Katholieke
Universiteit
Leuven

**Department of
Computer Science**

Shared Internet Of Things Infrastructure Platform:

ADD Application

Software Architecture (H09B5a and H07Z9a) – Part 2a

Filipcikova-Halilovic

Monika Filipcikova (r0683254)
Armin Halilovic(r0679689)

Academic year 2016–2017

Contents

1. Attribute-driven design documentation

1.1 Decomposition 1: SIoTIP System (Av3, UC14, UC15, UC18)

1.1.1 Module to decompose

In this run we decompose the `SIoTIP System`.

1.1.2 Selected architectural drivers

The non-functional drivers for this decomposition are:

- *Av3*: Pluggable device or mote failure

The related functional drivers are:

- *UC14*: Send heartbeat (Av3)
  This use case checks whether or not motes and pluggable devices are still operational.
- *UC15*: Send notification (Av3)
  This use case sends a notification to a registered user.
- *UC18*: Check and deactivate applications (Av3)
  This use case deactivates any application that requires deactivation, because of unavailability of essential pluggable devices or unassigned mandatory roles.

Rationale We chose Av3 first since it had high priority and it was more relevant to the core of the system (pluggable device data) than attributes M1 and U2. We chose P2 along with Av3 as it would force us to think about the way sensor data is handled. We believe this combination of pluggable device connectivity and storage of sensor data is the most defining feature of the system, and that handling this combination would give a better starting point than M1+U2 for later ADD iterations.

1.1.3 Architectural design

Application redundancy settings for Av3 Discussion of the solution selected for (a part of) one of the architectural drivers.

Failure detection for Av3 timers?
heartbeat/timestamp tactic

Application deactivation for Av3 Applications need pluggable devices for the proper functioning. When the pluggable devices fail, `the PluggableDeviceManager` sends command and `the ApplicationManager` deactivate one or more applications using those devices. Availability and reliability of the shared platform offered to applications is important. To reduce the risk of frequent application downtime, an application provider can require a redundancy in the available pluggable devices. Multiple sensors or actuators for one application can be in one room. If one of sensor or actuator failed application just start using the other available sensor or actuator in room.

Notifications for Av3 One of the important things for Av3 is notification. In the case of failure of sensor, it is mandatory to inform all involved parties about the failure to resolve the problem as soon as possible. `The NotificationHandler` notify an infrastructure owner of any persistent pluggable device or mote failures. The infrastructure owner has to receive the notification in ten seconds in case mote failed or in thirty seconds if a pluggable device failed. Notification is also send to a customer organisation, when one or more of their application are susspended or re-activated. Applications using a failed pluggable device should be also notified via `The NotificationHandler`.

Alternatives considered

Alternatives for application deactivation for Av3 As mentioned above, when some of pluggable devices fail the applation can operate normally, because of using the next available sensor. ???? could be?

1.1.4 Instantiation and allocation of functionality

Decomposition Main aspects of the resulting decomposition.

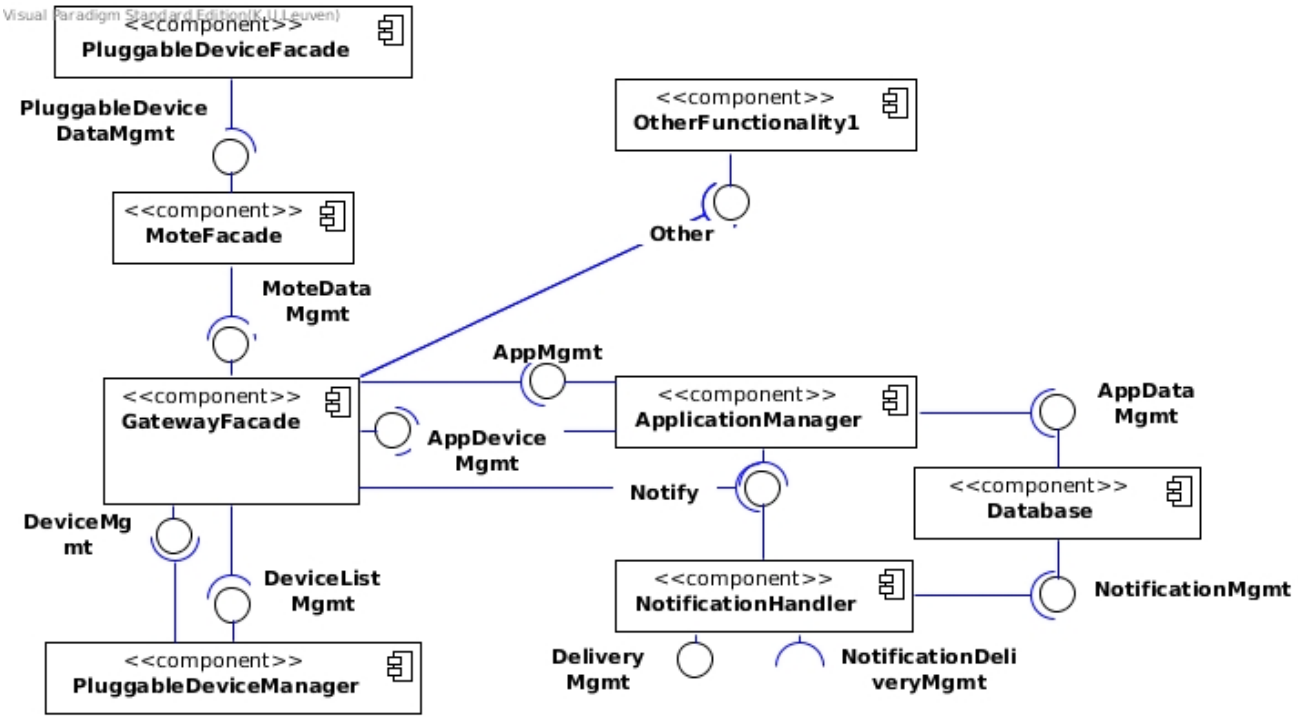

Figure 1.1: Component-and-connector diagram of this decomposition.

ApplicationManager deactivate apps ???? check mandatory user roles set redundancy in the available pluggable devices If application suspended or re-activated, notify cust. org. If application uses failed plugg device, notify application (Av3)

Database General database for other data. Storage of notifications for now.

GatewayFacade receive heartbeats, send heartbeats/device lists, send application shutdown, send notification trigger (Av3)
forward data to applications

MoteFacade sends heartbeats

NotificationHandler Send notifications.
stored by system − > contact DB?
lookup communication channel
users choose delivery method?

PluggableDeviceFacade send heartbeats

PluggableDeviceManager check list of devices and see if there are pluggable devices for applications check redundancy in the available pluggable devices contains application preferences (e.g. amount of sensors required) can send command to deactivate application If failure detected, notify inf owner (Av3). reactivate application if new/needed hardware detected

Deployment Rationale of the allocation of components to physical nodes.

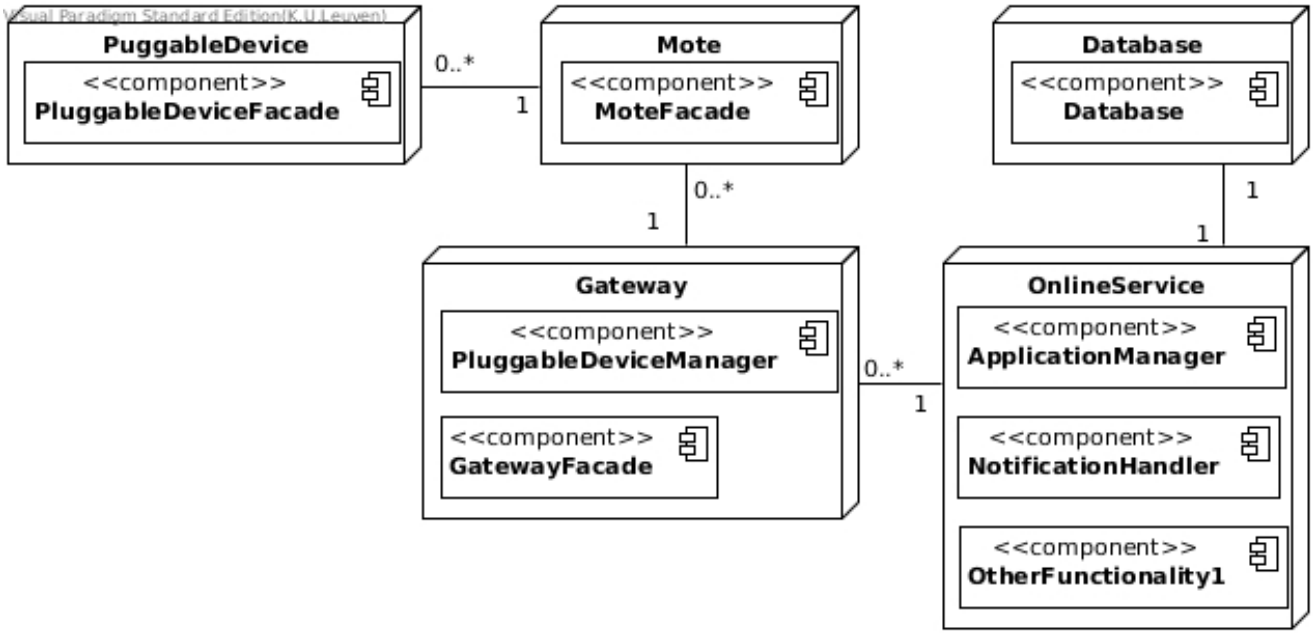

Figure 1.2: Deployment diagram of this decomposition.

1.1.5 Interfaces for child modules

ApplicationManager

- ForwardData
  - `void sendData(PluggableDeviceData data)`
    - Effect: Send pluggable device data to an application that wants to use it
    - Exceptions: None
- AppMgmt
  - `void deactivateApplicationInstance(int applicationInstanceID)`
    - Effect: Deactivates a running instance of an application.
    - Exceptions: None
  - `void activateApplicationInstance(int applicationInstanceID)`
    - Effect: Activates a new instance of an application.
    - Exceptions: None

Database

- NotificationMgmt
  - `int storeNotification(NotificationData data)`
    - Effect: Stores a new notification entry in the database. Returns the id of the new notification.
    - Exceptions: None
  - `void updateNotification(NotificationData data)`
    - Effect: Updates an existing notification (e.g. change status to "sent").
    - Exceptions: None
  - `int lookupNotificationChannelForUser(int userID)`
    - Effect: Returns the type of communication channel a user prefers. Different communication channels are mapped to integers.
    - Exceptions: None
- AppDataMgmt
  - `void updateApplication(ApplicationData data)`
    - Effect: Updates an application in the database (e.g. change state to 'inactive').
    - Exceptions: None
  - `void updateSubscription(SubscriptionData data)`
    - Effect: Updates a subscription in the database (e.g. change state to 'disabled').
    - Exceptions: None

GatewayFacade

- MoteDataMgmt
  - `void sendHeartbeat(int moteID, List<PluggableDeviceInfo> devices)`
    - Effect: Sends a heartbeat to a certain gateway with information about operational devices.
    - Exceptions: None
- DeviceMgmt
  - `List<DeviceInfo> getConnectedDevices()`
    - Effect: Describe the effect of calling this operation.
    - Exceptions: None
  - `void timerExpired(int deviceID)`
    - Effect: Lets the gateway know that a timer for pluggable device or mote has expired. This will generate a notification for an infrastructure owner.
    - Exceptions: None
  - `void deactivateApplicationInstance(int applicationInstanceID)`
    - Effect: Deactivates a certain application. This could happen when mandatory pluggable devices for the application are missing.
    - Exceptions: None
  - `void reactivateApplicationInstance(int applicationInstanceID)`
    - Effect: Reactivate an application instance. This could happen automatically after a broken sensor has been replaced.
    - Exceptions: None
- AppDeviceMgmt

- `bool areEssentialDevicesOperational(int applicationID)`
  - Effect: Returns true if all essential devices for the application with id "applicationID" are operational.
  - Exceptions: None

MoteFacade

- PluggableDeviceDataMgmt
  - `List<DeviceInfo> getConnectedDevices()`
    - Effect: Returns a list of information about devices that are connected to the mote.
    - Exceptions: None

NotificationHandler

- Notify
  - `void notify(int userID, String message)`
    - Effect: Describe the effect of calling this operation.
    - Exceptions: None
- DeliveryMgmt
  - `void sendAcknowledgement(int notificationID)`
    - Effect: Sends an acknowledgement to the system for a certain notification.
    - Exceptions: None

External notification delivery serivce

- NotificationDeliveryMgmt
  - `void notify(JSONObject data)`
    - Effect: Deliver a notification to an end user using a specific delivery service.
    - Exceptions: None

PluggableDeviceManager

- DeviceListMgmt
  - `void sendHeartbeat(int moteID, List<PluggableDeviceInfo> devices)`
    - Effect: Send a heartbeat from a mote to check/update timers for operational devices.
    - Exceptions: None
  - `bool areEssentialDevicesOperational(int applicationID)`
    - Effect: Returns true if all essential devices for the application with id "applicationID" are operational.
    - Exceptions: None

1.1.6 Data type definitions

PluggableDeviceData contains data from a pluggable device at a certain point in time (value, type, date) (e.g. a sensor reading, an actuator status)

PluggableDeviceSettings contains settings for a pluggable device (power status, data update rate, ...)

PluggableDeviceInfo contains information about a pluggable device (device id, power status, data update rate, ...)

NotificationData contains data about a notification (message text, recipient, communication channel, date, status, source, ...).

ApplicationData contains data about an application instance (instance id, running status, ...)

SubscriptionData contains data about a subscription (subscription id, subscription status, subscription period, ...).

1.1.7 Verify and refine

Completely handled: Av3, UC14, UC15, UC18

This section describes per component which (parts of) the remaining requirements it is responsible for.

ApplicationManager

- *Av2*: Application failure
  Prevention: a, b
  Detection: a, b, c
  Resolution: a, b, c
- *P1*: Large number of users: c
- *M1*: Integrate new sensor or actuator manufacturer: 1.c, 2.a
- *M2*: Big data analytics on pluggable data and/or application usage data: d, e
- *U1*: Application updates: a, b, c, d
- *U2*: Easy Installation: e
- *U12*: Perform actuation command
- *UC17*: Activate an application: 3, 4

Database

- None

GatewayFacade

- *Av1*: Communication between SIoTIP gateway and Online Service
  Resolution: b, c, d
- *M1*: Integrate new sensor or actuator manufacturer: 1.a, 2.b
- *U2*: Easy Installation: a, c, d
- *UC11*: Send pluggable device data: 1

MoteFacade

- *M1*: Integrate new sensor or actuator manufacturer: 1.a, 2.b
- *U2*: Easy Installation: b, c, d
- *UC4*: Install mote: 1, 2
- *UC5*: Uninstall mote: 1
- *UC6*: Insert a pluggable device into a mote: 2
- *UC7*: Remove a pluggable device from its mote: 2
- *UC11*: Send pluggable device data: 1

NotificationHandler

- *UC16*: Consult notification message: 5
- *UC17*: Activate an application: 5, 6

OtherFunctionality

- *Av1*: Communication between SIoTIP gateway and Online Service
  Detection: a, b, c, d Resolution: a
- *P1*: Large number of users: a
- *P2*: Requests to the pluggable data database
- *M1*: Integrate new sensor or actuator manufacturer: 1.d
- *M2*: Big data analytics on pluggable data and/or application usage data: a
- *U2*: Easy Installation: e
- *UC1*: Register a customer organisation
- *UC2*: Register an end-user
- *UC3*: Unregister an end user
- *UC4*: Install mote: 3
- *UC5*: Uninstall mote: 2.b
- *UC6*: Insert a pluggable device into a mote: 3: topology part; alternative 3a.1.b
- *UC7*: Remove a pluggable device from its mote: 3.b
- *UC8*: Initialise a pluggable device: 1, 2, 4
- *UC9*: Configure pluggable device access rights
- *UC10*: Consult and configure the topology
- *UC11*: Send pluggable device data: 3
- *UC13*: Configure pluggable device
- *UC16*: Consult notification message: 1, 2, 3, 4
- *UC17*: Activate an application: 1, 2

- *UC19*: Subscribe to application
- *UC20*: Unsubscribe from application
- *UC21*: Send invoice
- *UC22*: Upload an application
- *UC23*: Consult application statistics
- *UC24*: Consult historical data
- *UC25*: Access topology and available devices
- *UC26*: Send application command or message to external front-end
- *UC27*: Receive application command or message to external front-end
- *UC28*: Log in
- *UC29*: Log out

PluggableDeviceDB

- *M1*: Integrate new sensor or actuator manufacturer: 1.a, 1.b, 2.b
- *M2*: Big data analytics on pluggable data and/or application usage data: b

PluggableDeviceFacade

- *U2*: Easy Installation: d

PluggableDeviceManager

- *U2*: Easy Installation: c, d
- *UC4*: Install mote: 4
- *UC5*: Uninstall mote: 2
- *UC6*: Insert a pluggable device into a mote: 3: uninitialised part; alternative 3a.1 3a.2 3a.4; 4
- *UC7*: Remove a pluggable device from its mote: 3.a, 3.c
- *UC8*: Initialise a pluggable device: 3
- *UC11*: Send pluggable device data: 2, 3a

PluggableDeviceDataScheduler

- *P1*: Large number of users: b
- *M1*: Integrate new sensor or actuator manufacturer: 1.a, 2.b
- *M2*: Big data analytics on pluggable data and/or application usage data: b, c

1.2 Decomposition 2: OtherFunctionality (M1, P2, UC11)

1.2.1 Module to decompose

In this run we decompose OtherFunctionality.

1.2.2 Selected architectural drivers

The non-functional drivers for this decomposition are:

- *M1*: Integrate new sensor or actuator manufacturer
- *P2*: Requests to the pluggable data database

The related functional drivers are:

- *UC11*: Send pluggable device data (P2)
  This use case stores pluggable device data in the pluggable device data storage. This could be a sensor reading, or an actuator status.

Rationale We choose M1 because it belogs to the quality attributes with hight priority. M1 is about integration of new sensor or actuator. And it is very important to easily add new devices, because market grows very fast and new applications are developing. So we want to focus on this quality attribute in the early stages and then based on that create other functionality and components. And we also choose P2 because it is related to M1. M1 required minimal changes to data processing and storage, so we have to deal with good solution for this topic.

1.2.3 Architectural design

Handling new types of pluggable devices for M1 The developers have to make changes to: component1, component2, datatype X. The new type of sensor needs to be able to be initialised so that it can send data. Thus, the PluggableDeviceFacade code that initialises devices should be updated for each new type of sensor. The PluggableDeviceData datatype should be updated to represent the new type of data. In this case, the new type will have to be added to the database that contains all different types of sensor data.

Data conversions for M1 `The PluggableDeviceDataConverter` is resposible for converting data in system, for instance converting temperature in degrees Fahrenheit to degrees Celsius. System has to work with relevant data, otherwise problem may arise.

Usage of new data by applications for M1 This is possible through the RequestData interface provided by PluggableDeviceDataScheduler. The application manager can get device data from the PluggableDeviceDB and return this data to applications in the PluggableDeviceData datatype. This datatype can easily be updated for new types of pluggable devices.

Configuration of new device by infrastructure owners for M1 Initialisation: IO triggers the initialise() function which has been updated for the new pluggable device -¿ OK
Configure access rights: has absolutely fucking nothing to do with the new sensor type -¿ OK
Consult and configure topology: same as configure access rights

Scheduling for P2 dynamic priority scheduling
tactics: schedule resource, prioritize events, also limit event response?
starvation avoidance

Pluggable data separation for P2 ”pluggable data has no impact on other data” two databases

Alternatives considered

Alternatives for solution A discussion of the alternative solutions and why that were not selected.

1.2.4 Instantiation and allocation of functionality

Decomposition Main aspects of the resulting decomposition.

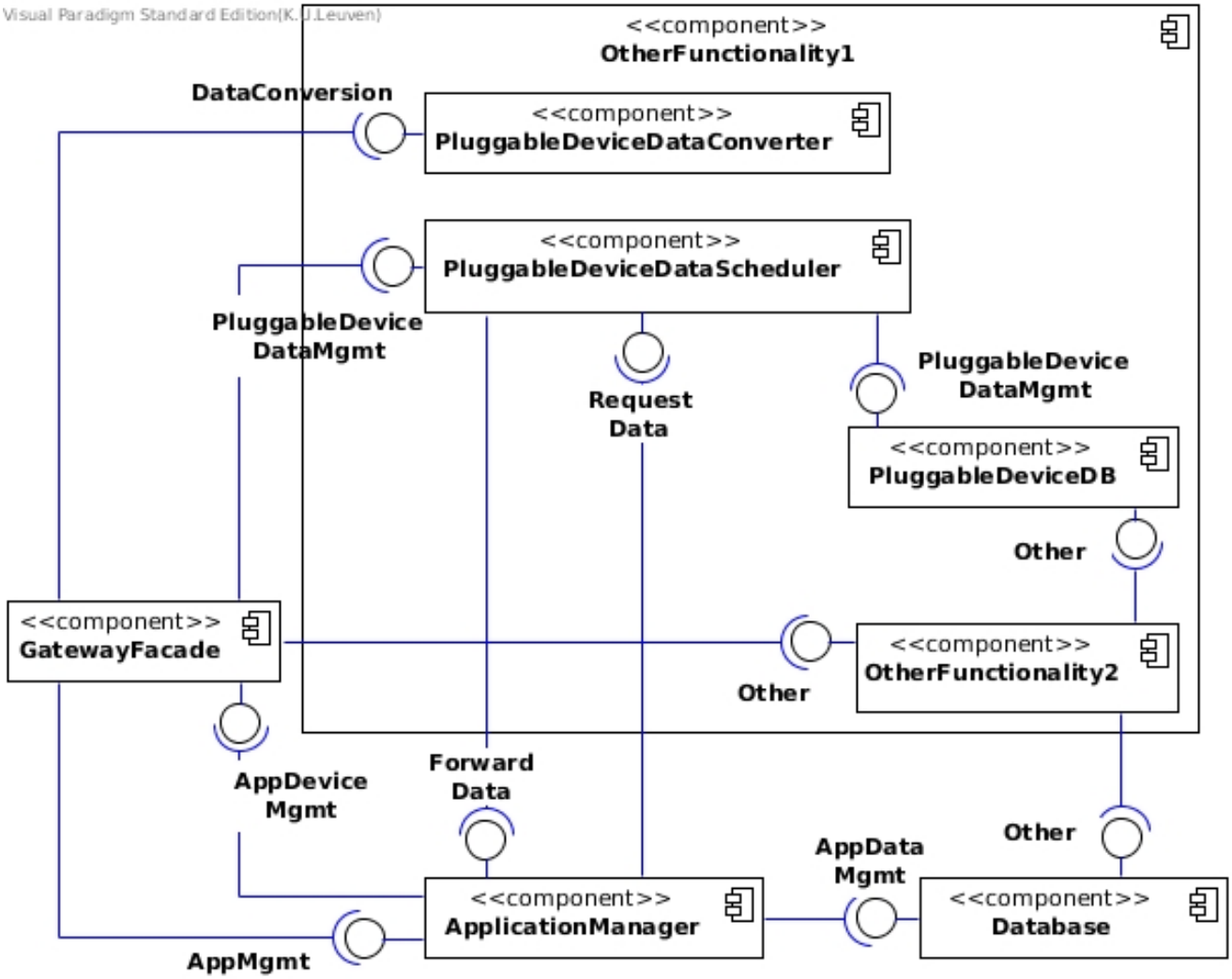

Figure 1.3: Component-and-connector diagram of this decomposition.

PluggableDeviceDB store data related to pluggable devices

PluggableDeviceDataScheduler scheduling, detect overload mode, store data, forward data

PluggableDeviceDataConverter M1: conversion of new type of data of new type of device

Deployment Rationale of the allocation of components to physical nodes.

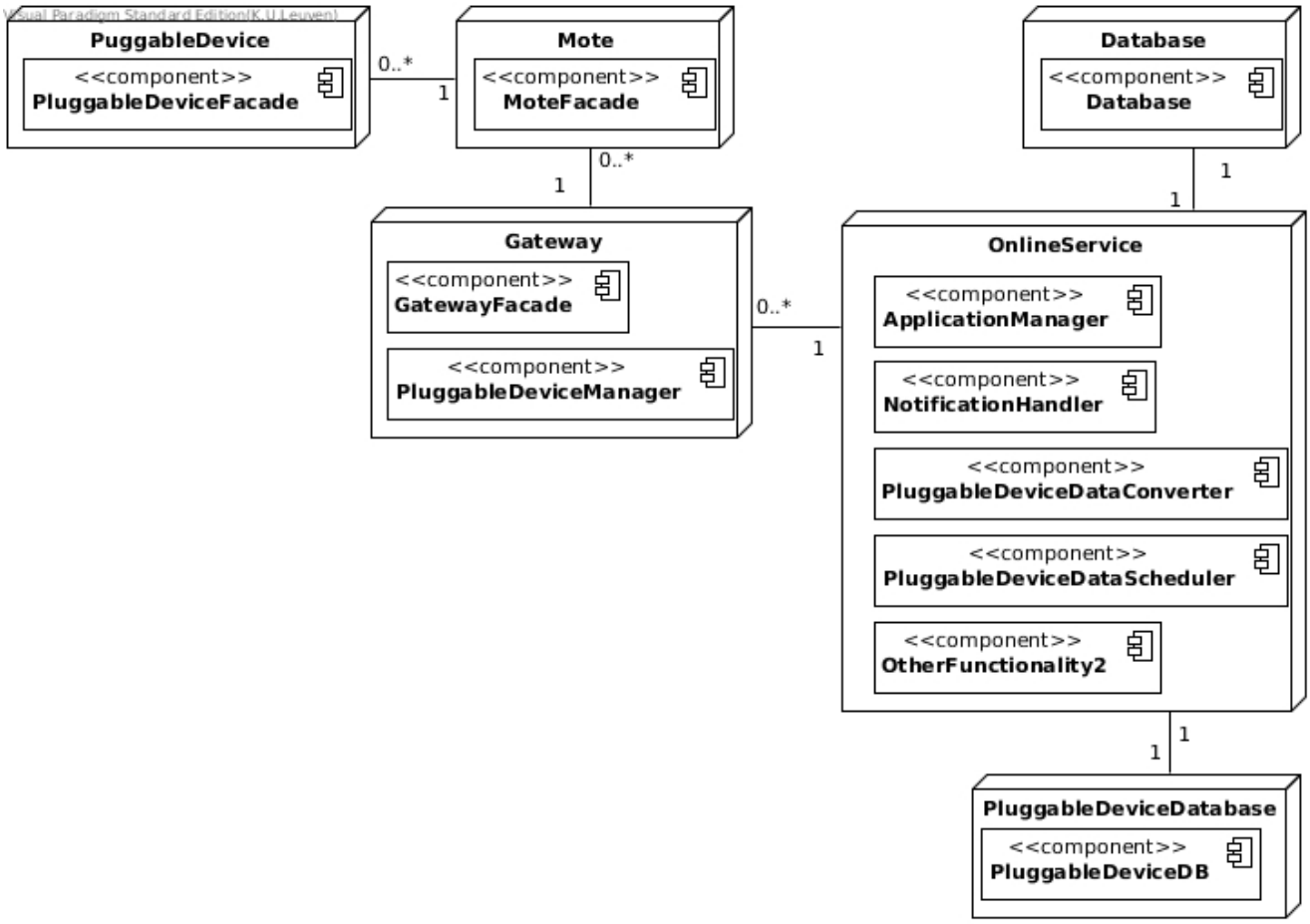

Figure 1.4: Deployment diagram of this decomposition.

1.2.5 Interfaces for child modules

GatewayFacade

See "**??**: GatewayFacade" for the rest of the interfaces provided by this component.

- MoteDataMgmt
  - `void sendData(PluggableDeviceData data)`
    - Effect: Sends pluggable device data to the connected mote.
    - Exceptions: None
- DeviceMgmt
  - `void initialiseDevice(int deviceID, PluggableDeviceSettings settings)`
    - Effect: Initialises a pluggable device for use with the system.
    - Exceptions: None
- AppDeviceMgmt
  - `void configurePluggableDevice(int deviceID, PluggableDeviceSettings settings)`
    - For: Use case 11 step 3.b
    - Effect: Causes certain settings to be set on a pluggable device that the gateway is connected to.
    - Exceptions: None

MoteFacade

See "1.1.5: MoteFacade" for the rest of the interfaces provided by this component.

- PluggableDeviceDataMgmt
  - `void sendData(PluggableDeviceData data)`
    - Effect: Sends pluggable device data to the connected mote.
    - Exceptions: None
- PluggableDeviceMgmt
  - `void initialise(int deviceID, PluggableDeviceSettings settings)`
    - Effect: Initialises a connected pluggable device according to some settings
    - Exceptions: None

PluggableDeviceFacade

- PluggableDeviceMgmt
  - `void initialise(PluggableDeviceSettings settings)`
    - Effect: Initialises the pluggable device according to some settings
    - Exceptions: None

PluggableDeviceManager

- DeviceListMgmt
  - `bool isDeviceInitialised(int deviceID)`
    - Effect: Returns true if the device with id "deviceID" has been initialized.
    - Exceptions: None

PluggableDeviceDataScheduler

- RequestData
  - `List<PluggableDeviceData> requestData(int applicationID, int deviceID, DateTime from, DateTime to)`
    - Effect: Request data from a specific device in a certain time period
    - Exceptions: None
- PluggableDeviceDataMgmt
  - `void sendData(PluggableDeviceData data)`
    - Effect: Sends pluggable device data to the scheduler to be processed.
    - Exceptions: None

PluggableDeviceDB

- PluggableDeviceDataMgmt
  - `void sendData(PluggableDeviceData data)`
    - Effect: Sends pluggable device data to the DB to be stored.
    - Exceptions: None
  - `List<PluggableDeviceData> getData(int deviceID, DateTime from, DateTime to)`
    - Effect: Returns data from a specific device in a certain time period.
    - Exceptions: None
  - `List<int> getApplicationsForDevice(int deviceID)`
    - Effect: Returns a list of applications that can use the device with id "deviceID."
    - Exceptions: None

1.2.6 Data type definitions

DateTime Represents an instant in time, typically expressed as a date and time of day.

1.2.7 Verify and refine

Completely handled: M1, P2, UC11

This section describes per component which (parts of) the remaining requirements it is responsible for.

ApplicationManager

- *Av2*: Application failure
  Prevention: a, b
  Detection: a, b, c
  Resolution: a, b, c
- *P1*: Large number of users: c
- *M2*: Big data analytics on pluggable data and/or application usage data: d, e
- *U1*: Application updates: a, b, c, d
- *U2*: Easy Installation: e
- *U12*: Perform actuation command
- *UC17*: Activate an application: 3, 4

Database

- None

GatewayFacade

- *Av1*: Communication between SIoTIP gateway and Online Service
  Resolution: b, c, d
- *U2*: Easy Installation: a, c, d

MoteFacade

- *U2*: Easy Installation: b, c, d
- *UC4*: Install mote: 1, 2
- *UC5*: Uninstall mote: 1
- *UC6*: Insert a pluggable device into a mote: 2
- *UC7*: Remove a pluggable device from its mote: 2

NotificationHandler

- *UC16*: Consult notification message: 5
- *UC17*: Activate an application: 5, 6

OtherFunctionality

- *Av1*: Communication between SIoTIP gateway and Online Service
  Detection: a, b, c, d Resolution: a
- *P1*: Large number of users: a
- *M2*: Big data analytics on pluggable data and/or application usage data: a
- *U2*: Easy Installation: e
- *UC1*: Register a customer organisation
- *UC2*: Register an end-user
- *UC3*: Unregister an end user
- *UC4*: Install mote: 3
- *UC5*: Uninstall mote: 2.b
- *UC6*: Insert a pluggable device into a mote: 3: topology part; alternative 3a.1.b
- *UC7*: Remove a pluggable device from its mote: 3.b
- *UC8*: Initialise a pluggable device: 1, 2, 4
- *UC9*: Configure pluggable device access rights
- *UC10*: Consult and configure the topology
- *UC13*: Configure pluggable device
- *UC16*: Consult notification message: 1, 2, 3, 4

- *UC17*: Activate an application: 1, 2
- *UC19*: Subscribe to application
- *UC20*: Unsubscribe from application
- *UC21*: Send invoice
- *UC22*: Upload an application
- *UC23*: Consult application statistics
- *UC24*: Consult historical data
- *UC25*: Access topology and available devices
- *UC26*: Send application command or message to external front-end
- *UC27*: Receive application command or message to external front-end
- *UC28*: Log in
- *UC29*: Log out

PluggableDeviceDB

- *M2*: Big data analytics on pluggable data and/or application usage data: b

PluggableDeviceFacade

- *U2*: Easy Installation: d

PluggableDeviceManager

- *U2*: Easy Installation: c, d
- *UC4*: Install mote: 4
- *UC5*: Uninstall mote: 2
- *UC6*: Insert a pluggable device into a mote: 3: uninitialised part; alternative 3a.1 3a.2 3a.4; 4
- *UC7*: Remove a pluggable device from its mote: 3.a, 3.c
- *UC8*: Initialise a pluggable device: 3,

PluggableDeviceDataScheduler

- *P1*: Large number of users: b
- *M2*: Big data analytics on pluggable data and/or application usage data: b, c

2. Resulting partial architecture

~~This section provides an overview of the architecture constructed through ADD.~~
"Since you are a two-student team, you can skip the final step of the assignment/report ("2. Resulting architecture This section should present the component diagram of the overall system (after two decompositions). At this point, you are not required to provide the deployment diagram of the overall system.")"