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, P2, UC11, 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
- *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.
- *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 same as Application redundancy settings for Av3?
degradation/removal from service tactic

Notifications for Av3 from application manager to cust orgs and apps
from gateway to infr owners
notifications tactic

Scheduling for P2 dynamic priority scheduling
tactics: schedule resource, prioritize events, also limit event response?
starvation avoidance
TODO ASK: The system checks whether the pluggable device has been initialised (cf. UC8: Initialise a pluggable device)

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

Alternatives considered

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

1.1.4 Instantiation and allocation of functionality

Decomposition Main aspects of the resulting 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?

OtherFunctionality1 Responsible for delivery of notifications.

PluggableDeviceDB store data related to pluggable devices

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

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

Deployment Rationale of the allocation of components to physical nodes.

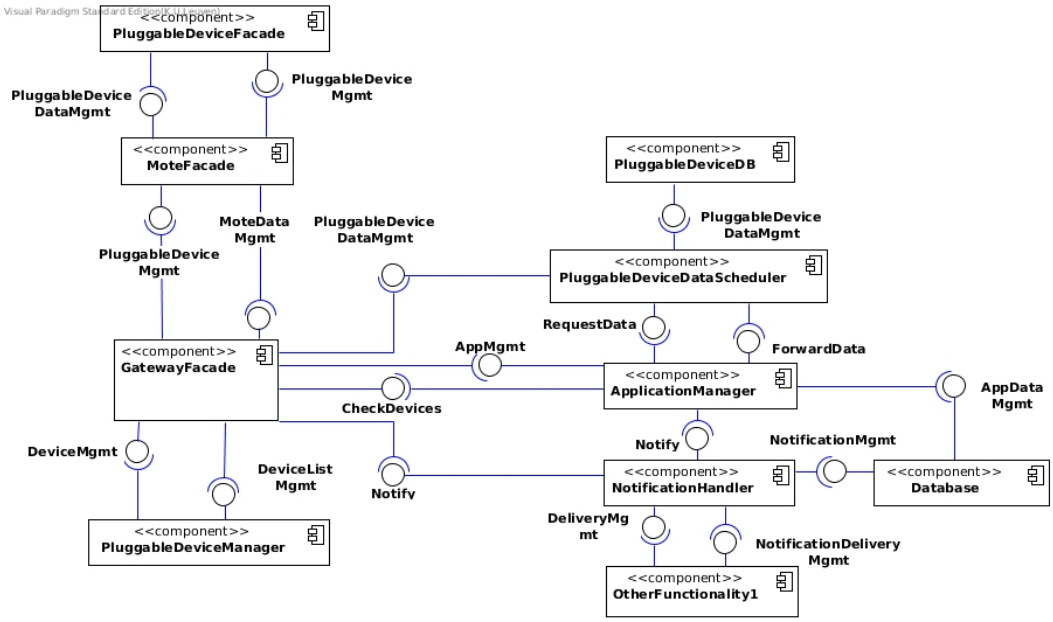

Figure 1.1: Component-and-connector diagram of this decomposition.

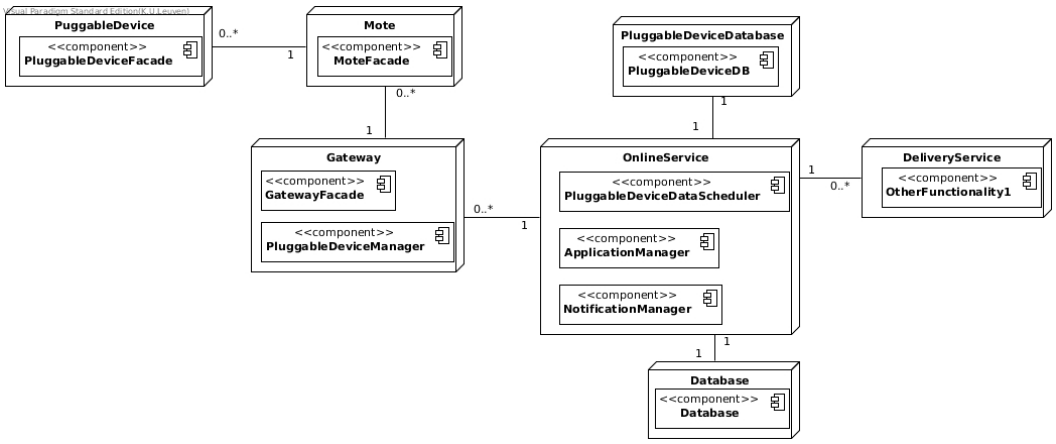

Figure 1.2: Deployment diagram of this decomposition.

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
  - `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. TODO check QA's text
    * Exceptions: None
  - `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

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

MoteFacade

- PluggableDeviceDataMgmt
  - `void sendData(PluggableDeviceData data)`
    * Effect: Sends pluggable device data to the connected mote.
    * Exceptions: None
  - `List<DeviceInfo> getConnectedDevices()`
    * Effect: Returns a list of information about devices that are connected to the mote.
    * Exceptions: None

- PluggableDeviceMgmt
  - `void initialise(int deviceID, PluggableDeviceSettings settings)`
    * Effect: TODO check this with QA's: Initialises a connected pluggable device according to some settings
    * 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

OtherFunctionality1

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

PluggableDeviceFacade

- PluggableDeviceMgmt
  - `void initialise(PluggableDeviceSettings settings)`
    - Effect: TODO check this with QA's: Initialises the pluggable device according to some settings
    - 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 isDeviceInitialised(int deviceID)`
    - Effect: Returns true if the device with id "deviceID" has been initialized.

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

1.1.6 Data type definitions

PluggableDeviceData contains data from a pluggable device at a certain point in time (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, ...)

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

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, P2, UC11, UC14, UC15, UC18

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

PluggableDeviceFacade

- *Z1*: name
- *UCd*: name

GatewayFacade

- *UCba*: name
  Description which part of the original use case is the responsibility of this component.

HeartbeatProcessor

- : *UCX or QAX*: which part of the original use case is the responsibility of this component

PluggableDeviceManager

- : *UCX or QAX*: which part of the original use case is the responsibility of this component

RequestHandler

- : *UCX or QAX*: which part of the original use case is the responsibility of this component

ApplicationManager

- : *UCX or QAX*: which part of the original use case is the responsibility of this component

NotificationHandler

- : *UCX or QAX*: which part of the original use case is the responsibility of this component

Av1: Communication between SIoTIP gateway and Online Service Av2: Application failure P1: Large number of users M1: Integrate new sensor or actuator manufacturer M2: Big data analytics on pluggable data and/or application usage data U1: Application updates U2: Easy installation

UC1: Register a customer organisation UC2: Register an end-user UC3: Unregister an end-user UC4: Install mote UC5: Uninstall mote UC6: Insert a pluggable device into a mote UC7: Remove a pluggable device from its mote UC8: Initialise a pluggable device UC9: Configure pluggable device access rights UC10: Consult and configure the topology UC12: Perform actuation command UC13: Configure pluggable device UC16: Consult notification message UC17: Activate an application 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 from external front-end UC28: Log in UC29: Log out

1.2 Decomposition 2: Module (drivers)

1.2.1 Module to decompose

In this run we decompose .

1.2.2 Selected architectural drivers

The non-functional drivers for this decomposition are:

- x: x

The related functional drivers are:

- x: x

Rationale

1.2.3 Architectural design

Topic Discussion of the solution selected for (a part of) one of the architectural drivers.

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.

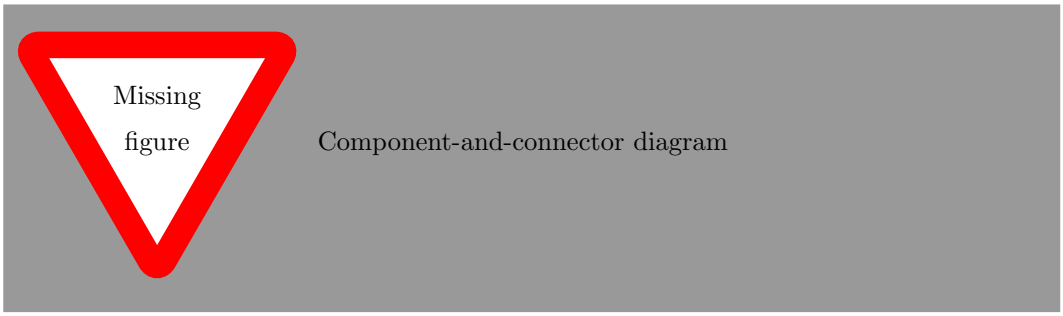

Figure 1.3: Component-and-connector diagram of this decomposition.

Behaviour If needed and explanation of the behaviour of certain aspects of the design so far.

Deployment Rationale of the allocation of components to physical nodes.

1.2.5 Interfaces for child modules

ModuleB

- InterfaceA
  - `returnType operation1(ParamType param1)` throws TypeOfException
    - Effect: Describe the effect of calling this operation.

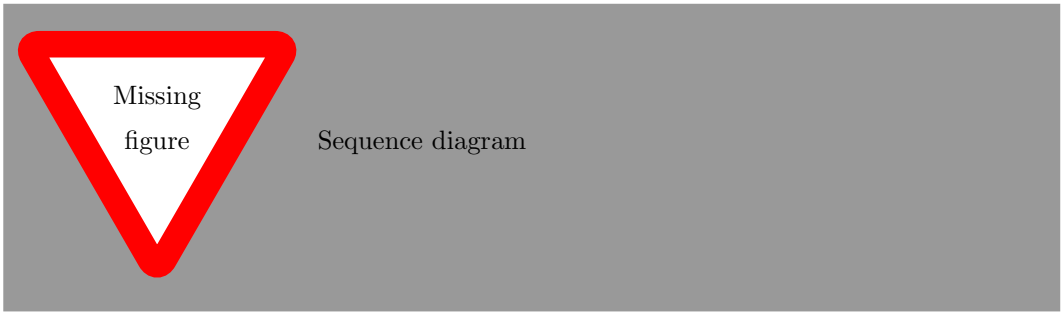

Figure 1.4: Sequence diagram illustrating a key behavioural aspect.

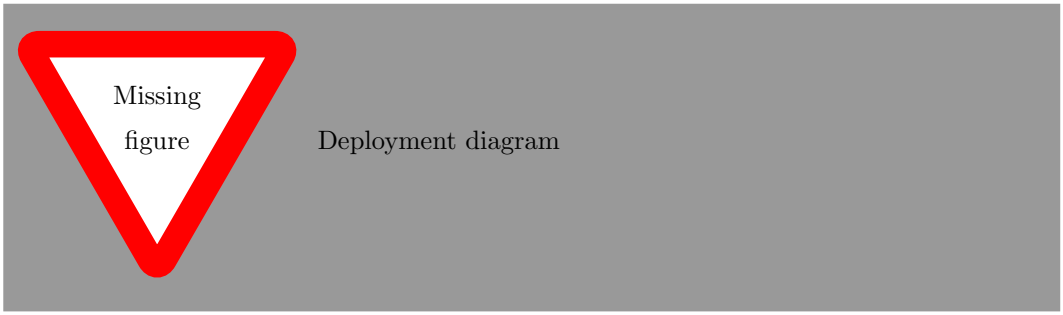

Figure 1.5: Deployment diagram of this decomposition.

    - Exceptions:
        - TypeOfException: Describe when this exception is thrown.
- `returnType operation2()`
    - Effect: Describe the effect of calling this operation.
    - Exceptions: None

1.2.6 Data type definitions

Describe per complex data type used in the interfaces what it represents.

returnType This data element represents X.

ParamType This data element represents Y.

1.2.7 Verify and refine

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

ModuleB

- *Z1*: name
- *UCd*: name

ModuleC

- *UCba*: name
  Description which part of the original use case is the responsibility of this component.

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.")"