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 Av3 was chosen first since it has high priority and it is more relevant to the core of the system than the other quality requirements with high priority (M1 and U2). We believe that handling pluggable device failure/connectivity is more important to the whole of the system than M1 and U2, and that handling this first would give a stronger starting point for later ADD iterations than M1 or U2.

1.1.3 Architectural design

This section describes what needs to be done to satisfy the requirements for this decomposition and how involved problems/obstacles are solved.

Av3: Failure detection Gateway need to be able to autonomously detect failure of one of its connected motes and pluggable devices. This is achieved by making motes send heartbeats to their connected gateways. The gateways can then monitor their connected devices. The heartbeats contain a list of devices that are connected/operational at the moment the mote sends the heartbeat. Each gateway makes use of a `PluggableDeviceManager` component to monitor the devices. This component uses timers to keep track of how long it has been since a device has sent a heartbeat or occured in a list of connected devices. Once a timer expires, this is treated as a failure.

A mote has failed when 3 consecutive heartbeats do not arrive within 1 second of their expected arrival time. A pluggable device has failed when it does not occur in a heartbeat of the mote in which it is expected to be in. This is is detected within 2 seconds after the arrival of the heartbeat.

Av3: Automatic application deactivation and redundancy settings Applications should be automatically suspended when they can no longer operate due to failure of a pluggable device or mote and reactivated once the failure is resolved. Application providers can design their applications such that they explicitly require redundancy in the available pluggable devices.
This problem is tackled by the `PluggableDeviceManager`. It stores the requirements for pluggable devices set by applications for all applications that use the gateway that the the `PluggableDeviceManager` runs on. When it detects that an application can no longer operate due to failures, it will send a command to the `ApplicationManager` (via the `GatewayFacade`) to suspend that application. When the required devices are operational again, the

`PluggableDeviceManager` detects this and sends a command to reactivate the application.

Applications are suspended within 1 minute after detecting the failure of an essential pluggable device.
Application are reactivated within 1 minute after the failure is resolved.

Av3: Notifications The infrastructure owner should be notified of any persistent pluggable device or mote failures. Customer organisations should be notified if one or more of their applications is suspended or reactivated. Applications using a failed pluggable device or any device on a failed mote should be notified.
The `NotificationHandler` was put in place to deal with notifications. Other components can use it to generate notifications for certain users in the system. The `NotificationHandler` will then insert information relevant to the notification in the database (message, status, date and time, source, ...), and use an external delivery service to deliver the notification to users. The used delivery medium is based on the user's preferences.
Since they are stored in the database, users can always view their notifications via their dashboard. However, this funcionality is not expanded on in this decomposition yet.

Infrastructure owners are notified within 1 minute after detecting a mote outage lasting at least 10 seconds.
Infrastructure owners are notified within 1 minute after the detection of the unavailability of a pluggable device for 30 seconds.
Applications are notified of the failure of relevant pluggable devices within 10 seconds.

Alternatives considered

Av3: Failure detection An alternative would have been to move the `PluggableDeviceManager` component from gateways to the Online Service. This solution would make the gateways do less work, but would be very unscalable. The reason is that as the customer base (and thus the amount of devices) increases, the Online Service would need to keep track of huge amounts of devices. This would also flood the network to the Online Service with heartbeats.

Av3: Failure detection Another alternative for failure detection could have been the use of a Ping/Echo mechanism instead of Heartbeats. Pings could then be used to check if a device is currently operational. However, as a device could not be operational for a moment because of e.g. interference, timers would still be necessary to keep track of operational devices. We opted to use heartbeats, as this would reduce the amount of data sent over the network used by the motes, and as motes would have to do slightly more work to process each Ping request in order to generate a reply.

Av3: Notifications Reliable and quick delivery of notifications is crucial to the system in order to solve problems should things go wrong. Currently, the solution is to use a third party service for delivery of notifications. In the case that no external services are found satisfactory, or if this dependency on an external service is unwanted, it is possible to build an internal solution for this. For example, a `NotificationSender` component could make use of the `Factory pattern` for different message channels for different delivery methods (each with their own sendNotification method). This solution allows us to easily add new message channels in the future with little effort.

1.1.4 Instantiation and allocation of functionality

This section describes the components which instantiate our solutions described in the section above and how the components are deployed on physical nodes.

Decomposition Figure 1.1 shows the components resulting from the decomposition in this run.
The responsibilities of the components are as follows:

ApplicationManager Responsible for activating/deactivating applications, setting pluggable device redundancy requirements on `PluggableDeviceManager` components, and using `the NotificationHandler` to send notifications to customer organisations. (Av3)

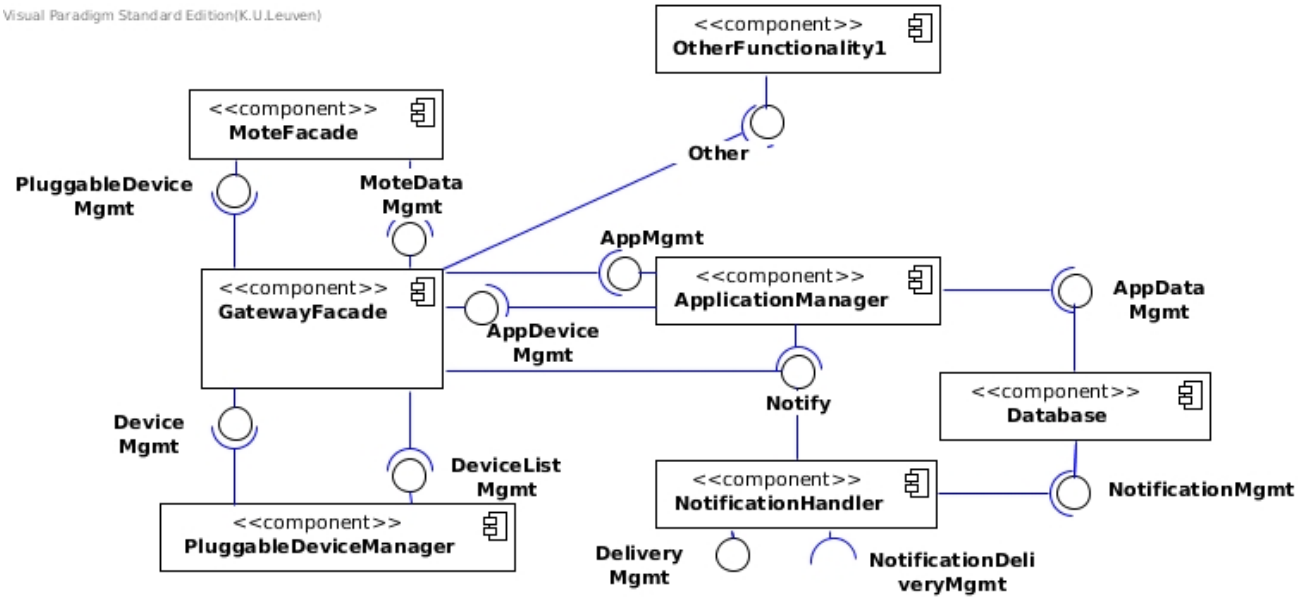

Figure 1.1: Component-and-connector diagram of this decomposition.

Database General database for data. For example, the Database stores the data about notifications.

GatewayFacade Main component on the gateway that allows different components to work with each other. E.g. transmits heartbeats from motes to `PluggableDeviceManager`, transmits commands to shut down applications, triggers notifications to be generated, ...

MoteFacade Sends heartbeats to the `GatewayFacade`. Includes a list of connected pluggable devices in the heartbeats. (UC14)

NotificationHandler Responsible for generation, storage, and delivery of notifications based on users' preferred communication channel. (UC15)

PluggableDeviceManager Monitors connected/operational devices on a gateway. Sends notifications in case of hardware failure. Can send a command to disable or reactivate applications when necessary. (Av3)

Deployment Figure 1.2 shows the allocation of components to physical nodes.

1.1.5 Interfaces for child modules

This section describes the interfaces assigned to the components defined in the section above. Per interface, we list its methods by means of its syntax. The data types used in these interfaces are defined in the following section.

Each method shows which (part of a) quality attribute or use case caused a need for the method. However, this does not mean that a method is only to be used to satisfy that quality attribute or use case, it could be used for other causes not yet mentioned here.

ApplicationManager

- AppMgmt
  - `void deactivateApplicationInstance(int applicationInstanceID)`

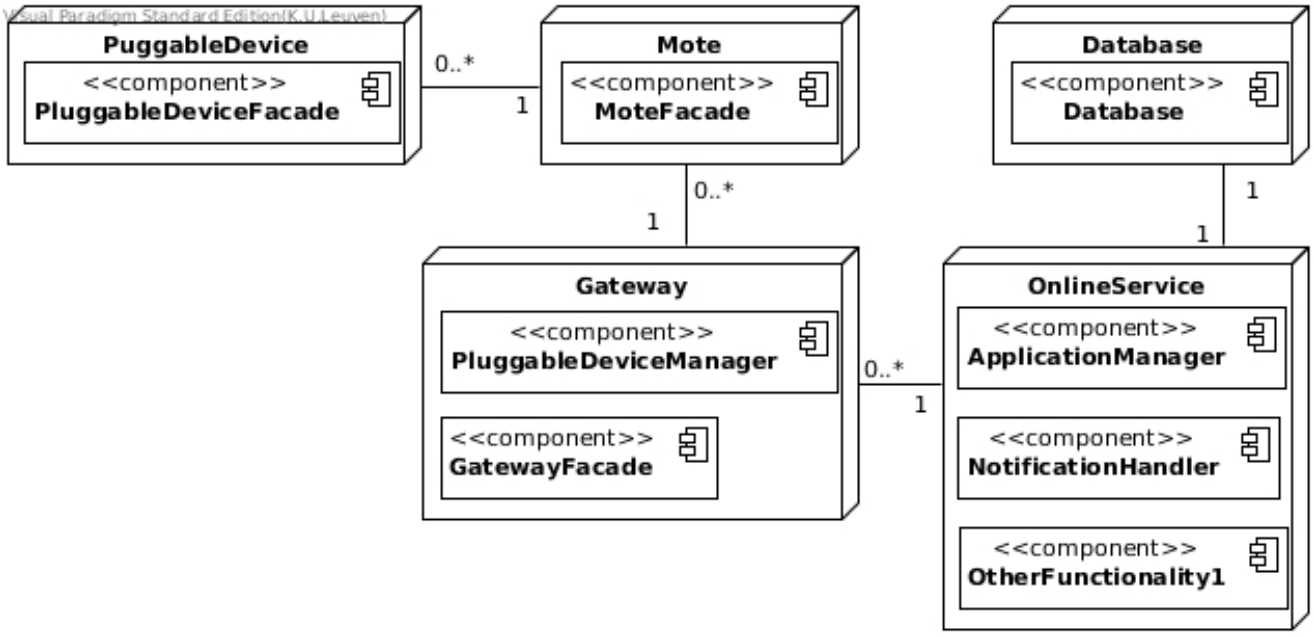

Figure 1.2: Deployment diagram of this decomposition.

    - Effect: Deactivates a running instance of an application.
  - `void activateApplicationInstance(int applicationInstanceID)`
    - Effect: Activates a new instance of an application.

Database

- NotificationMgmt
  - `int storeNotification(NotificationData data)`
    - Effect: Stores a new notification entry in the database. Returns the id of the new notification.
  - `void updateNotification(NotificationData data)`
    - Effect: Updates an existing notification (e.g. change status to "sent").
  - `int lookupNotificationChannelForUser(int userID)`
    - Effect: Returns the type of communication channel a user prefers. Different communication channels are mapped to integers.
- AppDataMgmt
  - `void updateApplication(ApplicationData data)`
    - Effect: Updates an application in the database (e.g. change state to 'inactive').
  - `void updateSubscription(SubscriptionData data)`
    - Effect: Updates a subscription in the database (e.g. change state to 'disabled').

GatewayFacade

- MoteDataMgmt
  - `void sendHeartbeat(int moteID, List<PluggableDeviceInfo> devices)`
    - Effect: Sends a heartbeat to a certain gateway with information about operational devices.
- DeviceMgmt
  - `List<DeviceInfo> getConnectedDevices()`

        * Effect: Describe the effect of calling this operation.
    - `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.
    - `void deactivateApplicationInstance(int applicationInstanceID)`
        * Effect: Deactivates a certain application. This could happen when mandatory pluggable devices for the application are missing.
    - `void reactivateApplicationInstance(int applicationInstanceID)`
        * Effect: Reactivate an application instance. This could happen automatically after a broken sensor has been replaced.
- AppDeviceMgmt
    - `void setPluggableDevicesRequirements(int applicationID, List<PluggableDeviceInfo> devices)`
        * Effect: Sets an application's requirements for pluggable devices.
    - `bool areEssentialDevicesOperational(int applicationID)`
        * Effect: Returns true if all essential devices for the application with id "applicationID" are operational.

MoteFacade

- PluggableDeviceMgmt
    - `List<DeviceInfo> getConnectedDevices()`
        * Effect: Returns a list of information about devices that are connected to the mote.

NotificationHandler

- Notify
    - `void notify(int userID, String message)`
        * Effect: Describe the effect of calling this operation.
- DeliveryMgmt
    - `void sendAcknowledgement(int notificationID)`
        * Effect: Sends an acknowledgement to the system for a certain notification.

External notification delivery serivce

- NotificationDeliveryMgmt
    - `void notify(JSONObject data)`
        * Effect: Deliver a notification to an end user using a specific delivery service.

PluggableDeviceManager

- DeviceListMgmt
    - `void sendHeartbeat(int moteID, List<PluggableDeviceInfo> devices)`
        * Effect: Send a heartbeat from a mote to check/update timers for operational devices.
    - `void setPluggableDevicesRequirements(int applicationID, List<PluggableDeviceInfo> devices)`
        * Effect: Sets an application's requirements for pluggable devices.
    - `bool areEssentialDevicesOperational(int applicationID)`
        * Effect: Returns true if all essential devices for the application with id "applicationID" are operational.

1.1.6 Data type definitions

This section defines the data types used in the interface descriptions above.

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

The selected architectural drivers have been handled completely in this decomposition. This section describes per component which (parts of) the remaining requirements it is responsible for. If requirements are split in multiple parts, this is indicated by the addition of a letter (or number, depending on the structure of the requirement) after their title.

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
- *UC12*: 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
- *UC04*: Install mote: 1, 2
- *UC05*: Uninstall mote: 1
- *UC06*: Insert a pluggable device into a mote: 2
- *UC07*: 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

OtherFunctionality1

- *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
- *UC01*: Register a customer organisation
- *UC02*: Register an end-user
- *UC03*: Unregister an end user
- *UC04*: Install mote: 3
- *UC05*: Uninstall mote: 2.b
- *UC06*: Insert a pluggable device into a mote: 3: topology part; alternative 3a.1.b
- *UC07*: Remove a pluggable device from its mote: 3.b
- *UC08*: Initialise a pluggable device: 1, 2, 4
- *UC09*: 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

PluggableDeviceManager

- *U2*: Easy Installation: c, d
- *UC04*: Install mote: 4
- *UC05*: Uninstall mote: 2

- *UC06*: Insert a pluggable device into a mote: 3: uninitialised part; alternative 3a.1 3a.2 3a.4; 4
- *UC07*: Remove a pluggable device from its mote: 3.a, 3.c
- *UC08*: Initialise a pluggable device: 3
- *UC11*: Send pluggable device data: 2, 3a

1.2 Decomposition 2: OtherFunctionality1 (M1, P2, UC11)

1.2.1 Module to decompose

In this run we decompose OtherFunctionality1.

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 chose M1 as it was one of the remaining quality attributes with high priority. M1's focus on easily introducing new types of devices to the system is very important because of the fast growing market for IoT and development of applications for IoT. Thus, we want to handle this quality attribute before U2 (the other remaining attribute with high priority), as we presume that customer organisations are more interested in using new devices than the effort it takes for infrastructure owners to install the devices.
We also chose P2 because it is strongly related to M1; the whole data flow from devices to storage/applications needs to exist before modifications can even be made. This combination of M1 and P2 would force us to handle processing and storage of data while making the involved components as simple as possible to modify.

1.2.3 Architectural design

This section describes what needs to be done to satisfy the requirements for this decomposition and how involved problems/obstacles are solved.

M1: Handling new types of pluggable devices The new types of sensor or actuator data should be transmitted, processed and stored, and should be made available to applications. The infrastructure managers must be able to initialize the new type of pluggable device (UC8 : Initialise a pluggable device), configure access rights for these devices (UC9 : Con- figure pluggable device access rights), and view detailed information about the new type of pluggable device (UC10 : Consult and configure the topology).

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.

M1: Data conversions The pluggable data processing subsystem should be extended with relevant data conversions, e.g. converting temperature in degrees Fahrenheit to degrees Celsius. `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.

M1: Usage of new data by applications The available applications can be updated to use any new pluggable devices. 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.

M1: Configuration of new device by infrastructure owners Initialisation: IO triggers the initialise() method 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

P2: Scheduling - In normal mode, the database processes incoming requests in a first-in-first-out order. - If the system fails to comply to the deadlines specified below, it goes in overload mode: requests are handled in the order that returns the system to normal mode the fastest, taking into account: the nature of the requests: storing new pluggable data (UC11 : Send pluggable device data) has priority over specific lookup queries (e.g. retrieving most recent measurement for a few sensors), which in turn have priority over broad queries (e.g. retrieving all sensor data for the last month). the priority of an application: requests from applications marked as critical by their sub- scribers are processed before non-critical applications. - Also a mechanism should be in place to avoid starvation of any type of request.

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

P2: Pluggable data separation The processing of (large amounts of) requests concerning pluggable data has no impact on requests concerning other data, e.g. available applications.

"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

This section describes the new components which instantiate our solutions described in the section above and how components are deployed on physical nodes.
Unless stated otherwise the responsibilities assigned in the first decomposition are unchanged.

Decomposition Figure 1.3 shows the components resulting from the decomposition in this run.
The responsibilities of the components are as follows:

PluggableDeviceDB stores data related to pluggable devices. In `the PluggableDeviceDB` is stored basic information about devices.

PluggableDeviceDataScheduler `the PluggableDeviceDB` receives a large amount of parallel request and it is neccesary to handle with that. `The PluggableDeviceDataScheduler` is resposnsible for scheduling jobs that interact with database. Jobs are processed in a first-in-first-out order normaly. In case te overload mode is detected, jobs are proccesed in the order that returns the system. In case the data has to be saved, `The PluggableDeviceDataScheduler` call method for saving the data, that `the PluggableDeviceDB` provides.

PluggableDeviceDataConverter is component for conversions of new type of data of new type of device (M1). For instance converting temperature is very useful in the system.

Deployment Figure 1.4 shows the allocation of components to physical nodes.

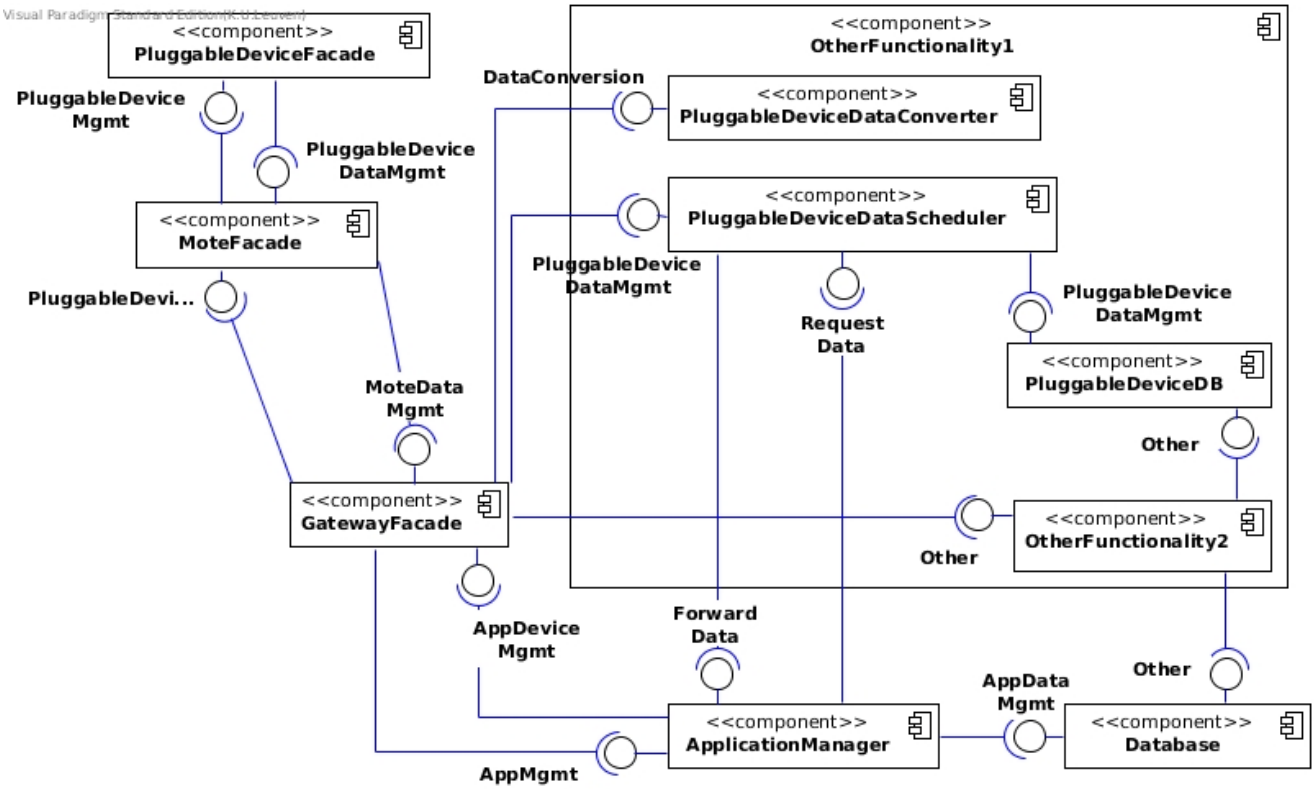

Figure 1.3: Component-and-connector diagram of this decomposition.

1.2.5 Interfaces for child modules

This section describes the interfaces assigned to the components defined in the section above. Per interface, we list its methods by means of its syntax. The data types used in these interfaces are defined in the following section.

Each method shows which (part of a) quality attribute or use case caused a need for the method. However, this does not mean that a method is only to be used to satisfy that quality attribute or use case, it could be used for other causes not yet mentioned here.
The interfaces and methods defined here are to be seen as an extension of the interfaces defined in previous sections, unless explicitly stated otherwise.

ApplicationManager

- ForwardData
  - `void sendData(PluggableDeviceData data)`
    - Effect: Send pluggable device data to an application that wants to use it

GatewayFacade

- MoteDataMgmt, last defined in section 1.1.5
  - `void sendData(PluggableDeviceData data)`
    - Effect: Sends pluggable device data to the connected mote.
- DeviceMgmt, last defined in section 1.1.5
  - `void initialiseDevice(int deviceID, PluggableDeviceSettings settings)`
    - Effect: Initialises a pluggable device for use with the system.

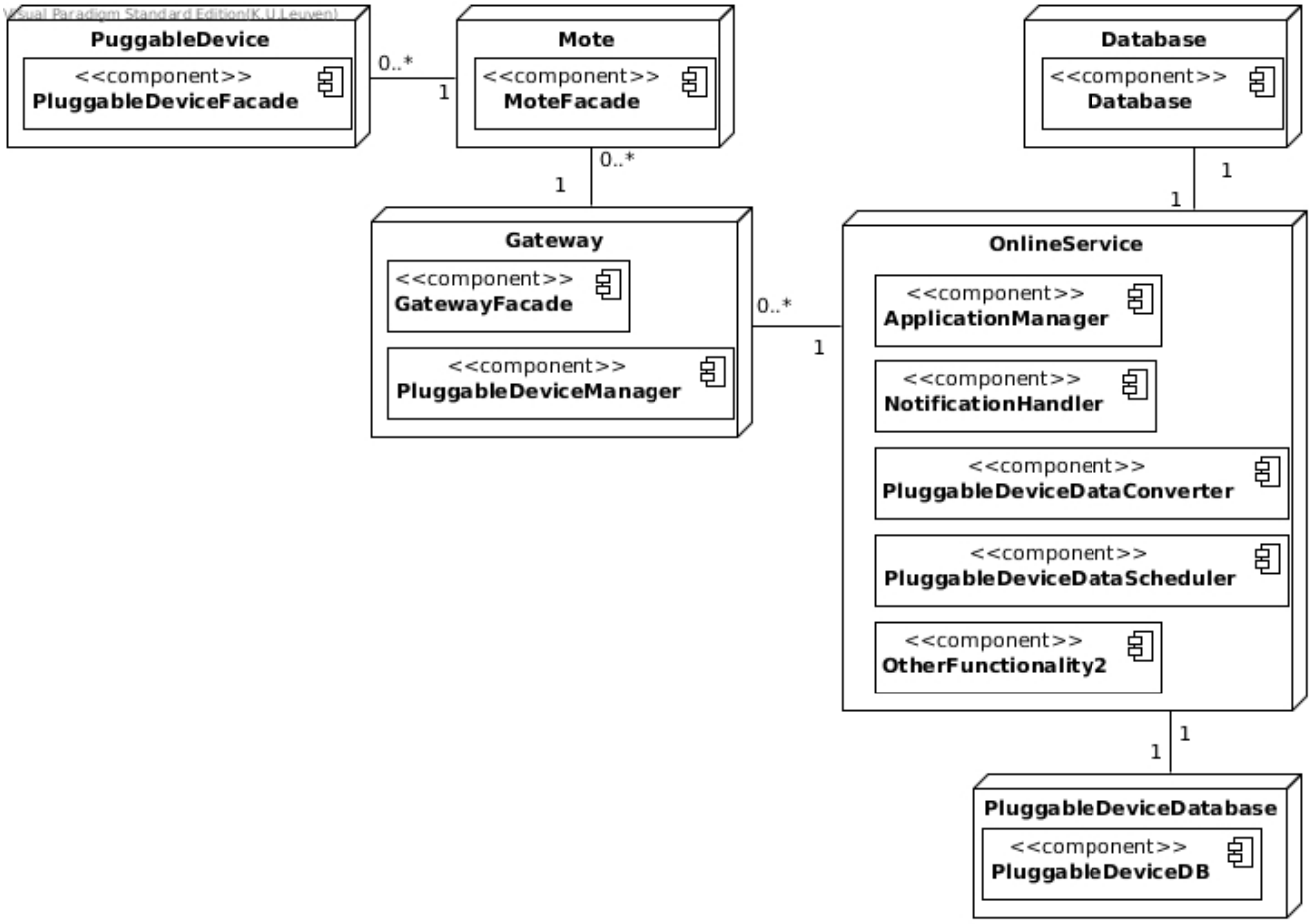

Figure 1.4: Deployment diagram of this decomposition.

- AppDeviceMgmt, last defined in section 1.1.5
  - `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.

MoteFacade

- PluggableDeviceDataMgmt, last defined in section 1.1.5
  - `void sendData(PluggableDeviceData data)`
    - Effect: Sends pluggable device data to the connected mote.
- PluggableDeviceMgmt
  - `void initialise(int deviceID, PluggableDeviceSettings settings)`
    - Effect: Initialises a connected pluggable device according to some settings

PluggableDeviceFacade

- PluggableDeviceMgmt
  - `void initialise(PluggableDeviceSettings settings)`
    - Effect: Initialises the pluggable device according to some settings

PluggableDeviceManager

- DeviceListMgmt, last defined in section 1.1.5
  - `bool isDeviceInitialised(int deviceID)`
    - Effect: Returns true if the device with id "deviceID" has been initialized.

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
- PluggableDeviceDataMgmt
  - `void sendData(PluggableDeviceData data)`
    - Effect: Sends pluggable device data to the scheduler to be processed.

PluggableDeviceDB

- PluggableDeviceDataMgmt
  - `void sendData(PluggableDeviceData data)`
    - Effect: Sends pluggable device data to the DB to be stored.
  - `List<PluggableDeviceData> getData(int deviceID, DateTime from, DateTime to)`
    - Effect: Returns data from a specific device in a certain time period.
  - `List<int> getApplicationsForDevice(int deviceID)`
    - Effect: Returns a list of applications that can use the device with id "deviceID."

1.2.6 Data type definitions

This section defines new data types that are used in the interface descriptions above.

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

1.2.7 Verify and refine

The selected architectural drivers have been handled completely in this decomposition. This section describes per component which (parts of) the remaining requirements it is responsible for. If requirements are split in multiple parts, this is indicated by the addition of a letter (or number, depending on the structure of the requirement) after their title.

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
- *UC12*: 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
- *UC04*: Install mote: 1, 2
- *UC05*: Uninstall mote: 1
- *UC06*: Insert a pluggable device into a mote: 2
- *UC07*: Remove a pluggable device from its mote: 2

NotificationHandler

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

OtherFunctionality2

- *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
- *UC01*: Register a customer organisation
- *UC02*: Register an end-user
- *UC03*: Unregister an end user
- *UC04*: Install mote: 3
- *UC05*: Uninstall mote: 2.b
- *UC06*: Insert a pluggable device into a mote: 3: topology part; alternative 3a.1.b
- *UC07*: Remove a pluggable device from its mote: 3.b
- *UC08*: Initialise a pluggable device: 1, 2, 4
- *UC09*: 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
- *UC04*: Install mote: 4
- *UC05*: Uninstall mote: 2
- *UC06*: Insert a pluggable device into a mote: 3: uninitialised part; alternative 3a.1 3a.2 3a.4; 4
- *UC07*: Remove a pluggable device from its mote: 3.a, 3.c
- *UC08*: 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.")"

Figure 2.1 shows the whole deployment diagram after second decomposition.

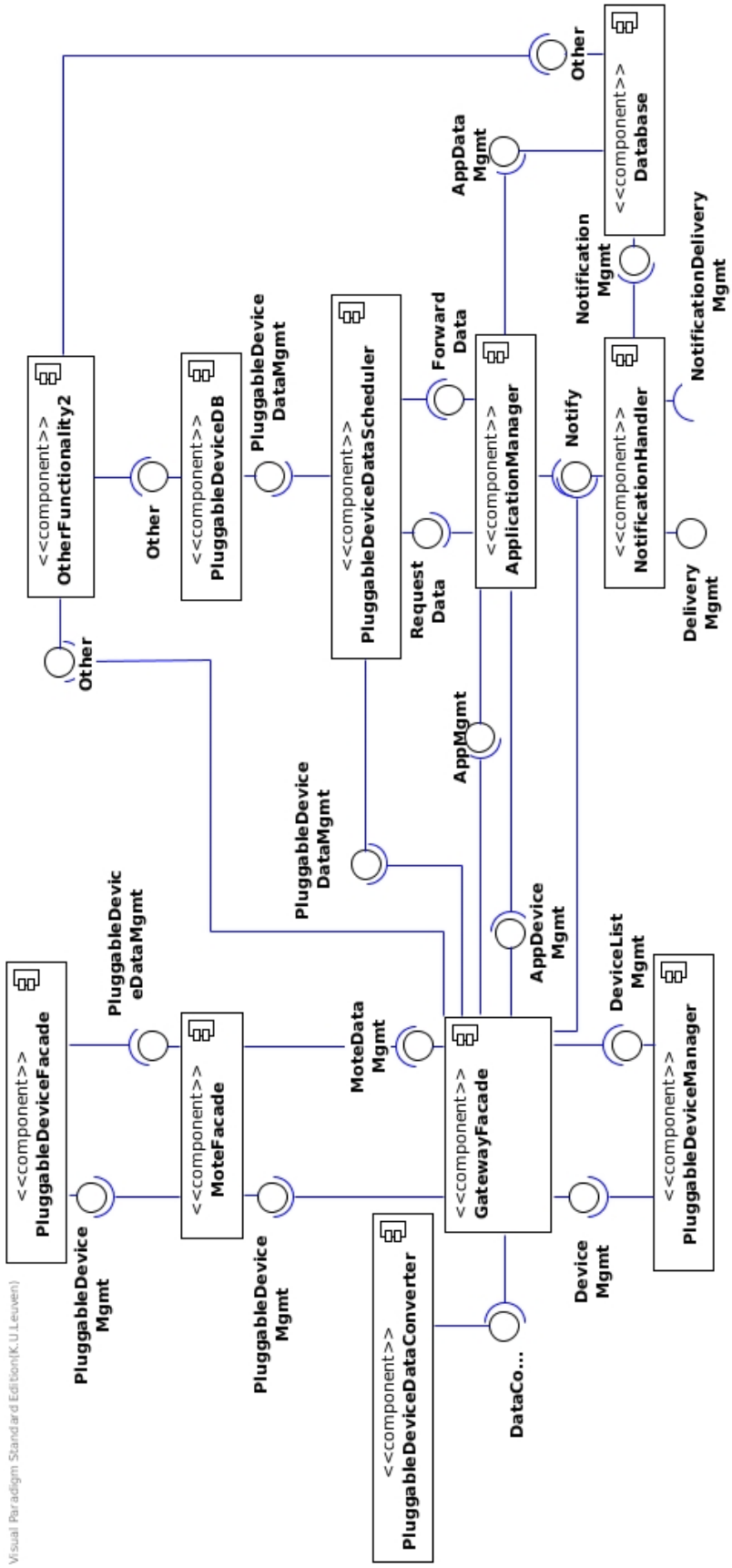

Figure 2.1: Complete deployment diagram after second decomposition.