtime Decomposition Rationale**

|  |
| --- |
| **Perspective**: Dynamic  **Associated Drawings**: Figure 6.2  **Associated Responsibilities**: Tables 6.4 and 6.5 |
| The rationale for this decomposition is to further divide the responsibilities assigned the IoT Server Runtime to elements into two separate element. The Web Server is responsible for access control and retrieve log data. This prevent an event bus from unauthorized access that will promote security communications, User App, SA node and 3rd Party venders. (*QA01 Security*) The Event Manager is responsible for all functionality which related to Event Bus, such as event logging, push notification and rule management.  The user and session information for the event bus will be stored in a RDBMS data store. The Web Server will be only one writer to this data store (with respect to the user data); the Event Bus will read the information, thus eliminating concurrent data access concerns. The sensor status information will be stored in a NoSQL data store. The Event Manager will be only one writer to this data store; the Web Server and Event Manager will read the this information, thus eliminating concurrent data access concerns. |

## 6.1.3 3rd Decomposition of IoT Server

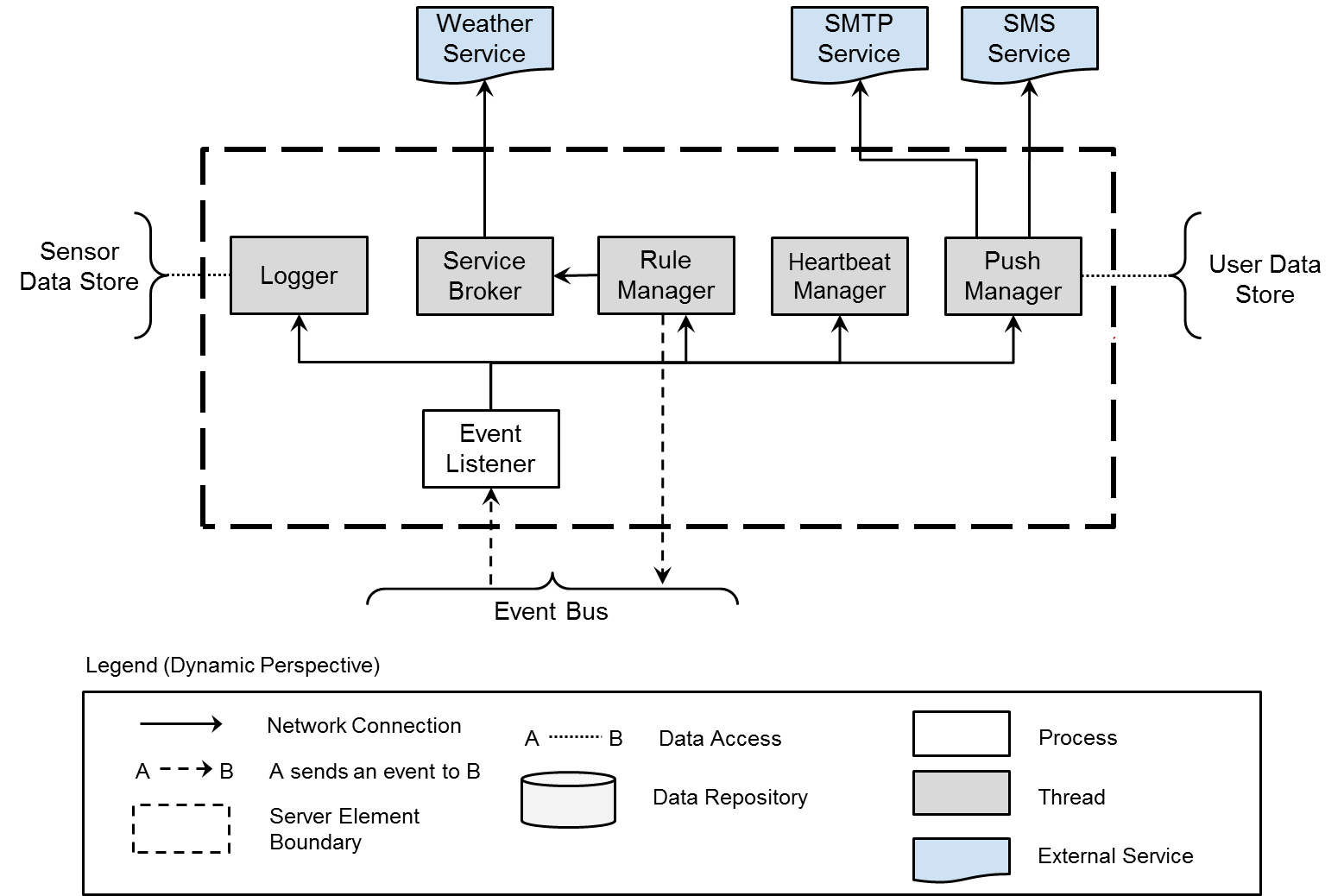
사용자에 대한 인증, SA Node 관리와 Log에 대한 조회는 서로 다른 Workload의 특성을 갖고 있기 때문에 해당 기능을 처리하는 모듈을 분리합니다. Log 조회는 1) 단방향으로 처리되는 Event가 아니라 응답이 필요한 Request/Response 형식이며, 2) Event Bus의 성능에 영향이 있을 수 있을 만큼 데이터 사이즈가 크기 때문에 Log Viwer에서 처리 합니다. Log Viewer는 log 조회 요청을 받으면, Auth Manager를 통해서 사용자가 설정한 logging timewindow와 사용자에게 등록된 SA node의 목록을 확인한 이후에, 관련된 log를 Sensor Data Store(NoSQL)에서 조회해서 결과를 응답합니다. Auth Manager 모듈은 시스템을 사용하려는 사용자에 대한 본인 인증을 위해, 사용자의 계정 등록 요청 시에 사용자의 email 주소로 인증 확인을 위한 link가 포함된 메일을 발송합니다. 해당 메일은 보안을 위해서 10분의 유효기간 이후에 만료됩니다.



**Figure 6.3 Third level of decomposition - focusing on the decomposition of the Web Server.**

Event Manager는 Event Server와의 연결을 생성합니다. 이후에 특정 topic에 대해서 처리를 위해서 Event handler thread들을 생성합니다. Event handler thread는 Event Manager에서 생성한 connection을 공유하며,

* Logger는 SA node에서 상태 변화 시에 발생하는 Event와, 사용자의 SA node에 대한 제어 Event에 대한 저장을 담당합니다.
* Rule Manager는 SA node와 관련된 rule을 관리하며, SA node의 현재 상태와, 외부 정보를 확인해서 일정 조건이 만족 될 경우에 SA node에게 제어를 요청 합니다. (i.e. 비가 내릴 것이 예보된 상황에서, SA node의 door가 열려 있는 상황). 관련해서 SA node의 내부 상태만으로 처리할 수 있는 rule에 대해서는, Server와 SA node의 연결이 끊어져도 일관성 있게 기능을 보장을 하기 위해서 SA node에 저장됩니다. 정보를 제공하는 service provider에 대한 의존성을 느슨하게 가져가기 위해서, 관련 처리는 Service Broker가 담당합니다.
* Heartbeat Manager는 SA node가 주기적으로 발생시키는 heartbeat를 관리하며, 3번 연속으로 heartbeat가 누락된 SA node에 대해서는 사용자에게 해당 내용을 전달 합니다.
* Push Manager는 SA node가 발생시킨 alarm에 대해서 사용자에게 전달 합니다. 발생된 SA node에 대한 alarm을 받을 사용자와 메시지 수신을 위한 상세 주소를 확인하기 위해서 User Data Store (RDBMS)를 통한 조회합니다. 사용자가 User App을 실행하고 있지 않은 경우에 대해서도 메시지를 전달 하기 위해서 Event Bus 이외의 Email/SMS를 추가로 사용 합니다.



**Figure 6.4 Third level of decomposition - focusing on the decomposition of the Event Manager.**

**Table 6.7 Element Responsibility Catalog for Third-Level Decomposition - focusing on the IoT Web Server.**

|  |  |
| --- | --- |
| **Perspective:** Dynamic  **Associated Drawings:** Figure 6.3 | |
| **Element** | **Responsibilities** |
| Auth Manager | * This element is a single process. * Authorization and authentication to access the event bus element. * Provide API for add new user. Response include two type of tokens, called access token and refresh token. The access token expires after some period of time. The refresh token never expires and this token used to refresh access token when access token is expired. * Provide API for retrieve user information. * Provide API for validate token. * Provide API for refresh token. * Provide API for revoke token. * Provide API for get logged sensor status and user access history within configured time windows. * Provide API for get/set logging time windows. * Provide API for register/unregister SA node. * Provide API for share/transfer SA node. |
| Log  Viewer | * This element is a single process. * Provide API for get logged sensor status and user access history within configured time windows. |

**Table 6.8 Element Responsibility Catalog for Third-Level Decomposition - focusing on the Event Manager.**

|  |  |
| --- | --- |
| **Perspective:** Dynamic  **Associated Drawings:** Figure 6.4 | |
| **Element** | **Responsibilities** |
| Event  Listener | * This element is a single process. * Instantiate Logger, Push Manager and Rule Manager * Relay subscribed event information. |
| Logger | * This element is a single thread. * Save event information to data store. |
| Push  Manager | * This element is a single thread. * Request sending push message to external push service provider. The recipients are users who registered SA node or shared from owner. |
| Rule  Manager | * This element is a single thread. * Managing rules which related with external services. Determine and send action event when rule is triggered. * Instantiate the Service Broker to query weather information. |
| Heartbeat Manager | * This element is a single thread. * Monitoring periodic heartbeat event from *SA node*. If server detects three consecutive missing heartbeat, notified it to user. |
| Service Broker | * This element is a single thread. * Request query to external weather service provider. |

**Table 6.9 Relationship Responsibility Catalog for Third-Level Decomposition**

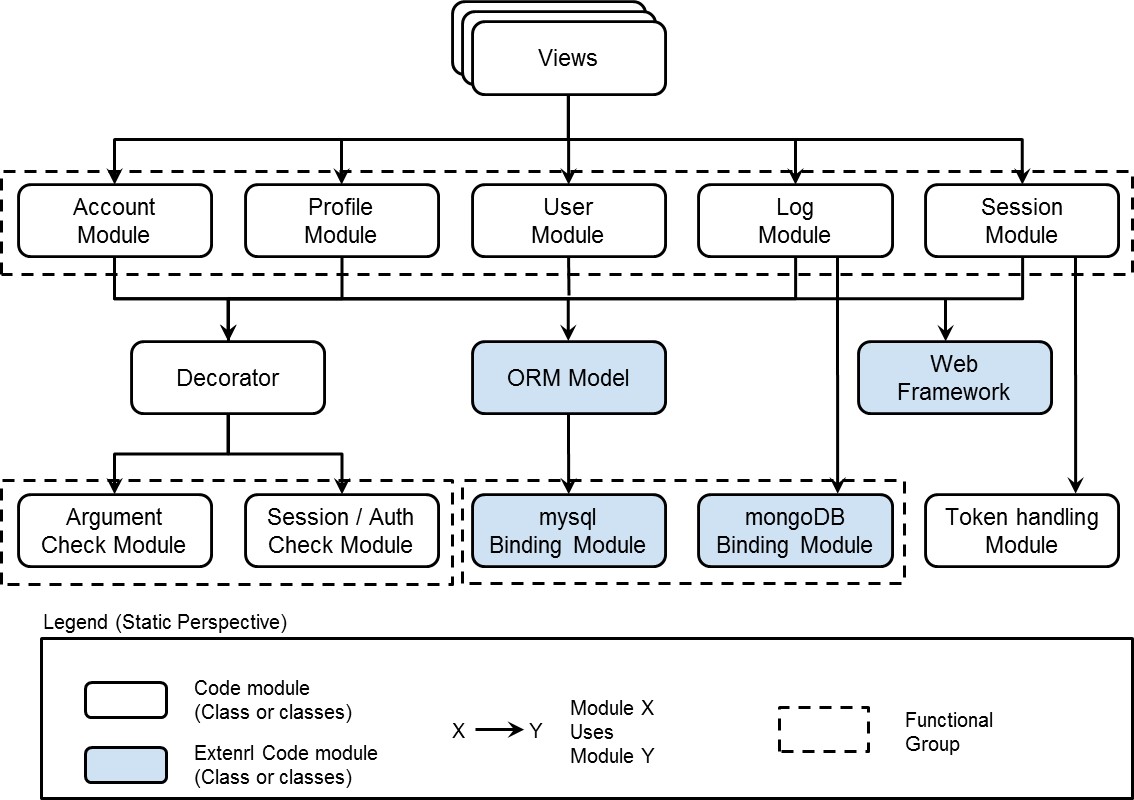
|  |  |
| --- | --- |
| **Perspective:** Dynamic  **Associated Drawings:** Figure 6.3 and 6.4 | |
| **Relationship** | **Responsibilities** |
| B  A | * This relationship will be used to connect 3rd Party App/Service and User App and SA node to an *IoT Anyware System* deployed in the field. This symbol indicates that element A connects to element B over a network connection. The connector envisioned for *the Web Server* will be a HTTPS. |
| B  A | * This relationship indicates that element A send event message to B at runtime. |
| B  A | * This relationship indicates that element A CURD (Create / Update / Read / Delete) B. The connector will be a SQL query. |

**Table 6.10 Third Level of IoT Web Server and Event Manager Decomposition Rationale**

|  |
| --- |
| **Perspective:** Dynamic  **Associated Drawings:** Figure 6.3  **Associated Responsibilities:** Tables 6.7, 6.8, and 6.9 |
| The rationale for this decomposition is to further divide the responsibilities assigned more separate element. The Log Viewer is responsible for read accumulated log and filter data which containing objects that match the given user and timestamp within period. Because the result of logged data could be large enough to inhibit performance of the event bus, we detached this responsibility from the event bus. The Logger is responsible for event logging which relate to user control event and device status change event. Because sensor generated data size could be exponentially growth, the gathered data of sensor status information will be stored in a NoSQL data store. NoSQL data store could scale out by adding new resource into system. *(QA06, Scalability)* The Push Manager is responsible for notification without using the event bus when user app is not presence. The Rule Manager is responsibility of handle pre-defined rules. These rules could be coordinate with external conditions such as weather, traffic and so forth. The Rule Manager instantiate corresponding Service Broker to request information. When rule is triggered, send message to the event bus to control appropriate SA node. |

## 6.2 Static Perspective

WebServer는 Web Framework를 사용해서, 서버에 대한 REST 요청과 응답을 처리합니다. REST 요청을 구성하는 API는 각각 Account, Profile, User, Log 그리고 Session Module로 책임을 나눕니다. 각각의 Module들이 공통으로 요청에 대해서 필요한 기능은 Decorator Module이 제공합니다. Decorator 모듈은 API에 정의된 mandatory 항목이 Header와 Body에모두 존재 하는 지와 해당 항목의 값이 유효한지를 확인하는 Argument Check Module과, 세션관리를 위한 Session / Auth Check Module로 구분이 됩니다. WebServer에서는 DB에 대한 의존성을 줄이기 위해서, DB에 저장되는 항목을 ORM Model로 추상화 해서 사용하며, ORM Model은 추상화된 class에 대해서 구체적인 SQL 언어로 변환해서 DBMS에 요청을 합니다.



**Figure 6.5 IoT Server class structure decomposition — switching to a static perspective and focusing on the structure of the Web Server.**

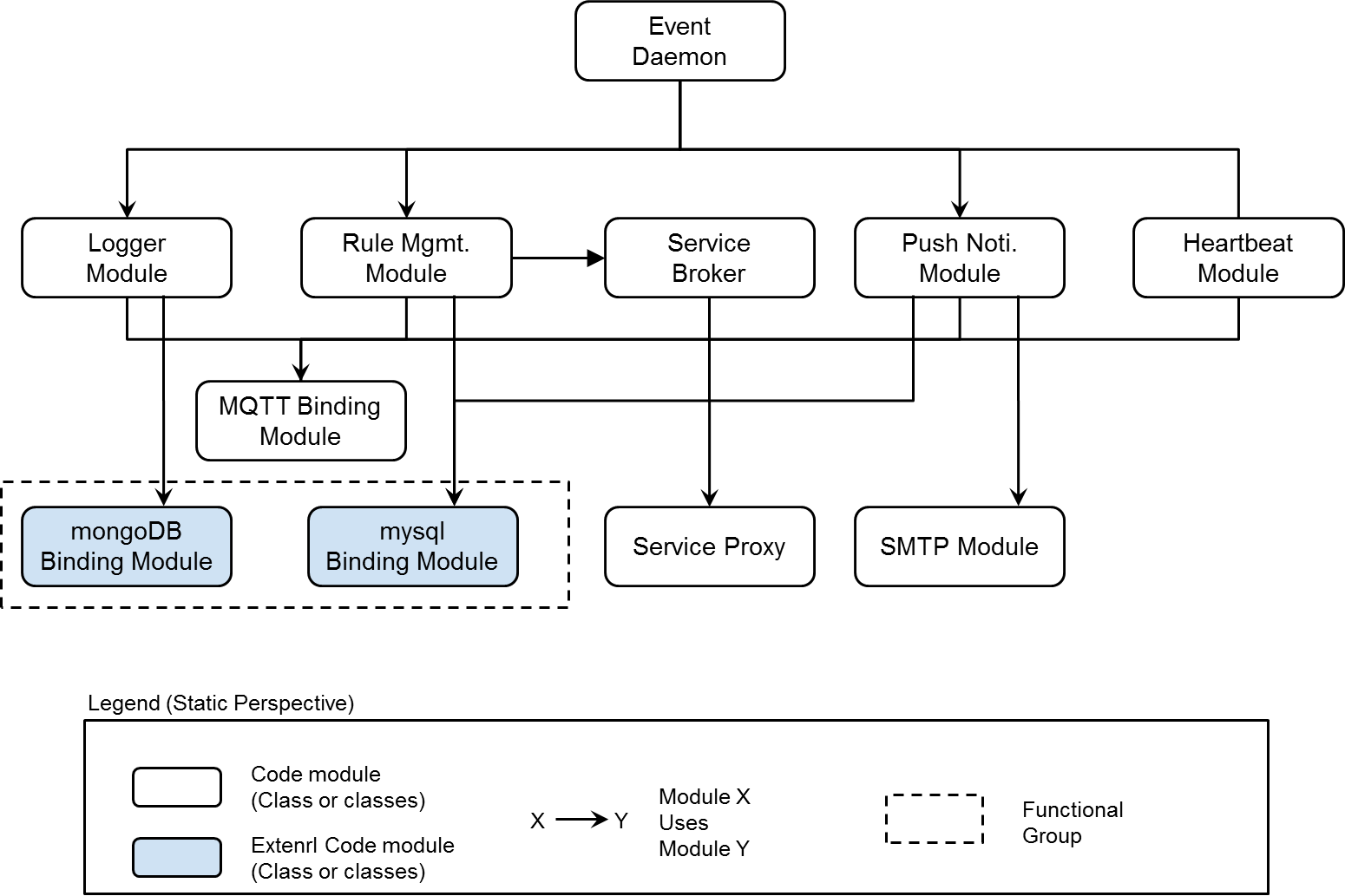
**Table 6.11 Element Responsibility Catalog for Static Decomposition**

|  |  |
| --- | --- |
| **Perspective:** Static  **Associated Drawings:** Figure 6.5 | |
| **Element** | **Responsibilities** |
| Web Framework | * Common structures and methods for web applications. * Common methods for sending and receiving via HTTP/HTTPS. * Provide MVC template engine. |
| Views | * Providing UI (User Interface) using static assets such as HTML/CSS/images. For dynamic contents, the template engine from the Web Framework would use. |
| Account Module | * Provide API for sign up. * Provide API for user identification (email confirmation) |
| Profile  Module | * Provide API for retrieve detail profile information. |
| User  Module | * Provide API for retrieve user information. * Provide API for register SA node. * Provide API for sharing and transfer SA node to other user. |
| Session  Module | * Provide API for validate token. * Provide API for refresh token. * Provide API for revoke token. |
| Log Module | * Provide API for get logged sensor status and user access history within configured time windows. |
| Decorator | * Common utility function for pre-request and post-request handling. |
| Argument Check Module | * Common utility function for check whether input argument (parameter) is valid or not. For example checking existence of mandatory parameter and its value. |
| Session / Auth Check Module | * Common utility function for check validity of input session and authentication of user app client. |
| ORM Model | * Common utility function for translate class definition with SQL query statement. |
| mysql Binding Module | * Common utility function for data transfer with the mysql database. |
| Token handling Module | * Common utility function for generating and management OTP token. |
| mongoDB Binding Module | * Common utility function for data transfer with the mongoDB database. |

**Table 6.12 Relationship Responsibility Catalog for Static Decomposition**

|  |  |
| --- | --- |
| **Perspective:** Static  **Associated Drawings:** Figure 6.5 | |
| **Relationship** | **Responsibilities** |
| X  Y  Module X  Uses Module Y | * This is traditional dependency. * One class depends on another if the independent class is a parameter variable or local variable of a method of the dependent class. |

Event Manager는 MQTT Binding Module을 사용해서, Event Bus에서 발생하는 Event를 수신하고, Event Bus에 새로운 Event를 발생 시킵니다. 사용자에 대한 정보를 조회할 필요가 있는 Push Notification Module과 Rule Management Module은 mysql Binding Module을 통해서 User Data Store에 접근을 하며, SA Node에서 발생하는 Event의 logging을 위해서 Logger Module은 mongoDB Binding Module을 사용해서 Sensor Data Store에 접근 합니다. Rule Management Module에서 외부 서비스의 정보를 조회하기 위해서 Service Broker를 사용하며, 의존성을 느슨하게 하기 위해서 각각의 서비스에 대한 상세 구현은 Proxy 모듈로 나뉩니다.



**Figure 6.6 IoT Server class structure decomposition — switching to a static perspective and focusing on the structure of the Event Manager.**

**Table 6.13 Element Responsibility Catalog for Static Decomposition**

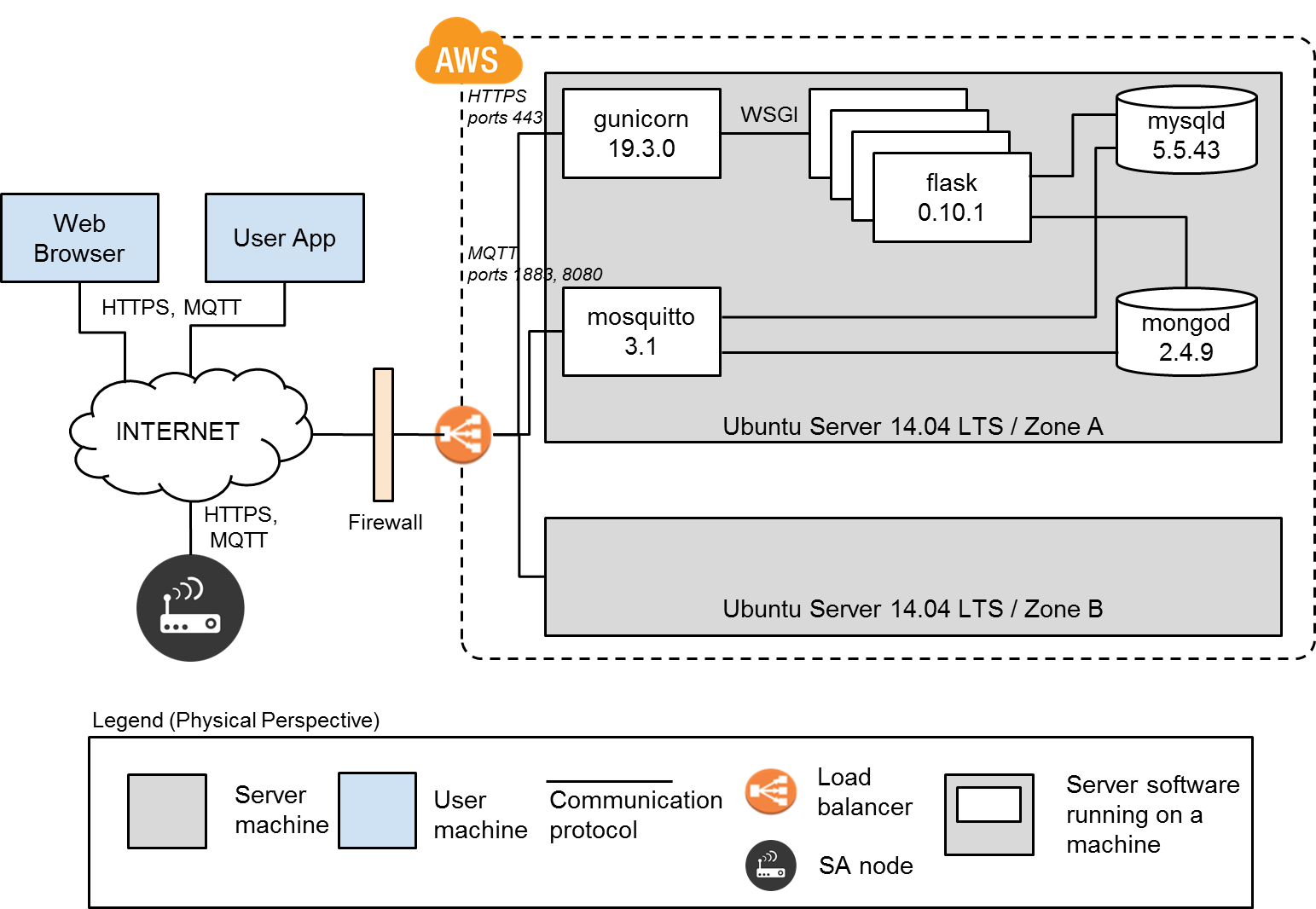
|  |  |
| --- | --- |
| **Perspective:** Static  **Associated Drawings:** Figure 6.6 | |
| **Element** | **Responsibilities** |
| Event Daemon | * Instantiating the event callback modules according to the input event message. |
| Logger  Module | * Save event information to database which relate to user control event and device status change event. |
| Rule Mgmt. Module | * Check input event whether event trigger condition is meet or not. * Instantiating the Service Broker to retrieve external information. |
| Service  Broker | * Provide API for retrieve information which related to external service. * Instantiating the Service Proxy. |
| Service Proxy | * Convert I/O data to relevant interface of external service. |
| Push Noti. Module | * Send request to external push service provider. The recipients are users who registered SA node or shared from owner. |
| Heartbeat Module | * Monitoring periodic heartbeat event from *SA node*. If server detects three consecutive missing heartbeat, notified it to user. |
| MQTT  Binding Module | * Common methods for publish and subscribe via message bus. * Common methods for register callback which related to subscribed topic. |
| mongoDB Binding Module | * Common utility function for data transfer with the mongoDB database. |
| mysql  Binding Module | * Common utility function for data transfer with the mysql database. |
| SMTP Module | * Composite SMTP mail and send it to user(s). |

**Table 6.14 Relationship Responsibility Catalog for Static Decomposition**

|  |  |
| --- | --- |
| **Perspective:** Static  **Associated Drawings:** Figure 6.6 | |
| **Relationship** | **Responsibilities** |
| X  Y  Module X  Uses Module Y | * This is traditional dependency. * One class depends on another if the independent class is a parameter variable or local variable of a method of the dependent class. |

## 6.3 Physical Perspective

The IoT Server infrastructure is serviced with Amazon Web Services (AWS). User App and Web Browser is connected to IoT Server via internet using MQTT and HTTPS protocol. Any port which not serviced is blocked by firewall. In order to support high availability and failover, IoT Server is deployed across multiple Availability Zones. Database also provisioned and maintained a synchronous standby replica in a different Availability Zone.



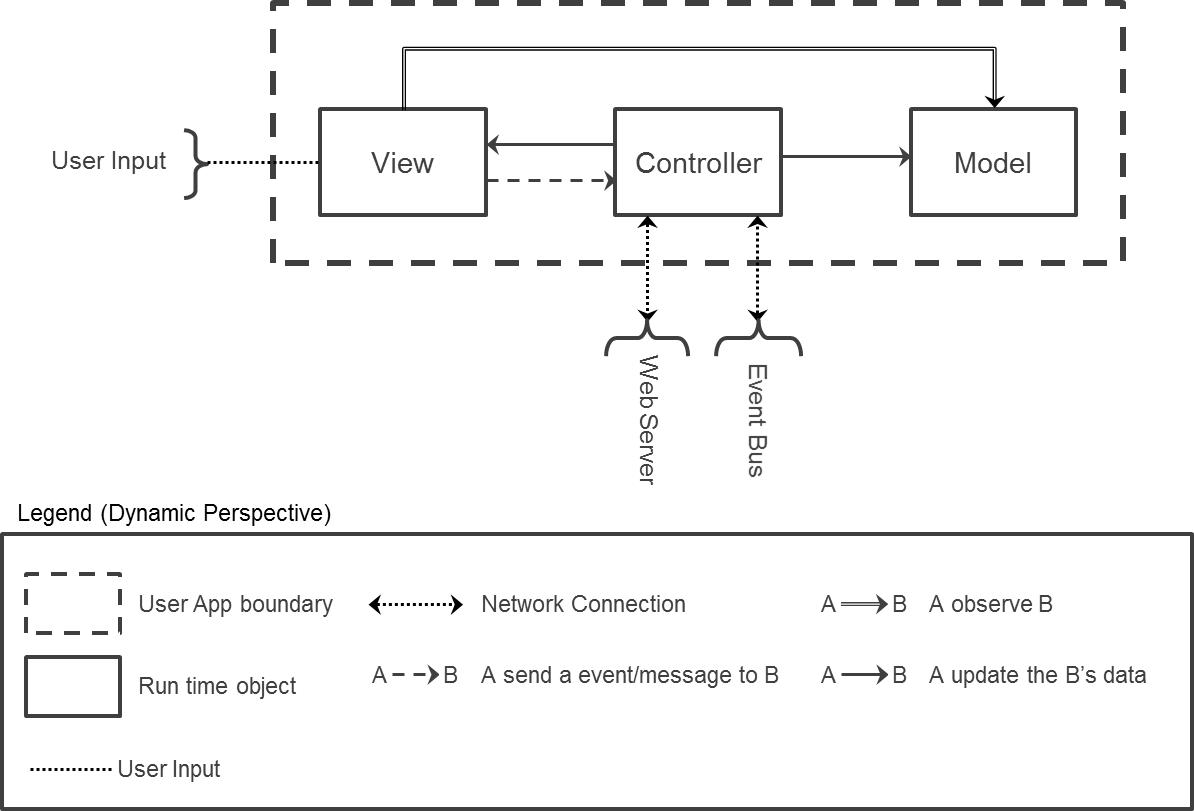
**Figure 6.7 IoT Server class structure decomposition — switching to a physical perspective.**

7. DECOMPOSITION OF USER APP

From the system context, *User App* provides the user interface to get a user input/action and interface with *IoT Server or SA node* through network. In this section we show the dynamic perspective (section 7.1) and static perspective (section 7.2) of *User App*. Through this section you can understand how to consist of User App. you can see the reason why we decompose into dynamic & static perspective like drawing of each section at the table of rationale.

## 7.1 Dynamic Perspective

*User App* consists of 3 components. One is *View* which is for the user interface. It shows the current status of SA nodes and some items what user can do. Another is controller which is interacting with user input and providing the interface with *IoT Server* and *SA node* as well. The other is model which represents the *SA node*. Figure7.2 is dynamic perspective of *User App* which is a part of *IoT system*. The general direction of the *User App* is conformed to MVC pattern. This pattern will provide the multi-view for 3rd application developer. And also, this pattern has been generalized and the advantage is known for well in the software world. In the Table 7.1, the reason of splitting is described briefly.



**Figure 7.1 1st decomposition of *User App***

**Table 7.1 Element Responsibilities catalog for 1st decomposition of User App**

|  |  |
| --- | --- |
| **Perspective:** Dynamic  **Associated Drawings:** Figure 7.1 | |
| **Element** | **Responsibilities** |
| *View* | * This element is run time object * This element shows the current information of SA nodes, or the action (or service) which user can do. * User can select action (or service) or enter the user data to interact with the system. * It observe the Model which represents the SA nodes. * This element is responsible for all of the GUI service. |
| *Controller* | * This element is run time object. * When user enters the user data or select the action (or service), it listen this user input. Then do the proper service like connect to server or publish the event to *SA node.* * It can select the view state by user input or event/message from network. * It is able to update the Model data when it needed. It is triggered by receiving network data from *IoT server* or *SA nodes.* |
| *Model* | * This element is run time object. * It represents a SA node. * It has the current status of SA nodes by controller. * When status was changed, it should send a event to View to notify status change. |

**Table 7.2 Relationship Responsibilities catalog for 1st decomposition of User App**

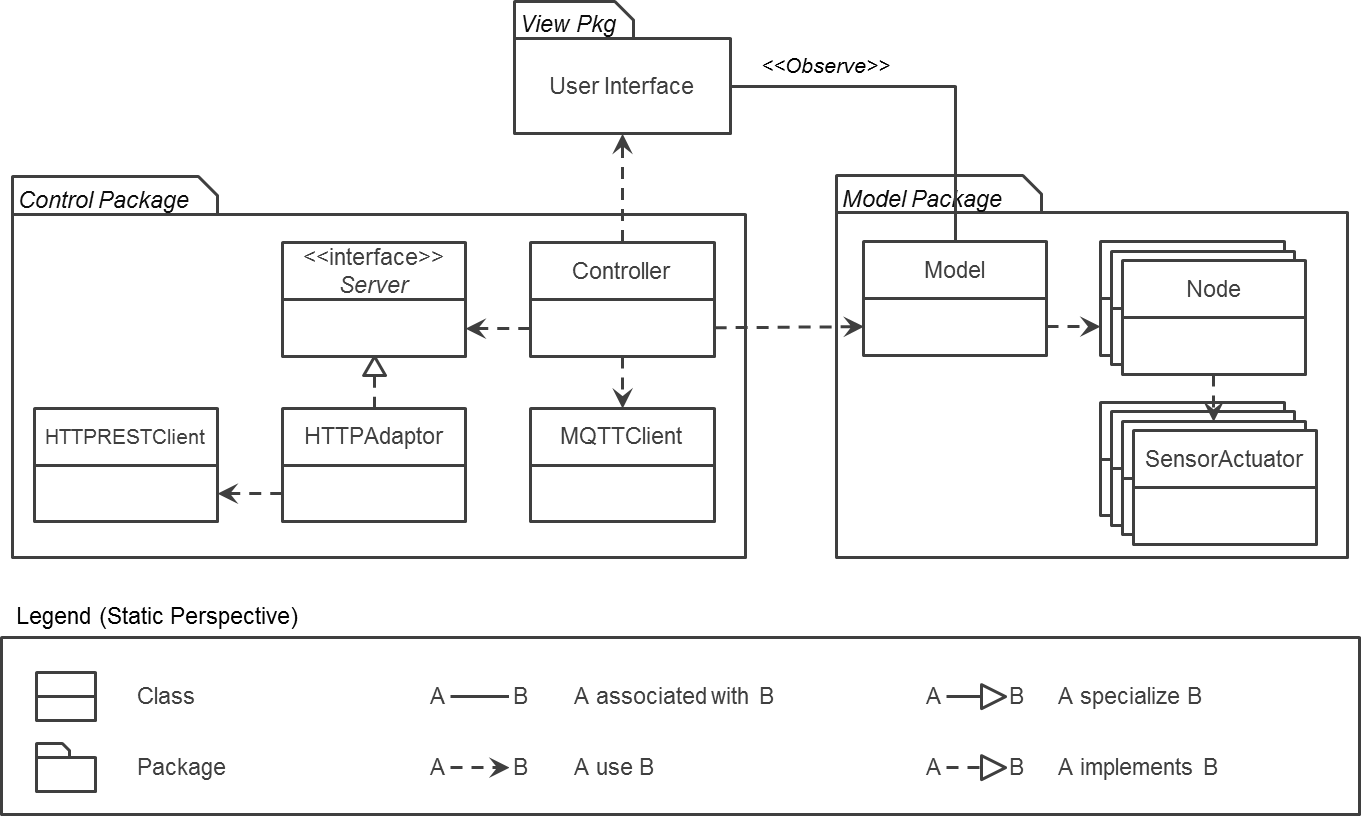
|  |  |
| --- | --- |
| **Perspective:** Dynamic  **Associated Drawings:** Figure 7.1 | |
| **Relationship** | **Responsibilities** |
|  | * This relationship indicates the data transition through network. * Network connection to the Event bus, it uses the MQTT. * Network connection to the Web server, it uses the HTTPREST. * On the relationship with the Event bus, elements do publish and subscribe. |
| A  B | * This relationship indicates that A sends an event or message to B. * It was used between View and Controller. Between View and Control, it will be used to tell user input to controller. |
| B | * This relationship indicates that A observe B. * It was used between View and Model. View shows the latest status of Model (SA node) by querying the status whenever model data was changed. |
| A  B  A | * This relationship indicates that A can update the status of B. * Controller can select View status by user input or web server response. * Controller can update the model status by listening the data from Web server or Event Bus. |
|  | * This relationship indicate user input. By providing the user interface, user can input his/her data or request. |

**Table 7.3 Rationale of 1st decomposition of User App**

|  |
| --- |
| **Perspective:** Dynamic  **Associated Drawings:** Figure 7.1  **Associated Responsibilities**: Table 7.1 and 7.2 |
| To provide the various user interfaces, GUI will be changed many times by many stakeholders. From this reason, we select MVC pattern. Because multiple views of the user interface can be created, maintained, and coordinated when *SA node* data changes. It is easy to develop and test them in parallel, and changes to one have minimal impact on the others. (*QA09. Extensibility*) |

## 7.2 Static Perspective

So far we have been concerned with decomposition of the *User App* in terms of the dynamic perspective. Now we will turn our attention to the static structures of the *User App*. We can map the dynamic view Figure 7.1 into static view like Figure 7.2 Each run time object mapped into same name of package. For example, run time object of Model mapped into model package.

****

**Figure 7.2 Static Perspective of *User App*.**

**Table 7.4 Element Responsibilities catalog for static perspective of *User App***

|  |  |
| --- | --- |
| **Perspective:** Static  **Associated Drawings:** Figure 7.2. | |
| **Element** | **Responsibilities** |
| View | * This package provide user interface. |
| Controller | * Controller can use the view. |
| Server interface | * Server interface used by controller to connect to remote server. |
| HTTPAdaptor | * One of the remote servers is HTTP. For this it implement server interface and use HTTPREST client. |
| HTTPRESTClient | * This is used for connect to remote HTTP server. |
| MQTTClient | * This was used to publish/subscribe to Event bus by controller. |
| Model | * This was used to represent the SA nodes which user can determine and control. It has many nodes. |
| Node | * Node represents the SA node general characteristics like nick name, serial number, ownership and so force. |
| SensorActuator | * SA node consists of various sensors or actuators. This elements means a sensor or actuator. This elements belong into Node. |

**Table 7.5 Relationship Responsibilities catalog for static perspective of *User App***

|  |  |
| --- | --- |
| **Perspective:** Static  **Associated Drawings:** Figure 7.2. | |
| **Relationship** | **Responsibilities** |
|  | * A and B are associated each other. For example, view observes the model. Whenever model status was changed, view know this situation and update view’s data with latest model data. |
|  | * A use B. |
|  | * A implements B |
|  | * A specialize B |

**[Table 7.6] Rationale of static perspective of *User App***

|  |
| --- |
| **Perspective:** Static  **Associated Drawings:** Figure 7.2  **Associated Responsibilities**: Table 7.4 and 7.5 |
| To provide the various *IoT server*, we apply the interface for remote server and then controller use this interface. Currently, we are considering the RESTful HTTP server to managing the account and security manager. Even it was changed into new one we just use the adapter for new remote server. In addition, to represent the *SA node* we separate the class into two parts one is Node description which has *SA node* characteristics (name, serial number, owner, and so forth). This node can have many Sensors&Actuators(*QA09. Extensibility*) |

8. DECOMPOSITION OF SA NODE

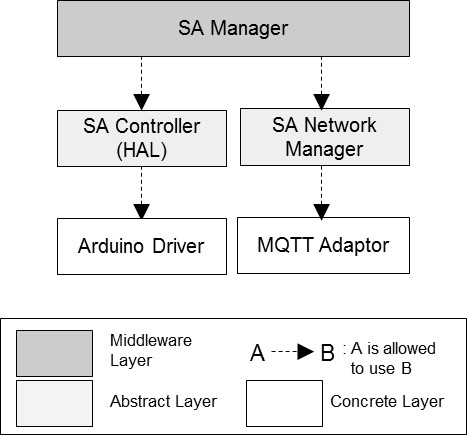
In this section, we will closely look into *SA node* that was described in the system context chapter 4 from the context diagram (Figure 4.1).

## 8.1 Static Perspective - 1st Decomposition of SA node

*SA node* is designed as the layered pattern for the modifiability and portability. Abstract Layer is consist of two modules; *SA Controller* and *SA Network Manager*. *SA Controller* provides the abstraction of the sensor and actuator to support different hardware of SA. *SA Network Manager* provides the abstraction of the publish-subscribe event bus to support emerging event bus and network protocol.

Modules - *Arduino Driver* and *MQTT Adaptor* in Concrete Layer implement *SA Controller* and *SA Network Manager* in the abstract layer respectively.

Finally, *SA Manager* in the middleware layer has the logic and policy of the functional requirements of the SA node (i.e. open or close the door upon the arrival of door control event) , and they are implemented by using the modules in the abstract layer.



**Figure 8.1 First level of decomposition - focusing on the decomposition of SA node**

**Table 8.1 Element Responsibility Catalog for First-Level Decomposition**

|  |  |
| --- | --- |
| **Perspective: Static**  **Associated Drawings: Figure 8.1** | |
| **Element** | **Responsibilities** |
| *SA Manager* | * This element implements logics and policies of the functional requirements of *SA node*. * This monitors the status of the sensor from *SA Controller* and publish it through *SA Network Manager*. * This uses the functions of SA controller to control the actuator after subscribing to the event through SA Network Manager. * This has the several rules and execute automatically some functions when the some condition is met. (i.e. : refer to FR06 Lock house automatically) |
| *SA Controller* (HAL) | * This element hides underlying driver implementations of sensors and actuators. * This includes the generic HAL interface that doesn’t need to be changed if the hardware of the sensor and actuator is replaced, * Various H/W drivers can be supported by implementing this generic HAL interface. |
| *SA Network Manager* | * This element hides the implementation details of the event bus. * It defines the generic publish-subscribe interface of the event bus. * *MQTT Adaptor* should be implemented from this interface. * In addition, this provides the communication channel for the emerging protocol. * This communication for the emerging protocol is encapsulated from *SA Manager*. * See “8.2 2nd Decomposition of SA Network Manager” for more detail. |
| *Arduino Driver* | * This element is the driver for the Arduino based sensor and actuator. * This should be implemented from HAL of *SA Controller*. |
| *MQTT Adaptor* | * This element is the adaptor that is implemented by MQTT event bus. * This adaptor should be implemented from the publish-subscribe interface of *SA Network Manager*. |

**Table 8.2 Relationship Responsibility Catalog for First-Level Decomposition**

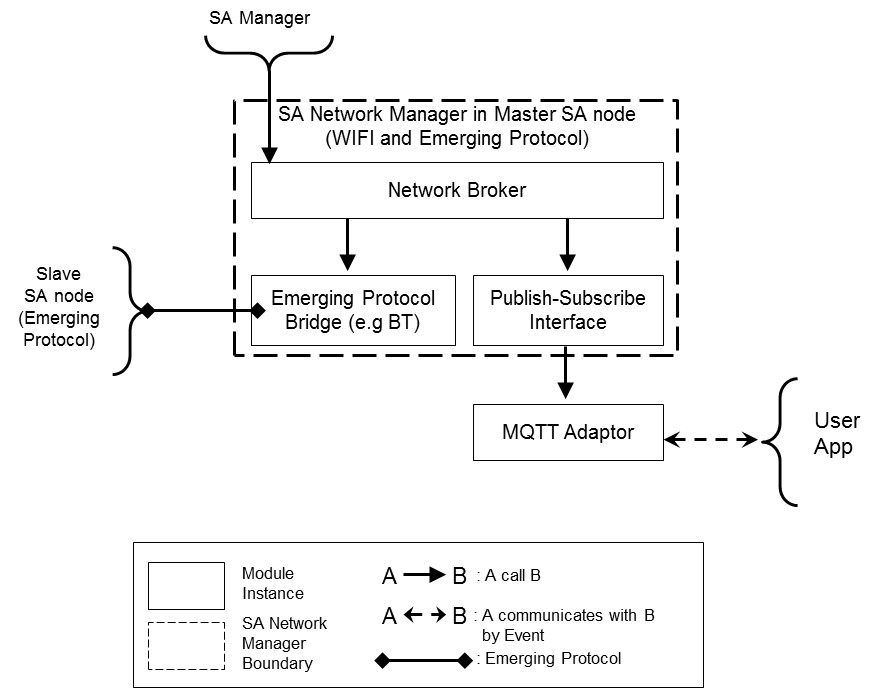
|  |  |
| --- | --- |
| **Perspective: Dynamic**  **Associated Drawings: Figure 8.1** | |
| **Relationship** | **Responsibilities** |
| B  A | * This relationship indicates that element A is allowed to use the function of B. The relations must be unidirectional. |

**Table 8.3 First Level of SA node Decomposition Rationale**

|  |
| --- |
| **Perspective: Static**  **Associated Drawings: Figure 8.1**  **Associated Responsibilities: Tables 8.1 and 8.2** |
| This decomposition is composed by the layered pattern to promote the modifiability and portability of SA node program.  First, it minimize the effort required to transfer SA node program from one hardware of the sensor and actuator to another. *SA controller* in the abstract layer provides the HAL interface of the sensor and actuator that need to be changed if the hardware of SA is replaced,  SA node program can be easily ported to various hardware, if they are implemented from HAL interface. (High portability for the different hardware)  Second, it reduces the cost of adding emerging event bus and network protocol. SA network manager in the abstract layer provides the generic publish-subscribe interface. High layer modules uses the interface, not using concrete MQTT event bus directly. So, the higher layer modules are not required to be changed, even though MQTT is replaced to other event bus. As a result, this structure makes it easy to add or change the emerging event bus and network protocols.(High modifiability for the emerging event bus and network protocols) |

## 8.2 Dynamic Perspective - 2nd Decomposition of SA Network Manager

SA node communicates with the *IoT Server* and *Event Bus* via WIFI using HTTP and MQTT protocol respectively. However, SA node is designed to support the emerging protocol without WIFI. *SA Network Manager* encapsulates the concrete network protocol-specific functionality from *SA manager* and other module instances. *Network broker* in *SA Network Manager* locates an appropriate network bus in accordance with the network capability of SA node. In addition, it provides the bridge between SA node with the emerging protocol and SA node with WIFI. SA node with WIFI communicate with the system in behalf of the node without WIFI. They have the relationship of master-slave and are named “Master SA node” and “Slave SA node”.



**Figure 8.2 Second level of decomposition - focusing on the decomposition of SA Network Manager in SA node**

**Table 8.4 Element Responsibility Catalog for Second-Level Decomposition**

|  |  |
| --- | --- |
| **Perspective:** Dynamic  **Associated Drawings:** Figure 8.2 | |
| **Element** | **Responsibilities** |
| *Network Broker* | * This element is the intermediary that locates an appropriate network bus in accordance with the network capability of SA node. * If SA node only supports the emerging protocol without WIFl, than the *Network Broker* sends the event to the other node’s *Network Broker* through the *Emerging Protocol Bridge*. (They are named as “Master SA node” and “Slave SA node”) * Master SA node communicates with the event bus in behalf of Slave SA node. |
| *Emerging Protocol Bridge* | * This element mediates between the local network broker and *Emerging Protocol Bridge* of remote network broker in other SA node. * This encapsulates the network protocol-specific functionality from the network broker. |
| *Publish-subscribe Bus Interface* | * This element is the abstract of the event bus. * This calls the *MQTT adaptor* for delegating the request from the network broker. |
| *MQTT Adaptor* | * This element is the adaptor to control MQTT event bus that has the real event bus mechanisms. |

**Table 8.5 Relationship Responsibility Catalog for Second-Level Decomposition**

|  |  |
| --- | --- |
| **Perspective: Dynamic**  **Associated Drawings: Figure 8.5** | |
| **Relationship** | **Responsibilities** |
| B  A | * This relationship indicates that element A calls functionality of B. This is just function call between the module instances at run time. |
| B  A | * This relationship indicates that element A communicates with by event at runtime. |
| B  A | * This relationship indicates the communication through emerging protocol. |

**Table 8.6 Second Level of SA Network Manager in SA node - Decomposition Rationale**

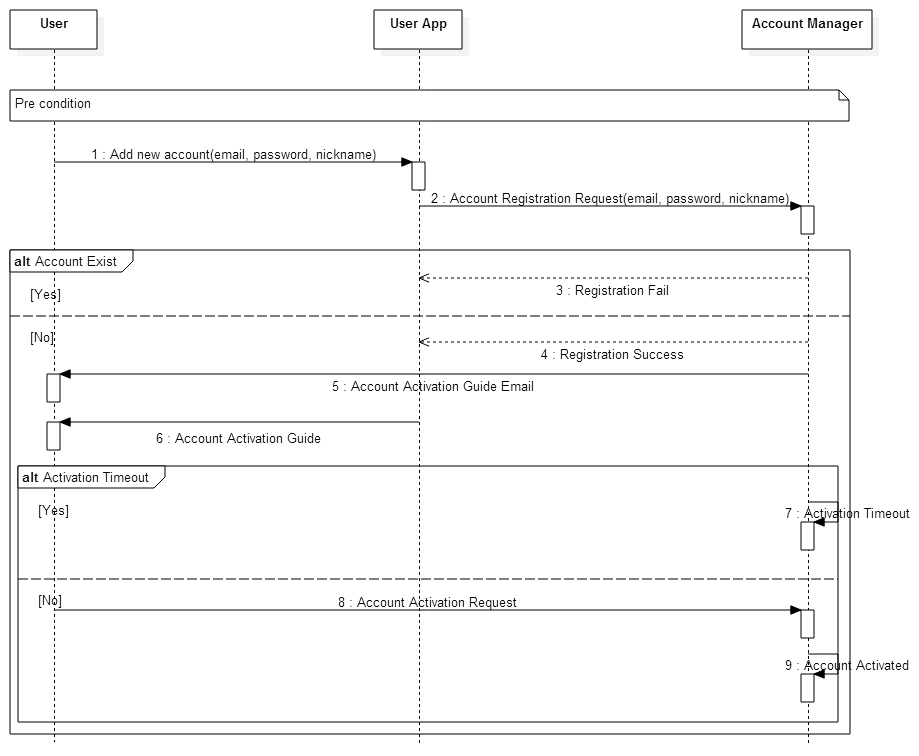
|  |
| --- |
| **Perspective: Dynamic**  **Associated Drawings: Figure 8.2**  **Associated Responsibilities: Tables 8.4 and 8.5** |
| This decomposition is composed by the light broker pattern to promote the modifiability and interoperability about adding new emerging protocols like Bluetooth without WIFI. Other module instances like SA manager only calls the *Network Broker* without knowing about the concrete network and protocol. The *Network Broker* locates an appropriate network bus in accordance with the network capability of SA node. If the SA node supports the WIFI, then the *Network Broker* calls the *Publish-subscribe Bus Interface* to send or receive the event. If the SA node supports the emerging protocol without WIFI, then the event is transferred to other SA node with WIFI and emerging protocol. (They are named “Master SA node” and “Slave SA node”.) At that time, The communication between Master SA node and Slave SA node are operated through the *Emerging Protocol Bridge*. Master SA node has the responsibility to communicate with the event bus on behalf of Slave SA node.  This structure provides the high interoperability between SA nodes based on different network protocol and system (High interoperability between SA node with different network protocol)  *Emerging protocol bridge* encapsulates the network protocol-specific functionality from the *Network Broker* and other module instances. So, it provides high modifiability for new emerging protocol requirement, by adding and developing new bridge without the change of other module instances. (High modifiability for new emerging protocol.) (*QA07 Modifiability*) |

9. SYSTEM DESIGN SUPPLEMENT

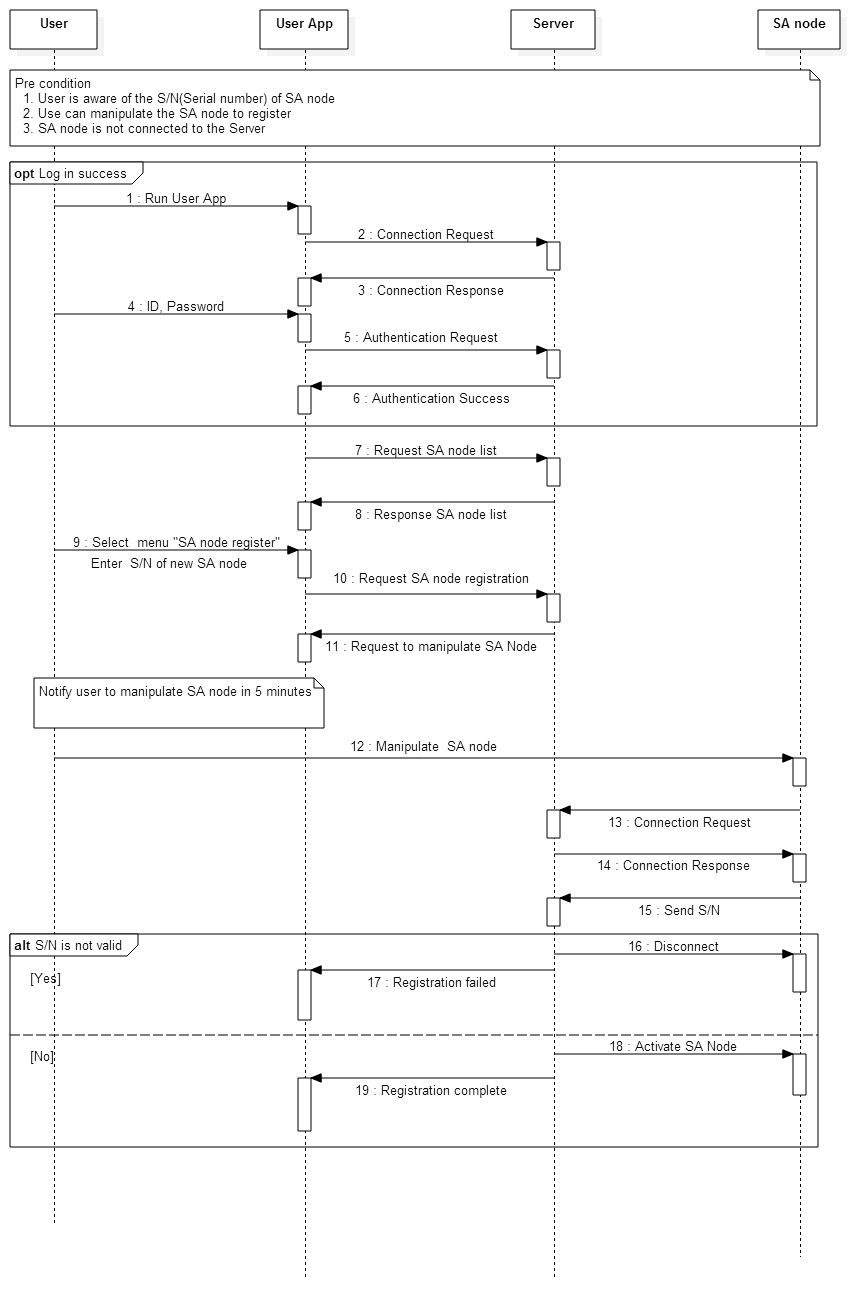
## 9.1 Sequence diagram

To help the understanding how to provide the service to user by interacting system components. We draw the sequence diagram in this section. Basic flow is: user action captured by user application, and then user application do the communication with IoT server which is provide the login service and SA node lists who can use by end user. After that it will publish or subscribe the event among IoT server and SA node as well through event bus.

For more detail flow, please refer the below sequence diagram for the node registration.



**Figure 9.1 Sequence diagram - Register new user account**



**Figure 9.2 Sequence diagram - SA node registration**

## 9.2 Dynamic Behavior of User App

To help the understanding how *User App* interacts with end user, we draw status diagram describing the view behavior of *User App*. ~~~