

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 (r<br/>0683254) Armin Halilovic(r<br/>0679689)

Academic year 2016–2017

# Contents

1	1.1 Av3: Pluggable device or mote failure       2         1.2 Other decisions       2         1.2.1 Decision 1       3	2 3 4 4
2	2.1 Context diagram	5 5
3	Decomposition view (UML Component diagram)	7
4	Deployment view (UML Deployment diagram)124.1 Context diagram154.2 Primary diagram15	2
5	Scenarios         18           5.1 Scenarios         18	
6	Element Catalog and Datatypes 28	5
7	Catalog         7.1       Components       26         7.1.1       AccessRightsManager       26         7.1.2       ApplicationContainer       26         7.1.3       ApplicationContainerManager       26         7.1.4       ApplicationContainerMonitor       26         7.1.5       ApplicationExecutionSubsystemMonitor       26         7.1.6       ApplicationFrontEndClient       27         7.1.7       ApplicationFrontEndFacade       26         7.1.8       ApplicationInstance       26         7.1.9       ApplicationManagementLogic       27         7.1.10       ApplicationManager       26         7.1.11       ApplicationProviderClient       26	666667777
	7.1.12 ApplicationProviderFacade       28         7.1.13 AuthenticationManager       22         7.1.14 ContainerLogic       22         7.1.15 CustomerOgranisationClient       23         7.1.16 CustomerOrganisationFacade       24         7.1.17 DeviceCommandConstructor       29         7.1.18 DeviceDataConverter       29         7.1.19 DeviceDataScheduler       29         7.1.20 DeviceDB       29         7.1.21 DeviceManager       29         7.1.22 Gateway       30         7.1.23 GatewayBroker       30         7.1.24 GatewayBrokerMonitor       30         7.1.25 GatewayMonitor       30         7.1.26 GWBrokerLogic       30         7.1.27 InfrastructreOwnerClient       31	8889999000000

	7.1.28	InfrastructureOwnerFacade
	7.1.29	InfrastructureOwnerManager
	7.1.30	InvoiceManager
	7.1.31	Mote
	7.1.32	NotificationHandler
	7.1.33	OnlineServiceBroker
	7.1.34	OnlineServiceBrokerMonitor
	7.1.35	OnlineServiceMonitor
	7.1.36	OSBrokerLogic
		OtherDataDB
	7.1.38	PluggableDevice
		PluggableDeviceDataDB
		RegisteredUserFacade
		RequestStore
		SessionDB
		SubscriptionManager
		SystemAdministratorFacade
		ThirdPartyInvoicingService
		Third arty invoicing Service
		TopologyManager
		1 00 0
		UnregisteredUserClient
		UnregisteredUserFacade
		UserManager
7.2		ces
	7.2.1	AccessRights
	7.2.2	AccessRightsMgmt
	7.2.3	Actuate
	7.2.4	Actuate
	7.2.5	AppDeviceData
	7.2.6	AppDeviceMgmt
	7.2.7	AppInstanceMgmt
	7.2.8	AppMessages
	7.2.9	AppMonitoring
	7.2.10	AppStatus
	7.2.11	Authentication
	7.2.12	Authentication
	7.2.13	Authentication
		Commands
		Config
		Config
		DataConversion
		DBAccessRightsMgmt
		DBAppDeviceMgmt
		DBAppMgmt
		DBDeviceData
		DBDeviceMgmt
		DBInvoiceMgmt
		$\Theta$
		DBIOMgmt
		DBNotificationMgmt
		DBSubscriptionMgmt
		DBTopologyMgmt
		DBUserMgmt
	7.2.30	DeliveryMgmt

		7.2.31 DeliveryMgmt	46
		7.2.32 DeviceCommands	46
		7.2.33 DeviceCommands	47
		7.2.34 DeviceData	47
		7.2.35 DeviceData	47
		7.2.36 DeviceMgmt	48
		7.2.37 ForwardData	48
			48
		7.2.38 FrontEndAppRequests	
		7.2.39 GatewayUpdates	49
		7.2.40 GWAppInstanceMgmt	49
		7.2.41 GWBrokerMonitoring	50
		7.2.42 GWCommunication	50
		7.2.43 Heartbeat	50
		7.2.44 InvoiceDeliveryMgmt	50
		7.2.45 InvoiceMgmt	50
		7.2.46 IOMgmt	51
		7.2.47 Monitoring	51
		7.2.48 NotificationDeliveryMgmt	51
		7.2.49 Notifications	51
		7.2.50 Notifications	51
		7.2.51 Notifiy	52
		7.2.52 OSBrokerMonitoring	52
		7.2.53 OSCommunication	52
		7.2.54 OSMonitoring	52
		7.2.55 OSUpdates	53
		7.2.56 Registration	53
		7.2.57 RequestData	53
		7.2.58 RequestData	54
		7.2.59 RequestData	54
		7.2.60 Requests	54
		7.2.61 RoleMgmt	54
		7.2.62 SubscriptionMgmt	55
		7.2.63 SubscriptionMgmt	55
		7.2.64 TopologyMgmt	56
			56
		7.2.65 TopologyMgmt	
		7.2.66 UserMgmt	57
		7.2.67 UserMgmt	58
	7.3	Exceptions	59
	7.4	Data types	59
	<b>A</b>		01
A		ribute-driven design documentation	61
		Introduction	61
	A.2	Adapted ADD process	61
		A.2.1 Decomposition X: DRIVERS (Elements/Subsystem to decompose/expand)	61
		A.2.2 New data types and Interfaces for child modules	61
		A.2.3 Verify and refine	61
	A.3	Decomposition 1: Av3, UC14, UC15, UC18 (SIoTIP System)	62
		A.3.1 Selected architectural drivers	62
		A.3.2 Architectural design	62
		A.3.3 Instantiation and allocation of functionality	63
		A.3.4 Interfaces for child modules	65
		A.3.5 New data types	65
		A.3.6 Verify and refine	65
	Δ 1	Decomposition 2: M1, P2, UC11 (OtherFunctionality1)	68
	11.4	Decomposition 2. WII, 12, UCII (Other functionality1)	00

	A.4.1	Selected architectural drivers
	A.4.2	Architectural design
	A.4.3	Instantiation and allocation of functionality
	A.4.4	Interfaces for child modules
	A.4.5	New data types
	A.4.6	Verify and refine
A.5	Decom	position 3: U2, UC4, UC6, UC9, UC10, UC17, UC19
	A.5.1	Selected architectural drivers
	A.5.2	Architectural design
	A.5.3	Instantiation and allocation of functionality
	A.5.4	Interfaces for child modules
		New data types
A.6		aposition 4: Av2, UC12, UC25, UC26, UC27 (application execution subsystem)
		Selected architectural drivers
		Architectural design
		Instantiation and allocation of functionality
	A.6.4	Interfaces for child modules
		New data types
A.7		position 5: Av1 (Gateway - Online Service communication subsystem)
	A.7.1	Selected architectural drivers
	A.7.2	Architectural design
	A.7.3	Instantiation and allocation of functionality
	A.7.4	Interfaces for child modules
		New data types
A.8		position 6: P1, UC1, UC2, UC3, UC5, UC7, UC8, UC16, UC20 (Elements/Subsystem to
	decom	pose/expand)
	A.8.1	Selected architectural drivers
	A.8.2	Architectural design
	A.8.3	Instantiation and allocation of functionality
	A.8.4	Interfaces for child modules
	A.8.5	New data types
A.9	Decom	position 7: UC28, UC29 (Elements/Subsystem to decompose/expand)
	A.9.1	Selected architectural drivers
	A.9.2	Instantiation and allocation of functionality
	A.9.3	Interfaces for child modules
A.10	) Decom	position 9: UC21 (Elements/Subsystem to decompose/expand)
	A.10.1	Selected architectural drivers
	A.10.2	Instantiation and allocation of functionality
	A.10.3	Interfaces for child modules 90

## 1. Architectural Decisions

**10** 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 Av3: Pluggable device or mote failure

**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).

- DeviceManager monitors connected/operational devices on a gateway.
- . . .

Employed tactics and patterns: ...

### Rationale

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 occurred 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 <code>DeviceManager</code>. It stores the requirements for pluggable devices set by applications for all applications that use the gateway that the the <code>DeviceManager</code> runs on. When it detects that an application can no longer operate due to failures, it will send a command to the <code>ApplicationManager</code> (via the <code>GatewayFacade</code>) to suspend that application. When the required devices are operational again, the <code>DeviceManager</code> 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 functionality 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.

### Considered Alternatives

Alternative for 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.

Alternative for 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.

### **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.

Nothing about application state -> maybe could force app developers to add a procedure to save application state so that we could save this and reload that state later.

Very high coupling for ApplicationManager

Something about Modifiability problems we didn't think about.

Too much communication between gateways and online service

# 2. Client-server 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.
- ✓ 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.

**A** 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.

**A** 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.

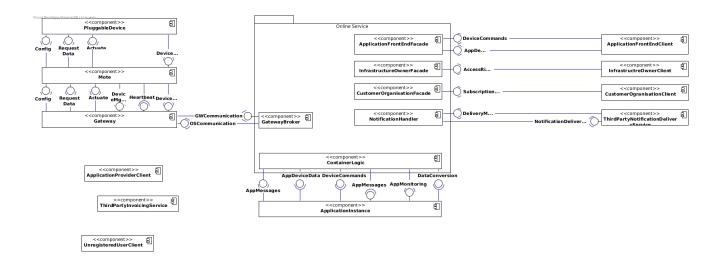


Figure 2.1: Context diagram for the client-server view.

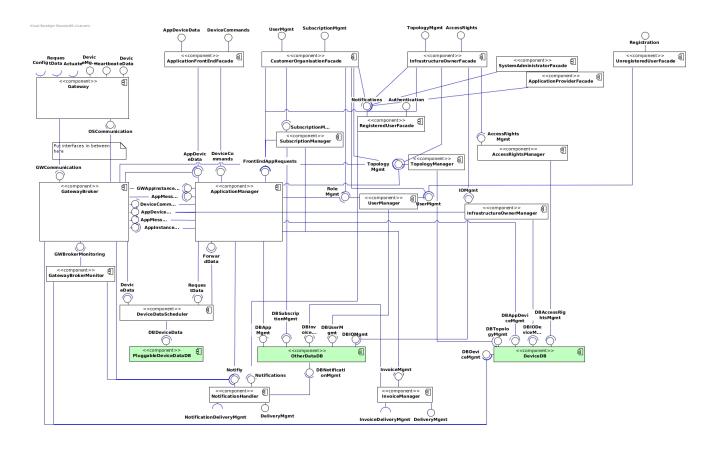


Figure 2.2: Primary diagram of the client-server view.

# 3. Decomposition view (UML Component diagram)

### **Figures**

3.1	Decomposition of Gateway
3.2	Decomposition of ApplicationManager 9
3.3	Decomposition of GatewayBroker
3.4	Decomposition of OnlineServiceBroker

✓ 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.

**A** 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.

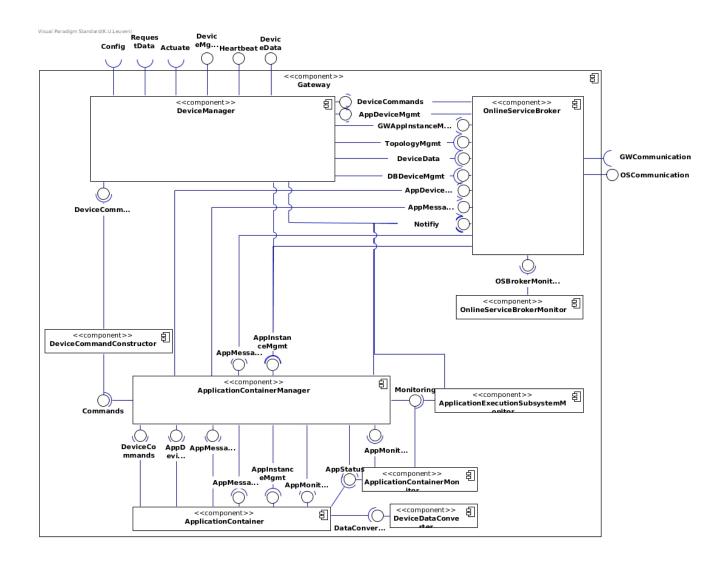


Figure 3.1: Decomposition of Gateway.

This caption contains a longer explanation over multiple lines. This additional explanation is not shown in the list of figures.

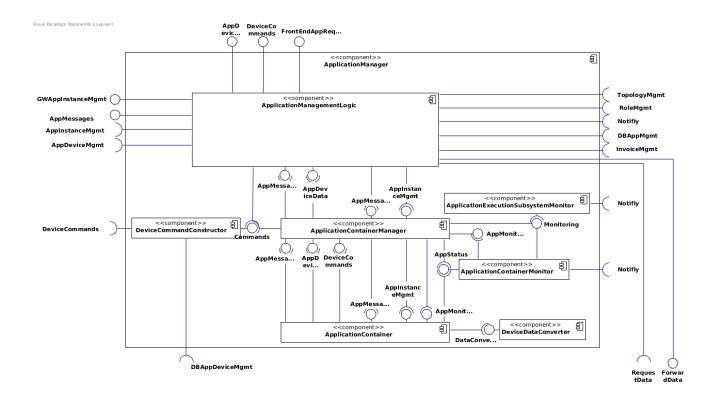


Figure 3.2: Decomposition of ApplicationManager.

This caption contains a longer explanation over multiple lines. This additional explanation is not shown in the list of figures.

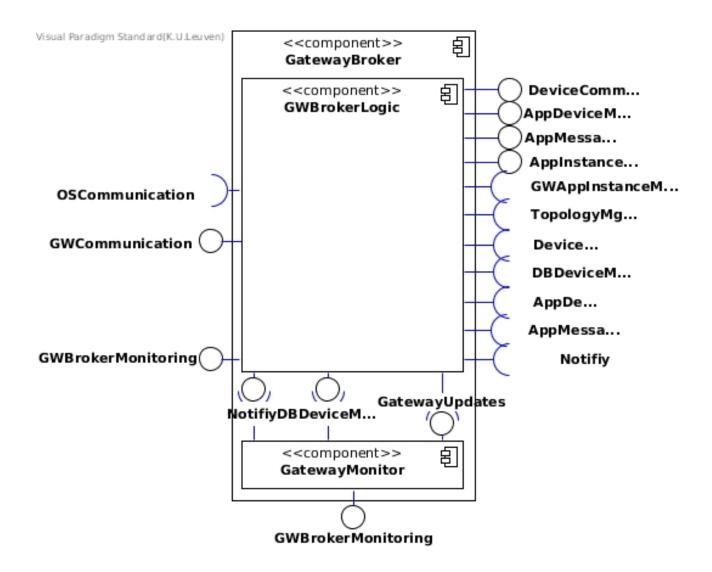


Figure 3.3: Decomposition of GatewayBroker.

This caption contains a longer explanation over multiple lines. This additional explanation is not shown in the list of figures.

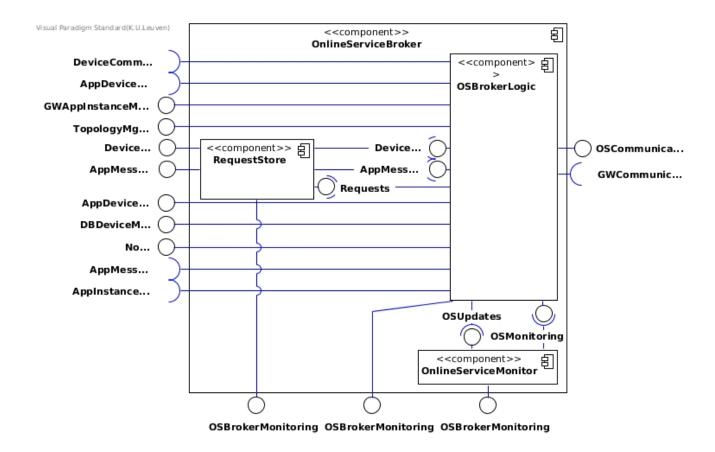


Figure 3.4: Decomposition of OnlineServiceBroker.

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.

**A** 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.

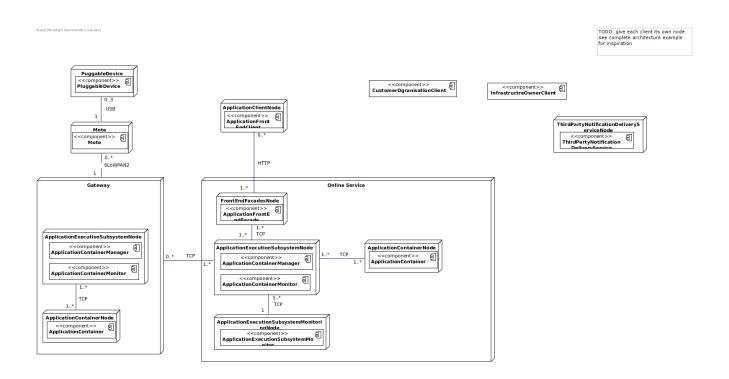


Figure 4.1: Context diagram for the deployment view.

No online service node here, because the primary diagram represents the whole online service.

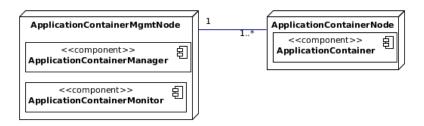




Figure 4.2: Primary diagram for the deployment view.

## 5. Scenarios

### **Figures**

5.1	Sensor data being processed by the system
5.2	Subscribing to an application
5.3	Applications issuing actuation commands
5.4	Scenario
5.5	Application crash
5.6	Plugging in a new pluggable device (sensor or actuator)
5.7	Detection and handling of communication channel failure
5.8	Upgrading an application
5.9	Sending actuation commands via a mobile app

**✓ 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.

**A** 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
- $\bullet$  Av1, UC15: Detection and handling of communication channel failure Figure 5.7
- $\bullet\,$  UC22, U1: Upgrading an application

Figure 5.8

 $\bullet$  UC26, UC27, UC12: Sending actuation commands via a mobile app Figure 5.9

# DeviceDataScheduler:

Figure 5.1: EXPLAIN WHAT HAPPENS IN THE SCENARIO. ADD COMMENTS.
LINK TO OTHER RELEVANT SCENARIO'S.

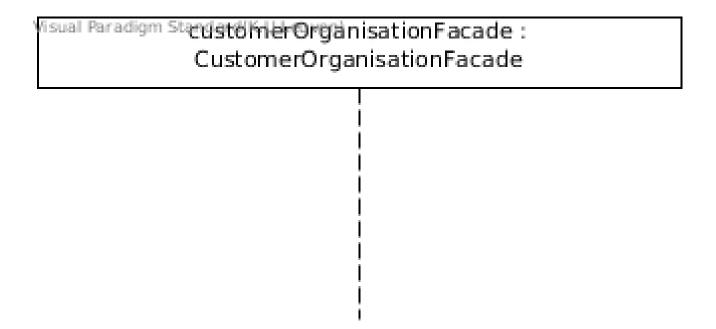


Figure 5.2: EXPLAIN WHAT HAPPENS IN THE SCENARIO. ADD COMMENTS. LINK TO OTHER RELEVANT SCENARIO'S.

# isual Paradi**applicationManage**r : ApplicationManager

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

# Visual ParadeviceMariágereuven) DeviceManager

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



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

# VisuagatewayFacadek:USateway

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

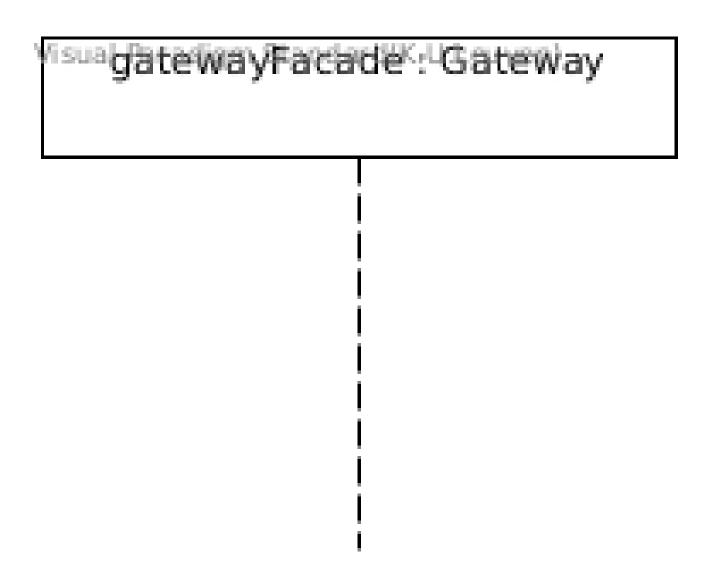


Figure 5.7: EXPLAIN WHAT HAPPENS IN THE SCENARIO. ADD COMMENTS. LINK TO OTHER RELEVANT SCENARIO'S.

# isual Paradi**applicationManage**r : ApplicationManager

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

# isual Paradi**applicationManage**r : 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

### 7.1Components

### AccessRightsManager

Responsibility: Responsible for all functionality related to access rights to pluggable devices. E.g. retrieving the access rights a customer organisation has for a device, updating access rights for customer organisations,

Super-components: None Sub-components: None

Provided interfaces: 

AccessRightsMgmt Required interfaces: < DBAccessRightsMgmt

### 7.1.2**ApplicationContainer**

Responsibility: This component contains a sandbox environment for an application instance to execute The container itself can detect when applications crash. In that case, the container tries to restart the application up to 3 times. If the application continues to crash, it is suspended and the ContainerMonitor is notified.

Super-components: 

ApplicationManager

Sub-components: 

ApplicationInstance, 
ContainerLogic

Provided interfaces: ⊸ AppInstanceMgmt, ⊸ AppMessages, ⊸ AppMonitoring

Required interfaces: < AppDeviceData, < AppMessages, < AppStatus, < DataConversion, < DeviceCommands

### **ApplicationContainerManager**

**Responsibility:** Responsible for management of ApplicationContainers. E.g. creation, suspending containers, forwarding data to containers, etc. All communication that is destined to ApplicationContainers goes through here first, since the ApplicationContainerManager knows where exactly ApplicationContainers are deployed.

Sub-components: None

Provided interfaces: → AppDeviceData, → AppInstanceMgmt, → AppMessages, → AppMonitoring,

→ Commands, → DeviceCommands, → Monitoring

Required interfaces: <a href="#">AppDeviceData</a>, <a href="#">AppInstanceMgmt</a>, <a href="#">AppMessages</a>, <a href="#">AppMonitoring</a>, ≺ AppStatus, ≺ Commands, ≺ Notifiy

### 7.1.4**ApplicationContainerMonitor**

Responsibility: Monitors ApplicationContainers and keeps track of their ApplicationInstances.

Super-components: 

ApplicationManager

Sub-components: None

Provided interfaces: ⊸ AppStatus, ⊸ Monitoring Required interfaces: ≺ AppMonitoring, ≺ Notifiy

### 7.1.5ApplicationExecutionSubsystemMonitor

Responsibility: Monitors the application execution subsystem (ApplicationContainerManager, ApplicationContainerMon

Sub-components: None Provided interfaces: None Required interfaces: < Monitoring, < Notifiy

### 7.1.6ApplicationFrontEndClient

**Responsibility:** Represents the front end client a an application.

**Super-components:** None Sub-components: None Provided interfaces: None

Required interfaces: < AppDeviceData, < DeviceCommands

### 7.1.7**ApplicationFrontEndFacade**

Responsibility: Acts as an access point for ApplicationClients and handles all functionality that can be done by application's external front ends.

Super-components: None Sub-components: None

Provided interfaces: - AppDeviceData, - DeviceCommands Required interfaces:  $\prec$  AppDeviceData,  $\prec$  DeviceCommands

### **ApplicationInstance** 7.1.8

**Responsibility:** Represents an application instance running in the system. Super-components: ¶ ApplicationManager ▷ ¶ ApplicationContainer

Sub-components: None

Provided interfaces: ⊸ AppMessages, ⊸ AppMonitoring

Required interfaces: < AppDeviceData, < AppMessages, < DataConversion, < DeviceCommands

### 7.1.9ApplicationManagementLogic

Responsibility: The entry point for ApplicationManager. Responsible for tying together all components of the ApplicationManager and handling requests related to applications.

Sub-components: None

→ AppDeviceData, → AppMessages, → DeviceCommands, Provided interfaces: → ForwardData, ¬ FrontEndAppRequests, ¬ GWAppInstanceMgmt

Required interfaces: < AppDeviceMgmt, < AppInstanceMgmt, < AppMessages, < Commands, < DBAppMgmt, ¬ InvoiceMgmt, ¬ Notifiy, ¬ RequestData, ¬ RoleMgmt, ¬ TopologyMgmt

### 7.1.10ApplicationManager

**Responsibility:** Responsible for all functionality related to applications and application instances. activating/deactivating application instances, updating an ApplicationInstance's pluggable device redundancy relationships, etc.

Super-components: None

Sub-components: DeviceCommandConstructor, ApplicationContainerMonitor, ApplicationContainerManager, ¶ ApplicationManagementLogic, ¶ ApplicationExecutionSubsystemMonitor, ¶ DeviceDataConverter, <sup>₹</sup> ApplicationContainer

Provided interfaces: → AppDeviceData, → AppMessages, → DeviceCommands, → ForwardData, → FrontEndAppRequests, → GWAppInstanceMgmt

Required interfaces: AppDeviceMgmt, AppInstanceMgmt, AppMessages, DBAppDeviceMgmt, 

TopologyMgmt

### 7.1.11 ApplicationProviderClient

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

### 7.1.12 ApplicationProviderFacade

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

Provided interfaces: ○ Authentication, ○ Notifications Required interfaces: ≺ Authentication, ≺ Notifications

### 7.1.13 AuthenticationManager

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

Provided interfaces: •• Authentication Required interfaces: •• Authentication

### 7.1.14 ContainerLogic

 $\textbf{Responsibility:} \quad \text{The logic of an Application Container.} \quad \text{Handles communication from/to} \quad$ 

ApplicationInstances and can maintain/monitor them.

Super-components: <sup>¶</sup> ApplicationManager ▷ <sup>¶</sup> ApplicationContainer

Sub-components: None

Provided interfaces: O AppDeviceData, O AppMessages, O DataConversion, O DeviceCommands

Required interfaces: ¬AppMessages, ¬AppMonitoring

### 7.1.15 CustomerOgranisationClient

**Responsibility:** Represents the client used by a customer organisation. This is the user's dashboard.

Super-components: None Sub-components: None Provided interfaces: None

Required interfaces:  $\prec$  SubscriptionMgmt

### 7.1.16 CustomerOrganisationFacade

**Responsibility:** Acts as an access point for CustomerOrganisationClients and handles all functionality that can be done by customer organisations.

Super-components: None Sub-components: None

Provided interfaces: → Authentication, → Notifications, → SubscriptionMgmt, → UserMgmt

 $\textbf{Required interfaces:} \quad \textit{``Authentication, '} \quad \textit{``FrontEndAppRequests, '} \quad \textit{``Notifications, '} \quad \textit{``RoleMgmt, } \quad \textit{``RoleMgmt, } \quad \textit{``RoleMgmt, } \quad \textit{``Authentication, '} \quad \textit{``RoleMgmt, } \quad \textit{``Ro$ 

¬ SubscriptionMgmt, ¬ TopologyMgmt, ¬ UserMgmt

### 7.1.17 DeviceCommandConstructor

**Responsibility:** Is responsible for verifying, constructing, and sending commands from applications for pluggable devices. On the Online Service, It fetches data about pluggable devices from the DeviceDB. On a Gateway, it must be configured to fetch that data from the DeviceManager.

Super-components: 

ApplicationManager

Sub-components: None

Provided interfaces: - Commands

Required interfaces: < DBAppDeviceMgmt, < DeviceCommands

### 7.1.18 DeviceDataConverter

**Responsibility:** The DeviceDataConverter is responsible for conversions pluggable device data for applications. There is one in the Online Service for application instances running in the Online Service and one in Gateways for application instances running on gateways.

Super-components: 

ApplicationManager

Sub-components: None

Provided interfaces: - DataConversion

Required interfaces: None

### 7.1.19 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: OPEVICEData, RequestData Required interfaces: Described BDEVICEData, ForwardData

### 7.1.20 DeviceDB

**Responsibility:** Contains all information related to devices in the system, but not pluggable device data such as sensor data or actuation statuses. The data includes information about pluggable devices, motes, gateways, topologies, access rights, etc.

Super-components: None Sub-components: None

 $\textbf{Provided interfaces:} \ \, ^{\circ} \ \, \texttt{DBAccessRightsMgmt}, \ \, ^{\circ} \ \, \texttt{DBAppDeviceMgmt}, \ \, ^{\circ} \ \, \texttt{DBDeviceMgmt}, \ \, ^{\circ} \ \, \texttt{DBIODeviceMgmt}, \\$ 

→ DBTopologyMgmt Required interfaces: None

### 7.1.21 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.

Keeps a local cache of data about pluggable devices from the DeviceDB, such as specific formatting syntax, possible configuration parameters, etc. This data can then be used by the DeviceCommandConstructor on the DeviceManager to convert commands from ApplicationContainers on the gateway before they are sent to pluggable devices.

Sub-components: None

Provided interfaces: 
¬ AppDeviceMgmt, ¬ DeviceCommands, ¬ DeviceData, ¬ DeviceMgmt, ¬ Heartbeat

Required interfaces: ¬ Actuate, ¬ AppInstanceMgmt, ¬ Config, ¬ DBDeviceMgmt, ¬ DeviceData,
¬ GWAppInstanceMgmt, ¬ Notifiy, ¬ RequestData, ¬ TopologyMgmt

### 7.1.22 Gateway

Responsibility: Represents a SIoTIP gateway.

Super-components: None

Sub-components: @ DeviceManager, @ OnlineServiceBrokerMonitor, @ OnlineServiceBroker

Provided interfaces:  $\neg$  DeviceData,  $\neg$  DeviceMgmt,  $\neg$  Heartbeat,  $\neg$  OSCommunication

Required interfaces: < Actuate, < Config, < GWCommunication, < RequestData

### 7.1.23 GatewayBroker

**Responsibility:** Isolates communication-related concerns between the Online Service Gateways and along with OnlineServiceBroker on Gateways. Forwards requests from one party to the other and transmits results and possible exceptions.

Sends acknowledgements for all messages sent by Gateways so that they can detect failures.

Super-components: None

Provided interfaces: - AppDeviceMgmt, - AppInstanceMgmt, - AppMessages, - DeviceCommands, - GWBrokerMonitoring, - GWCommunication

Required interfaces: AppDeviceData, 
AppMessages, 
DBDeviceMgmt, 
DeviceData, 
TopologyMgmt

### 7.1.24 GatewayBrokerMonitor

**Responsibility:** Monitors the communication component on the Online Service for communication with Gateways on the Online Service. Notifies SIoTIP system administrators in case of failures.

Super-components: None Sub-components: None Provided interfaces: None

Required interfaces: ¬ DBDeviceMgmt, 
GWBrokerMonitoring, 
Notifiy

### 7.1.25 GatewayMonitor

**Responsibility:** Monitors the connectivity status of all gateways. Can detect that a gateway is not sending data anymore based on the expected synchronisation interval. If 3 consecutive expected synchronisations do not arrive within 1 minute of their expected arrival time, this is detected as a gateway outage.

When outages of gateways are detected, the infrastructure owners that own the gateways and a SIoTIP system administrator are notified.

When the connectivity status change of a Gateway is detected, this is saved in the DeviceDB.

Quick math to check we are not doing something very dumb here: 5000 gateways \* (32 bit gatewayID + 4 bit countSynchronisationsMissed + 64 bit lastSyncPeriod + 64 bit nextSyncPeriod) == approx <math>100kb => Seems ok

Sub-components: None

Provided interfaces: → GatewayUpdates, → GWBrokerMonitoring

Required interfaces: < DBDeviceMgmt, < Notifiy

### 7.1.26 GWBrokerLogic

**Responsibility:** Handles all actual functionality related to communication. When Gateways synchronise with the Online Service, this component notifies the GatewayMonitor so that it can keep track of Gateway's status.

Sub-components: None

Provided interfaces: 
- AppDeviceMgmt, 
- AppInstanceMgmt, 
- AppMessages, 
- DBDeviceMgmt, 
- DeviceCommands, 
- GWBrokerMonitoring, 
- GWCommunication, 
- Notifiy

Required interfaces: 
AppDeviceData
AppMessages

<pr

### 7.1.27 InfrastructreOwnerClient

Responsibility: Represents the client used by an infrastructure owner. This is the user's dashboard.

Super-components: None Sub-components: None Provided interfaces: None

Required interfaces: < AccessRights

### 7.1.28 InfrastructureOwnerFacade

**Responsibility:** Acts as an access point for InfrastructureOwnerClients and handles all functionality that can be done by infrastructure owners.

Super-components: None Sub-components: None

Provided interfaces: ○ AccessRights, ○ Authentication, ○ Notifications, ○ TopologyMgmt

Required interfaces: < AccessRightsMgmt, < Authentication, < FrontEndAppRequests, < IOMgmt, < Notifications, < TopologyMgmt

### 7.1.29 InfrastructureOwnerManager

**Responsibility:** Responsible for all functionality related to infrastructure owners. E.g. looking up the devices they own, retrieving a list of customer organisations that they are associated to, etc.

 $\begin{tabular}{lll} \bf Super-components: & None \\ \bf Sub-components: & None \\ \end{tabular}$ 

 ${\bf Provided\ interfaces:}\ {\tiny \ \, {\tiny \ \, {\tiny }}{\tiny \ \, }{\tiny \ \, \ \, }{\tiny \$ 

Required interfaces: ¬ AppDeviceMgmt, < DBIODeviceMgmt, </pre> DBIODeviceMgmt, ¬ DBIOMgmt

### 7.1.30 InvoiceManager

**Responsibility:** Responsible for all functionality related to access rights to invoicing. E.g. creating invoices.

Super-components: None Sub-components: None

 $\textbf{Provided interfaces:} \ \ ^{\circ} \ \texttt{DeliveryMgmt}, \ ^{\circ} \ \texttt{InvoiceMgmt}$ 

Required interfaces: ≺ DBInvoiceMgmt, ≺ InvoiceDeliveryMgmt

### 7.1.31 Mote

Responsibility: Represents a MicroPnP mote.

Super-components: None Sub-components: None

Provided interfaces: ⊸ Actuate, ⊸ Config, ⊸ DeviceData, ⊸ RequestData

### 7.1.32 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, ⊸ Notifications, ⊸ Notifiy

Required interfaces: < DBNotificationMgmt, < NotificationDeliveryMgmt

### 7.1.33 OnlineServiceBroker

Responsibility: Isolates communication-related concerns between Gateways and the Online Service along with GatewayBroker on the Online Service. Forwards requests from one party to the other and transmits results and possible exceptions.

Keeps track of reachability status of the Online Service.

Can store at least 3 days of pluggable data and application commands before old data has to be overwritten.

Sub-components: 
© OSBrokerLogic, © OnlineServiceMonitor, © RequestStore

 $\textbf{Provided interfaces:} \quad \text{$^{\circ}$} \quad \texttt{AppDeviceData}, \quad \text{$^{\circ}$} \quad \texttt{AppMessages}, \quad \text{$^{\circ}$} \quad \texttt{DBDeviceMgmt}, \quad \text{$^{\circ}$} \quad \texttt{DeviceData},$ 

~ GWAppInstanceMgmt, ~ Notifiy, ~ OSBrokerMonitoring, ~ OSCommunication, ~ TopologyMgmt

 $\mathbf{Required\ interfaces:}\quad \textit{$^{\prec}$}\quad \texttt{AppDeviceMgmt},\quad \textit{$^{\prec}$}\quad \texttt{AppInstanceMgmt},\quad \textit{$^{\prec}$}\quad \texttt{AppMessages},\quad \textit{$^{\prec}$}\quad \texttt{DeviceCommands},$ 

 $\prec$  GWCommunication

### 7.1.34 OnlineServiceBrokerMonitor

**Responsibility:** Monitors the communication component on Gateways. If the communication component fails, the monitor tries to restart it. If the failure persists, makes the gateway reboots itself entirely.

Sub-components: None Provided interfaces: None

Required interfaces: < OSBrokerMonitoring

### 7.1.35 OnlineServiceMonitor

Responsibility: Monitors the Gateway's connectivity to the Online Service.

If the Online Service or the communication channel has failed, all requests to the Online Service will be stopped and stay stored in the RequestStore. An explicit command for this is not necessary, because the requests in the RequestStore will not be deleted, since no acknowledgements are received anymore from the Online Service.

After the monitor detects that a connection to the Online Service is possible again, it makes the gateway start synchronising again.

When the Online Service is unreachable, application parts running locally on the SIoTIP gateway continue to operate normally.

Super-components: <sup>②</sup> OnlineServiceBroker ▷ <sup>③</sup> Gateway

Sub-components: None

Provided interfaces: → OSBrokerMonitoring, → OSUpdates

Required interfaces:  $\prec$  OSMonitoring

### 7.1.36 OSBrokerLogic

Responsibility: Handles all actual functionality related to communication. When a message comes in from the Online Service, lets the OnlineServiceMonitor know that a message has been received. When an acknowledgements for requests are received from the Online Service, notifies the RequestStore of this so that it can delete the request from its storage.

Super-components: ¶ OnlineServiceBroker ▷ ¶ Gateway

Sub-components: None

```
Provided interfaces: 

AppDeviceData, 

AppMessages, 

DBDeviceMgmt, 

DeviceData, 

GWAppInstanceMgmt, 

Notifiy, 

OSBrokerMonitoring, 

OSCommunication, 

OSMonitoring, 

TopologyMgmt

Required interfaces: 

AppDeviceMgmt, 

AppInstanceMgmt, 

AppMessages, 

DeviceCommands, 

GWCommunication, 

OSUpdates, 

Requests
```

#### 7.1.37 OtherDataDB

Responsibility: General database for data. For example, storage of data about notifications.

Super-components: None Sub-components: None

 $\textbf{Provided interfaces:} \quad \neg \quad \texttt{DBAppMgmt}, \quad \neg \quad \texttt{DBInvoiceMgmt}, \quad \neg \quad \texttt{DBIOMgmt}, \quad \neg \quad \texttt{DBNotificationMgmt},$ 

→ DBSubscriptionMgmt, → DBUserMgmt

Required interfaces: None

# 7.1.38 PluggableDevice

**Responsibility:** Represents a pluggable device that plugs into MicroPnP motes. Can be a sensor or an actuator.

Sensors produce measurements and send them to the a gateway via a MicroPnP mote.

Actuators have one or more actions associated with them. For example, a switch can "turn on" and "turn off".

Super-components: None Sub-components: None

Provided interfaces: ⊸ Actuate, ⊸ Config, ⊸ RequestData

Required interfaces: < DeviceData

#### 7.1.39 PluggableDeviceDataDB

Responsibility: Database dedicated to pluggable device data only.

Super-components: None Sub-components: None

Provided interfaces: - DBDeviceData

Required interfaces: None

## 7.1.40 RegisteredUserFacade

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

Provided interfaces: 
• Authentication, • Notifications
Required interfaces: 
• Authentication, • Notifications

#### 7.1.41 RequestStore

Responsibility: Temporarily stores all pluggable data and issued application commands until they can be deleted (= until an acknowledgement has been received for the request by the Online Service). Passes all requests along to the BrokerLogic and includes a unique requestID in them, so that the Online Service can send an acknowledgement for those messages. Can store at least 3 days of pluggable data before old data has to be overwritten.

Super-components: ¶ OnlineServiceBroker ▷ ¶ Gateway

Sub-components: None

Provided interfaces: → AppMessages, → DeviceData, → OSBrokerMonitoring, → Requests

Required interfaces: ¬ AppMessages, ¬ DeviceData

#### 7.1.42 SessionDB

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

Provided interfaces: - Authentication

Required interfaces: None

## 7.1.43 SubscriptionManager

**Responsibility:** Responsible for all functionality related to access rights to subscriptions. E.g. retrieving the applications that a customer organisation can subscribe to, creating new subscriptions to ApplicationInstances, etc.

Super-components: None Sub-components: None

Provided interfaces: 

SubscriptionMgmt

Required interfaces: < DBSubscriptionMgmt, < FrontEndAppRequests, < InvoiceMgmt

## 7.1.44 SystemAdministratorFacade

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

Provided interfaces: ○ Authentication, ○ Notifications Required interfaces: ≺ Authentication, ≺ Notifications

# 7.1.45 ThirdPartyInvoicingService

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

## 7.1.46 ThirdPartyNotificationDeliveryService

**Responsibility:** Represents a third party notification delivery service.

Super-components: None Sub-components: None

 $\textbf{Provided interfaces:} \ \ ^{\circ} \ \texttt{NotificationDeliveryMgmt}$ 

Required interfaces: < DeliveryMgmt

# 7.1.47 TopologyManager

**Responsibility:** Responsible for all functionality related to topology. E.g. Adding a new mote to the topology of an infrastructure, checking whether or not all devices used by an application are active in the topology, etc.

 $\begin{array}{ll} \textbf{Super-components:} & \text{None} \\ \textbf{Sub-components:} & \text{None} \end{array}$ 

Provided interfaces: - TopologyMgmt Required interfaces: - DBTopologyMgmt

## 7.1.48 UnregisteredUserClient

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

# 7.1.49 UnregisteredUserFacade

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

Provided interfaces: •• Registration Required interfaces: •• UserMgmt

## 7.1.50 UserManager

**Responsibility:** Responsible for all functionality related to users and UserRoles. E.g. registering a new end user for a customer organisation, retrieving the user roles that are mandatory for a certain application, etc.

Super-components: None Sub-components: None

Provided interfaces: - RoleMgmt, - UserMgmt

Required interfaces: < DBUserMgmt

# 7.2 Interfaces

## 7.2.1 AccessRights

Provided by: ¶ InfrastructureOwnerFacade Required by: ¶ InfrastructreOwnerClient Operations:

- configureDevice(PluggableDeviceID plD)
  - 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: 4 AccessRightsManager

Required by: [1] InfrastructureOwnerFacade

- getCustomerOrganisationsRights(PluggableDeviceID plD, List<int> custOrglDs)
  - 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 Actuate

Provided by: 1 PluggableDevice

Required by: 5 Mote

Operations:

- void sendActuationCommand(string commandName)
  - Effect: Send an actuation command to the actuator. Sending an unknown actuation command has no effect.

#### 7.2.4 Actuate

Provided by: 1 Mote

Required by: DeviceManager, Gateway

Operations:

- void sendActuationCommand(**PluggableDeviceID** pID, string commandName)
  - Effect: Send an actuation command to an actuator. Sending an unknown actuation command has no effect.

## 7.2.5 AppDeviceData

 $\textbf{Provided by: } \P \textbf{ ApplicationContainerManager}, \P \textbf{ ApplicationFrontEndFacade}, \P \textbf{ ApplicationManagementLogic}, \P \textbf{ ApplicationManager}, \P \textbf{ ContainerLogic}, \P \textbf{ OSBrokerLogic}, \P \textbf{ OnlineServiceBroker}$ 

Operations:

- Map<PluggableDeviceID, List<DeviceData>> getDataForDevices(List<PluggableDeviceID> devices, Date-Time from, DateTime to)
  - Effect: Returns DeviceData of pluggable devices over a specified time period.
  - Created for: UC24.1
- Map<PluggableDeviceID, List<DeviceData>> getDataForRoom(RoomTopology room, DateTime from, Date-Time to)
  - Effect: Returns DeviceData of pluggable devices in a room over a specified time period.
  - Created for: UC24.1
- List<RoomTopology> getTopologyOverview(int applicationInstanceID, int customerOrganisationID)
  - Effect: Returns a list of RoomTopology containing devices that an ApplicationInstance has access
    to.
  - Created for: UC25.1

# 7.2.6 AppDeviceMgmt

Provided by: \* DeviceManager, \* GWBrokerLogic, \* GatewayBroker

Required by: ApplicationManagementLogic, ApplicationManager, InfrastructureOwnerManager, SorterLogic, OnlineServiceBroker

- bool areEssentialDevicesOperational(int applicationID)
  - Effect: Returns true if all essential devices for the application with id "applicationID" are operational.
  - Created for: UC18
- void initialisePluggableDevice(**PluggableDeviceID** pID)
  - Effect: Updates a pluggable device's status to 'active'.

- Created for: UC8.3
- boolean isDeviceInitialised(PluggableDeviceID plD)
  - 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 applicationInstanceID, List<**PluggableDeviceInfo**> devices, List<**Relationship**> relationships)
  - Effect: Sets an application's requirements for pluggable devices. e.g. a device needs to be used instead of another device if that other device fails.
  - Created for: Av3: "Application providers can design their applications such that they explicitly require redundancy in the available pluggable devices."

## 7.2.7 AppInstanceMgmt

Provided by: ¶ ApplicationContainer, ¶ ApplicationContainerManager, ¶ GWBrokerLogic, ¶ GatewayBroker Required by: ¶ ApplicationContainerManager, ¶ ApplicationManagementLogic, ¶ ApplicationManager, ¶ DeviceManager, ¶ OSBrokerLogic, ¶ OnlineServiceBroker

#### **Operations:**

- void activateApplicationInstance(int applicationInstanceID)
  - Effect: Activates an ApplicationInstance that is running on the gateway.
  - Created for: UC17.3, U2 easy applications
- void deactivateApplicationInstance(int applicationInstanceID)
  - Effect: Deactivates a running instance of an application.
  - Created for: UC18, Av3: automatic suspension/reactivation of applications.
- void destroyApplicationInstance(int applicationInstanceID)
  - Effect: Destroys an ApplicationInstance
  - Created for: UC20.4ii
- boolean isApplicationInstanceAvailable(int applicationInstanceID)
  - Effect: Returns true if a certain ApplicationInstance is available.
  - Created for: UC27.2

#### 7.2.8 AppMessages

<sup>¶</sup> GatewayBroker, <sup>¶</sup> OSBrokerLogic, <sup>¶</sup> OnlineServiceBroker, <sup>¶</sup> RequestStore

Required by: 1 ApplicationContainer, 2 ApplicationContainerManager, 1 ApplicationInstance, 2 ApplicationManagementLogic, 3 ApplicationManager, 5 ContainerLogic, 5 GWBrokerLogic, 5 CatewayBroker, 5 OSBrokerLogic, 5 OnlineServiceBroker, 5 RequestStore

- void applicationInstanceFailed(int applicationInstanceID, int onOnlineService)
  - Effect: Lets the other parts of an application know that one of their parts has become
    unavailable. This allowd the other parts to possibly run in a degraded mode. Is called by an
    ApplicationContainer after its ApplicationInstance has crashed 3 times in a row.
  - Created for: Av1
- void sendCommandToApplicationInstance(int applicationInstanceID, string command, boolean forOnlineService) throws AppMessageNotAllowedException
  - Effect: Sends a command to an ApplicationInstance that is running on the Online Service or on a Gateway.
  - Created for: UC26.1
- void sendMessageToExternalFrontEnd(string hostName, int port, string message) throws AppMessageNotAllowedException

- Effect: Sends a message from an ApplicationInstance to an external front end. The ApplicationInstance provides the hostname and port.

- Created for: UC26.1

# 7.2.9 AppMonitoring

Provided by: ¶ ApplicationContainer, ¶ ApplicationContainerManager, ¶ ApplicationInstance Required by: ¶ ApplicationContainerManager, ¶ ApplicationContainerMonitor, ¶ ContainerLogic Operations:

- **Echo** pingApplicationInstance(int applicationInstanceID)
  - Effect: Sends a ping request to an ApplicationContainer. The ApplicationContainer should respond with an Echo reply. The monitor uses this to check whether or not the container is still available.
  - Created for: The system is able to autonomously detect failures of its individual failing applications.

# **7.2.10** AppStatus

Provided by: 

ApplicationContainerMonitor

 $\mathbf{Required\ by:}\ ^{\P} \ \mathtt{ApplicationContainer},\ ^{\P} \ \mathtt{ApplicationContainerManager}$ 

**Operations:** 

- boolean isApplicationInstanceAvailable(int applicationInstanceID)
  - Effect: Returns true if a certain ApplicationInstance is available.
  - Created for: UC27.2
- void updateApplicationActive(int applicationInstanceID)
  - Effect: Updates an ApplicationInstance's status to 'active'.
- void updateApplicationInstanceSuspended(int applicationInstanceID)
  - Effect: Updates an ApplicationInstance's status to 'suspended'.
- void updateApplicationStatusUnavailable(int applicationInstanceID)
  - Effect: Effect: Notifies the ApplicationContainerMonitor that an ApplicationInstance has crashed.
  - Created for: Av2 The system is able to autonomously detect failing applications

## 7.2.11 Authentication

Provided by: 
SessionDB

Required by: 

AuthenticationManager

Operations:

## 7.2.12 Authentication

Provided by: None Required by: None

Operations:

## 7.2.13 Authentication

Provided by: ~ §I ~ ApplicationProviderFacade, ~ §I ~ AuthenticationManager, ~ §I ~ CustomerOrganisationFacade,

¶ InfrastructureOwnerFacade, ¶ RegisteredUserFacade, ¶ SystemAdministratorFacade

 $\mathbf{Required\ by:}\ ^{\P} \mathtt{ApplicationProviderFacade}, ^{\P} \mathtt{CustomerOrganisationFacade}, ^{\P} \mathtt{InfrastructureOwnerFacade}, ^{\P} \mathtt{InfrastructureOwne$ 

 $^{\P}$  Registered UserFacade,  $^{\P}$  SystemAdministratorFacade

#### **7.2.14** Commands

Provided by: ¶ ApplicationContainerManager, ¶ DeviceCommandConstructor Required by: ¶ ApplicationContainerManager, ¶ ApplicationManagementLogic Operations:

- void sendActuationCommand(List<PluggableDeviceID> device, string commandName) throws UnknownCommandException
  - Effect: Sends a command for a list of actuators to DeviceCommandConstructor for construction
    of actuation command messages according to the specific formatting syntax for the given actuators.
    The given command 'commandName' is a command that application developers use for a group of
    devices
  - Created for:  $\;$  UC12 commands from applications
- void verifyAndConstructConfigurationsForDevice(**PluggableDeviceID** pID, Map<string, string> config) throws *UnknownConfigurationParameterException* 
  - Effect: Verifies the configuration parameters for a pluggable device. If the parameters have been successfuly verified, constructs a reconfiguration command according to the specific formatting syntax for the pluggable device and sends it to the DeviceManager if everything is correct.
  - Created for: UC13.2-3

# 7.2.15 Config

Provided by: \* PluggableDevice

Required by: ¶ Mote

**Operations:** 

- Map<String, String> getConfig()
  - Effect: Returns the current configuration of a pluggable device as a parameter-value map.
- boolean setConfig(void Map<string, string>)
  - 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 Config

Provided by: 1 Mote

**Operations:** 

- Map<String, String> getConfig(PluggableDeviceID pID)
  - Effect: Returns the current configuration of a pluggable device as a parameter-value map.
- boolean setConfig(PluggableDeviceID pID, Map<string, string> config)
  - 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

#### 7.2.17 DataConversion

 $\mathbf{Provided} \ \mathbf{by:} \ {}^{\{\!\!\!\ p\ \!\!\!\}} \ \mathtt{ContainerLogic}, \ {}^{\{\!\!\!\ p\ \!\!\!\}} \ \mathtt{DeviceDataConverter}$ 

Required by: 

ApplicationContainer, 
ApplicationInstance

- **DeviceData** convert(**DeviceData** data, string targetType)
  - Effect: Converts pluggable device data into other pluggable device data that contains the same information in a different measurement type.

- Created for: M1: data processing subsystem should be extended with relevant data conversions

## 7.2.18 DBAccessRightsMgmt

Provided by: 1 DeviceDB

Required by: 4 AccessRightsManager

Operations:

- getCustomerOrganisationsRights(PluggableDeviceID plD, List<int> custOrglDs)
  - 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

## 7.2.19 DBAppDeviceMgmt

Provided by: 

DeviceDB

Required by: 4 Application Manager, 5 Device Command Constructor

Operations:

- Map<PluggableDeviceID, string> getFormattingSyntaxForDevices(List<PluggableDeviceID> devices)
  - Effect: Retrieves the specific formatting syntax for a list of pluggable devices.
  - Created for: UC12.2-3
- Map < Pluggable Device ID, Gateway Info > get Gateways For Devices (List < Pluggable Device ID > devices)
  - Effect: Returns a map of PluggableDevices with the gateways they are connected to.
  - Created for: UC12.2
- string getPossibleConfigurationParametersForDevice(**PluggableDeviceID** pID)
  - Effect: Retrieves the possible configuration parameters for a pluggable device.
  - Created for: UC13.2

## 7.2.20 DBAppMgmt

Provided by: 5 OtherDataDB

Required by: 4 ApplicationManagementLogic, 4 ApplicationManager

- 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
- GatewayInfo getApplicationIstanceGateway()
  - Effect: Returns information about a gateway that an application instance is running on.
  - Created for: Av2 Need to know which gateways to send messages to for e.g. application commands.
- 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< Pluggable Device ID> getDevices For Application (int application Instance ID)

- 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 softDeleteApplicationInstance(int applicationInstanceID)
  - Effect: Updates an ApplicationInstance's status in the database to reflect that the customer organisation has unsubscribed from it.
  - Created for: UC20.4ii
- void updateApplicationDevicesSettings(int applicationInstanceID, List<**PluggableDeviceID**> devices, List<**Relationship**> 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 updateApplicationInstance(int applicationInstanceID)
  - Effect: Updates an ApplicationInstance in the database (e.g. change state to 'inactive').
  - Created for: UC18, Av3: automatic suspension/reactivation of applications.
- void updateApplicationInstanceGateway(int applicationInstanceID, int gatewayID)
  - Effect: Updates the gateway on which an application instance is running.
  - Created for: Av2 Need to know which gateways to send messages to for e.g. application commands.
- 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

#### 7.2.21 DBDeviceData

Provided by: ¶ PluggableDeviceDataDB Required by: ¶ DeviceDataScheduler Operations:

- List<DeviceData> getData(PluggableDeviceID plD, DateTime from, DateTime to)
  - Effect: Returns data from a specific device in a certain time period.
  - Created for: P2: lookup queries
- getDataForDevices(List<PluggableDeviceID> devices, DateTime from, DateTime to)
  - Effect: Returns DeviceData of pluggable devices over a specified time period.
  - Created for: UC24.2
- void rcvData(PluggableDeviceID plD, DeviceData data)
  - Effect: Sends pluggable device data to the DB to be stored.
  - Created for: UC11, P2: storing new pluggable data

## 7.2.22 DBDeviceMgmt

#### **Operations:**

- void addDevice(PluggableDeviceID plD, PluggableDeviceType type, Map<string, string> configurations, int motelD)
  - 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 motelPAddress)
  - 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
- void deactivateMote(int moteID)
  - Effect: Changes the status of a mote to 'inactive'.
  - Created for: UC5.2a
- void deactivatePluggableDevice(PluggableDeviceID plD)
  - Effect: Changes the status of a pluggable device to 'inactive'.
  - Created for: UC7.3a
- void deactivatePluggableDevices(List<PluggableDeviceID> plDs)
  - Effect: Changes the status of a list of pluggable devices to 'inactive'.
  - Created for: UC5.2b
- 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
- double getPercentageOfGatewaysUnreachable()
  - Effect: Returns the percentage of deployed gateways that have status 'unreachable'.
  - Created for: Av1 A SIoTIP system administrator should be notified of a simultaneous outage of more than 1% of the registered gateways.
- Map<string> getPluggableDeviceData(PluggableDeviceID plD)
  - Effect: Effect: Returns all data about a pluggable device in the DeviceDB, such as status, type information, specific formatting syntax (for applications), etc.
  - Created for: UC12 construction of commands, UC13 verification of configuration parameters
- void reactivateDevice(PluggableDeviceID plD)
  - 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 conguration.
- void registerGateway(int gatewayID, **IPAddress** address)
  - Effect: Sets a gateway's status to 'active' and updates its IP address.
  - Created for: U2 gateway installation
- void updateGatewayStatus(int gatewayID)
  - Effect: Changes a gateway's status.
  - Created for: Av1 Connectivity of gateways

## 7.2.23 DBInvoiceMgmt

Provided by: \*\* OtherDataDB Required by: \*\* InvoiceManager

- void createInvoice(Invoice invoice)
  - Effect: Stores a new invoice in the database.
  - Created for: UC20.4iii

- Invoice getInvoicesForCustomerOrganisation(int customerOrganisationID)
  - Effect: Returns outstanding invoices for a customer organisation.
  - Created for: UC21.1
- 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

# 7.2.24 DBIODeviceMgmt

Provided by: 5 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
- void initialisePluggableDevice(**PluggableDeviceID** pID)
  - Effect: Updates a pluggable device's status to 'active'.
  - Created for: UC8.3

## 7.2.25 DBIOMgmt

Provided by: [ OtherDataDB

Required by: [1] 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

#### 7.2.26 DBNotificationMgmt

Provided by: [ OtherDataDB

- int createNotification(Notification notification)
  - Effect: Stores a new notification entry in the database. Returns the id of the new notification.
  - Created for: UC15, Av3: notifications
- **Notification** getDetailedNotification(int userID, int notificationID)
  - Effect: Returns detailed information about a notification for a certain registered user.
  - Created for: UC16.4
- List<Notification> getNotifications(int userID)
  - Effect: Returns a list of Notification for a certain registered user. Not all details in the Notification objects are filled in.
  - Created for: UC16.2
- int lookupNotificationChannelForUser(int userID)
  - Effect: Returns the id of the type of communication channel a user prefers.
  - Created for: UC15
- void setNotificationRead(int notificationID)
  - Effect: Marks a notification as 'read'.
  - Created for: UC16.5
- int updateNotification(Notification notification)
  - Effect: Updates an existing notification (e.g. change status to "sent").
  - Created for: UC15

## 7.2.27 DBSubscriptionMgmt

Provided by: 5 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 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
- List<**Subscription**> getSubscriptions(int customerOrganisationID)
  - Effect: Returns a list of the subscriptions of a customer organisation.
  - Created for: UC20.2
- void softDeleteSubscription(int subscriptionID)
  - Effect: Ends a subsription.Created for: UC20.4i

## 7.2.28 DBTopologyMgmt

Provided by: \* DeviceDB

Required by: TopologyManager

- void addDevice(PluggableDeviceID plD, 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 motelD, 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
- void deactivateMote(int moteID)
  - Effect: hanges the status of a mote in the topology to 'inactive'.
  - Created for: UC5.2a
- void deactivatePluggableDevice(PluggableDeviceID plD)
  - Effect: Changes the status of a pluggable device in the topology to 'inactive'.
  - Created for: UC7.3b
- void deactivatePluggableDevices(List<PluggableDeviceID> plDs)
  - Effect: Changes the status of a list of pluggable devices in the topology to 'inactive'.
  - Created for: UC5.2b
- List< DeviceStatus> getStatusDetailsForGateway(int gatewayID)
  - Effect: Returns details about the status of a certain gateway.
     e.g. indicating that the device was unavailable between 2 and 3 a.m.
  - Created for: UC10.3a
- List< DeviceStatus> getStatusDetailsForMote(int moteID)
  - Effect: Returns details about the status of a certain mote. e.g. indicating that the device was unavailable between 2 and 3 a.m.
  - Created for: UC10.3a

- List<DeviceStatus> getStatusDetailsForPluggableDevice(PluggableDeviceID plD)
  - Effect: Returns details about the status of a certain pluggable device. e.g. indicating that the device was unavailable between 2 and 3 a.m.
  - Created for: UC10.3a
- List<RoomTopology> getTopologyOfCustomerOrganisation(int custOrgID)
  - Effect: Returns a list of RoomTopology associated to a customer organisation.
  - Created for: UC19.5, U2 easy applications
- List<RoomTopology> getTopologyOfInfrastructureOwner(int infrastructureOwnerID)
  - Effect: Returns the topology owned by an infrastructure owner.
  - Created for: UC10.2
- List<MoteInfo> getUnplacedMotes(int infrastructureOwnerID)
  - Effect: Returns info about unplaced motes that are owned by an infrastructure owner
  - Created for: UC10.2
- List< Pluggable DeviceInfo> getUnplacedPluggable Devices (int infrastructureOwnerID)
  - Effect: Returns info about unplaced motes that are owned by an infrastructure owner
  - Created for: UC10.2
- 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 conguration.
- void updateTopology(int infrastructureOwnerID, List<RoomTopology> updatedTopology)
  - Effect: Updates the topology of an infrastructure owner. The 'updatedTopology' is the new one submitted by the infrastructure owner.
  - Created for: UC10.5a

## 7.2.29 DBUserMgmt

Provided by: ① OtherDataDB Required by: ② UserManager

- 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
- boolean canCustomerOrganisationRegister(int infrastructureOwnerID, string hash)
  - Effect: Returns true if the combination of infrastructure owner and hash exists in the system.
     This means that the infrastructure owner has allowed for a certain customer organistaion to register.
  - Created for: UC1.1
- boolean confirmEmail(string emailAddress, string randomString)
  - Effect: Return true if the given e-mail and randomString match in the database.
     Returns false if user did not click the link in the received e-mail within 30 minutes.
  - Created for: UC1.5
- int createCustomerOrganisation(Map<string> data) throws AddressInvalidException, UserNameAlreadyExistsException, PaymentInformationInvalidException
  - Effect: Creates a customer organisation and returns its id. Links the customer organisation to an infrastructure owner by using the information stored in 'createEmailAddressForRegistration'.
  - Created for: UC1.8
- boolean createEmailAddressForRegistration(string emailAddress, int infrastructureOwnerID) throws *EmailAddress-InvalidException*, *EmailAddressAlreadyExistsException* 
  - Effect: If the given e-mail address is valid and not yet in the database, saves the e-mail address in

the database, links it to the infrastructure owner and a unique random string for the confirmation e-mail, and the current timestamp. Afterwards, returns true.

Otherwise, returns false.

- Created for: UC1.4
- boolean createEndUser(int customerOrganisationID, Map<string> data) throws *EmailAddressAlreadyExist-sException*, *EmailAddressInvalidException*, *EndUserAlreadyExistsException*, *PhoneNumberInvalidException* 
  - Effect: Verifies provided data for a new end-user, checks that the end-user does not exist in the system already, and then stores the end-user information and associates it with the customer organisation. Returns true if everything succeeded.
  - Created for: UC2.3-4
- 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 softDeleteEndUser(int customerOrganisationID, int userID)
  - Effect: Marks the profile of an end-user as 'inactive'.
  - Created for UC3.4
- 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

## 7.2.30 DeliveryMgmt

Provided by: [9] NotificationHandler

Required by: [9] ThirdPartyNotificationDeliveryService

**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

# 7.2.31 DeliveryMgmt

Provided by: [1] InvoiceManager

Required by: None

Operations:

- void acknowledgement(int invoiceID)
  - Effect: Sends an acknowledgement to the system for a certain invoice to denote that the invoice has been received.

#### 7.2.32 DeviceCommands

Provided by: DeviceManager, GWBrokerLogic, GatewayBroker

Required by: ¶ ApplicationManager, ¶ DeviceCommandConstructor, ¶ OSBrokerLogic, ¶ OnlineServiceBroker

#### **Operations:**

- Map<PluggableDeviceID, string> getFormattingSyntaxForDevices(List<PluggableDeviceID> devices)
  - Effect: Retrieves the specific formatting syntax for a list of pluggable devices from the DeviceManager. Used by the DeviceCommandConstructor on Gateways.
  - Created for: UC12 commands from applications on gateways
- string getPossibleConfigurationParametersForDevice(PluggableDeviceID pID)
  - Effect: Retrieves the possible configuration parameters for a pluggable device. Used by the DeviceCommandConstructor on Gateways.
  - Created for: UC13 configuration commands from applications on gateways
- void sendActuationCommand(Map<**PluggableDeviceID**, string> commandsForDevices)
  - Effect: Sends correctly constructed actuation commands for a group of actuators to the DeviceManager on a Gateway.
  - Created for: UC12 send command message to the intended actuators
- void setConfiguration(**PluggableDeviceID** pID, Map<string, string> config)
  - Effect: Sets configuration parameters of a pluggable device. The DeviceManager first determines whether the pluggable device needs to be reconfigured. To do this, it checks the data it has about configurations set by other applications on the pluggable device. If the device can be reconfigured, then the configuration command is propagated to the pluggable device.
  - Created for: UC13.3

#### 7.2.33 DeviceCommands

Provided by: \*\* ApplicationContainerManager, \*\* ApplicationFrontEndFacade, \*\* ApplicationManagementLogic, \*\* ApplicationManager, \*\* ContainerLogic

Required by: ApplicationContainer, ApplicationFrontEndClient, ApplicationFrontEndFacade, ApplicationInstance

#### **Operations:**

- void sendActuationCommand(List<**PluggableDeviceID**> devices, string command)
  - Effect: Sends a command for a list of actuators for construction of the actuation command messages.
  - Created for: UC12.1
- void setConfiguration(PluggableDeviceID plD, Map<string, string> config)
  - Effect: Sends a configuration command for a pluggable device for construction of the actual command messages.
  - Created for: UC13.1

#### 7.2.34 DeviceData

Provided by: <sup>1</sup> DeviceDataScheduler, <sup>1</sup> OSBrokerLogic, <sup>1</sup> OnlineServiceBroker, <sup>1</sup> RequestStore Required by: <sup>1</sup> DeviceManager, <sup>1</sup> GWBrokerLogic, <sup>1</sup> GatewayBroker, <sup>1</sup> RequestStore 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

## 7.2.35 DeviceData

Provided by: DeviceManager, Sateway, Mote

Required by: [9] Mote, [9] PluggableDevice

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

## 7.2.36 DeviceMgmt

Provided by: DeviceManager, Gateway

Required by: ¶ Mote

**Operations:** 

- void pluggableDevicePluggedIn(MoteInfo mInfo, PluggableDeviceID pID, PluggableDeviceType type)
  - Effect: Notify the gateway that a new PluggableDevice of the given type is connected to the mote.
- void pluggableDeviceRemoved(PluggableDeviceID pID)
  - Effect: Notify the gateway that a PluggableDevice is removed.

#### 7.2.37 ForwardData

Provided by: 

ApplicationManagementLogic, 
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

# 7.2.38 FrontEndAppRequests

Provided by: [9] ApplicationManagementLogic, [9] ApplicationManager

- void activateApplication(int applicationInstanceID)
  - Effect: Checks and activates an ApplicationInstance (UC17).
  - Created for: UC19.14, U2 easy applications
- 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 checkApplicationsForActivationForInfrastructureOwner(int infrastructureOwnerID)
  - Effect: Checks and activates ApplicationInstances 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: UC10.5b
- void checkApplicationsForDeactivationForCustomerOrganisations(List<int> custOrgIDs)
  - Effect: Checks for ApplicationInstances that require deactivation for a list of customer organisations.
  - Created for: UC9.7
- void checkApplicationsForDectivationForInfrastructureOwner(int infrastructureOwnerID)
  - Effect: Checks and deactivates ApplicationInstances which require deactivation. The applications
    checked are those that are subscribed to by customers organisations associated to the given
    infrastructure owner.
  - Created for: UC10.5c
- 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
- void destroyApplicationInstance(int applicationInstanceID)
  - Effect: Destroys an ApplicationInstance
  - Created for: UC20.4ii

- 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 softDeleteApplicationInstance(int applicationInstanceID)
  - Effect: Updates an ApplicationInstance's status in the database to reflect that the customer organisation has unsubscribed from it.
  - Created for: UC20.4ii
- void updateApplicationDevicesSettings(int applicationInstanceID, List<**PluggableDeviceID**> devices, List<**Relationship**> 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

## 7.2.39 GatewayUpdates

 $\begin{array}{ll} \textbf{Provided by:} \; ^{\underline{e}_{\underline{l}}} \; \texttt{GatewayMonitor} \\ \textbf{Required by:} \; ^{\underline{e}_{\underline{l}}} \; \texttt{GWBrokerLogic} \end{array}$ 

**Operations:** 

- void gatewayUpdate(int gatewayID, **DateTime** time)
  - Effect: Lets the GatewayMonitor know that data has been received from a certain Gateway. Is called after a synchronisation of the gateway. When 3 consecutive expected synchronisations that do not arrive within 1 minute of their expected arrival time, a gateway outage is detected by the monitor.
  - Created for: Av1 The Online Service is able to detect that a SIoTIP gateway is not sending data anymore based on the expected synchronisation interval.

## 7.2.40 GWAppInstanceMgmt

Provided by: ¶ ApplicationManagementLogic, ¶ ApplicationManager, ¶ OSBrokerLogic, ¶ OnlineServiceBroker Required by: ¶ DeviceManager, ¶ GWBrokerLogic, ¶ GatewayBroker 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 ApplicationInstances 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 checkApplicationsForDeactivationForPluggableDevice(PluggableDeviceID plD)
  - Effect: Checks and possibly deactivates ApplicationInstances which are affected by the unavailability
    of a pluggable device (Include: UC18: Check and deactivate applications).
  - Created for: UC7.3c
- void checkApplicationsForDeactivationForPluggableDevices(List<PluggableDeviceID> plDs)
  - Effect: Checks and possibly deactivates ApplicationInstances which are affected by the unavailability
    of a list of pluggable devices (Include: UC18: Check and deactivate applications).
  - Created for: UC5.2c
- void deactivateApplicationInstance(int applicationInstanceID)

- Effect: Deactivates a running instance of an application.
- Created for: UC18, Av3: automatic suspension/reactivation of applications.

# 7.2.41 GWBrokerMonitoring

Provided by: 9 GWBrokerLogic, 9 GatewayBroker, 9 GatewayMonitor

Required by: [ GatewayBrokerMonitor

Operations:

- Echo ping()
  - Effect: Sends a ping request to a component. The component should respond with an Echo reply.
     The monitor uses this to check whether or not the component is still available.
  - Created for: Av1 The Online Service is able to autonomously detect failures of its individual internal communication components.

#### 7.2.42 GWCommunication

Required by: 4 Gateway, 4 OSBrokerLogic, 4 OnlineServiceBroker

**Operations:** 

void receive(void ...)

- Effect: Undefined

• void send(void ...)

- Effect: <u>Undefined</u>

## 7.2.43 Heartbeat

Provided by: DeviceManager, Gateway

Required by: 4 Mote

**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

## 7.2.44 InvoiceDeliveryMgmt

Provided by: None

Required by: [InvoiceManager

**Operations:** 

- void sendInvoice(Map<string, string> invoiceData)
  - Effect: Sends an invoice using a third party invoicing service.
  - Created for: UC21.2

## 7.2.45 InvoiceMgmt

Provided by: 1 InvoiceManager

Required by:  $\P$  ApplicationManagementLogic,  $\P$  ApplicationManager,  $\P$  SubscriptionManager Operations:

- void constructInvoiceForApplicationInstance(int applicationInstanceID, int customerOrganisationID)
  - Effect: Constructs an invoice for outstanding payments for an ApplicationIntance
  - Created for: UC20.4iii
- 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

## 7.2.46 IOMgmt

Provided by:  ${}^{\{\!\!\mid\,}$  InfrastructureOwnerManager Required by:  ${}^{\{\!\!\mid\,\}}$  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

- void initialisePluggableDevice(PluggableDeviceID plD, int gatewayID)
  - Effect: Updates a pluggable device's status to 'active'.

- Created for: UC8.3

## 7.2.47 Monitoring

Provided by: [1] ApplicationContainerManager, [1] ApplicationContainerMonitor

Required by: 4 ApplicationExecutionSubsystemMonitor

Operations:

- Echo ping()
  - Effect: Sends a ping request to a component. The component should respond with an Echo reply.
     The monitor uses this to check whether or not the component is still available.
  - Created for: The system is able to autonomously detect failures of its individual application execution components

## 7.2.48 NotificationDeliveryMgmt

Provided by: \* ThirdPartyNotificationDeliveryService

Required by: 1 NotificationHandler

**Operations:** 

- void notify(Map<string, string> data)
  - Effect: Delivers a notification to an end user using a third party notification delivery service.
  - Created for: UC15

#### 7.2.49 Notifications

Provided by: <sup>1</sup> NotificationHandler Required by: <sup>1</sup> RegisteredUserFacade

Operations:

- **Notification** getDetailedNotification(int userID, int notificationID)
  - Effect: Returns detailed information about a notification for a certain registered user.

- Created for: UC16.4

- List<Notification> getNotifications(int userID)
  - Effect: Returns a list of Notification for a certain registered user. Not all details in the Notification objects are filled in.
  - Created for: UC16.2

## 7.2.50 Notifications

Provided by: ApplicationProviderFacade, CustomerOrganisationFacade, InfrastructureOwnerFacade, RegisteredUserFacade, SystemAdministratorFacade

Required by: [1] ApplicationProviderFacade, [2] CustomerOrganisationFacade, [2] InfrastructureOwnerFacade, [3] SystemAdministratorFacade

- Notification consultNotification(int userID, int notificationID)
  - Effect: Returns detailed information about a notification for a certain registered user.
  - Created for: UC16.3
- List<Notification> consultNotifications(int userID)
  - Effect: Returns a list of Notification for a certain registered user.
  - Created for: UC16.1

# **7.2.51** Notifiy

Provided by: @ GWBrokerLogic, @ NotificationHandler, @ OSBrokerLogic, @ OnlineServiceBroker

Required by: 

ApplicationContainerManager, 

ApplicationContainerMonitor, 

ApplicationExecutionSubsystemMo

- ApplicationManagementLogic, ApplicationManager, DeviceManager, GWBrokerLogic,
- <sup>§</sup> GatewayBroker, <sup>§</sup> GatewayBrokerMonitor, <sup>§</sup> GatewayMonitor

## 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
- void notifySystemAdministrator(string message)
  - Effect: Stores a new notification in the system and causes it to be sent to a SIoTIP system administrator.

# 7.2.52 OSBrokerMonitoring

Provided by: 9 OSBrokerLogic, 9 OnlineServiceBroker, 9 OnlineServiceMonitor, 9 RequestStore

Required by: 4 OnlineServiceBrokerMonitor

# **Operations:**

- Echo ping()
  - Effect: Sends a ping request to a component. The component should respond with an Echo reply.
     The monitor uses this to check whether or not the component is still available.
  - Created for: Av1 The SIoTIP gateway is able to autonomously detect failures of its individual internal communication components.
- void restart()
  - Effect: Restarts the Gateways' communication component.
  - Created for: Av1 If an internal SIoTIP gateway component fails, the gateway first tries to restart the affected component.

#### 7.2.53 OSCommunication

Provided by: [9] Gateway, [9] OSBrokerLogic, [9] OnlineServiceBroker

Required by: [9] GWBrokerLogic, [9] GatewayBroker

#### **Operations:**

- void acknowledgement(int requestID)
  - Effect: Sends an acknowledgement for a request sent by the SIoTIP gateway.
  - Created for: Av1 The Online Service should acknowledge each message sent by the SIoTIP gateway so that the gateway can detect failures.
- void receive(void ...)
  - Effect: <u>Undefined</u>
- void send(void ...)
  - Effect: Undefined

## 7.2.54 OSMonitoring

Provided by: [ OSBrokerLogic

Required by:  $\P$  OnlineServiceMonitor

#### **Operations:**

- Echo pingOnlineService()
  - Effect: Makes the BrokerLogic send a ping to the Online Service. The Online Service should respond with an Echo reply. The monitor uses this to check whether or not the component is still available.
  - Created for: The SIoTIP gateway will start synchronising with the Online service within 1 minute after the communication channel becomes available.
- void synchroniseWithOnlineService()
  - Effect: Makes the BrokerLogic fetch all pluggable data and application commands as new requests to start sending to the Online Service again.
  - Created for: Av1 The SIoTIP gateway will start synchronising with the Online service within 1 minute after the communication channel becomes available.

## 7.2.55 OSUpdates

Provided by: [9] OnlineServiceMonitor

Required by: @ OSBrokerLogic

**Operations:** 

- void onlineServiceUpdate()
  - Effect: Lets the OnlineServiceMonitor know that a message has been received from the Online Service
  - Created for: Av1 The Online Service should acknowledge each message sent by the SIoTIP gateway so that the gateway can detect failures.

## 7.2.56 Registration

Provided by: UnregisteredUserFacade

Required by: None

**Operations:** 

- boolean canCustomerOrganisationRegister(int infrastructureOwnerID, string hash)
  - Effect: Returns true if the combination of infrastructure owner and hash exists in the system.
     This means that the infrastructure owner has allowed for a certain customer organistaion to register.
  - Created for: UC1.1
- boolean confirmEmail(string emailAddress, string randomString)
  - Effect: Return true if the given e-mail and randomString match in the database.
     Returns false if user did not click the link in the received e-mail within 30 minutes.
  - Created for: UC1.5
- int createCustomerOrganisation(Map<string, string> data) throws AddressInvalidException, UserNameAlreadyExistsException, PaymentInformationInvalidException
  - Effect: Creates a customer organisation and returns its id. Links the customer organisation to an infrastructure owner by using the information stored in 'createEmailAddressForRegistration'.
  - Created for: UC1.7
- boolean createEmailAddressForRegistration(string emailAddress, int infrastructureOwnerID) throws EmailAddress-sAlreadyExistsException, EmailAddressInvalidException
  - Effect: If the given e-mail address is valid and not yet in the database, saves the e-mail address in the database, links it to the infrastructure owner and a unique random string for the confirmation e-mail, and the current timestamp. Afterwards, returns true.
     Otherwise, returns false.
  - Created for: UC1.3-4

#### 7.2.57 RequestData

Provided by: \* PluggableDevice

Required by: 4 Mote

#### **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).

## 7.2.58 RequestData

Provided by: Mote

Operations:

• DeviceData getData(PluggableDeviceID plD)

- Effect: Synchronously retrieve the device data of a pluggable device.

• void getDataAsync(PluggableDeviceID plD, int requestID)

- Effect: Asynchronously retrieve the device data of a pluggable device (by calling rcvDataCallback).

## 7.2.59 RequestData

Provided by: \* DeviceDataScheduler

Required by: 

ApplicationManagementLogic, 

ApplicationManager

**Operations:** 

- List<DeviceData> getData(PluggableDeviceID plD, DateTime from, DateTime to)
  - Effect: Requests data from a specific device in a certain time period.
  - Created for: P2: requests from applications
- Map<PluggableDeviceID, List<DeviceData>> getDataForDevices(List<PluggableDeviceID> devices, Date-Time from, DateTime to)
  - Effect: Returns DeviceData of pluggable devices over a specified time period.
  - Created for: UC24.2

## 7.2.60 Requests

Provided by: TequestStore Required by: ToSBrokerLogic

Operations:

- void acknowledgement(int requestID)
  - Effect: Sends an acknowledgement for a request sent by the SIoTIP gateway. Deletes a request that is stored in the RequestStore.
  - Created for: Av1 the SIoTIP gateway will temporarily store all incoming pluggable data and any issued application commands internally
- List<List<byte>>> getNewRequestsForSynchronisation()
  - Effect: Returns pluggable data and application commands as new requests to start sending to the Online Service again.
  - Created for: Av1 The SIoTIP gateway will start synchronising with the Online service within 1 minute after the communication channel becomes available.

## 7.2.61 RoleMgmt

Provided by: UserManager

Required by:  $\P$  ApplicationManagementLogic,  $\P$  ApplicationManager,  $\P$  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<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

## 7.2.62 SubscriptionMgmt

Provided by: <sup>{§}</sup> CustomerOrganisationFacade Required by: <sup>{§}</sup> 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
- List<**Subscription**> getSubscriptions(int customerOrganisationID)
  - Effect: Returns a list of the subscriptions of a customer organisation.
  - Created for: UC20.1
- 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 unsubscribeFromApplication(int customerOrganisationID, int subscriptionID, int applicationInstanceID)
  - Effect: Unsubscribes a customer organisation from an application.
  - Created for: UC20.3
- void updateApplicationDevicesSettings(int applicationInstanceID, List<**PluggableDeviceID**> devices, List<**Relationship**> 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

## 7.2.63 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
- List<**Subscription**> getSubscriptions(int customerOrganisationID)
  - Effect: Returns a list of the subscriptions of a customer organisation.
  - Created for: UC20.2
- void softDeleteSubscription(int subscriptionID)
  - Effect: Ends a subsription.
  - Created for: UC20.4i

# 7.2.64 TopologyMgmt

Provided by: [1] InfrastructureOwnerFacade

Required by: None

Operations:

- List< **DeviceStatus**> getStatusDetailsForGateway(int gatewayID)
  - Effect: Returns details about the status of a certain gateway.
     e.g. indicating that the device was unavailable between 2 and 3 a.m.
  - Created for: UC10.3a
- List<**DeviceStatus**> getStatusDetailsForMote(int moteID)
  - Effect: Returns details about the status of a certain mote. e.g. indicating that the device was unavailable between 2 and 3 a.m.
  - Created for: UC10.3a
- List<DeviceStatus> getStatusDetailsForPluggableDevice(PluggableDeviceID plD)
  - Effect: Returns details about the status of a certain pluggable device. e.g. indicating that the device was unavailable between 2 and 3 a.m.
  - Created for: UC10.3a
- List<**RoomTopology**> getTopology(int infrastructureOwnerID)
  - Effect: Returns the topology owned by an infrastructure owner.
  - Created for: UC10.1
- void updateTopology(int infrastructureOwnerID, List<RoomTopology) updatedTopology)</li>
  - Effect: Updates the topology of an infrastructure owner. The 'updatedTopology' is the new one submitted by the infrastructure owner.
  - Created for: UC10.4

# 7.2.65 TopologyMgmt

Provided by: @ OSBrokerLogic, @ OnlineServiceBroker, @ TopologyManager

- 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
- void deactivateMote(int moteID)
  - Effect: Changes the status of a mote in the topology to 'inactive'.
  - Created for: UC5.2a
- void deactivatePluggableDevice(PluggableDeviceID plD)
  - Effect: Changes the status of a pluggable device in the topology to 'inactive'.
  - Created for: UC7.3b
- void deactivatePluggableDevices(List<**PluggableDeviceID**> plDs)
  - Effect: Changes the status of a list of pluggable devices in the topology to 'inactive'.
  - Created for: UC5.2b
- List< **DeviceStatus**> getStatusDetailsForGateway(int gatewayID)
  - Effect: Returns details about the status of a certain gateway. e.g. indicating that the device was unavailable between 2 and 3 a.m.
  - Created for: UC10.3a
- List< DeviceStatus> getStatusDetailsForMote(int moteID)
  - Effect: Returns details about the status of a certain mote. e.g. indicating that the device was unavailable between 2 and 3 a.m.
  - Created for: UC10.3a
- List< DeviceStatus> getStatusDetailsForPluggableDevice(PluggableDeviceID plD)
  - Effect: Returns details about the status of a certain pluggable device.
     e.g. indicating that the device was unavailable between 2 and 3 a.m.
  - Created for: UC10.3a
- List<RoomTopology> getTopologyOfCustomerOrganisation(int custOrgID)
  - Effect: Returns a list of RoomTopology associated to a customer organisation.
  - Created for: UC19.5, U2 easy applications
- List<RoomTopology> getTopologyOfInfrastructureOwner(int infrastructureOwnerID)
  - Effect: Returns the topology owned by an infrastructure owner.
  - Created for: UC10.2
- List< **MoteInfo**> getUnplacedMotes(int infrastructureOwnerID)
  - Effect: Returns info about unplaced motes that are owned by an infrastructure owner
  - Created for: UC10.2
- List<PluggableDeviceInfo> getUnplacedPluggableDevices(int infrastructureOwnerID)
  - Effect: Returns info about unplaced motes that are owned by an infrastructure owner
  - Created for: UC10.2
- 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.
- void updateTopology(int infrastructureOwnerID, List<RoomTopology> updatedTopology)
  - Effect: Updates the topology of an infrastructure owner. The 'updatedTopology' is the new one submitted by the infrastructure owner.
  - Created for: UC10.5a

## 7.2.66 UserMgmt

Provided by: UserManager

Required by: [9] CustomerOrganisationFacade, [9] UnregisteredUserFacade

#### **Operations:**

- boolean canCustomerOrganisationRegister(int infrastructureOwnerID, string hash)
  - Effect: Returns true if the combination of infrastructure owner and hash exists in the system. This means that the infrastructure owner has allowed for a certain customer organistaion to register.
  - Created for: UC1.1
- boolean confirmEmail(string emailAddress, string randomString)
  - Effect: Return true if the given e-mail and randomString match in the database.
     Returns false if user did not click the link in the received e-mail within 30 minutes.
  - Created for: UC1.5
- int createCustomerOrganisation(Map<string, string> data) throws AddressInvalidException, UserNameAlreadyExistsException, PaymentInformationInvalidException
  - Effect: Creates a customer organisation and returns its id. Links the customer organisation to an infrastructure owner by using the information stored in 'createEmailAddressForRegistration'.
  - Created for: UC1.8
- boolean createEmailAddressForRegistration(string emailAddress, int infrastructureOwnerID) throws *EmailAddress-sAlreadyExistsException*, *EmailAddressInvalidException* 
  - Effect: If the given e-mail address is valid and not yet in the database, saves the e-mail address in the database, links it to the infrastructure owner and a unique random string for the confirmation e-mail, and the current timestamp. Afterwards, returns true.
    - Otherwise, returns false.
  - Created for: UC1.3-4
- boolean createEndUser(int customerOrganisationID, Map<string> data) throws *EmailAddressAlreadyExist-sException*, *EmailAddressInvalidException*, *EndUserAlreadyExistsException*, *PhoneNumberInvalidException* 
  - Effect: Verifies provided data for a new end-user, checks that the end-user does not exist in the system already, and then stores the end-user information and associates it with the customer organisation. Returns true if everything succeeded.
  - Created for: UC2.3-4
- List<User> getEndUsers(int customerOrganisationID)
  - Effect: Returns a list of Users which are associated to a customer organisation.
  - Created for: UC19.8, U2 easy applications
- void softDeleteEndUser(int customerOrganisationID, int userID)
  - Effect: Marks the profile of an end-user as 'inactive'.
  - Created for: UC3.4

## 7.2.67 UserMgmt

Provided by:  ${}^{\{\!\!\mid\,}$  CustomerOrganisationFacade

Required by: None

- List<User> getEndUsers(int customerOrganisationID)
  - Effect: Returns a list of all end-users associated with a customer organisation.
  - Created for: UC3.1
- boolean registerEndUser(int customerOrganisationID, Map<string, string> data) throws EmailAddressAlreadyExistsException, EmailAddressInvalidException, PhoneNumberInvalidException, EndUserAlreadyExistsException
  - Effect: Verifies provided data for a new end-user, checks that the end-user does not exist in the system already, and then stores the end-user information and associates it with the customer organisation. Returns true if everything succeeded.
  - Created for: UC2.3-4
- void unregisterEndUser(int customerOrganisationID, int userID)
  - Effect: Unregisters an end-user.
  - Created for: UC3.3

# 7.3 Exceptions

- AddressInvalidException Thrown if a given address is invalid. e.g. does not exist.
- AppMessageNotAllowedException Thrown if one part of an application is not allowed to send a certain message or command to another part of the application. Contains the reason why the message is not allowed.
- EmailAddressAlreadyExistsException Thrown if a new given email address already exists in the system.
- EmailAddressInvalidException Thrown if a given email address is invalid. e.g. badly formatted
- EndUserAlreadyExistsException Thrown if an entered end user already exists in the system.
- PaymentInformationInvalidException Thrown if given payment information is invalid.
- PhoneNumberInvalidException Thrown if a given phone number is invalid. e.g. invalid country code
- UnknownCommandException Thrown if a command is sent by an application that is not recognised.
- UnknownConfigurationParameterException Thrown if an application tries to set a configuration parameter on a device that is not recognised.
- UserNameAlreadyExistsException Thrown if a given username already exists in the system.

# 7.4 Data types

#### • Application:

Contains information about an application uploaded to the SIoTIP system.

#### • ApplicationInstance:

Attributes: int id, int status, int customerOrganisationID, boolean isGatewayVersion Contains information on an application instance. When an ApplicationInstance is running on the Online Service and on a Gateway, the ApplicationIntances in the ApplicationContainers share the same id.

#### • 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.

#### DeviceStatus:

Attributes: PluggableDeviceID pID, int moteID, int gatewayID, DateTime from, DateTime to, string status, string details

Contains status details for a specific gateway/mote/pluggable device.

#### • Echo:

Is a reply to an echo request (ping). Contains an Identifier and Sequence Number that matches the request that caused the Echo reply.

#### • GatewayInfo:

Attributes: int gatewayID, **IPAddress** IPAddress, int infrastructureOwnerID An object containing information on a Gateway.

#### • Invoice:

Contains information about an invoice, such as true provided service or product, the amount to be paid, the date the payment is due, the contact details of the receiver of the bill, etc.

#### • IPAddress:

Represents an Internet Protocol address.

## • 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, **DateTime** dateTime, int triggerID, int status

Contains information about a notification. The communication ChannelID represents the communication channel that will be used to send the notification to the user. The notification TypeID 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:

Attributes: int manufacturerID, int productID, int type, int measurementUnits

Denotes the type of a pluggable device. It specifies the manufacturer, a model identifier (if any) of the pluggable (e.g. heat-o sensor 5000), the type of pluggable device (e.g. temperature sensor or power socket actuator) and the used measurement units (e.g. degrees Celsius, degrees Fahrenheit or decibel). Within MicroPnP this information is also used to retrieve and install the correct drivers (this is outside the scope of the system),

This description is taken from the Discussion Board on Toledo.

#### • Relationship:

Attributes: PluggableDeviceID pID1, PluggableDeviceID pID2, string relationship

Represents a relationship between two devices. e.g. device2 can be used for the purpose that device1 fulfills, device2 is preferred over device1 for a responsibility, etc.

#### • RoomTopology:

Attributes: int roomID, List<**PluggableDeviceInfo>** pluggableDevices, **Relationship** relationships, int buildingID, List<**MoteInfo>** motes, List<**GatewayInfo>** gateways

Represents a room in a topology. Contains the room's devices and the relationships between those devices. Depending on whether this data type is used for customer organisation requests or infrastructure owner requests, the motes and gateways field will be left blank.

#### • Subscription:

Attributes: int id, int status, int customerOrganisationID, int applicationInstanceID, int applicationID

Contains data about a subscription by a customer organisation for an application instance. Data about period/length of the subscription is stored in invoices.

## • User:

Represents a user in the SIoTIP system.

#### • UserRole:

Attributes: int userID, int roleID, int applicationInstanceID, int customerOrganisationID Contains information on the role that a User fulfills for a certain ApplicationInstance.

# 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 New data types 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 occurred 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 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 functionality 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:
- Mote: UC14
- NotificationHandler: UC15

**Decomposition** Figure A.1 shows the components resulting from the decomposition in this run.

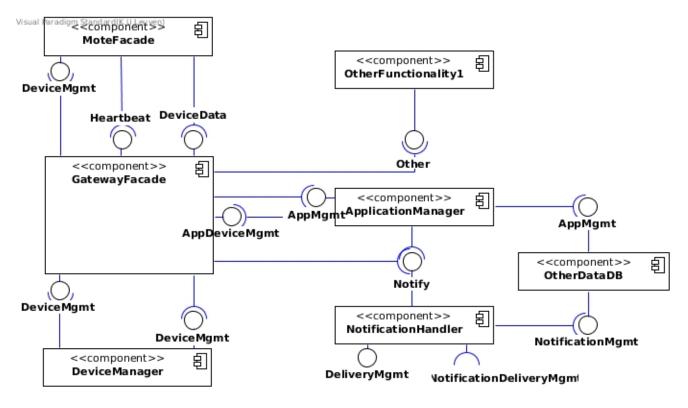


Figure A.1: Component-and-connector diagram of this decomposition.

**Deployment** Figure A.2 shows the allocation of components to physical nodes.

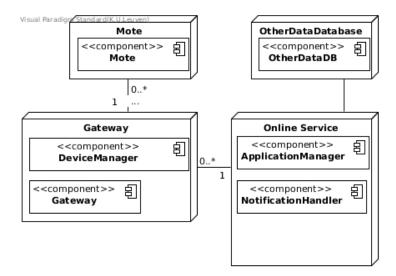


Figure A.2: Deployment diagram of this decomposition.

#### 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

• GWAppInstanceMgmt

#### Database

- NotificationMgmt
- AppMgmt

## GatewayFacade

- Heartbeat
- DeviceData
- DeviceMgmt
- AppDeviceMgmt

#### Mote

• DeviceMgmt

#### NotificationHandler

- Notify
- DeliveryMgmt

## External notification delivery service

• NotificationDeliveryMgmt

## DeviceManager

• DeviceMgmt

## A.3.5 New data types

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:

• 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:

*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

#### Mote

• 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:  $\epsilon$ 

• *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
- $\bullet~UC26\colon$  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:
- 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:

- *PluggableDevice*: 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
PluggableDevice: 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

#### Mote

• DeviceData

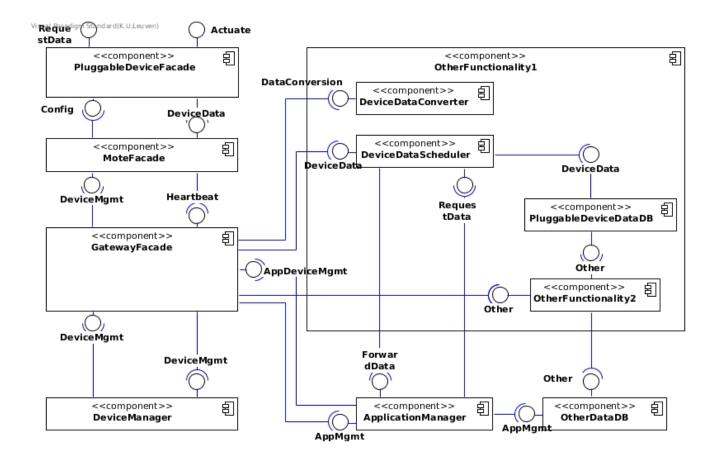


Figure A.3: Component-and-connector diagram of this decomposition.

# PluggableDevice

- Actuate
- Config
- RequestData

#### ${\bf Device Data Converter}$

• DataConversion

### ${\bf Device Data Scheduler}$

- RequestData
- DeviceData

### ${\bf Pluggable Device Data DB}$

• DeviceData

# A.4.5 New data types

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

• DateTime

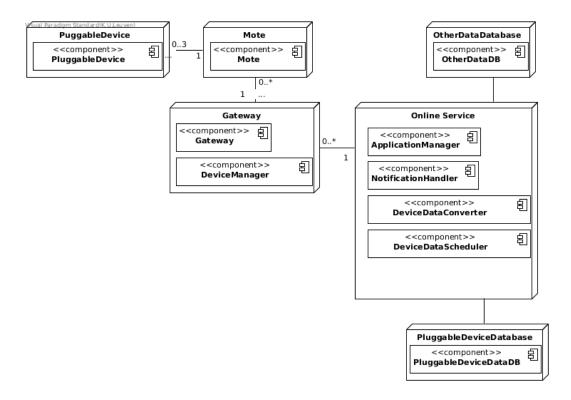


Figure A.4: Deployment diagram of this decomposition.

# 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**

- U2: Easy Installation: a, c, d

#### Mote

- U2: Easy Installation: b, c, d
- *UC04*: Install mote: 1, 2
- *UC05*: Uninstall mote:
- *UC06*: Insert a pluggable device into a mote: 2
- *UC07*: Remove a pluggable device from its mote:

#### 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
- $\bullet$  *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
- $\bullet$  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

### Pluggable Device Data DB

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

## ${\bf Pluggable Device}$

• U2: Easy Installation: d

# ${\bf Device Manager}$

- UC04: Install mote: 4
- $\bullet$  *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,

# ${\bf Device Data Scheduler}$

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

# A.5 Decomposition 3: U2, UC4, UC6, UC9, UC10, UC17, UC19

#### A.5.1 Selected architectural drivers

The non-functional drivers for this decomposition are:

• *U2*: Easy installation

The related functional drivers are:

- *UC4*: Install mote
- UC6: Insert a pluggable device into a mote
- UC9: Configure pluggable device access rights
- *UC10*: Consult and configure topology
- UC17: Activate an application UC19: Subscribe to application

## 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.

Did you ever hear the tragedy of Darth Plagueis The Wise? I thought not. It's not a story the Jedi would Darth Plagueis was a Dark Lord of the Sith, so powerful and so wise he could It's a Sith legend. use the Force to influence the midichlorians to create life... He had such a knowledge of the dark side that The dark side of the Force is a pathway to many he could even keep the ones he cared about from dying. abilities some consider to be unnatural. He became so powerful... the only thing he was afraid of was losing his power, which eventually, of course, he did. Unfortunately, he taught his apprentice everything he knew, then his apprentice killed him in his sleep. Ironic. He could save others from death, but not himself.

**U2: Gateway installation** The gateway should not require any configuration, other than being connected to the local wired or WiFi network, after it is plugged into an electrical socket. An infrastructure owner should be able get the SIoTIP gateway up-and-running (connected) within 10 minutes given that the information (e.g. WiFi SSID and passphrase) is available to the person responsible for the installation.

A connection to the internet is a constraint of the GatewayFacade. After the gateway is connected to the internet (we don't model this), it connects to the gateway (we don't model this?) and registers itself (we model this).

When an infrastructure owner orders a gateway, that gateway is linked to the IO. Gateway was already in the DeviceDB, but it was not linked to anyone. It has a gatewayID => unique identifier gatewayID same like motes.

Important info related to gateways: GatewayID (new class), infrastructureOwnerID, IPAddress, status (active/inactive), location (in topology table) GatewayInfo(int gatewayID, int manufacturerID, int productID, int infrastructureOwnerID, IPAddress ip, int status)

gateway registers with online service: TODO: gateway is added in the infrastructure owners topology as 'unplaced'. it will be visible to the infrastructure owner after the gateway connects and registers with the online service

We cannot link the gateway to an exact location for the infrastructure owner, because he might be managing multiple buildings an IP addresses can be dynamic. If we let the IO choose for which building the gateway is, then this is bad for usability and he has to configure anyways. Also in that case he cannot buy spare gateways, unless he buys spare gateways for every gateway in the building

**U2: Mote installation** Installing a new mote should not require more configuration than adding it to the topology. Adding new motes, sensors or actuators should not involve more than just starting motes, and plugging devices into motes âĂŞ plug-and-play! Reintroducing a previously known mote, with the same pluggable devices attached to it, should not require any configuration. It is automatically re-added on

its last known location on the topology. The attached pluggable devices are automatically initialised and configured with their last known configuration and access rights.

Thing that need to happen automatically: \*) mote should find the gateway (mote sends a broadcast message->ReceiveBroadcast) => this is done automatically? see remarks of the use case \*) gateway should register the mote (DeviceManager update, store entry in DB) \*) on reintroduction of motes: DeviceManager notices this, makes the gateway send a message to online service to reuse some old topology

- **U2:** Pluggable device installation Adding new sensors or actuators should require no further customer actions besides plugging it into the mote. Configurable sensors and actuators should have a working default configuration. Pluggable devices added to an already known mote are automatically added in the right location on the topology. Making (initialised) sensors and actuators available to customer organisations and applications should not require more effort than configuring access rights (cf. UC9).
- ) After devices are plugged in: connect to mote, set up default configurations \*) if the mote is already known, the device is added to the right location on the topology \*) need something for configuration of access rights, can only happen for initialised devices
- \*) for reactivating last configurations: just set status to active and don't change configuration field, it will still be the same as in the past alternative: current\_configuration and last\_configuration in DB alternative: store all configurations on Gateway -> but it has bad resources alternative: store all versions on DeviceDB -> but lots of useless data then = extra work for db
- \*) Pluggable devices added to an already known mote are automatically added in "the right location" on the topology. what exactly is a location? => when a pluggable device is connected to a new mote, the pluggable device gets the location of the mote by default
- **U2: Easy applications** Applications should work out of the box if the required sensors and actuators are available. Only when mandatory end-user roles must be assigned, additional explicit configuration actions are required from a customer organisation (cf. UC17, UC19).
- ) if there is a subsription and new hardware is plugged in: need something to check if some application can be activated now => see UC6: checkApplicationsForActivationForInfrastructureOwner \*) need something to assign user roles to users during UC19

#### 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.

- AccessRightsManager: U2, UC9
- ApplicationContainerManager: U2, UC17
- CustomerOrganisationClient: U2, UC19
- CustomerOrganisationFacade: U2, UC19
- DeviceDB: U2, UC4, UC6
- InfrastructureOwnerClient: U2, UC9
- InfrastructureOwnerFacade: U2, UC9
- InfrastructureOwnerManager: U2, UC9
- InvoiceManager: U2, UC17
- TopologyManager: U2, UC4, UC6, UC17
- SubscriptionManager: U2, UC19
- UserManager: U2, UC17, UC19

#### 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.

### AccessRightsManager

 $\bullet \ AccessRightsMgmt \\$ 

## ApplicationContainerManager

• AppMgmt

# ${\bf Application Manager}$

- FrontEndAppMgmt
- IOAppMgmt

## CustomerOrganisationFacade

• SubscriptionMgmt

# $\mathbf{DeviceDB}$

- AccessRightsMgmt
- DeviceMgmt
- TopologyMgmt
- IODeviceMgmt

## ${\bf Infrastructure Owner Facade}$

 $\bullet$  AccessRights

### In frastructure Owner Manager

• IOMgmt

# ${\bf Invoice Manager}$

• InvoiceMgmt

#### OtherDataDB

- $\bullet$  InvoiceMgmt
- IOMgmt
- SubscriptionMgmt
- UserRoleMgmt

# ${\bf Topology Manager}$

• TopologyMgmt

# ${\bf Subscription Manager}$

 $\bullet \;\; Subscription Mgmt$ 

### UserManager

• RoleMgmt

# A.5.5 New data types

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

- Application
- $\bullet$  IPAddress
- Relationship
- $\bullet$  RoomTopology
- User
- $\bullet$  UserRole

# A.6 Decomposition 4: Av2, UC12, UC25, UC26, UC27 (application execution subsystem)

### A.6.1 Selected architectural drivers

The non-functional drivers for this decomposition are:

• Av2: Application failure

The related functional drivers are:

- UC12: Perform actuation command
- UC13: Configure pluggable device
- UC25: Access topology and available devices
- UC24: Consult historical data
- UC26: Send application command or message to external front-end
- UC27: Receive application command or message from external front-end

**Rationale** At this point the remaining drivers were Av1, Av2, and P1, which all had medium priority. We chose decompositions 4, 5, and 6 based on the priorities of the use cases that are related to the quality attributes.

The related use cases from now on are the ones that would use components that are going to be changed in the decomposition.

## 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.

If you read this, you are cool!

RATIONALE: ApplicationContainerManager and DeviceCommandConstructor need to be reconfigured ivm some used interfaces

**Av2: Detection of failures** The system is able to autonomously detect failures of its individual application execution components, failing applications, and failing application containers.

Upon detection, a SIoTIP system administrator is notified.

The failure of an internal application execution component is detected within 30 seconds. Detection of failed hardware or crashed software happens within 5 seconds. SIoTIP system administrators are notified within 1 minute.

To detect failures, we made use of the Container pattern. The application execution subsystem is composed of:

- ApplicationContainer
- ApplicationContainerMonitor
- ApplicationContainerManager
- ApplicationExecutionSubsystemMonitor

The ApplicationContainers are deployed in groups on different nodes.

ApplicationContainer: is a container/sandbox that has 1 running application instance ApplicationContainerMonitor: monitors the ApplicationContainer instances

To detect failing applications, ApplicationContainer and ApplicationContainerMonitor ApplicationContainer -> ApplicationContainerMonitor: void applicationCrashed(id applicationInstanceID)

To detect failling application containers, ApplicationContainerMonitor ApplicationContainerMonitor -> ApplicationContainer: Echo ping() -> we say container has crashed/failed when the following has no response:

To detect failures of individual application execution components, This means that one of ApplicationContainer, ApplicationContainerMonitor, ApplicationContainerManager crashed. If the ApplicationContainer failed, then the ApplicationContainerMonitor would detect this. other two components failed, the ApplicationExecutionSubsystemMonitor is put in place to detect this. These components will be deployed on the Online Service and on gateways. Since gateways are weaker machines than the ones on the Online Service, the ApplicationContainer can be configured differently for The ApplicationContainers will then have stricter limits on resources used of the node they are working on. DeviceCommandConstructor and ApplicationContainerManager interfaces need to begin re-routed depening on whether they are in a Gateway or on the Online Service.

Av2: Resolution of application failures and application execution component failures In case of application crash, the system autonomously restarts failed applications. If part of an application fails, the remaining parts remain operational, possibly in a degraded mode (graceful degradation).

After 3 failed restarts the application is suspended, and the application developer and customer organisation are notified within 5 minutes.

In case of failure of application execution components or an application container, a system administrator is notified.

SIoTIP system administrators are notified within 1 minute.

When an application instance fails, the ApplicationContainerMonitor detects this and sends a command to the ApplicationContainerManager to restart the application instance. The ApplicationContainerMonitor keeps track of how many times the application instance has been restarted after a failure. After 3 failed restarts, the monitor send a command to the ApplicationContainerManager to suspend the application instance and send a notification to the application developers of the application and to the affected customer organisaiton. Also, to achieve graceful degradation, the ApplicationContainerManager notifies other parts of the application instance of its suspension.

If one of the components of the application execution subsystem fails, a SIoTIP system administrator is notified.

**Av2:** Failures do not impact other applications or other functionality of the system This does not affect other applications that are executing on the Online service or SIoTIP gateway. This does not affect the availability of other functionality of the system, such as the dashboards.

Applications fail independently: they are executed within their own container to avoid application crashes to affect other applications.

Each ApplicationContainer contains one application instance. If an application fails, then this will be handled by the application execution subsystem so this does not affect any other application or other functionality of the system. The ApplicationContainers are constructed such that failures of applications do not affect the containers. The ApplicationContainers are to be deployed on different nodes alone or grouped with other containers. Write something here.

# 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: ApplicationClientComponent: ApplicationContainer

Component: ApplicationContainerManagerComponent: ApplicationContainerMonitor

• Component: ApplicationExecutionSubsystemMonitor

• Component: ApplicationFacade

Component: ApplicationManagementLogicComponent: DeviceCommandConstructor

#### 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.

### **ApplicationContainer**

- AppMessages
- AppInstanceMgmt
- AppMonitoring

#### ApplicationContainerManager

- AppMessages
- AppInstanceMgmt
- DeviceCommands
- DeviceData

### **ApplicationContainerMonitor**

- Monitoring
- AppStatus

### ApplicationFacade

- DeviceData
- DeviceCommands

# ApplicationManagementLogic

- DeviceData
- DeviceCommands
- FrontEndAppRequests
- GWAppInstanceMgmt
- AppMessages

#### ${\bf Device Command Constructor}$

 $\bullet$  Commands

#### DeviceManager

• DeviceCommands

#### DeviceDB

• AppDeviceMgmt

### A.6.5 New data types

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

• Echo the response to a ping message

# A.7 Decomposition 5: Av1 (Gateway - Online Service communication subsystem)

### A.7.1 Selected architectural drivers

The non-functional drivers for this decomposition are:

• Av1: Communication between SIoTIP gateway and Online Service

# 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.

Av1: New Gateway responsibilities The SIoTIP gateway is able to autonomously detect failures of its individual internal communication components.

The Online Service should acknowledge each message sent by the SIoTIP gateway so that the gateway can detect failures.

If an internal SIoTIP gateway component fails, the gateway first tries to restart the affected component. If the failure persists, the SIoTIP gateway reboots itself entirely. Note that the SIoTIP gateway, due to the occurred failure, cannot contact a system administrator itself.

If (an internal communication component of) the Online Service or the communication channel has failed, the SIoTIP gateway will temporarily store all incoming pluggable data and any issued application commands internally.

If the Online Service becomes unreachable, application parts running locally on the SIoTIP gateway continue to operate normally.

The SIoTIP gateway will start synchronising with the Online service within 1 minute after the communication channel becomes available.

The SIoTIP gateway can store at least 3 days of pluggable data before old data has to be overwritten.

OnlineServiceBroker: Isolates communication-related concerns between Gateways and the Online Service along with GatewayBroker on the Online Service. Forwards requests from one party to the other and transmits results and possible exceptions.

OnlineServiceBrokerMonitor: Monitors the communication component on Gateways. If the communication component fails, the monitor tries to restart it. If the failure persists, the gateway reboots itself entirely.

In OnlineServiceBroker: BrokerLogic: Handles all functionality related to communication. Temporarily stores all pluggable data and issued application commands until they can be deleted (= until an acknowledgement has been received for the request by the Online Service). Can store at least 3 days of pluggable data before old data has to be overwritten. OnlineServiceMonitor: Monitors the Gateway's connectivity to the Online Service. If the Online Service or the communication channel has failed, all requests to the Online Service will be stopped and stored in the RequestStore. An explicit command for this is not necessary, because the requests in the RequestStore will not be deleted, since no acknowledgements are received anymore from the Online Service. After the monitor detects that a connection to the Online Service is possible again, it makes the gateway start synchronising again. When the Online Service is unreachable, application parts running locally on the SIoTIP gateway continue to operate normally.

INTERFACES: OnlineServiceBroker: HAS ALL INTERFACES PROVIDED BY COMPONENTS INSIDE OF IT

 $interface \ Communication Component Monitoring \ used \ by \ Online Service Broker Monitor: \ boolean \ check() \ void \ restart()$ 

BrokerLogic: interface OSCommunication used by Online Service: void acknowledgement(int requestID) void send(...) void receive(...)

 $interface\ OSMonitoring\ used\ by\ Online Service Monitor:\ Echo\ ping Online Service ()\ void\ synchronise With Online Service ()$ 

RequestStore: All interfaces that are going to the online service now go to the broker All requests come in here and are stored The requests are then forwarded to BrokerLogic containing a unique requestID

Interface Communication used by BrokerLogic: void deleteRequest(int requestID) List<Object>getNewRequestsForSynchronisation()

OnlineServiceMonitor: Online service keeps track of connection to Online Service. If the Online Service becomes unreachable (= does not send ACKs anymore), then start pinging the Online Service

Interface OSUpdates used by BrokerLogic: void onlineServiceUpdate()

Av1: New Online Service responsibilities The Online Service is able to autonomously detect failures of its individual internal communication components.

The Online Service is able to detect that a SIoTIP gateway is not sending data anymore based on the expected synchronisation interval.

The Online Service notifies the infrastructure manager and a SIoTIP system administrator when the outage of a SIoTIP gateway is detected.

The failure of an internal SIoTIP Online Service component is detected within 30 seconds.

The detection time for a failed SIoTIP gateway or channel depends on the transmission rate of the gateway. An outage is de

ned as 3 consecutive expected synchronisations that do not arrive within 1 minute of their expected arrival time.

The infrastructure owner is noti

ed within 5 minutes after the detection of an outage of their gateway.

A SIoTIP system administrator should be noti

ed within 1 minute after the detection of a simultaneous outage of more than 1% of the registered gateways.

GatewayBroker: Isolates communication-related concerns between the Online Service Gateways and along with OnlineServiceBroker on Gateways. Forwards requests from one party to the other and transmits results and possible exceptions.

Sends acknowledgements for all messages sent by Gateways so that they can detect failures. GatewayBrokerMonitor: Monitor the communication component for communication with gateways on the Online Service.

In GatewayBroker: BrokerLogic: Handles all functionality related to communication. GatewayMonitor: Monitors the connectivity status all gateways. Can detect that a gateway is not sending data anymore based on the expected synchronisation interval. If 3 consecutive expected synchronisations do not arrive within 1 minute of their expected arrival time, this is detected as a gateway outage. When outages of gateways are detected, the infrastructure owners that own the gateways and a SIoTIP system administrator are notified. When the connectivity status change of a Gateway is detected, this is saved in the DeviceDB.

INTERFACES: GatewayBroker: HAS ALL INTERFACES PROVIDED BY COMPONENTS INSIDE OF IT interface CommunicationComponentMonitoring used by GatewayBrokerMonitor: boolean check()

BrokerLogic: ALL INTERFACES FROM OS TO GATEWAY ARE NOW TO THIS COMPONENT interface GWCommunication used by Gateway/BrokerLogic:

GatewayMonitor:

interface GatewayUpdates used by BrokerLogic: void gatewayUpdate(int gatewayID, DateTime time)

NotificationHandler: Interface Notify used by GatewayMonitor

DeviceDB: Interface DeviceMgmt used by GatewayMonitor: void setGatewayUnreachable() void getPercentageOfUnreachableGateways()

In BROKER, many clients can make remote method invocations on specific remote component objects hosted by a server. Clients thus communicate with the server objects in a many-to-one fashion, and their functional interfaces are often statically typed. The remote method invocation style of communication provided by B ROKER is best suited for systems that try to hide the presence of the network.

MESSAGING relaxes this coupling and typing: clients send dynamically typed messages to specific remote services that reside at communication endpoints, not (necessarily) to specific methods. M ESSAGING thus enables many-to-one communication without statically pre- defining the interface dependencies of clients to services. Distribution Infrastructure

PUBLISHER-SUBSCRIBER decouples an applicationâĂŹs components even more: they can exchange events in a one-to-many manner with- out knowing one anotherâĂŹs identity explicitly, and without having to make a request each time new events are available. P UBLISHER - S UBSCRIBER middleware is therefore responsible for tracking which subscribers receive specific events sent asynchronously by publishers. Subscribers react when receiving an event by performing some action, but publishers do not directly initiate the execution of a specific method on the subscribers.

BROKER makes invocations on remote component objects look and act as much as possible like invocations on component objects in the same address space as their clients. MESSAGING and PUBLISHER-SUBSCRIBER are most appropriate for integration scenarios in which multiple, independently developed and self-contained services or applications must collaborate and form a coherent software system.

A message channel does not come without cost, however, since it needs memory, networking resources, and persistent storage to support GUARANTEED DELIVERY [HoWo03]. Developers must therefore plan and configure the number and types of message channels explicitly and thoughtfully to ensure the desired quality of service in a given system deployment. A well-designed set of message channels forms a MESSAGE BUS [HoWo03] that acts like a messaging API for the clients and services in the distributed system.

Employed tactics and patterns: broker, heartbeat, ping/echo

DEPLOYMENT RATIONALE: OnlineServiceBroker OnlineServiceBrokerMonitor

GatewayBroker GatewayBrokerMonitor

For both the Online Service and Gateways, the components used for communication (OnlineServiceBroker, GatewayBroker) are to be deployed on different nodes than their monitoring components GatewayBrokerMonitor). (OnlineServiceBrokerMonitor, Otherwise, if the node of a communication component fails, its monitoring component would also fail and thus nothing would be detected.

Alternative for monitoring of gateways: Gateway updated come to GatewayMonitor We could make GatewayMonitor ping all the gateways However, this would increase traffic on the network Alternative for communication: Messaging, Publishher Subscriber

### 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.

• BrokerLogic: Av1

• OnlineServiceBroker: Av1

• OnlineServiceBrokerMonitor: Av1

• GatewayBroker: Av1

• GatewayBrokerMonitor: Av1

RequestStore: Av1GatewayMonitor: Av1

• OnlineServiceMonitor: Av1

## 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.

#### BrokerLogic

- Communication
- OSCommunication
- GWCommunication
- OSMonitoring

### OnlineServiceBroker

 $\bullet \ \ Communication Component Monitoring$ 

# GatewayBroker

• CommunicationComponentMonitoring

# ${\bf RequestStore}$

 $\bullet$  Communication

# ${\bf Gateway Monitor}$

• GatewayUpdates

### ${\bf Online Service Monitor}$

• OSMonitoring

# A.7.5 New data types

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

• None

# A.8 Decomposition 6: P1, UC1, UC2, UC3, UC5, UC7, UC8, UC16, UC20 (Elements/Subsystem to decompose/expand)

## A.8.1 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
- *UC2*: Register an end-user
- UC3: Unregister an end-user
- UC5: Uninstall mote
- UC7: Remove a pluggable device from its mote
- UC8: Initialise a pluggable device
- *UC10*: Consult and configure the topology
- UC16: Consult notification message
- *UC20*: Unsubscribe from application

# 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.

https://www.alertra.com/blog/2010/improve-availability-performance-using-database-replication https://serverfault.com/questare-the-performance-implications-for-using-sql-server-replication " Ik dacht aan misschien de meeste van de componenten gewoon duplicaten maar dan is de DB wel u bottleneck en dan moet ge zo'n systeem van distributed systems gebruiken om ook de databases juist te kunnen replicaten. Maar daarnaast echt geen idee ivm performance. Ge kunt wel bullshit van de "tactics" in u rationale zetten zoals the developers have to "increase computation efficiency" and "reduce computational overhead". Manage event rate en scheduling policy kunt ge misschien wel gebruiken om er voor te zorgen dat bepaalde taken gebeuren op een moment dat de load op de online service wat lager is maar ik weet Ni echt welke taken "

- \*) Most components can be duplicated for load balancing \*) Related DB's should use a DBMS that can be scaled horizontally. \*) DB calls caused by IO vs CO have been mostly split up in DeviceDB and OtherDataDB, so those 2 groups of users don't influence each other too much
- **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.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.

ApplicationProviderClient: UC16
 ApplicationProviderFacade: UC16
 RegisteredUserFacade: UC16
 SystemAdministratorFacade: UC16

UnregisteredUserClient: UC1UnregisteredUserFacade: UC1

#### 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.

## ApplicationProviderFacade

Notifications

#### ${\bf Customer Organisation Facade}$

- Notifications
- UserMgmt

#### InfrastructureOwnerFacade

- Notifications
- TopologyMgmt

#### NotificationHandler

Notifications

## RegisteredUserFacade

Notifications

#### SystemAdministratorFacade

Notifications

#### UnregisteredUserFacade

• Registration

#### UserManager

• UserMgmt

# A.8.5 New data types

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

- Invoice
- $\bullet \ Email Address Already Exists Exception$
- $\bullet \ Email Address Invalid Exception$
- $\bullet \ \ End User Alrea dy Exists Exception$
- $\bullet \ \ Payment Information Invalid Exception$
- $\bullet \ \ Phone Number Invalid Exception$
- $\bullet \ \ UserNameAlreadyExistsException$

# A.9 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.9.1 Selected architectural drivers

The functional drivers are:

*UC28*: Log in *UC29*: Log out

Sessions are used to check whether or not a user is currently logged in.

On successful login, a new session is created and stored in the SessionDB with a unquie sessionID which is returned to the front end. The client then sends this sessionID in future requests to represent that the user is logged in.

A database is used for checking the sessions, so that replication can be done on components that handle front end requests. If the session was stored in files, this would add extra overhead to keep all replica's that store session files consistent. It is easier to do this using a database. Also, a separate database is used to not add a lot of extra requests for sessions to another database.

# A.9.2 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.

• SessionDB: UC28, UC29

• AuthenticationManager: UC28, UC29

## A.9.3 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.

#### **ApplicationProviderFacade**

Authentication

# AuthenticationManager

• Authentication

#### ${\bf Customer Organisation Facade}$

• Authentication

#### InfrastructureOwnerFacade

Authentication

### NotificationHandler

• Authentication

# ${\bf Other Data DB}$

ullet Authentication

# ${\bf Registered User Facade}$

• Authentication

# ${\bf Session DB}$

• Authentication

# ${\bf System Administrator Facade}$

ullet Authentication

# A.10 Decomposition 9: UC21 (Elements/Subsystem to decompose/expand)

### A.10.1 Selected architectural drivers

The functional drivers are:

• UC21: Send invoice

# A.10.2 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.

• ThirdPartyInvoicingService: UC21

#### A.10.3 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.

### ThirdPartyInvoicingService

• InvoiceDeliveryMgmt

### Invoice Manager

• DeliveryMgmt