Katholieke
Universiteit
Leuven

**Department of
Computer Science**

Shared Internet Of Things Infrastructure Platform:

The Complete Architecture
Software Architecture (H09B5a and H07Z9a) – Part 2b

FILIPCIKOVA-HALILOVIC

Monika Filipcikova (r0683254)
Armin Halilovic(r0679689)

Academic year 2016–2017

Contents

1.   Architectural Decisions

> ❶ **Note:** This section discusses *all* your architectural decisions *in-depth*. First, *all* decisions related to the non-functionals are discussed in detail. Next, *all* other decisions are listed and discussed.

> ✔ **Hint:** Don't just say *what* you have done. Explain *why* you have done it.

1.1   ReqX: Requirement Name

> 📑 **TODO:** Use this section structure for each requirement

Key Decisions

> 📑 **TODO:** Briefly list your key architectural decisions. Pay attention to the solutions that you employed (in your own terms or using tactics and/or patterns).

- decision 1
- . . .

Employed tactics and patterns: . . .

Rationale

> 📑 **TODO:** Describe the design choices related to *ReqX* together with the rationale of why these choices where made.

Considered Alternatives

Alternative(s) for choice 1   Explain what alternative(s) you considered for this design choice and why they where not selected.

Deployment Decisions

. . .

Considered Deployment Alternatives

. . .

1.2   Other decisions

> 📑 **TODO:** *Optional* If you have made any other important architectural decisions that do not directly fit in the sections of the other qualities you can mention them here.
> Follow the same structure as above.

1.2.1   Decision 1

KeyDecisions

. . .

Rationale

. . .

Considered Alternatives

. . .

Deployment Decisions

. . .

Considered Deployment Alternatives

. . .

1.3   Discussion

> 🗋 **TODO:** Use this section to discuss your architecture in retrospect. For example, what are the strong points of your architecture? What are the weak points? Is there anything you would have done otherwise with your current experience? Are there any remarks about the architecture that you would give to your customers? Etc.

2. Client-server view (UML Component diagram)

> ✔ **Hint:** No need to just repeat what we can see on the diagram.
> Don't do this: *As you can see on fig. x: comp A consists of B and C, and C connects to D.*
> But, please do explain if there is anything non-trivial (e.g., a custom mapping from actors to external components on the context diagram).

> ✔ **Hint:** Add any essential information, necessary for interpreting the figure, in the caption. Be sure to add a separate short title for inclusion in the list of figures: `\caption[shorttitle]{longtitle}`.
> If your explanation becomes too long for the caption, you can create a separate subsection. Don't forget to refer to the figure and vice versa.

> ✔ **Hint:** If you have any doubts about the size of your figures, it is better to make your figure too large than too small. Alternatively, you can test the readability by printing it.

> ⚠ **Attention:** With regard to the context diagram, recall the lectures on what it means and should contain. Be sure not to miss any elements here. This is a frequent source of errors.

> ⚠ **Attention:** Make sure your main component-and-connector and context diagrams are consistent.

2.1 Context diagram

TODO: find a way to display the page horizontally with the image covering the whole page.

The context diagram of the client-server view is displayed in figure 2.1.

The external components are as follows.

- NotificationDeliveryService: blabla
- InfrastructureOwnerClient: blabla
- CustomerOrganisationClient: blabla

> 📄 **TODO:** The context diagram of the client-server view: Discuss which components communicate with external components and what these external components represent.

2.2 Primary diagram

The primary diagram of the client-server view is displayed in figure 2.2.

> 📄 **TODO:** The primary diagram and accompanying explanation.

Figure 2.1: Context diagram for the client-server view.
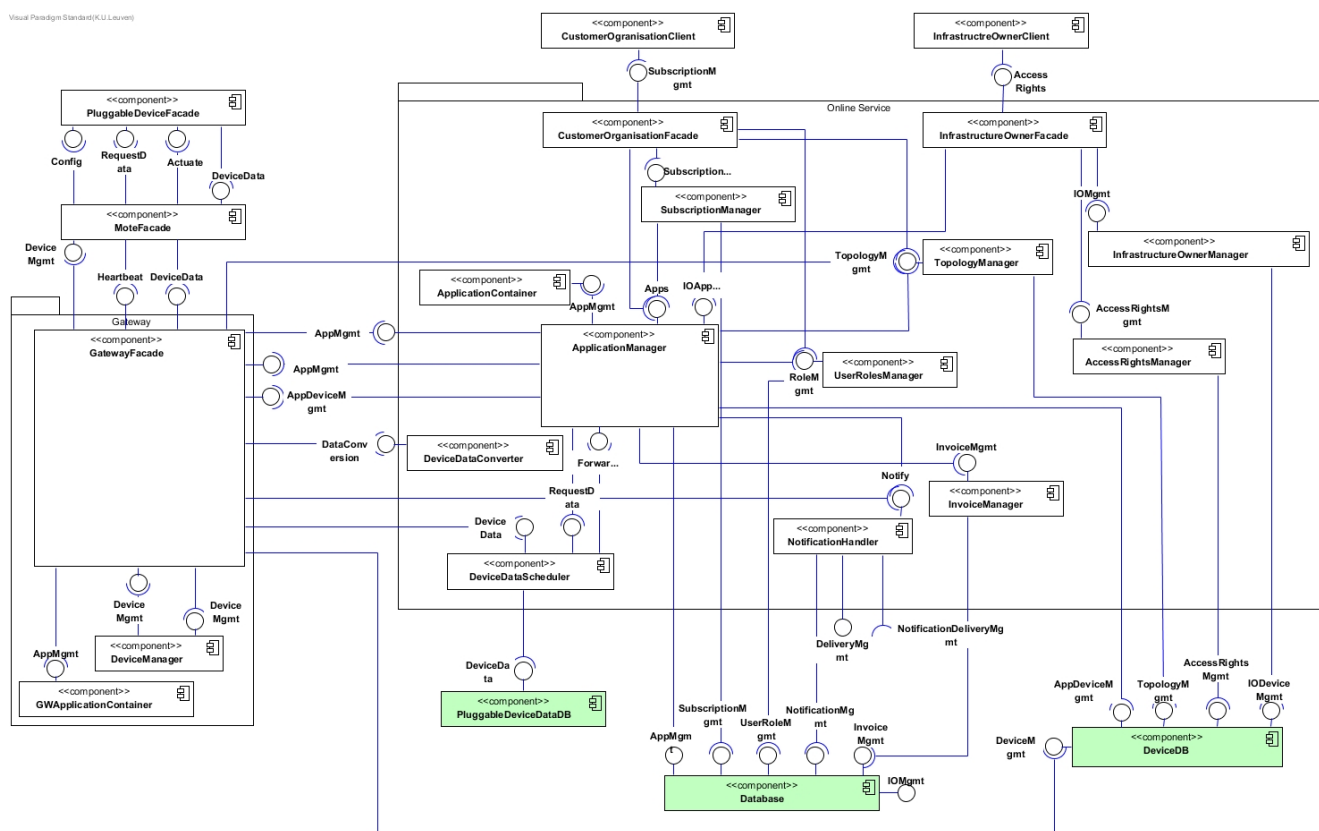
Figure 2.2: Primary diagram of the client-server view.

3. Decomposition view (UML Component diagram)

Figures

> ✔ **Hint:** No need to just repeat what we can see on the diagram.
> Don't do this: *As you can see on fig. x: comp A consists of B and C, and C connects to D.*
> But, please do explain if there is anything non-trivial (e.g., a custom mapping from actors to external components on the context diagram).

> ✔ **Hint:** Add any essential information, necessary for interpreting the figure, in the caption. Be sure to add a separate short title for inclusion in the list of figures: `\caption[shorttitle]{longtitle}`.
> If your explanation becomes too long for the caption, you can create a separate subsection. Don't forget to refer to the figure and vice versa.

> ⚠ **Attention:** *Consistency between views!* Be sure to check for consistency between the client-server view and your decompositions.

> ⚠ **Attention:** *Consistency of a single decomposition!* Make sure that every interface provided or required by the decomposed component, is provided or required by a subcomponent in the decomposition.

Missing figure   Diagram showing decomposition of ComponentX
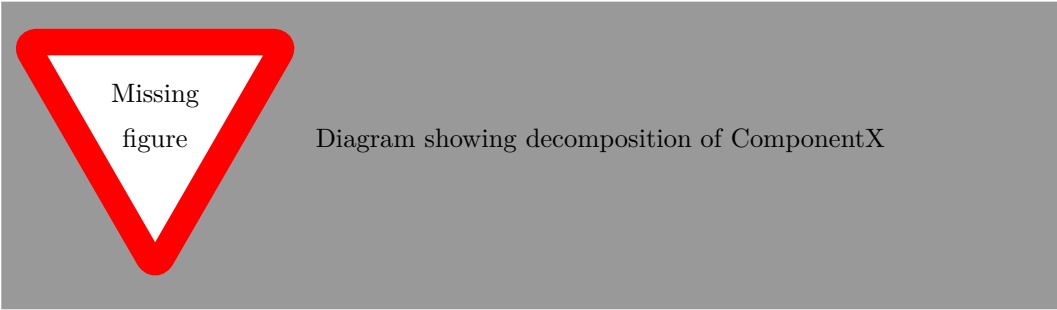
Figure 3.1: Decomposition of `ComponentX`
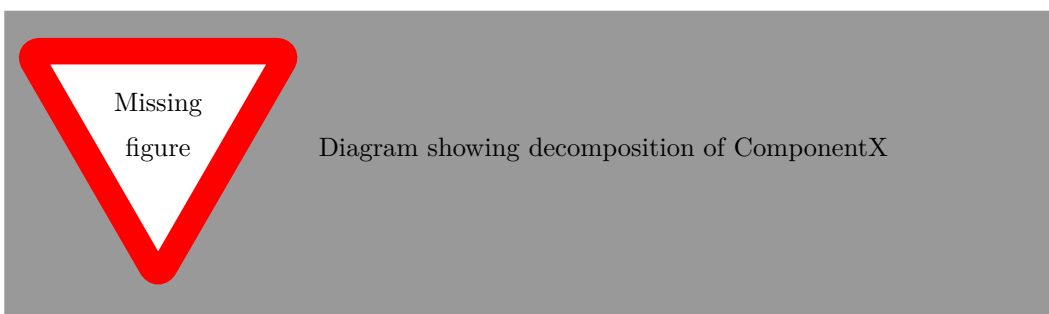
Figure 3.2: Decomposition of `ComponentY`.
This caption contains a longer explanation over multiple lines. This additional explanation is not shown in the list of figures.

4. Deployment view (UML Deployment diagram)

Figures

> ✔ **Hint:** No need to just repeat what we can see on the diagram.
> Don't do this: *As you can see on fig. x: components A and B are deployed on node C.*
> But, please do explain if there is anything non-trivial (e.g., a custom mapping from actors to external components on the context diagram).

> ✔ **Hint:** Add any essential information, necessary for interpreting the figure, in the caption. Be sure to add a separate short title for inclusion in the list of figures: `\caption[shorttitle]{longtitle}`.
> If your explanation becomes too long for the caption, you can create a separate subsection. Don't forget to refer to the figure and vice versa.

> ⚠ **Attention:** Connect nodes on the deployment diagram, *not* components.

> ⚠ **Attention:** *Consistency between views!* Be sure to check for consistency between the client-server/decomposition view and your deployment view.

4.1 Context diagram

TODO: find a way to display the page horizontally with the image covering the whole page.

The context diagram for the deployment view is displayed in figure 4.1.

Components X, Y, Z are deployed on multiple nodes for bla bla bla.
Components A and B communicate using the C protocol...

> 📑 **TODO:** Describe the context diagram for the deployment view. For example, which protocols are used for communication with external systems and why?

4.2 Primary diagram

The primary diagram for the deployment view is displayed in figure 4.2.
TODO: add references to "architectural decisions" where we made some choices related to deployment of components.

> 📑 **TODO:** The primary deployment diagram itself. This discussion on the parts of the deployment diagram which are crucial for achieving certain non-functional requirements, and any alternative deployments that you considered, should be in the architectural decisions chapter.

InfrastructureOwnerClientNode

<<component>>
InfrastructreOwnerClient

CustomerOrganisationClientNode

<<component>>
CustomerOgranisationClient

1..*

1..*

1..*

1..*

OnlineService

1..*

1..*

NotificationDeliveryServiceNode

<<component>>
NotificationDeliveryService

Figure 4.1: Context diagram for the deployment view.

OnlineService

1

1

1

1

PluggableDeviceDatabase

<<component>>
PluggableDeviceDataDB

Database

<<component>>
Database

Figure 4.2: Primary diagram for the deployment view.

5. Scenarios

Figures

> ✔ **Hint:** No need to just repeat what we can see on the diagram.
> Don't do this: *As you can see on fig. x: component A calls operation b, next component C calls operation d.*
> But, please do explain if there is anything non-trivial (e.g., a custom mapping from actors to external components on the context diagram).

> ✔ **Hint:** Add any essential information, necessary for interpreting the figure, in the caption. Be sure to add a separate short title for inclusion in the list of figures: `\caption[shorttitle]{longtitle}`.
> If your explanation becomes too long for the caption, you can create a separate subsection. Don't forget to refer to the figure and vice versa.

> ⚠ **Attention:** Do include a list of which sequence diagrams together illustrate a which scenario from the assignment.

> ✔ **Hint:** Don't only model the 'happy path' in your sequence diagrams. Take into account the quality attributes. For example, what happens when a certain component fails (Av) or overloads (P)? Use the sequence diagrams to illustrate how you have achieved the qualities in your architecture.

5.1   Scenarios

> 🔖 **TODO:** Illustrate how your architecture fulfills the most important data flows. As a rule of thumb, focus on the scenario of the assignment. Describe the scenario in terms of architectural components using UML Sequence diagrams and further explain the most important interactions in text. Illustrating the scenarios serves as a quick validation of the completeness of your architecture. If you notice at this point that for some reason, certain functionality or qualities are not addressed sufficiently in your architecture, it suffices to document this, together with a rationale of why this is the case according to you. You do not have to further refine you architecture at this point.

This section lists which sequence diagrams belong to which scenarios:

- UC11: Sensor data being processed by the system
  Figure 5.1
- UC19: Subscribing to an application
  Figure 5.2
- UC12: Applications issuing actuation commands
  Figure 5.3
- UC14, Av3, UC18: Sensors/actuators failing
  Figure 5.4
  This scenario displays the data flow when sensors/actuators fail, causing

- deactivation of specific applications
  - a redundant sensor/actuator to take over in the context of a single application
- Av2: Application crash
  Figure 5.5
- U2, UC4: Plugging in a new pluggable device (sensor or actuator)
  Figure 5.6
- Av1, UC15: Detection and handling of communication channel failure
  Figure 5.7
- UC22, U1: Upgrading an application
  Figure 5.8
- UC26, UC27, UC12: Sending actuation commands via a mobile app
  Figure 5.9
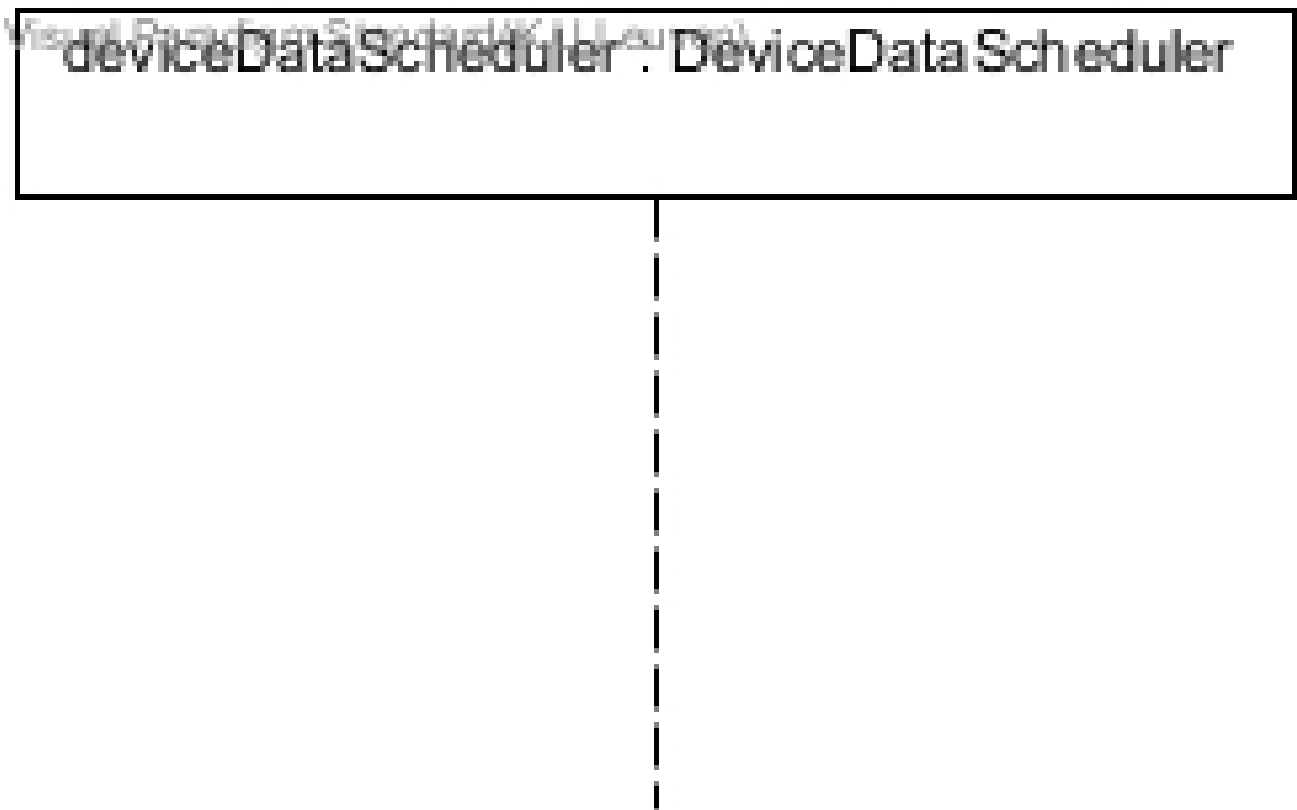
Figure 5.1: EXPLAIN WHAT HAPPENS IN THE SCENARIO.
ADD COMMENTS.
LINK TO OTHER RELEVANT SCENARIO'S.
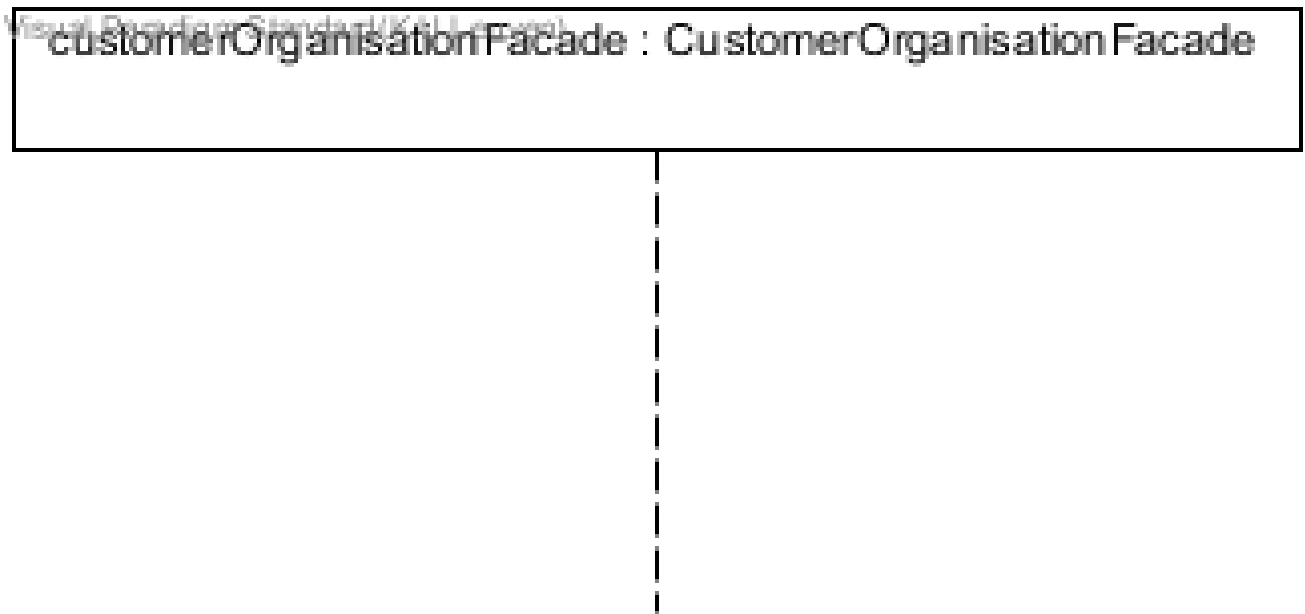
Figure 5.2: EXPLAIN WHAT HAPPENS IN THE SCENARIO.
ADD COMMENTS.
LINK TO OTHER RELEVANT SCENARIO'S.

application Manager : ApplicationManager
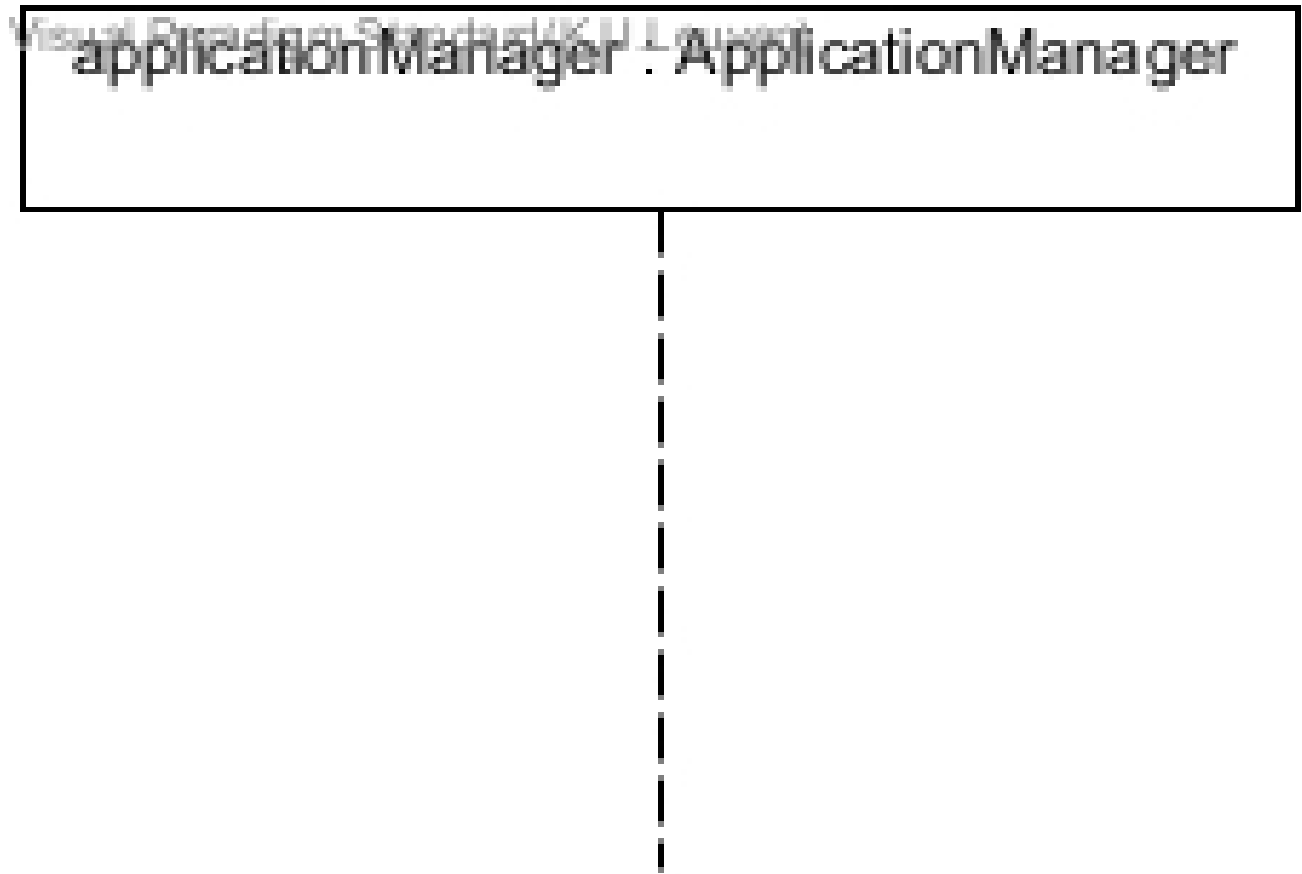
Figure 5.3: EXPLAIN WHAT HAPPENS IN THE SCENARIO.
ADD COMMENTS.
LINK TO OTHER RELEVANT SCENARIO'S.
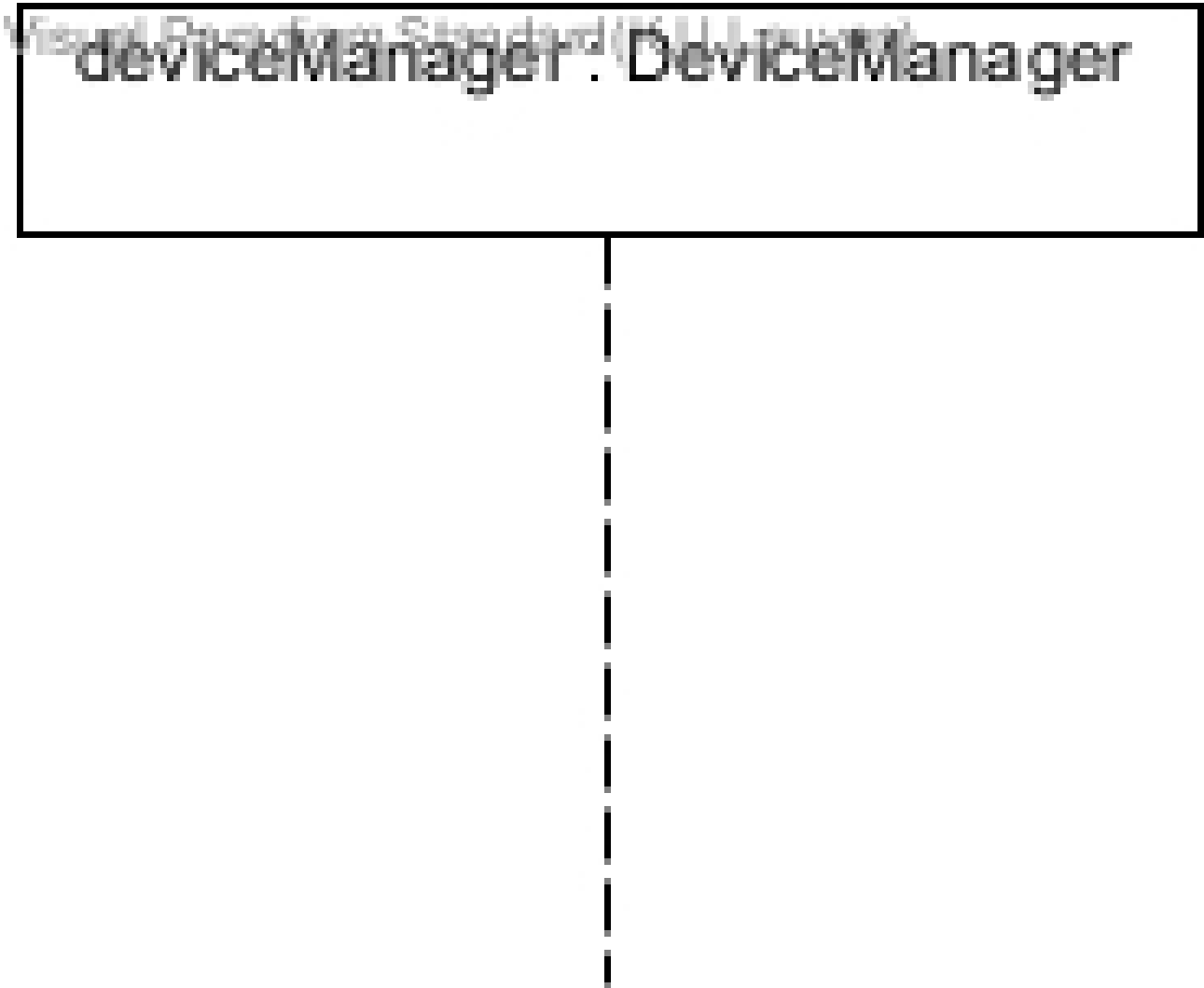
Figure 5.4: EXPLAIN WHAT HAPPENS IN THE SCENARIO.
ADD COMMENTS.
LINK TO OTHER RELEVANT SCENARIO'S.

application Manager : ApplicationManager
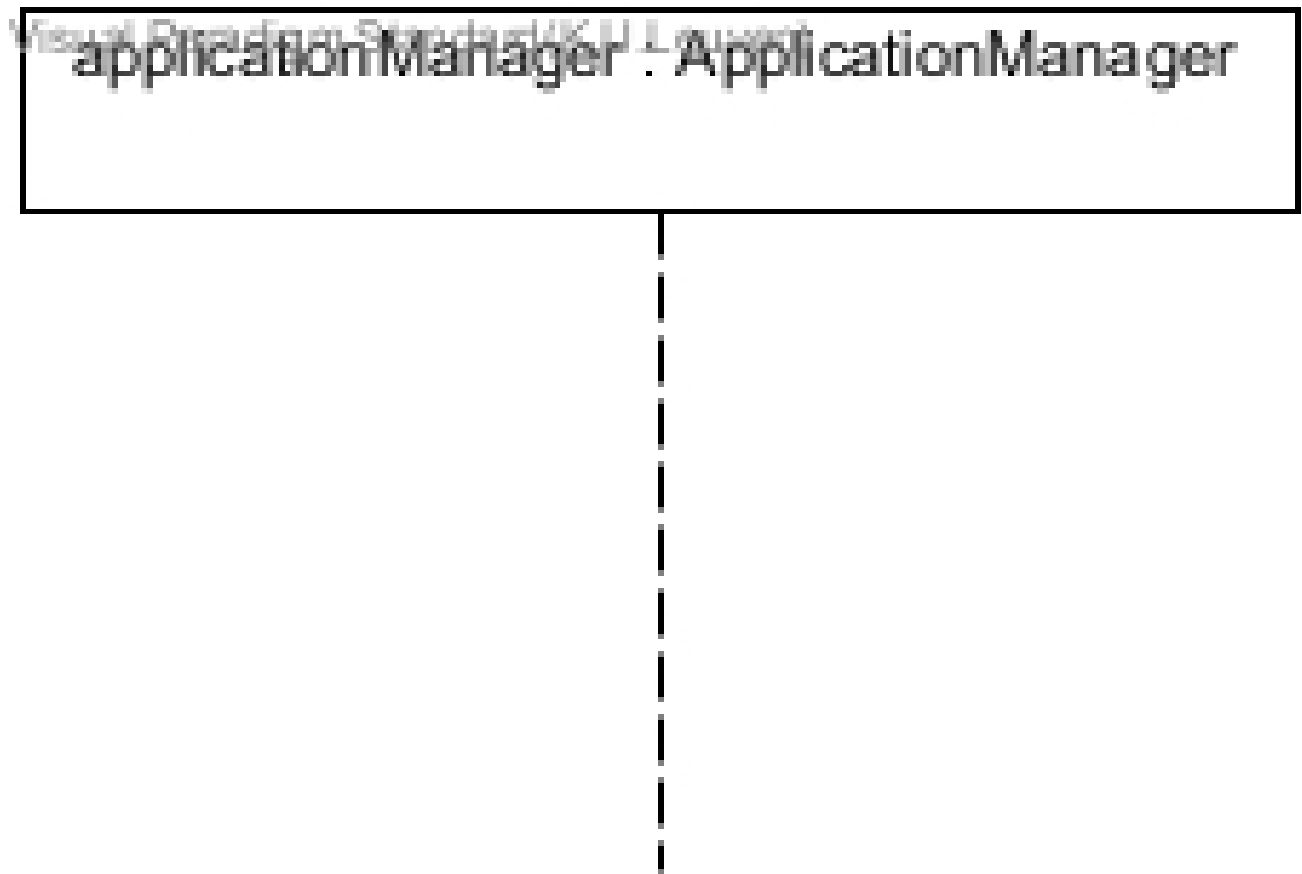
Figure 5.5: EXPLAIN WHAT HAPPENS IN THE SCENARIO.
ADD COMMENTS.
LINK TO OTHER RELEVANT SCENARIO'S.

gatewayFacade : GatewayFacade
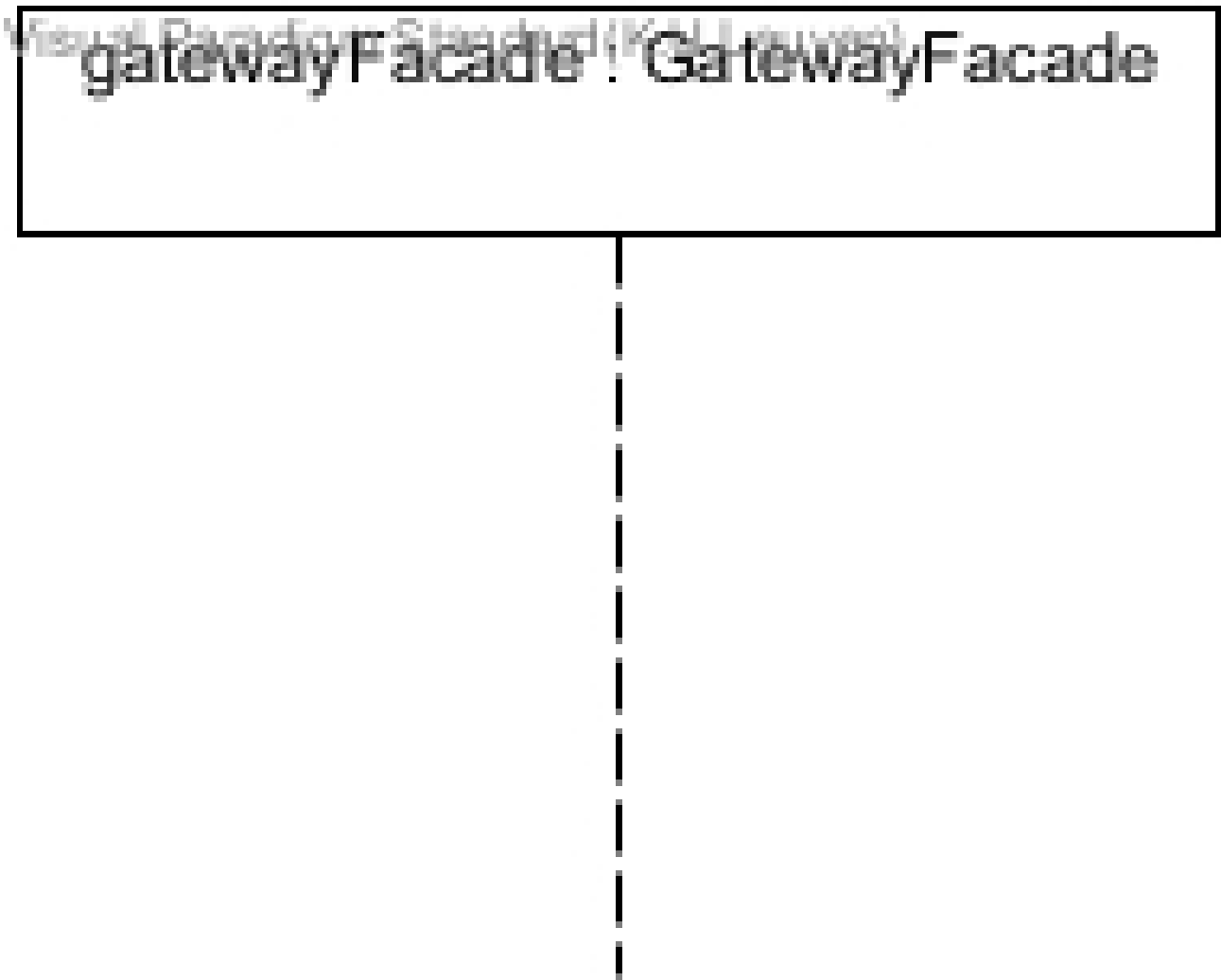
Figure 5.6: EXPLAIN WHAT HAPPENS IN THE SCENARIO.
ADD COMMENTS.
LINK TO OTHER RELEVANT SCENARIO'S.

gatewayFacade : GatewayFacade
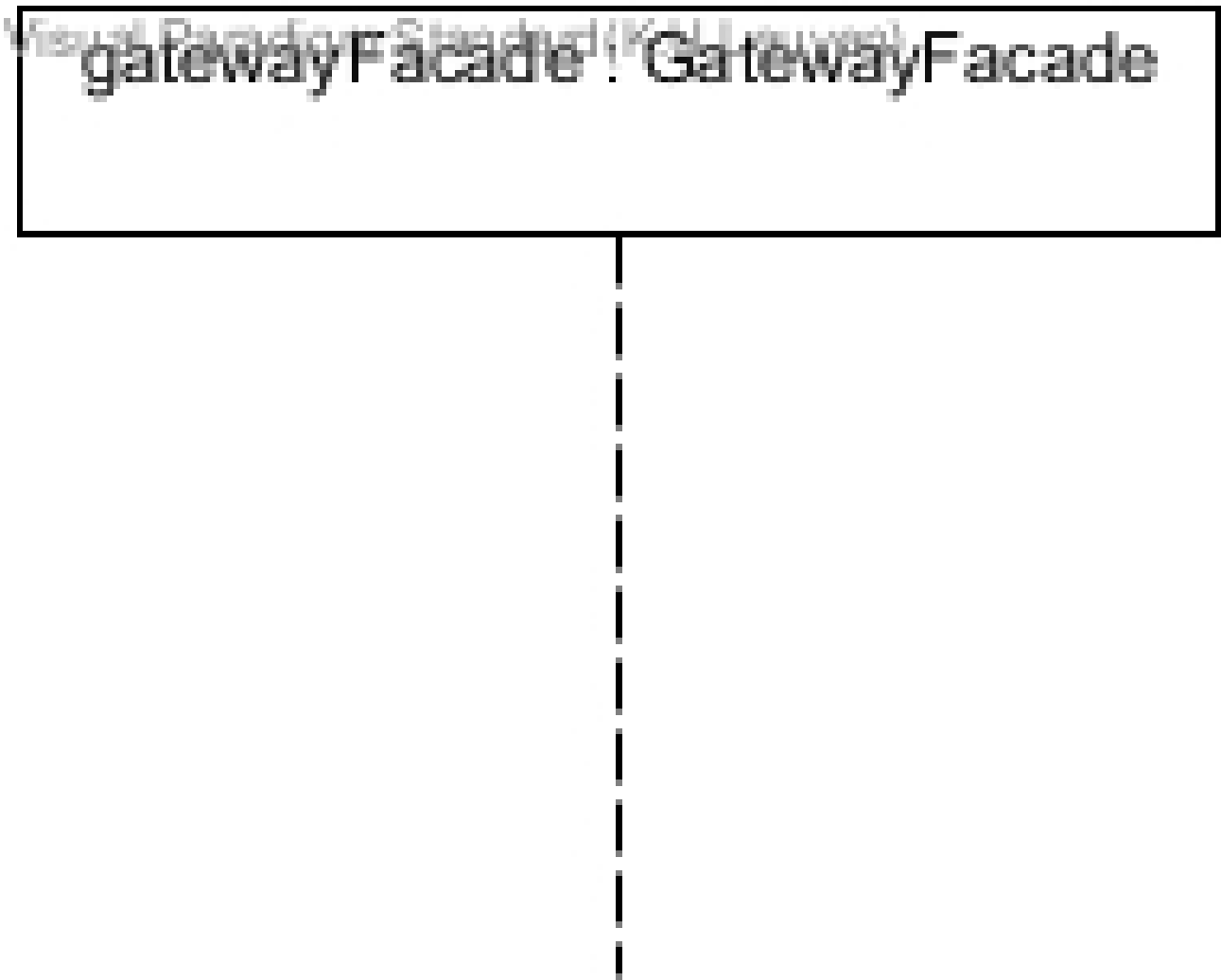
Figure 5.7: EXPLAIN WHAT HAPPENS IN THE SCENARIO.
ADD COMMENTS.
LINK TO OTHER RELEVANT SCENARIO'S.

applicationManager : ApplicationManager
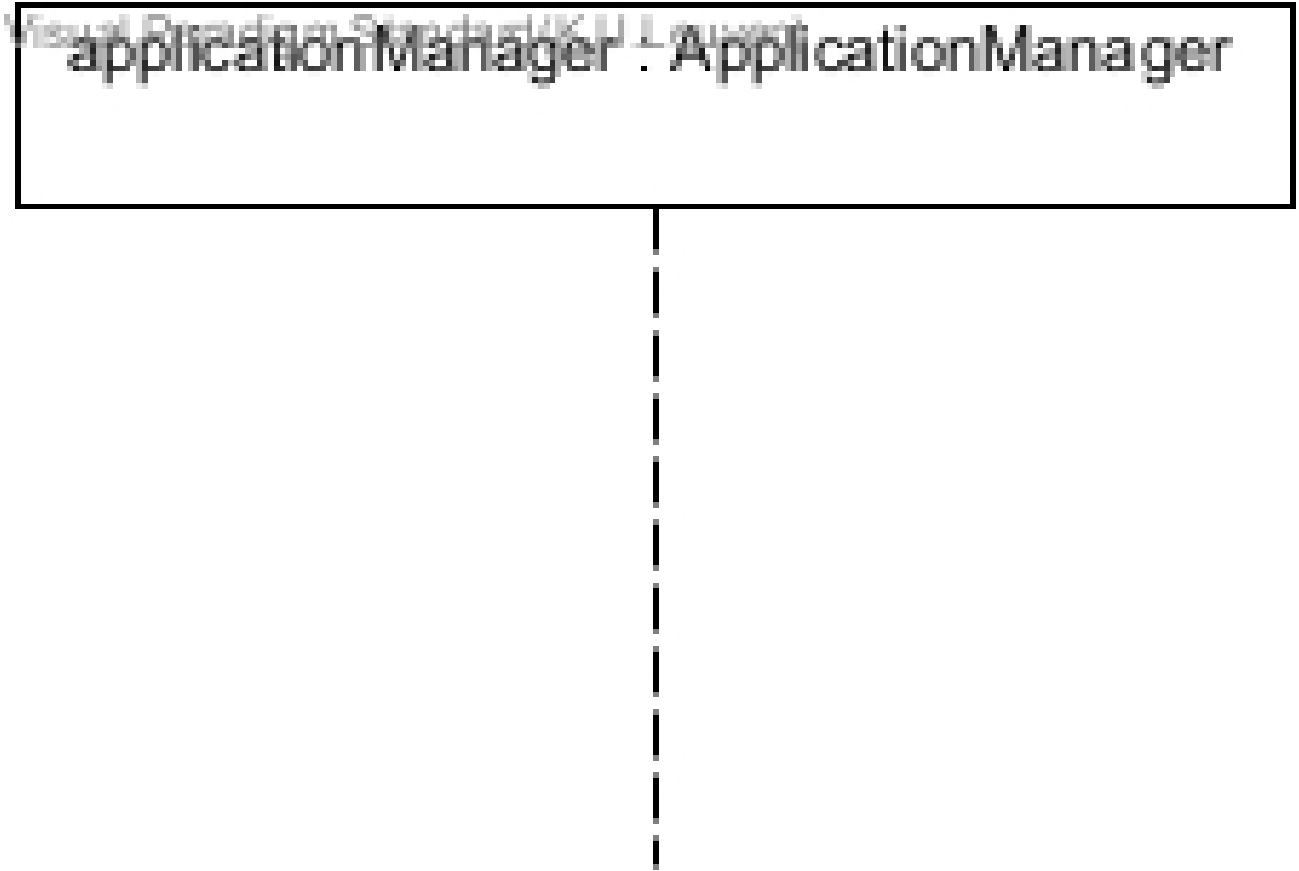
Figure 5.8: EXPLAIN WHAT HAPPENS IN THE SCENARIO.
ADD COMMENTS.
LINK TO OTHER RELEVANT SCENARIO'S.

application Manager : ApplicationManager
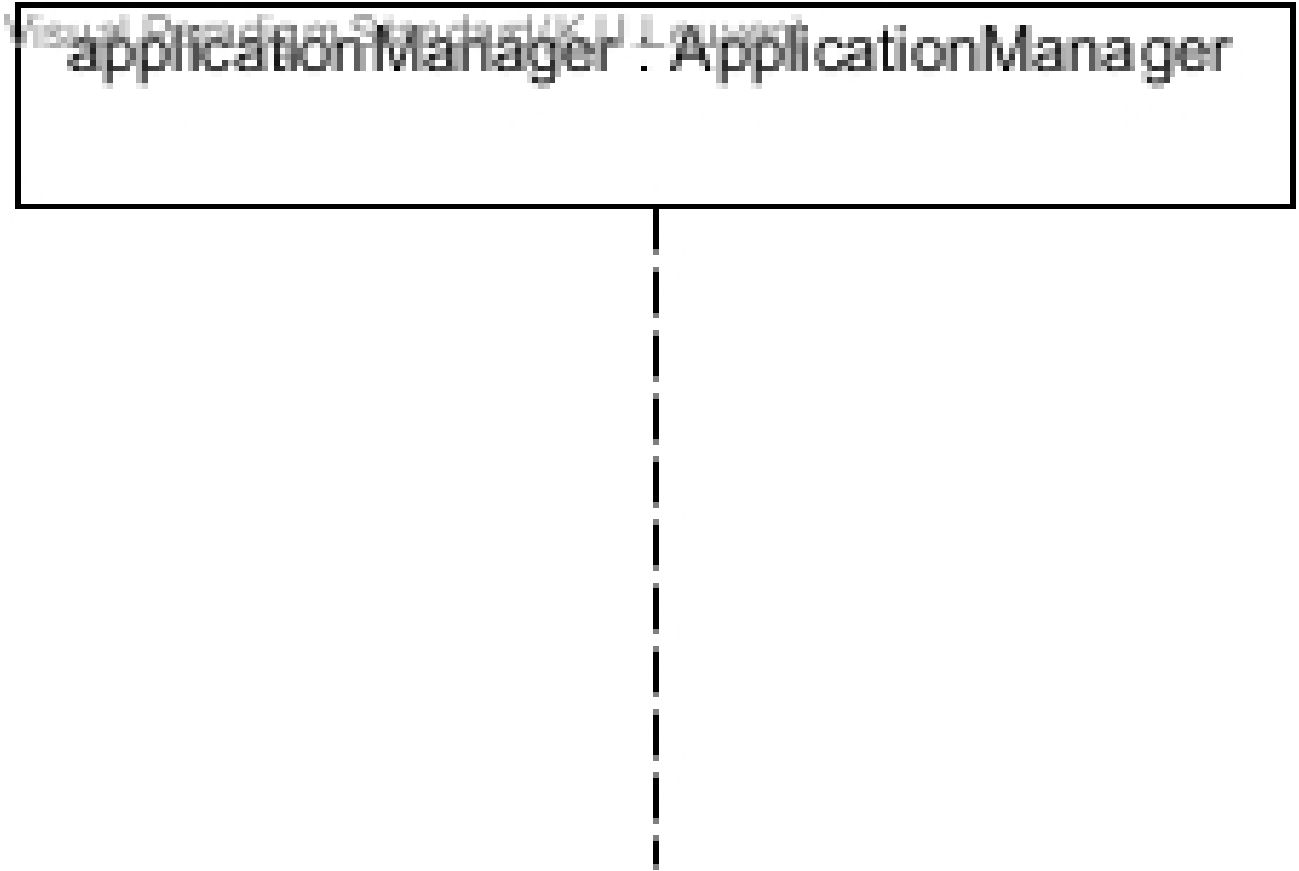
Figure 5.9: EXPLAIN WHAT HAPPENS IN THE SCENARIO.
ADD COMMENTS.
LINK TO OTHER RELEVANT SCENARIO'S.

6.   Element Catalog and Datatypes

Each method contains a short note on why the method was added (under "Created for"). This was done to keep track of our decisions and does not mean that the methods can only be used for the Quality Attribute/Use Case referenced in the "Created for" note.

7. Catalog

7.1   Components

7.1.1   AccessRightsManager

Responsibility: Undefined
Super-components: None
Sub-components: None
Provided interfaces: ⊶ AccessRightsMgmt
Required interfaces: ⊰ AccessRightsMgmt

7.1.2   ActuationCommandConstructor

Responsibility: Undefined
Super-components: ⊟ ApplicationManager
Sub-components: None
Provided interfaces: ⊶ Actuation
Required interfaces: ⊰ AppDeviceMgmt

7.1.3   ApplicationClient

Responsibility: Undefined
Super-components: None
Sub-components: None
Provided interfaces: None
Required interfaces: ⊰ AppData, ⊰ DeviceMgmt, ⊰ TopologyOverview

7.1.4   ApplicationContainer

Responsibility: Undefined
Super-components: ⊟ ApplicationManager
Sub-components: None
Provided interfaces: ⊶ AppInstanceMgmt
Required interfaces: None

7.1.5   ApplicationContainerManager

Responsibility: Undefined
Super-components: ⊟ ApplicationManager
Sub-components: None
Provided interfaces: None
Required interfaces: ⊰ AppInstanceMgmt

7.1.6   ApplicationContainerMonitor

Responsibility: Undefined
Super-components: ⊟ ApplicationManager
Sub-components: None
Provided interfaces: None

Required interfaces: ⊰ `AppInstanceMgmt`

7.1.7 ApplicationFacade

Responsibility: Undefined
Super-components: None
Sub-components: None
Provided interfaces: ⊸ `AppData`, ⊸ `DeviceMgmt`, ⊸ `TopologyOverview`
Required interfaces: ⊰ `Actuation`, ⊰ `Apps`

7.1.8 ApplicationManager

Responsibility: Responsible for activating/deactivating applications, setting pluggable device redundancy requirements on `DeviceManager` components, and using `NotificationHandler` to send notifications to customer organisations.
Super-components: None
Sub-components: ▣ `ActuationCommandConstructor`, ▣ `ApplicationContainerMonitor`, ▣ `ApplicationContainer`, ▣ `ApplicationContainerManager`
Provided interfaces: ⊸ `Actuation`, ⊸ `AppMgmt`, ⊸ `Apps`, ⊸ `ForwardData`, ⊸ `IOAppMgmt`
Required interfaces: ⊰ `AppDeviceMgmt`, ⊰ `AppMgmt`, ⊰ `AppMgmt`, ⊰ `InvoiceMgmt`, ⊰ `Notify`, ⊰ `RequestData`, ⊰ `RoleMgmt`, ⊰ `TopologyMgmt`

7.1.9 CustomerOgranisationClient

Responsibility: Undefined
Super-components: None
Sub-components: None
Provided interfaces: None
Required interfaces: ⊰ `SubscriptionMgmt`

7.1.10 CustomerOrganisationFacade

Responsibility: Undefined
Super-components: None
Sub-components: None
Provided interfaces: ⊸ `SubscriptionMgmt`
Required interfaces: ⊰ `Apps`, ⊰ `RoleMgmt`, ⊰ `SubscriptionMgmt`, ⊰ `TopologyMgmt`

7.1.11 DeviceDataConverter

Responsibility: The `DeviceDataConverter` is resposible for converting pluggable device data in the data processing subsystem.
Super-components: None
Sub-components: None
Provided interfaces: ⊸ `DataConversion`
Required interfaces: None

7.1.12 DeviceDataScheduler

Responsibility: Responsible for scheduling incoming read and write requests for pluggable device data. Monitors throughput of requests and switches between normal and overload mode when appropriate. Avoids starvation of any type of request.
Super-components: None
Sub-components: None

Provided interfaces: ⊶ DeviceData, ⊶ RequestData
Required interfaces: ⊰ DeviceData, ⊰ ForwardData

7.1.13   DeviceDB

Responsibility: Undefined
Super-components: None
Sub-components: None
Provided interfaces: ⊶ AccessRightsMgmt, ⊶ DeviceMgmt, ⊶ IODeviceMgmt, ⊶ TopologyMgmt
Required interfaces: None

7.1.14   DeviceManager

Responsibility: 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.
Super-components: None
Sub-components: None
Provided interfaces: ⊶ DeviceMgmt
Required interfaces: ⊰ DeviceMgmt

7.1.15   GatewayFacade

Responsibility: Main component on the gateway that allows different components to work with each
other.   E.g.   transmits heartbeats from motes to `DeviceManager`, transmits commands to shut down
applications, triggers notifications to be generated, ...
Super-components: None
Sub-components: None
Provided interfaces: ⊶ AppDeviceMgmt, ⊶ AppMgmt, ⊶ DeviceData, ⊶ DeviceMgmt, ⊶ Heartbeat
Required interfaces: ⊰ AppMgmt, ⊰ AppMgmt, ⊰ DataConversion, ⊰ DeviceData, ⊰ DeviceMgmt,
⊰ DeviceMgmt, ⊰ DeviceMgmt, ⊰ Notify, ⊰ Other, ⊰ TopologyMgmt

7.1.16   GWApplicationContainerManager

Responsibility: Undefined
Super-components: None
Sub-components: None
Provided interfaces: ⊶ AppMgmt
Required interfaces: None

7.1.17   InfrastructreOwnerClient

Responsibility: Undefined
Super-components: None
Sub-components: None
Provided interfaces: None
Required interfaces: ⊰ AccessRights

7.1.18   InfrastructureOwnerFacade

Responsibility: Undefined
Super-components: None
Sub-components: None
Provided interfaces: ⊶ AccessRights
Required interfaces: ⊰ AccessRightsMgmt, ⊰ IOAppMgmt, ⊰ IOMgmt

7.1.19 InfrastructureOwnerManager

Responsibility: `Undefined`
Super-components: None
Sub-components: None
Provided interfaces: ⊸ IOMgmt
Required interfaces: ⊰ IODeviceMgmt, ⊰ IOMgmt

7.1.20 InvoiceManager

Responsibility: `Undefined`
Super-components: None
Sub-components: None
Provided interfaces: ⊸ InvoiceMgmt
Required interfaces: ⊰ InvoiceMgmt

7.1.21 MoteFacade

Responsibility: Sends heartbeats to the `GatewayFacade`. Includes a list of connected pluggable devices in the heartbeats.
Super-components: None
Sub-components: None
Provided interfaces: ⊸ DeviceData, ⊸ DeviceMgmt
Required interfaces: ⊰ Actuate, ⊰ Config, ⊰ DeviceData, ⊰ Heartbeat, ⊰ RequestData

7.1.22 NotificationDeliveryService

Responsibility: `Undefined`
Super-components: None
Sub-components: None
Provided interfaces: None
Required interfaces: None

7.1.23 NotificationHandler

Responsibility: Responsible for generation, storage, and delivery of notifications based on users' preferred communication channel.
Super-components: None
Sub-components: None
Provided interfaces: ⊸ DeliveryMgmt, ⊸ Notify
Required interfaces: ⊰ NotificationDeliveryMgmt, ⊰ NotificationMgmt

7.1.24 OtherDataDB

Responsibility: General database for data. For example, storage of data about notifications.
Super-components: None
Sub-components: None
Provided interfaces: ⊸ AppMgmt, ⊸ InvoiceMgmt, ⊸ IOMgmt, ⊸ NotificationMgmt, ⊸ Other, ⊸ SubscriptionMgmt, ⊸ UserRoleMgmt
Required interfaces: None

7.1.25   OtherFunctionality1

Responsibility: Undefined
Super-components: None
Sub-components: None
Provided interfaces: ⊶ Other
Required interfaces: ⊰ Other, ⊰ Other

7.1.26   OtherFunctionality2

Responsibility: Undefined
Super-components: None
Sub-components: None
Provided interfaces: ⊶ Other
Required interfaces: ⊰ Other, ⊰ Other

7.1.27   PluggableDeviceDataDB

Responsibility: Database dedicated to pluggable device data only.
Super-components: None
Sub-components: None
Provided interfaces: ⊶ DeviceData, ⊶ Other
Required interfaces: None

7.1.28   PluggableDeviceFacade

Responsibility: Responsible for sending pluggable device data to `MoteFacade`.   Needs to be initialised in order for the data to be used/stored.
Super-components: None
Sub-components: None
Provided interfaces: ⊶ Actuate, ⊶ Config, ⊶ RequestData
Required interfaces: ⊰ DeviceData

7.1.29   SubscriptionManager

Responsibility: Undefined
Super-components: None
Sub-components: None
Provided interfaces: ⊶ SubscriptionMgmt
Required interfaces: ⊰ Apps, ⊰ SubscriptionMgmt

7.1.30   TopologyManager

Responsibility: Undefined
Super-components: None
Sub-components: None
Provided interfaces: ⊶ TopologyMgmt
Required interfaces: ⊰ TopologyMgmt

7.1.31   UserRolesManager

Responsibility: Undefined
Super-components: None
Sub-components: None

Provided interfaces: ⊸ RoleMgmt
Required interfaces: ⊸ UserRoleMgmt

7.2 Interfaces

7.2.1 AccessRights

Provided by: ⌑ InfrastructureOwnerFacade
Required by: ⌑ InfrastructreOwnerClient
Operations:

- configureDevice(**PluggableDeviceID** pID)
  - Effect: Returns a map of AccessRights and the IDs of customer organisations that have those AccessRights.
  - Created for: UC9.3 - UC9.4
- List<**PluggableDeviceInfo**> getAccessRights(int infrastructureOwnerID)
  - Effect: Returns a list of PluggableDeviceInfo to display so an infrastructure owner can select a device to configure access rights.
  - Created for: UC9.1
- void updateAccessRights()
  - Effect: Updates the access rights on a certain pluggable device for a group of customer organisations.
  - Created for: UC9.6

7.2.2 AccessRightsMgmt

Provided by: ⌑ DeviceDB
Required by: ⌑ AccessRightsManager
Operations:

- getCustomerOrganisationsRights(**PluggableDeviceID** pID, List<int> custOrgIDs)
  - Effect: Returns a map of AccessRights and the IDs of customer organisations that have those AccessRights on a certain pluggable device.
  - Created for: UC9.4
- void updateAccessRights()
  - Effect: Updates the access rights on a certain pluggable device for a group of customer organisations.
  - Created for: UC9.7

7.2.3 AccessRightsMgmt

Provided by: ⌑ AccessRightsManager
Required by: ⌑ InfrastructureOwnerFacade
Operations:

- getCustomerOrganisationsRights(**PluggableDeviceID** pID, List<int> custOrgIDs)
  - Effect: Returns a map of AccessRights and the IDs of customer organisations that have those AccessRights on a certain pluggable device.
  - Created for: UC9.4
- void updateAccessRights()
  - Effect: Updates the access rights on a certain pluggable device for a group of customer organisations.
  - Created for: UC9.7

7.2.4 Actuate

Provided by: ⌑ PluggableDeviceFacade
Required by: ⌑ MoteFacade
Operations:

- void sendActuationCommand(string commandName)

– Effect: Send an actuation command to the actuator.   Sending an unknown actuation command has
no effect.

7.2.5   Actuation

Provided by: ⬚ `ActuationCommandConstructor`, ⬚ `ApplicationManager`
Required by: ⬚ `ApplicationFacade`
Operations:

7.2.6   AppData

Provided by: ⬚ `ApplicationFacade`
Required by: ⬚ `ApplicationClient`
Operations:

7.2.7   AppDeviceMgmt

Provided by: ⬚ `GatewayFacade`
Required by: ⬚ `ActuationCommandConstructor`, ⬚ `ApplicationManager`
Operations:
- bool areEssentialDevicesOperational(int applicationID)
  – Effect: Returns true if all essential devices for the application with id "applicationID" are
  operational.
  – Created for: UC18
- void setPluggableDevicesRequirements(int applicationID, List<**PluggableDeviceInfo**> devices)
  – Effect: Sets an application's requirements for pluggable devices.
  – Created for: Av3:   "Application providers can design their applications such that they explicitly
  require redundancy in the available pluggable devices."
  TODO: update this with Relationship type?

7.2.8   AppInstanceMgmt

Provided by: ⬚ `ApplicationContainer`
Required by: ⬚ `ApplicationContainerManager`, ⬚ `ApplicationContainerMonitor`
Operations:

7.2.9   AppMgmt

Provided by: ⬚ `OtherDataDB`
Required by: ⬚ `ApplicationManager`
Operations:
- void activateApplication(int applicationInstanceID, string status)
  – Effect: Sets an ApplicationInstance's status in the `OtherDataDB` to 'active'.
  – Created for: UC17.4, U2 - easy applications
- int createNewApplicationInstance(int custOrgID, int applicationID)
  – Effect: Creates a new ApplicationInstance for an application for a customer organisation and returns
  its id.
  – Created for: UC19.4, U2 - easy applications
- List< Application > getApplications()
  – Effect: Returns a list of applications in the system.
  – Created for: UC19.2, U2 - easy applications
- List<int> getApplicationsForDevice()
  – Effect: Returns a list of applications that can use the device with id "pID".
  – Created for: UC11:   the system looks up the list of applications that use the pluggable device

- List<**PluggableDeviceID**> getDevicesForApplication(int applicationInstanceID)
  - Effect: Returns a list of PluggableDeviceID of pluggable devices that an ApplicationInstance can use.
  - Created for: UC17.2, U2 - easy applications
- string getInstallationInstructions(int applicationID)
  - Effect: Returns the installation instructions of a certain application. If there are no installation instructions set, returns an empty string.
  - Created for: UC17.6, U2 - easy applications
- List< RoomTopology > getNecessaryDevicesAndTopologyConfigurations(int applicationID)
  - Effect: Returns a list of RoomTopology which is a minimal requirement for a certain application to run. This can used to display the requirements to a user or to check if requirements are fulfilled.
  - Created for: UC19.5, U2 - easy applications
- void updateApplication(**ApplicationInstance** instance)
  - Effect: Updates an application in the database (e.g. change state to 'inactive').
  - Created for: UC18, Av3: automatic suspension/reactivation of applications.
- void updateApplicationDevicesSettings(int applicationInstanceID, List<**PluggableDeviceID**> devices, List< Relationships > relationships)
  - Effect: Updates an ApplicationInstance's device settings. This includes which devices the instance can use and which relationships exist between those devices.
  - Created for: UC19.6, U2 - easy applications
- void updateCriticality(int applicationInstanceID, int isCritical)
  - Effect: Updates the criticality of an ApplicationInstance.
  - Created for: UC19.11, U2 - easy applications
- void updateSubscription(**Subscription** subscription)
  - Effect: Updates a subscription in the database (e.g. change state to 'disabled').
  - Created for: UC18

7.2.10 AppMgmt

Provided by: ▣ GatewayFacade
Required by: ▣ ApplicationManager
Operations:
- void activateApplicationInstance(int applicationInstanceID)
  - Effect: Activates an ApplicationInstance that is running on the gateway.
  - Created for: UC17.3, U2 - easy applications

7.2.11 AppMgmt

Provided by: ▣ GWApplicationContainerManager
Required by: ▣ GatewayFacade
Operations:
- void activateApplicationInstance(int applicationInstanceID)
  - Effect: Activates an ApplicationInstance that is running on the gateway.
  - Created for: UC17.3, U2 - easy applications

7.2.12 AppMgmt

Provided by: ▣ ApplicationManager
Required by: ▣ GatewayFacade
Operations:
- void activateApplicationInstance(int applicationInstanceID)
  - Effect: Activates a new instance of an application.
  - Created for: UC18, Av3: automatic suspension/reactivation of applications.

- void checkApplicationsForActivationForInfrastructureOwner(int infrastructureOwnerID)
  - Effect: Checks and activates applications which can now execute again. The applications checked are those that are subscribed to by customers organisations associated to the given infrastructure owner.
  - Created for: UC17, UC6.3 - reintroduced device
- void deactivateApplicationInstance(int applicationInstanceID)
  - Effect: Deactivates a running instance of an application.
  - Created for: UC18, Av3: automatic suspension/reactivation of applications.
- void updateApplicationDevicesSettings(int applicationInstanceID, List<**PluggableDeviceID**> devices, List< Relationships > relationships)
  - Effect: Updates an ApplicationInstance's device settings. This includes which devices the instance can use and which relationships exist between those devices.
  - Created for: UC19.6, U2 - easy applications

7.2.13 AppMgmt

Provided by: None
Required by: None
Operations:

7.2.14 Apps

Provided by: ▣ ApplicationManager
Required by: ▣ ApplicationFacade, ▣ CustomerOrganisationFacade, ▣ SubscriptionManager
Operations:
- void activateApplication(int applicationInstanceID)
  - Effect: Activates an ApplicationInstance.
  - Created for: UC19.14, U2 - easy applications
- int createNewApplicationInstance(int custOrgID, int applicationID)
  - Effect: Creates a new ApplicationInstance for an application for a customer organisation and returns its id.
  - Created for: UC19.4, U2 - easy applications
- List< Application > getApplications()
  - Effect: Returns a list of applications in the system.
  - Created for: UC19.2, U2 - easy applications
- List< RoomTopology > getNecessaryDevicesAndTopologyConfigurations(int applicationID)
  - Effect: Returns a list of RoomTopology which is a minimal requirement for a certain application to run. This can used to display the requirements to a user or to check if requirements are fulfilled.
  - Created for: UC19.5, U2 - easy applications
- void updateCriticality(int applicationInstanceID, boolean isCritical)
  - Effect: Updates the criticality of an ApplicationInstance.
  - Created for: UC19.11, U2 - easy applications

7.2.15 Config

Provided by: ▣ PluggableDeviceFacade
Required by: ▣ MoteFacade
Operations:
- Map<String, String> getConfig()
  - Effect: Returns the current configuration of a pluggable device as a parameter-value map.
- boolean setConfig()
  - Effect: Set the given configuration parameters of the pluggable device to the given values. Setting unknown parameters on a pluggable device (e.g., 'noise threshold' -> '3' on a light sensor) has no

effect.

   – Created for: Given constraint, UC11:   pluggable device needs to be initialised, M1:   pluggable
     device must be able to be initialised

7.2.16   DataConversion

Provided by: ▯ `DeviceDataConverter`
Required by: ▯ `GatewayFacade`
Operations:
- **DeviceData** convert(**PluggableDeviceID** pID, **DeviceData** data, string type)
   – Effect: Converts pluggable device data into other pluggable device data that contains the same
     information in a different type.
   – Created for: M1:   data processing subsystem should be extended with relevant data conversions

7.2.17   DeliveryMgmt

Provided by: ▯ `NotificationHandler`
Required by: None
Operations:
- void acknowledgement(int notificationID)
   – Effect: Sends an acknowledgement to the system for a certain notification to denote that a
     notification has been received.
   – Created for: UC15

7.2.18   DeviceData

Provided by: ▯ `PluggableDeviceDataDB`
Required by: ▯ `DeviceDataScheduler`
Operations:
- List<**DeviceData**> getData(**PluggableDeviceID** pID, **DateTime** from, **DateTime** to)
   – Effect: Returns data from a specific device in a certain time period.
   – Created for: P2:   lookup queries
- void rcvData(**PluggableDeviceID** pID, **DeviceData** data)
   – Effect: Sends pluggable device data to the DB to be stored.
   – Created for: UC11, P2:   storing new pluggable data

7.2.19   DeviceData

Provided by: ▯ `MoteFacade`
Required by: ▯ `PluggableDeviceFacade`
Operations:
- void rcvData()
   – Effect: Propagates pluggable device data to the connected gateway by calling rcvData on the
     gateway.   (Initiated by the device).
   – Created for: UC11, P2:   storing new pluggable data
- void rcvDataCallback(**PluggableDeviceID** pID, **DeviceData** data, int requestID)
   – Effect: Propagates pluggable device data to the connected gateway by calling rcvData on the
     gateway.   (Callback of getDataAsync).
   – Created for: UC11, P2:   storing new pluggable data

7.2.20   DeviceData

Provided by: ▯ `GatewayFacade`
Required by: ▯ `MoteFacade`

Operations:

- void rcvData(**PluggableDeviceID** pID, **DeviceData** data)
    - Effect: Provides pluggable device data to the gateway (Initiated by the device).
- void rcvDataCallback(**PluggableDeviceID** pID, **DeviceData** data, int requestID)
    - Effect: Provides device data to the gateway (Callback of getDataAsync).

7.2.21   DeviceData

Provided by: [⊡] `DeviceDataScheduler`
Required by: [⊡] `GatewayFacade`
Operations:

- void rcvData(**PluggableDeviceID** pID, **DeviceData** data)
    - Effect: Sends pluggable device data to the scheduler to be processed.
    - Created for: UC11, P2:   storing new pluggable data

7.2.22   DeviceMgmt

Provided by: [⊡] `MoteFacade`
Required by: [⊡] `GatewayFacade`
Operations:

- List<**PluggableDeviceInfo**> getConnectedDevices()
    - Effect: Effect: Returns a list of information about devices that are connected to the mote.
    - Created for: UC18
    - Tradeoff: send PluggableDeviceID instead of DeviceInfo.   If you send DeviceInfo, then `ApplicationManager` does not have to fetch this info.   If you send PluggableDeviceID's, then less bandwidth is used and the Gateways do less work.
- void setConfig(**PluggableDeviceID** pID, Map<String, String> config)
    - Effect: Set the given configuration parameters of a PluggableDevice to the given values.   Setting unknown parameters on a PluggableDevice has no effect.

    - Created for: UC11:   pluggable device needs to be initialised, M1:   pluggable device must be able to be initialised

7.2.23   DeviceMgmt

Provided by: [⊡] `DeviceDB`
Required by: [⊡] `GatewayFacade`
Operations:

- void addDevice(**PluggableDeviceID** pID, **PluggableDeviceType** type, Map<string, string> configurations, int moteID)
    - Effect: Adds a new pluggable device in the `DeviceDB` and adds a reference to a mote.   The device's status is 'uninitialised' by default and it's current configurations (which are now the default configurations) are stored as well.   If the device already exists, removes the data first (in case the device is plugged into a different mote or on a different network).
    - Created for: UC6.3, U2 - easy pluggable device installation
- int addMote(**MoteInfo** mote, int gatewayID, IPAddress moteIPAddress)
    - Effect: Adds a new mote in the `DeviceDB` along with an IP address and a reference to a gateway. Returns the DB id for the mote.
    - Created for: UC4.3, U2 - easy mote installation
- Map<string, string> getConfigDB(**PluggableDeviceID** pID)
    - Effect: Gets the last set configurations of a pluggable device from the `DeviceDB`.
    - Created for: UC6.3 - reintroduced device
- void reactivateDevice(**PluggableDeviceID** pID)
    - Effect: Changes the status of a pluggable device to 'active'.

– Created for: UC6.3 - reintroduced device
- void reactivateMote(int moteID)
    – Effect: Changes the status of the mote with DB id 'moteID' to 'active'.
    – Created for: U2 - Reintroducing a previously known mote should not require any con
      guration.
- void registerGateway(int gatewayID, IPAddress address)
    – Effect: Sets a gateway's status to 'active' and updates its IP address.
    – Created for: U2 - gateway installation

7.2.24   DeviceMgmt

Provided by: ⬚ DeviceManager
Required by: ⬚ GatewayFacade
Operations:
- bool areEssentialDevicesOperational(int applicationID)
    – Effect: Returns true if all essential devices for the application with id "applicationID" are
      operational.
    – Created for: UC18
- void heartbeat(int moteID, List<**PluggableDeviceInfo**> devicesmeter)
    – Effect: Sends a heartbeat from a mote to check/update timers for operational devices.
    – Created for: UC14, Av3:   failure detection
- bool isDeviceInitialised(**PluggableDeviceID** pID)
    – Effect: Returns true if the device with id "pID" has been initialized.
    – Created for: UC11:   pluggable device needs to be initialised, M1:   pluggable device must be able
      to be initialised
    – TODO: need this check?   is 'initialized' status stored in DB or on gateways?   or both?
- void setPluggableDevicesRequirements(int applicationID, List<**PluggableDeviceInfo**> devices)
    – Effect: Sets an application's requirements for pluggable devices.
    – Created for: Av3:   "Application providers can design their applications such that they explicitly
      require redundancy in the available pluggable devices."

7.2.25   DeviceMgmt

Provided by: ⬚ GatewayFacade
Required by: ⬚ DeviceManager
Operations:
- void addDevice(**PluggableDeviceID** pID, **PluggableDeviceType** type, Map<string, string> configurations, int
  moteID)
    – Effect: Adds a new pluggable device in the DeviceDB and adds a reference to a mote.   The
      device's status is 'uninitialised' by default and it's current configurations (which are now the default
      configurations) are stored as well.   If the device already exists, removes the data first (in case the
      device is plugged into a different mote or on a different network).
      Adds a new pluggable device to the topology of the infrastructure owner and links it to a mote.
      The device gets the mote's location by default.   If the device is already linked to another mote,
      overwrites that link.
      Notifies the infrastructure owner that owns the gateway that the pluggable device was detected.
    – Created for: UC6.3, U2 - easy pluggable device installation
- int addMote(**MoteInfo** mote, int gatewayID, IPAddress moteIPAddress)
    – Effect: Adds a new mote in the DeviceDB along with an IP address and a reference to a gateway.
      Returns the DB id for the mote.
      Adds a new mote to the topology of the infrastructure owner.   The mote is linked to a gateway and
      gets status 'unplaced' by default.
      Notifies the infrastructure owner that owns the gateway that a new mote is available for con
      guration in the topology.

- – Created for: UC4.3, U2 - easy mote installation
- void checkApplicationsForActivationForInfrastructureOwner(int infrastructureOwnerID)
    - – Effect: Checks and activates applications which can now execute again.   The applications checked are those that are subscribed to by customers organisations associated to the given infrastructure owner.
    - – Created for: UC17, UC6.3 - reintroduced device
- void deactivateApplicationInstance(int applicationInstanceID)
    - – Effect: Deactivates a certain application.   This could happen when mandatory pluggable devices for the application are missing.
    - – Created for: Av3:   automatic suspension/reactivation of applications.
- Map<string, string> getConfigDB(**PluggableDeviceID** pID)
    - – Effect: Gets the last set configurations of a pluggable device from the `DeviceDB`.
    - – Created for: UC6.3 - reintroduced device
- List<**PluggableDeviceInfo**> getConnectedDevices()
    - – Effect: Returns a list of information about devices that are connected to the gateway.
    - – Created for: UC18
    - – Tradeoff: send PluggableDeviceID instead of DeviceInfo.   If you send DeviceInfo, then `ApplicationManager` does not have to fetch this info.   If you send PluggableDeviceID's, then less bandwidth is used and the Gateways do less work.
- void pluggableDevicePersisentFailure()
    - – Effect: Lets the gateway know that a timer for pluggable device or mote has expired.   This will generate a notification for an infrastructure owner.
    - – Created for: Av3:   The infrastructure owner should be notified of any persistent pluggable device or mote failures.
- void pluggableDevicePluggedIn(Map<string, string> mInfo, **PluggableDeviceID** pID, **PluggableDeviceType** type)
    - – Effect: Notifies the gateway that a new pluggable device of the given type is connected to the mote.
- void pluggableDeviceRemoved(**PluggableDeviceID** pID)
    - – Effect: Notifies the gateway that a pluggable device is removed.
- void reactivateApplicationInstance(int applicationInstanceID)
    - – Effect: Reactivate an application instance.   This could happen automatically after a broken sensor has been replaced.
    - – Created for: Av3:   automatic suspension/reactivation of applications.
- void reactivateDevice(**PluggableDeviceID** pID)
    - – Effect: Changes the status of a pluggable device to 'active'.
    Changes the status of a pluggable device in the topology to 'placed'.
    Notifies the infrastructure owner that owns the gateway that the pluggable device was detected and reactivated.
    - – Created for: UC6.3 - reintroduced device
- void reactivateMote(int moteID)
    - – Effect: Changes the status of the mote with DB id 'moteID' to 'active'.
    Changes the status of the mote in the topology to 'placed'.
    Notifies the infrastructure owner that owns the gateway that the mote was detected and reactivated.
    - – Created for: U2 - Reintroducing a previously known mote should not require any con guration.
- void setConfig(**PluggableDeviceID** pID, Map<String, String> config)
    - – Effect: Set the given configuration parameters of a PluggableDevice to the given values.   Setting unknown parameters on a PluggableDevice has no effect.

    - – Created for: UC11:   pluggable device needs to be initialised, M1:   pluggable device must be able to be initialised

7.2.26   DeviceMgmt

Provided by: ⬡ `ApplicationFacade`

Required by: [⊡] `ApplicationClient`
Operations:

7.2.27 ForwardData

Provided by: [⊡] `ApplicationManager`
Required by: [⊡] `DeviceDataScheduler`
Operations:
- List<int> getApplicationsForDevice(**PluggableDeviceID** pID)
  - Effect: Returns a list of application instances that can use the device with id "pID".

  - Created for: UC11:   the system looks up the list of applications that use the pluggable device
- void rcvData(**PluggableDeviceID** pID, **DeviceData** data)
  - Effect: Sends pluggable device data to an application that wants to use it

  - Created for: UC11:   system relays data to applications

7.2.28 Heartbeat

Provided by: [⊡] `GatewayFacade`
Required by: [⊡] `MoteFacade`
Operations:
- void heartbeat(Map<string, string> moteInfo, List<Tuple<**PluggableDeviceID**, **PluggableDeviceType**>> pds)
  - Effect: Sends a heartbeat from a mote to a gateway, including a list of the pluggable devices and their device types (i.e.   those currently plugged into the mote)
  - Created for: Given constraint, UC14, Av3:   failure detection

7.2.29 InvoiceMgmt

Provided by: [⊡] `OtherDataDB`
Required by: [⊡] `InvoiceManager`
Operations:
- void markActivatedApplication(int applicationInstanceID, int custOrgID, **DateTime** date)
  - Effect: Updates an ApplicationInstance's billing information: marks the start of a billing period.
  - Created for: UC17.4, U2 - easy applications

7.2.30 InvoiceMgmt

Provided by: [⊡] `InvoiceManager`
Required by: [⊡] `ApplicationManager`
Operations:
- void markActivatedApplication(int applicationInstanceID, int custOrgID, **DateTime** date)
  - Effect: Updates an ApplicationInstance's billing information: marks the start of a billing period.
  - Created for: UC17.4, U2 - easy applications

7.2.31 IOAppMgmt

Provided by: [⊡] `ApplicationManager`
Required by: [⊡] `InfrastructureOwnerFacade`
Operations:
- void checkApplicationsForActivationForCustomerOrganisations(List<int> custOrgIDs)
  - Effect: Checks and activates 'inactive' ApplicationInstances which can now execute again for a list of customer organisations.
  - Created for: UC9.7

- void checkApplicationsForDeactivationForCustomerOrganisations(List<int> custOrgsIDs)
  - Effect: Checks for ApplicationInstances that require deactivation for a list of customer organisations.
  - Created for: UC9.7

7.2.32   IODeviceMgmt

Provided by: [⊡] DeviceDB
Required by: [⊡] InfrastructureOwnerManager
Operations:
- List<**PluggableDeviceInfo**> getDevices(int infrastructureOwnerID)
  - Effect: Returns a list of PluggableDeviceInfo of devices owned by an infrastructure owner.
  - Created for: UC9.2

7.2.33   IOMgmt

Provided by: [⊡] OtherDataDB
Required by: [⊡] InfrastructureOwnerManager
Operations:
- List<int> getCustomerOrganisations(int infrastructureOwnerID)
  - Effect: Returns a list of IDs of all customer organisations associated with an infrastructure owner.
  - Created for: UC9.4

7.2.34   IOMgmt

Provided by: [⊡] InfrastructureOwnerManager
Required by: [⊡] InfrastructureOwnerFacade
Operations:
- List<int> getCustomerOrganisations(int infrastructureOwnerID)
  - Effect: Returns a list of IDs of all customer organisations associated with an infrastructure owner.
  - Created for: UC9.4
- List<**PluggableDeviceInfo**> getDevices(int infrastructureOwnerID)
  - Effect: Returns a list of PluggableDeviceInfo of devices owned by an infrastructure owner.
  - Created for: UC9.2

7.2.35   NotificationDeliveryMgmt

Provided by: None
Required by: [⊡] NotificationHandler
Operations:
- void notify(Map<string, string> data)
  - Effect: Delivers a notification to an end user using a specific delivery service.
  - Created for: UC15

7.2.36   NotificationMgmt

Provided by: [⊡] OtherDataDB
Required by: [⊡] NotificationHandler
Operations:
- int lookupNotificationChannelForUser()
  - Effect: Returns the id of the type of communication channel a user prefers.
  - Created for: UC15
- int storeNotification(**Notification** notification)
  - Effect: Stores a new notification entry in the database.   Returns the id of the new notification.

– Created for: UC15, Av3:   notifications
- int updateNotification(**Notification** notification)
    – Effect: Updates an existing notification (e.g.   change status to "sent").
    – Created for: UC15

7.2.37   Notify

Provided by: ▯ NotificationHandler
Required by: ▯ ApplicationManager, ▯ GatewayFacade
Operations:
- void notify(int userID, string message)
    – Effect: Stores a new notification in the system and causes it to be sent to a user.
    – Created for: UC14, Av3:   notifications

7.2.38   Other

Provided by: ▯ PluggableDeviceDataDB
Required by: ▯ OtherFunctionality1, ▯ OtherFunctionality2
Operations:

7.2.39   Other

Provided by: ▯ OtherDataDB
Required by: ▯ OtherFunctionality1, ▯ OtherFunctionality2
Operations:

7.2.40   Other

Provided by: ▯ OtherFunctionality1, ▯ OtherFunctionality2
Required by: ▯ GatewayFacade
Operations:

7.2.41   RequestData

Provided by: ▯ PluggableDeviceFacade
Required by: ▯ MoteFacade
Operations:
- **DeviceData** getData()
    – Effect: Synchronously retrieve the device data of a device.

- void getDataAsync(int requestID)
    – Effect: Asynchronously retrieve the device data of a device (by calling rcvDataCallback).

7.2.42   RequestData

Provided by: ▯ DeviceDataScheduler
Required by: ▯ ApplicationManager
Operations:
- List<**DeviceData**> getData(int applicationID, **PluggableDeviceID** pID, **DateTime** from, **DateTime** to)
    – Effect: Requests data from a specific device in a certain time period.
    – Created for: P2:   requests from applications

7.2.43   RoleMgmt

Provided by: ⏺ `UserRolesManager`
Required by: ⏺ `ApplicationManager`, ⏺ `CustomerOrganisationFacade`
Operations:

- boolean areMandatoryUserRolesAssigned(int applicationInstanceID)
  - Effect: Returns true if all mandatory UserRoles for the application have been assigned to users. Finds the relevant customer organisations through the ApplicationInstance.
  - Created for: UC17.1, U2 - easy applications
- List< User > getEndUsers(int custOrgID)
  - Effect: Returns a list of Users which are associated to a customer organisation.
  - Created for: UC19.8, U2 - easy applications
- List< UserRole > getMandatoryUserRoles(int applicationID)
  - Effect: Returns a list of UserRoles that which need to be assigned in order for an ApplicationInstance to run.
  - Created for: UC19.7, U2 - easy applications
- List< UserRole > getOptionalUserRoles(int applicationID)
  - Effect: Returns a list of UserRoles which can optionally be assigned for an ApplicationInstance.
  - Created for: UC19.7, U2 - easy applications
- List< User > getUsersWithRoles(int applicationInstanceID)
  - Effect: Returns a list of Users associated to an ApplicationInstance that were assigned UserRoles.
  - Created for: UC17.6, U2 - easy applications
- void updateUserRoles(int applicationInstanceID, Map<int, int> usersAndRoles)
  - Effect: Updates the UserRoles assigned to Users for a certain ApplicationInstance.   'usersAndRoles' maps User IDs to UserRole IDs.
  - Created for: UC19.9, U2 - easy applications

7.2.44   SubscriptionMgmt

Provided by: ⏺ `OtherDataDB`
Required by: ⏺ `SubscriptionManager`
Operations:

- void createSubscription(int custOrgID, int applicationInstanceID)
  - Effect: Creates a subscription for a customer organisation to an ApplicationInstance.   If the customer organisation is already subscribed to an older version of the the application, then the organisation is unsubscribed from that earlier version.
  - Created for: UC19.12-13, U2 - easy applications
- List<**Subscription**> getSubscriptions(int custOrgID)
  - Effect: Returns a list of subscriptions a customer organisation has.
  - Created for: UC19.2, U2 - easy applications

7.2.45   SubscriptionMgmt

Provided by: ⏺ `CustomerOrganisationFacade`
Required by: ⏺ `CustomerOgranisationClient`
Operations:

- Map< Application , **Subscription**> getApplicationsToSubscribe(int custOrgID)
  - Effect: Returns a map of Applications and Subscriptions a given customer organisation has to those applications
  - Created for: UC19.1, U2 - easy applications
- int subscribeToApplication(int custOrgID, int applicationID)
  - Effect: Creates a new ApplicationInstance for an application for a customer organisation and returns its id.
  - Created for: UC19.4, U2 - easy applications

- void updateApplicationDevicesSettings(int applicationInstanceID, List<**PluggableDeviceID**> devices, List< Relationships > relationships)
  - Effect: Updates an ApplicationInstance's device settings. This includes which devices the instance can use and which relationships exist between those devices.
  - Created for: UC19.6, U2 - easy applications
- void updateCriticality(int applicationInstanceID, boolean isCritical)
  - Effect: Updates the criticality of an ApplicationInstance.
  - Created for: UC19.11, U2 - easy applications
- void updateUserRoles(int applicationInstanceID, Map<int, int> usersAndRoles)
  - Effect: Updates the UserRoles assigned to Users for a certain ApplicationInstance. 'usersAndRoles' maps User IDs to UserRole IDs.
  - Created for: UC19.9, U2 - easy applications

7.2.46 SubscriptionMgmt

Provided by: ⧉ SubscriptionManager
Required by: ⧉ CustomerOrganisationFacade
Operations:
- void createSubscription(int custOrgID, int applicationInstanceID)
  - Effect: Creates a subscription for a customer organisation to an ApplicationInstance. If the customer organisation is already subscribed to an older version of the the application, then the organisation is unsubscribed from that earlier version.
  - Created for: UC19.12-13, U2 - easy applications
- Map< Application , **Subscription**> getApplicationsToSubscribe(int custOrgID)
  - Effect: Returns a map of Applications and Subscriptions a given customer organisation has to those applications
  - Created for: UC19.2, U2 - easy applications

7.2.47 TopologyMgmt

Provided by: ⧉ DeviceDB
Required by: ⧉ TopologyManager
Operations:
- void addDevice(**PluggableDeviceID** pID, int moteID)
  - Effect: Adds a new pluggable device to the topology of the infrastructure owner and links it to a mote. The device gets the mote's location by default. If the device is already linked to another mote, overwrites that link.
  - Created for: UC6.3, U2 - easy pluggable device installation
- void addMote(int moteID, int infrastructureOwnerID, int gatewayID)
  - Effect: Adds a new mote to a topology of an infrastructure owner. The mote is linked to a gateway and gets status 'unplaced' by default.
  - Created for: UC4.3, U2 - easy mote installation
- boolean arePluggableDevicesPlaced(List<**PluggableDeviceID**> devices)
  - Effect: Returns true if all pluggable devices in the given list have status 'placed' in the topology.
  - Created for: UC17.2, U2 - easy applications
- List< RoomTopology > getTopology(int custOrgID)
  - Effect: Returns a list of RoomTopology associated to a customer organisation.
  - Created for: UC19.5, U2 - easy applications
- void reactivateDevice(**PluggableDeviceID** id)
  - Effect: Changes the status of a pluggable device in the topology to 'placed'.
  - Created for: UC6.3 - reintroduced device
- void reactivateMote(int moteID)

- Effect: Changes the status of the mote in the topology to 'placed'. The location of the mote is unchanged, it has already been set.
- Created for: U2 - Reintroducing a previously known mote should not require any con guration.

7.2.48 TopologyMgmt

Provided by: ⊡ `TopologyManager`
Required by: ⊡ `ApplicationManager`, ⊡ `CustomerOrganisationFacade`, ⊡ `GatewayFacade`
Operations:
- void addDevice(**PluggableDeviceID** id, int moteID)
  - Effect: Adds a new pluggable device to the topology of the infrastructure owner and links it to a mote. The device gets the mote's location by default. If the device is already linked to another mote, overwrites that link.
  - Created for: UC6.3, U2 - easy pluggable device installation
- void addMote(int moteID, int gatewayID, int infrastructureOwnerID)
  - Effect: Adds a new mote to a topology of an infrastructure owner. The mote is linked to a gateway and gets status 'unplaced' by default.
  - Created for: UC4.3, U2 - easy mote installation
- boolean arePluggableDevicesPlaced(List<**PluggableDeviceID**> devices)
  - Effect: Returns true if all pluggable devices in the given list have status 'placed' in the topology.
  - Created for: UC17.2, U2 - easy applications
- List< RoomTopology > getTopology(int custOrgID)
  - Effect: Returns a list of RoomTopology associated to a customer organisation.
  - Created for: UC19.5, U2 - easy applications
- void reactivateDevice(**PluggableDeviceID** id)
 - Effect: Changes the status of a pluggable device in the topology to 'placed'.
  - Created for: UC6.3 - reintroduced device
- void reactivateMote(int moteID)
  - Effect: Changes the status of the mote in the topology to 'placed'. The location of the mote is unchanged, it has already been set.
  - Created for: U2 - Reintroducing a previously known mote should not require any con guration.

7.2.49 TopologyOverview

Provided by: ⊡ `ApplicationFacade`
Required by: ⊡ `ApplicationClient`
Operations:

7.2.50 UserRoleMgmt

Provided by: ⊡ `OtherDataDB`
Required by: ⊡ `UserRolesManager`
Operations:
- boolean areMandatoryUserRolesAssigned(int applicationInstanceID)
  - Effect: Returns true if all mandatory UserRoles for the application have been assigned to users. Finds the relevant customer organisations through the ApplicationInstance.
  - Created for: UC17.1, U2 - easy applications
- List< User > getEndUsers(int custOrgID)
  - Effect: Returns a list of Users which are associated to a customer organisation.
  - Created for: UC19.8, U2 - easy applications
- List< UserRole > getMandatoryUserRoles(int applicationID)

– Effect: Returns a list of UserRoles that which need to be assigned in order for an ApplicationInstance to run.
– Created for: UC19.7, U2 - easy applications
- List< UserRole > getOptionalUserRoles(int applicationID)
  – Effect: Returns a list of UserRoles which can optionally be assigned for an ApplicationInstance.
  – Created for: UC19.7, U2 - easy applications
- List< User > getUsersWithRoles(int applicationInstanceID)
  – Effect: Returns a list of Users associated to an ApplicationInstance that were assigned UserRoles.
  – Created for: UC17.6, U2 - easy applications
- void updateUserRoles(int applicationInstanceID, Map<int, int> usersAndRoles)
  – Effect: Updates the UserRoles assigned to Users for a certain ApplicationInstance. 'usersAndRoles' maps User IDs to UserRole IDs.
  – Created for: UC19.9, U2 - easy applications

7.3 Exceptions

No exceptions

7.4 Data types

- **ApplicationInstance**:
  Attributes: int id, int status, int customerOrganisationID
  Contains information on an application instance.
- **DateTime**:
  Represents an instant in time, expressed as a date and time of day.
- **DeviceData**:
  Data from a pluggable device. For sensors, this contains sensor values. For actuators, this contains the state of the actuator. The data is encapsulated within a JSON message, and should be converted into something meaningful based on the device type of the pluggable device that sent the data.
- **MoteInfo**:
  Attributes: int moteID, int manufacturerID, int productID, int batteryLevel
  An object containing information on a mote. This is a list of key-value pairs. The values depend on the type of mote. For example, only a battery-powered mote would include the batterylevel info.
- **Notification**:
  Attributes: int id, int recipientUserID, string message, int communicationChannelID, int notificationTypeID
  Contains information about a notification. The communicationChannelID represents the communication channel that will be used to send the notification to the user. The notificationTypeID denotes the type of the notification (normal / alarm / ...).
- **PluggableDeviceID**:
  A unique identifier of a pluggable device.
- **PluggableDeviceInfo**:
  Attributes: **PluggableDeviceID** id, **PluggableDeviceType** type, Map<string, string> config
  Contains information on a pluggable device.
- **PluggableDeviceType**:
  Denotes the type of a pluggable device.
- **Subscription**:
  Attributes: int id, int status, int customerOrganisationID, int applicationInstanceID
  Contains data about a subscription by a customer organisation for an application instance. Data about period/length of the subscription is stored in invoices.

A. Attribute-driven design documentation

A.1 Introduction

This chapter contains our ADD log. First, we list the changes we made the ADD process so it fits our workflow better. The remaining part of this chapter is the ADD log. Decompositions 1 and 2 have been changed relative to phase 2a of this project, because we forgot about the given interfaces for gateways and pluggable devices and made up our own (but similar) interfaces instead. The decompositions have been updated to use the given interfaces.

A.2 Adapted ADD process

We left off step a ("Pick an element that needs to be decomposed") since we never really chose a single Element to decompose. Instead, we chose the drivers for each decomposition first and then looked at which elements/subsystems would require changes or which new elements we would need to satisfy those drivers. For component, interfaces, datatypes: we list the new ones, but refer to the plugin exported catalog for descriptions

A.2.1 Decomposition X: DRIVERS (Elements/Subsystem to decompose/expand)

We changed these titles to reflect the architectural drivers we chose first and then denote which elements/subsystems needed changes to satisfy the drivers.

A.2.2 Data type definitions and Interfaces for child modules

For each decomposition, we have listed all new interfaces and data types that we added during the decomposition, but all details have been left out. We used the Visual Paradigm plugin provided by the SA team to generate the element catalog of chapter 6. All details can be found in there.

Also, since an ADD log was no longer a requirement for phase 2b, we have left out intermediary "OtherFunctionality" components and figures of diagrams. This was done to save time.

A.2.3 Verify and refine

We have skipped the verify and refine step because we chose to handle all chosen architectural drivers completely in every decomposition. We did not find this step to be useful after decompositions 1 and 2.

A.3 Decomposition 1: Av3, UC14, UC15, UC18 (SIoTIP System)

A.3.1 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.

A.3.2 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 `DeviceManager` 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 `DeviceManager`. It stores the requirements for pluggable devices set by applications for all applications that use the gateway that the the `DeviceManager` 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 `DeviceManager` 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 `DeviceManager` 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. The disadvantage of this is that an internal solution takes a lot more time to implement.

A.3.3 Instantiation and allocation of functionality

This section lists the new components which instantiate our solutions described in the section above. For each component we note the quality attribute or use case that prompted us to create it. Descriptions about the components can be found under chapter 6.

- ApplicationManager: Av3
- Database: /
- DeviceManager: Av3
- GatewayFacade: /
- MoteFacade: UC14
- NotificationHandler: UC15

Decomposition Figure A.1 shows the components resulting from the decomposition in this run.
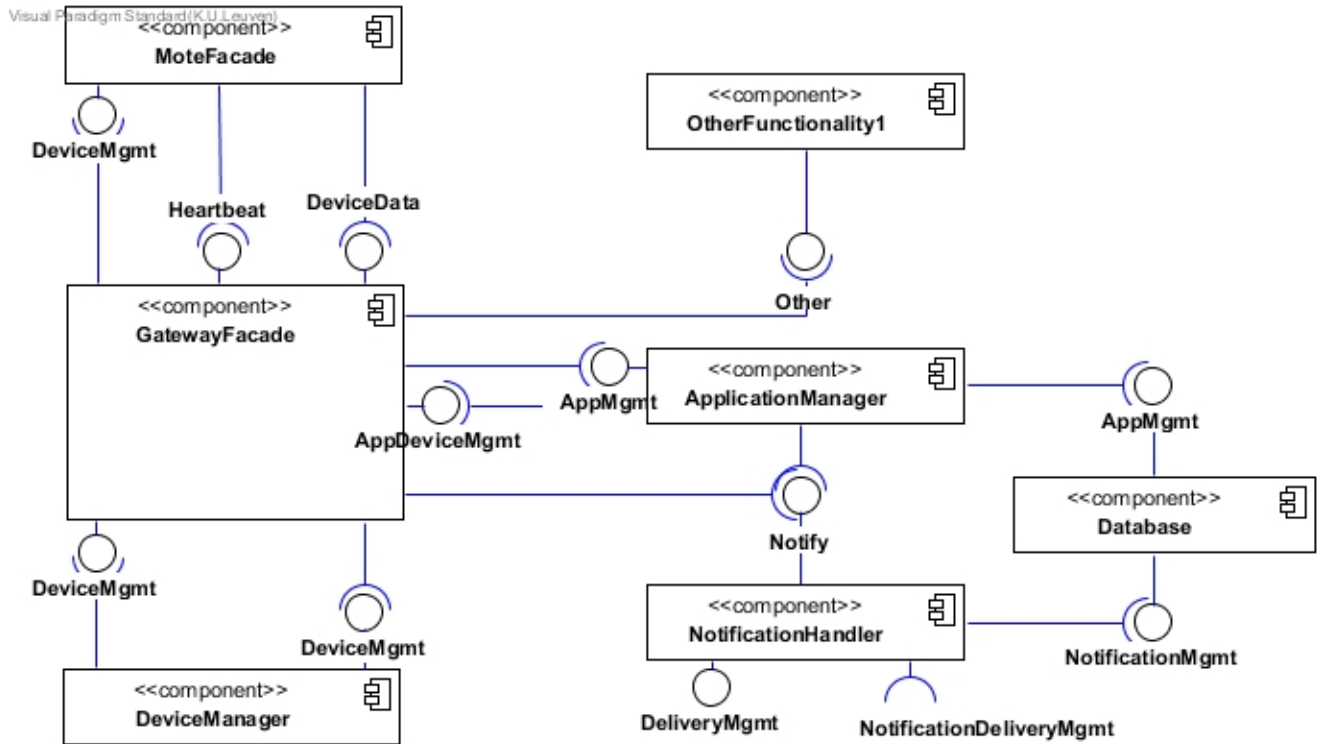
Figure A.1:   Component-and-connector diagram of this decomposition.

Deployment Figure A.2 shows the allocation of components to physical nodes.
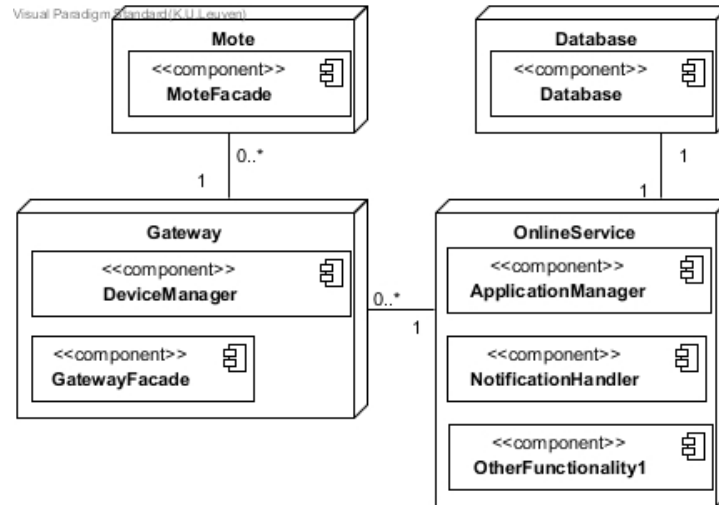
Figure A.2:   Deployment diagram of this decomposition.

A.3.4 Interfaces for child modules

This section lists new interfaces assigned to the components defined in the section above. Detailed information about each interface and its methods can be found under chapter 6.

ApplicationManager

- AppMgmt

Database

- NotificationMgmt
- AppMgmt

GatewayFacade

- Heartbeat
- DeviceData
- DeviceMgmt
- AppDeviceMgmt

MoteFacade

- DeviceMgmt

NotificationHandler

- Notify
- DeliveryMgmt

External notification delivery service

- NotificationDeliveryMgmt

DeviceManager

- DeviceMgmt

A.3.5 Data type definitions

This section lists the data types introduced in this decomposition.

- PluggableDeviceInfo
- Notification
- ApplicationInstance
- Subscription
- PluggableDeviceID
- PluggableDeviceType
- DeviceData
- Map<String,String>

A.3.6 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

DeviceManager

- *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

A.4 Decomposition 2: M1, P2, UC11 (OtherFunctionality1)

A.4.1 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.

A.4.2 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: Data conversion   With new types of devices, the pluggable data processing subsystem should be extended with relevant data conversions, e.g.   converting temperature in degrees Fahrenheit to degrees Celsius.
The `DeviceDataConverter` is put in place to handle the task of converting pluggable device data to data of a different type in the system.   This component can easily be modified for new types of data simply by adding a new conversion method for the new.

M1: Usage of new data by applications   The available applications in the system can be updated to use any new pluggable devices.
This is made possible by the RequestData interface provided by `DeviceDataScheduler`.   Data of the new type of device can be requested in the same way as for older devices: by using the device's unique id.   The application manager can get pluggable device data from the `PluggableDeviceDataDB` and return this data to applications in the DeviceData datatype.   This datatype can easily be updated for new types of pluggable devices.

P2: Scheduling   The pluggable data processing subsystem needs to be able to run in normal or overload mode, depending on whether or not the system can process requests within the deadlines given in the quality requirement.   Also, a mechanism should be in place to avoid starvation of any type of request.
The `DeviceDataScheduler` is used to deal with this problem.   It is responsible for scheduling requests that wish to interact with the `PluggableDeviceDataDB`. In normal mode, the system processes incoming requests in a FIFO order.   In overload mode, the requests are given a priority based on what the request is for and what the source of the request is.   The requests are then not simply processed in an order based on their priorities, but an aging technique is to be used such that starvation will be avoided.   Thus, in overload mode, requests are processed in an order based on a combination of the priorities of the requests and the age of the requests.

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.

In order to statisfy this constraint, all data directly related to pluggable data has been separated into the `PluggableDeviceDataDB`. All requests concerning pluggable data will be handled by this new component. `PluggableDeviceDataDB` will run on a node different from the node that the `Datbase` component runs on. This way requests concerning pluggable will have no impact on requests concerning other data.

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, configure access rights for these devices, and view detailed information about the new type of pluggable device.

The components created thus far have been created with high cohesion in mind so that updating them for new devices would be relatively straightforward. In order for this constraint to be satisfied, changes have to be made to the following elements:

- *PluggableDeviceFacade*: This component needs to be updated so that the new type of device can be initialised and configured, and thus so that the device's data can be sent to the system.
- *DeviceData*: Depending on how this data type is implemented, it might need an update in order for it to represent possible new data types (for example Temperature Filipcikova) and for the new data types to be serialized.
- *PluggableDeviceDataDB*: The database needs to be updated so that information can be retrieved about the new types of sensors and the new types of data. Data related to the displaying of sensor data will also need to be updated.
- *PluggableDeviceConverter*: see above.

A.4.3   Instantiation and allocation of functionality

This section lists the new components which instantiate our solutions described in the section above. For each component we note the quality attribute or use case that prompted us to create it. Descriptions about the components can be found under chapter 6.

- DeviceDataConverter: M1
- DeviceDataScheduler: P2
- PluggableDeviceDataDB: P2
- PluggableDeviceFacade: UC11

Decomposition Figure A.3 shows the components resulting from the decomposition in this run.

Deployment Figure A.4 shows the allocation of components to physical nodes.

A.4.4   Interfaces for child modules

This section lists new interfaces assigned to the components defined in the section above. Detailed information about each interface and its methods can be found under chapter 6.

ApplicationManager

- ForwardData

MoteFacade

- DeviceData
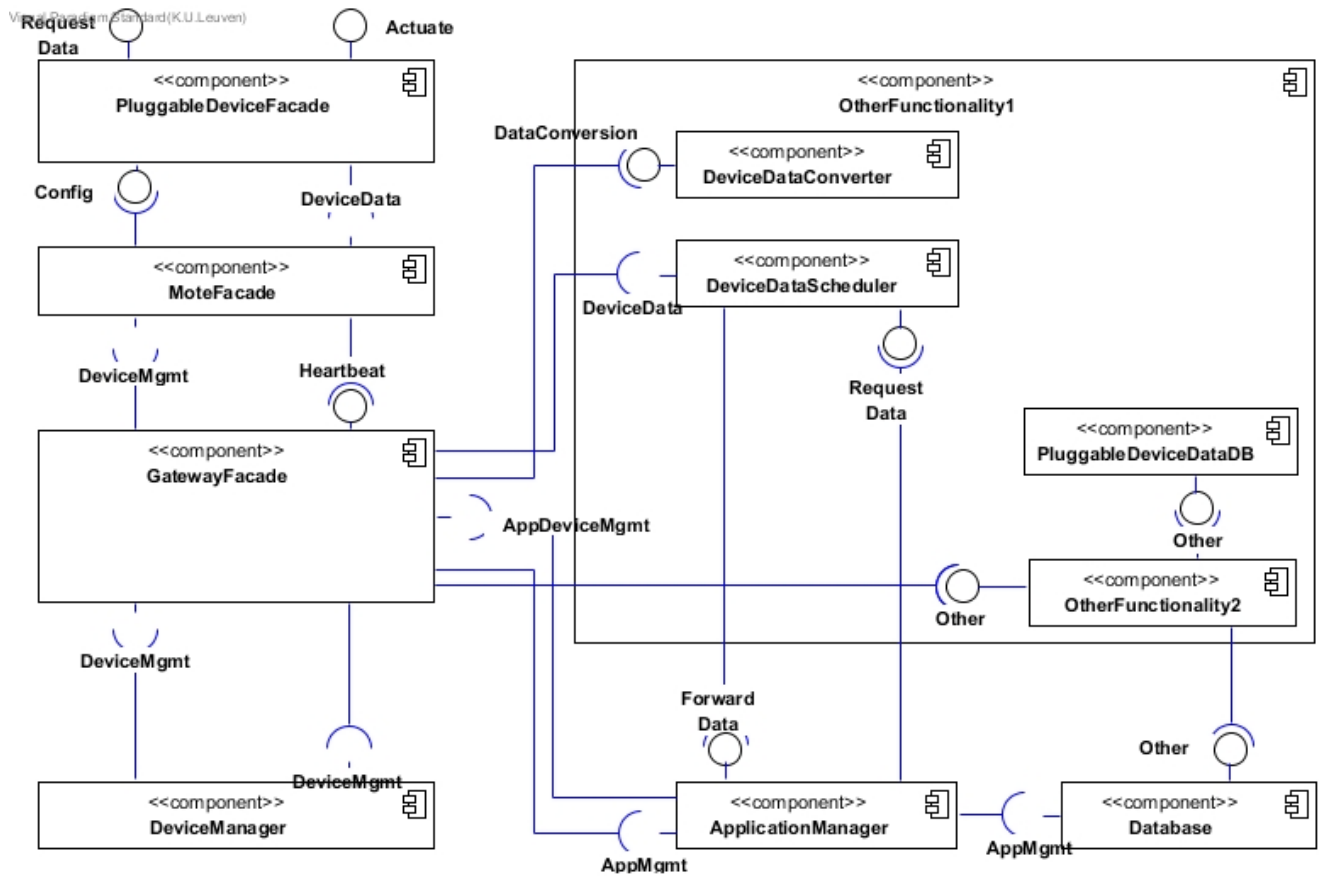
Figure A.3:   Component-and-connector diagram of this decomposition.

PluggableDeviceFacade

- Actuate
- Config
- RequestData

DeviceDataConverter

- DataConversion

DeviceDataScheduler

- RequestData
- DeviceData

PluggableDeviceDataDB

- DeviceData

A.4.5   Data type definitions

This section lists the new data types introduced during this decomposition.

- DateTime
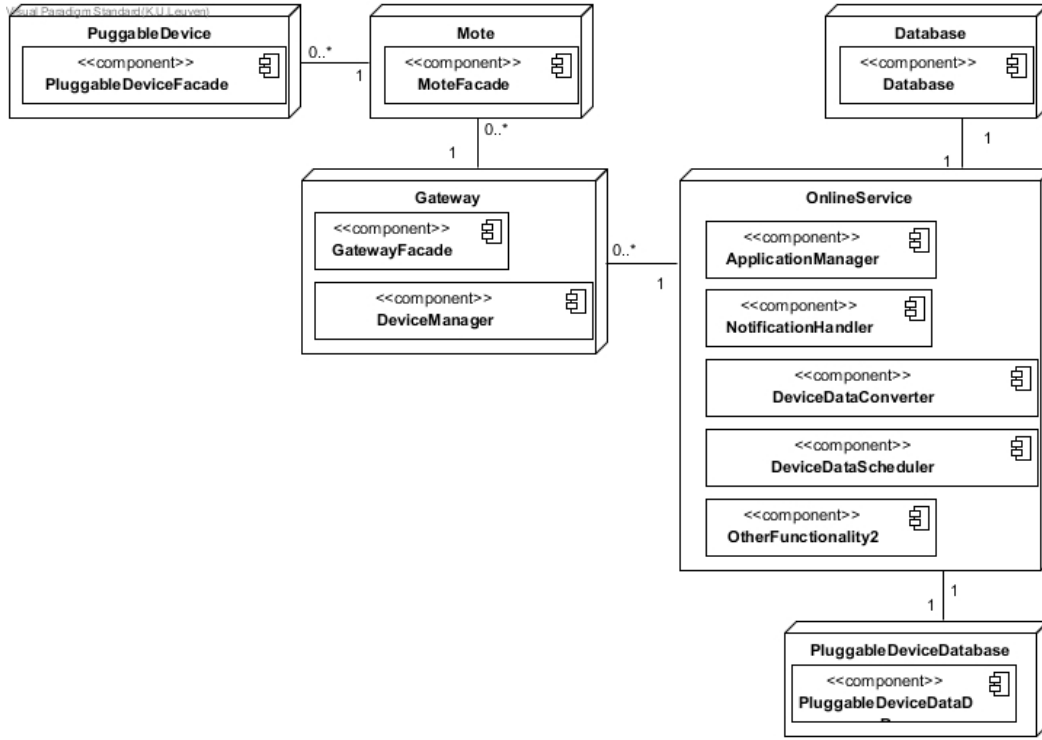
Figure A.4:   Deployment diagram of this decomposition.

A.4.6   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

OtherDataDB

- None

GatewayFacade

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

55

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

PluggableDeviceDataDB

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

PluggableDeviceFacade

- *U2*: Easy Installation: d

DeviceManager

- *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,

DeviceDataScheduler

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

A.5 Decomposition 5: Av1 (Elements/Subsystem to decompose/expand)

A.5.1 Selected architectural drivers

The non-functional drivers for this decomposition are:

- *Av1*: Communication between SIoTIP gateway and Online Service

A.5.2 Architectural design

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

Av1: Problem title Short description of the problem.
Solution for the problem.
Communication pattern: broker ?
Prevention: *) SLA 99.9% availability, can't model this *) Local networks 90% availability
Detection: *) Online Service can detect failures of its individual internal communication components: Ping/Echo, Monitor, Heartbeat, Timestamp The failure of an internal SIoTIP Online Service component is detected within 30 seconds.
*) SIoTIP gateway can detect failures of its individual internal communication components: Ping/Echo, Monitor, Heartbeat, Timestamp
*) Online service can detect that a SIoTIP gateway is not sending data anymore based on the expected synchronisation interval: Monitor: GWMonitor An outage is defined as 3 consecutive expected synchronisations that do not arrive within 1 minute of their expected arrival time.
*) Online Service should acknowledge each message sent by the SIoTIP gateway so that the gateway can detect failures.
CommunicationModule on gateway and Online Service. Sends all messages to Online Service and waits for acknowlegdements for all sent messages. If no acknowledgement within X seconds -> Online Service failure
Resolution: *) gateway outage (= complete failure) detected -> notify IO and SIoTIP system administrator The infrastructure owner is notified within 5 minutes after the detection of an outage of their gateway. A SIoTIP system administrator should be notified within 1 minute after the detection of a simultaneous outage of more than 1% of the registered gateways.
*) gateway component failure detected -> first, restart component. If still fails, reboot gateway entirely.
*) Online Service / communication channel failure detected -> -> temporarily store all incoming pluggable data and issued application commands internally The SIoTIP gateway can store at least 3 days of pluggable data before old data has to be overwritten.
*) When Online Service = unreachable, applications on SIoTIP gateway can still operate normally, TODO ask
Response Measure: *) The SIoTIP gateway will start synchronising with the Online service within 1 minute after the communication channel becomes available.

A.5.3 Instantiation and allocation of functionality

This section lists the new components which instantiate our solutions described in the section above. For each component we note the quality attribute or use case that prompted us to create it. Descriptions about the components can be found under chapter 6.

- Component: (Av1 or UC)

A.5.4 Interfaces for child modules

This section lists new interfaces assigned to the components defined in the section above. Detailed information about each interface and its methods can be found under chapter 6.

Component

- Interface

A.5.5   Data type definitions

This section lists the new data types introduced during this decomposition.

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

A.6 Decomposition 6: P1, UC1, UC2, UC3, UC5, UC7, UC8, UC16, UC20 (Elements/Subsystem to decompose/expand)

A.6.1 Selected architectural drivers

The non-functional drivers for this decomposition are:

- *P1*:   Large number of users

The related functional drivers are:

- *UC1*:   Register a customer organisation
  Short description of the UC.
- *UC2*:   Register an end-user
  Short description of the UC.
- *UC3*:   Unregister an end-user
  Short description of the UC.
- *UC5*:   Uninstall mote
  Short description of the UC.
- *UC7*:   Remove a pluggable device from its mote
  Short description of the UC.
- *UC8*:   Initialise a pluggable device
  Short description of the UC.
- *UC16*:   Consult notification message
  Short description of the UC.
- *UC20*:   Unsubscribe from application
  Short description of the UC.

A.6.2 Architectural design

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

P1: Problem title   The SIoTIP Online Service replies to the service requests of the infrastructure owner and customer organisations.

P1: Problem title   The Online Service processes the data received from the gateways.

P1: Problem title   The application execution subsystem should be able to execute an increasing number of active applications.

P1: Problem title   The initial deployment of SIoTIP should be able to deal with at least 5000 gateways in total, and should be provisioned to service at least 3000 registered users simultaneously connected to SIoTIP. -> 5000 gateways * 4 motes per gateway * 3 devices per mote = 60000 devices -> Keep communication with gateways at a minimum e.g.   gateway messages are of type "send data", "new device connected", ...   -> LOAD BALANCING

P1: Problem title Scaling up to service an increasing amount of infrastructure owners, customers organisations and applications should (in worst case) be linear ; i.e.   it should not require proportionally more resources (machines, etc.)   than the initial amount of resources provisioned per customer organisation/infrastructure owner and per gateway.

A.6.3 Instantiation and allocation of functionality

This section lists the new components which instantiate our solutions described in the section above. For each component we note the quality attribute or use case that prompted us to create it. Descriptions about the components can be found under chapter 6.

- Component: (P1 or UC)

A.6.4 Interfaces for child modules

This section lists new interfaces assigned to the components defined in the section above. Detailed information about each interface and its methods can be found under chapter 6.

Component

- Interface

A.6.5 Data type definitions

This section lists the new data types introduced during this decomposition.

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

A.7 Decomposition 7: UC28, UC29 (Elements/Subsystem to decompose/expand)

At this point, all quality attributes have been handled. The remaining decompositions handle all of the use cases that are left. The order is based on the priority of the use cases.

A.7.1 Selected architectural drivers

The functional drivers are:

- *UC28*: Log in
  Short description of the UC.
- *UC29*: Log out
  Short description of the UC.

A.7.2 Architectural design

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

UC: Problem title Short description of the problem.
Solution for the problem.

A.7.3 Instantiation and allocation of functionality

This section lists the new components which instantiate our solutions described in the section above. For each component we note the quality attribute or use case that prompted us to create it. Descriptions about the components can be found under chapter 6.

- Component: (Relevant UC)

A.7.4 Interfaces for child modules

This section lists new interfaces assigned to the components defined in the section above. Detailed information about each interface and its methods can be found under chapter 6.

Component

- Interface

A.7.5 Data type definitions

This section lists the new data types introduced during this decomposition.

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

A.8 Decomposition 8: UC22, UC23 (Elements/Subsystem to decompose/expand)

A.8.1 Selected architectural drivers

The functional drivers are:

- *UC22*:   Upload an application
  Short description of the UC.
- *UC23*:   Consult application statistics
  Short description of the UC.

A.8.2 Architectural design

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

UC: Problem title   Short description of the problem.
Solution for the problem.

A.8.3 Instantiation and allocation of functionality

This section lists the new components which instantiate our solutions described in the section above.   For each component we note the quality attribute or use case that prompted us to create it.   Descriptions about the components can be found under chapter 6.

- Component: (Relevant UC)

A.8.4 Interfaces for child modules

This section lists new interfaces assigned to the components defined in the section above.   Detailed information about each interface and its methods can be found under chapter 6.

Component

- Interface

A.8.5 Data type definitions

This section lists the new data types introduced during this decomposition.

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

A.9 Decomposition 9: UC21 (Elements/Subsystem to decompose/expand)

A.9.1 Selected architectural drivers

The functional drivers are:

- *UC21*:   Send invoice
  Short description of the UC.

A.9.2 Architectural design

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

UC: Problem title   Short description of the problem.
Solution for the problem.

A.9.3 Instantiation and allocation of functionality

This section lists the new components which instantiate our solutions described in the section above.   For each component we note the quality attribute or use case that prompted us to create it.   Descriptions about the components can be found under chapter 6.

- Component: (Relevant UC)

A.9.4 Interfaces for child modules

This section lists new interfaces assigned to the components defined in the section above.   Detailed information about each interface and its methods can be found under chapter 6.

Component

- Interface

A.9.5 Data type definitions

This section lists the new data types introduced during this decomposition.

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