

# Shared Internet-of-Things Infrastructure Platform (SIoTIP)

Part 2

Initial architecture

ACADEMIC YEAR 2018–2019

H09B5B: SOFTWARE ARCHITECTUUR

H07Z9B: SOFTWARE ARCHITECTURE

# Contents

<b>1</b>	<b>Client-server view (UML Component Diagram)</b>	<b>4</b>
<b>2</b>	<b>Decompositions (UML Component Diagram)</b>	<b>6</b>
<b>3</b>	<b>Deployment view (UML Deployment Diagram)</b>	<b>8</b>
<b>4</b>	<b>Scenarios (UML Sequence Diagram)</b>	<b>10</b>
<b>5</b>	<b>Element catalog</b>	<b>23</b>
5.1	Components . . . . .	23
5.1.1	Application . . . . .	23
5.1.2	ApplicationCommandPropagator . . . . .	23
5.1.3	ApplicationCommandRouter . . . . .	23
5.1.4	ApplicationContainer . . . . .	23
5.1.5	ApplicationDeveloperClient . . . . .	24
5.1.6	ApplicationDeveloperDashboard . . . . .	24
5.1.7	ApplicationHandler . . . . .	24
5.1.8	ApplicationHeartbeatMonitor . . . . .	24
5.1.9	ApplicationInstanceInfoDB . . . . .	25
5.1.10	ApplicationManager . . . . .	25
5.1.11	ApplicationScheduler . . . . .	25
5.1.12	ApplicationStarter . . . . .	25
5.1.13	AppTester . . . . .	26
5.1.14	AuthenticationHandler . . . . .	26
5.1.15	AvailabilityMonitor . . . . .	26
5.1.16	CommunicationHeartbeatMonitor . . . . .	26
5.1.17	CustomerOrganisationClient . . . . .	26
5.1.18	CustomerOrganisationDashboard . . . . .	27
5.1.19	DataHandler . . . . .	27
5.1.20	DataReceiver . . . . .	27
5.1.21	DBRequestHandler . . . . .	27
5.1.22	DeviceDataRouter . . . . .	28
5.1.23	DeviceInitCache . . . . .	28
5.1.24	DeviceStatusMonitor . . . . .	28
5.1.25	Gateway . . . . .	28
5.1.26	GatewayOS . . . . .	28
5.1.27	GWApplication . . . . .	29
5.1.28	GWApplicationContainer . . . . .	29
5.1.29	GWApplicationHeartbeatMonitor . . . . .	29
5.1.30	GWApplicationScheduler . . . . .	29
5.1.31	GWCommandRouter . . . . .	30
5.1.32	GWFailureHandler . . . . .	30
5.1.33	GWTempDataDB . . . . .	30
5.1.34	InfrastructureOwnerClient . . . . .	30
5.1.35	InfrastructureOwnerDashboard . . . . .	30
5.1.36	InvoiceManager . . . . .	31
5.1.37	MobileApp . . . . .	31
5.1.38	MobileAppCommunication . . . . .	31
5.1.39	Mote . . . . .	31
5.1.40	NotificationHandler . . . . .	31
5.1.41	OSSensorDataDB . . . . .	32
5.1.42	OSWithGWCommunication . . . . .	32
5.1.43	OtherDataDB . . . . .	32

5.1.44	SensorDataDB	32
5.1.45	SensorDataLoadBalancer	33
5.1.46	SensorDataScheduler	33
5.1.47	SessionDB	33
5.1.48	SIoTIP system	33
5.1.49	SysAdminClient	34
5.1.50	SysAdminDashboard	34
5.1.51	ThirdPartyInvoicingService	34
5.1.52	ThirdPartyNotificationDeliverySystem	34
5.1.53	TopologyDB	34
5.1.54	TopologyLoadBalancer	34
5.2	Interfaces	35
5.2.1	AckMessage	35
5.2.2	Actuate	35
5.2.3	AMReactToFailure	35
5.2.4	AppCalls	35
5.2.5	AppContainerMgmt	36
5.2.6	AppFailure	36
5.2.7	AppInstanceChange	36
5.2.8	AppInstanceMapChange	37
5.2.9	AppInstanceSuspension	37
5.2.10	ApplicationActivation	38
5.2.11	ApplicationInfo	38
5.2.12	ApplicationInstanceLookup	39
5.2.13	AppProvidesAPI	39
5.2.14	CallThroughContainer	40
5.2.15	ChangeSyncTimer	40
5.2.16	CommHeartbeat	40
5.2.17	Config	41
5.2.18	ConfirmDelivery	41
5.2.19	ConsultNotifications	41
5.2.20	CRReactToFailure	41
5.2.21	DBRequestResults	42
5.2.22	DeliverMsgMobileApp	42
5.2.23	DeliverNotification	42
5.2.24	DeviceAvailabilityCheck	42
5.2.25	DeviceAvailabilityMessage	42
5.2.26	DeviceInitCacheMgmt	43
5.2.27	DeviceManagement	43
5.2.28	DHReactToFailure	44
5.2.29	Failure	44
5.2.30	FailureDetection	44
5.2.31	GWAppCalls	44
5.2.32	GWAppChange	45
5.2.33	GWAppProvidesAPI	45
5.2.34	GWAvailability	45
5.2.35	GWTempDataMgmt	46
5.2.36	GWToOSCommand	46
5.2.37	HandleDBRequest	46
5.2.38	Heartbeat	46
5.2.39	IncomingCommand	47
5.2.40	InternalAppActivation	47
5.2.41	IPLookup	47
5.2.42	ManageDevices	47
5.2.43	NotifyRecipient	48
5.2.44	OSToGWCommand	48
5.2.45	OtherDataMgmt	48
5.2.46	RcvMobileAppMsg	51

5.2.47	RcvSensorData . . . . .	52
5.2.48	RebootGW . . . . .	52
5.2.49	Registration . . . . .	52
5.2.50	RequestData . . . . .	53
5.2.51	RouteDeviceData . . . . .	53
5.2.52	ScheduleApplication . . . . .	53
5.2.53	SendCommandToGW . . . . .	54
5.2.54	SendInvoice . . . . .	54
5.2.55	SendMobileAppMsg . . . . .	54
5.2.56	SensorData . . . . .	54
5.2.57	SensorDataMgmt . . . . .	55
5.2.58	SessionMgmt . . . . .	55
5.2.59	Subscription . . . . .	55
5.2.60	TestApplication . . . . .	56
5.2.61	TopologyMgmt . . . . .	56
5.2.62	UpdateMessage . . . . .	57
5.2.63	UploadApplication . . . . .	58
5.2.64	UserAuthentication . . . . .	58
5.2.65	VerifySession . . . . .	59
5.3	Exceptions . . . . .	59
5.4	Data types . . . . .	59

# 1. Client-server view (UML Component Diagram)

## Figures

1.1	cs_context	4
1.2	cs_primary	5

Visual Paradigm Standard(alex(K.U.Leuven))

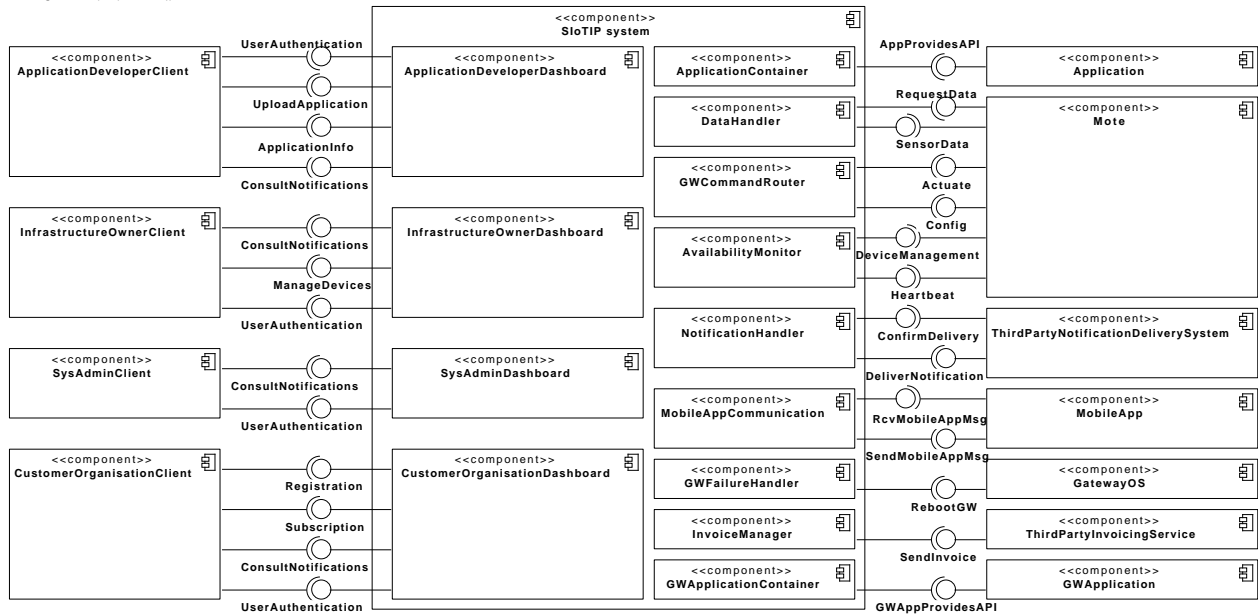


Figure 1.1: Context diagram for the client-server view. Note that the application execution subsystem on the **Gateway** a variation is on the one on the Online Service. This is because of the difference in computing power between the two and thus the difference in solution possibilities.



## 2. Decompositions (UML Component Diagram)

### Figures

2.1	dec_Gateway . . . . .	6
2.2	dec_OSWithGWCommunication . . . . .	7
2.3	dec_OSSensorDataDB . . . . .	7
2.4	dec_ApplicationHandler . . . . .	7

Visual Paradigm Standard(alex(K.U.Leuven))

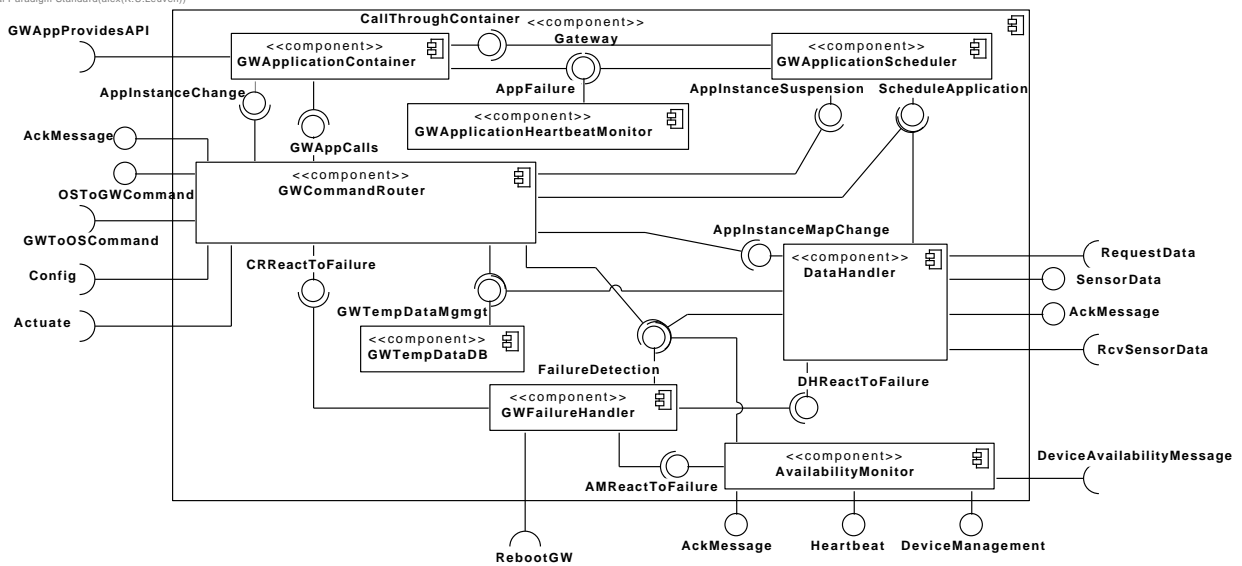


Figure 2.1: Decomposition of Gateway.

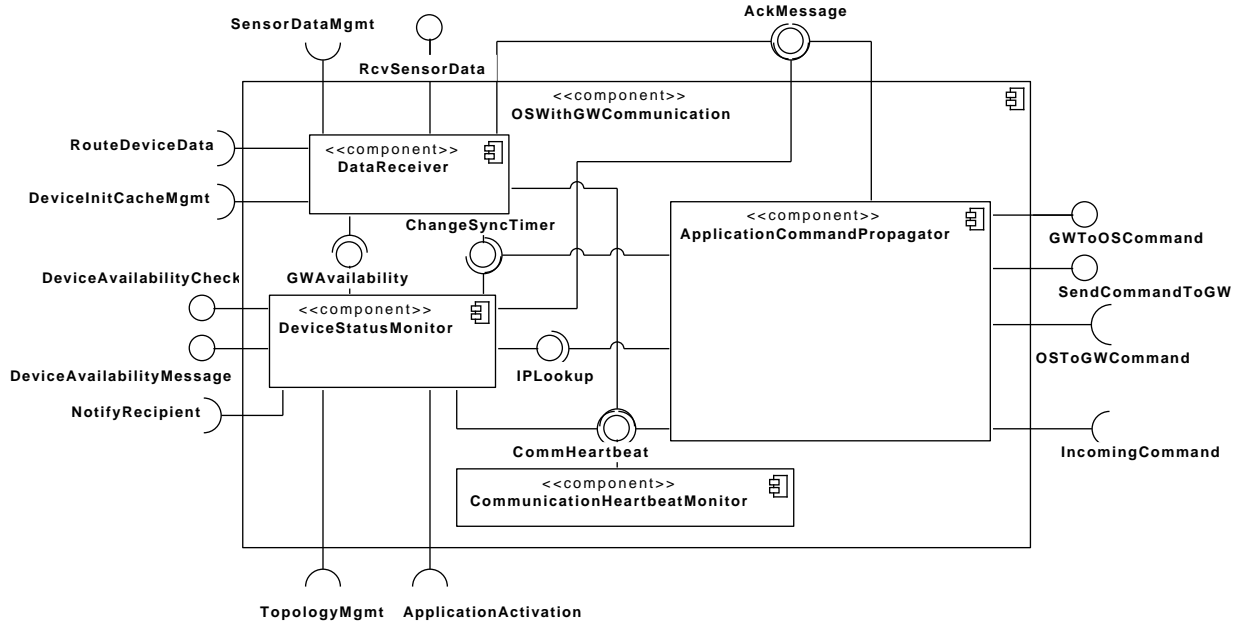


Figure 2.2: Decomposition of OSWithGWCommunication.

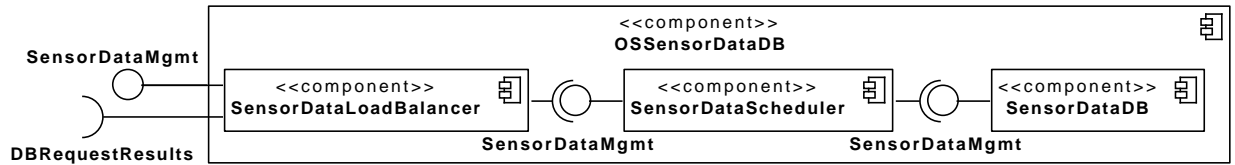


Figure 2.3: Decomposition of OSSensorDataDB.

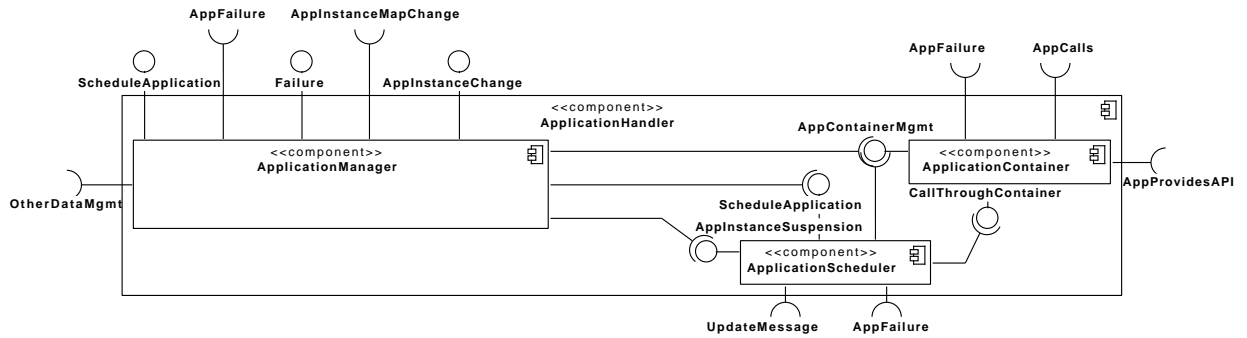


Figure 2.4: Decomposition of ApplicationHandler. Dangling internal interfaces at the stand-in components represent available interfaces which are not connected at all times.



### 3. Deployment view (UML Deployment Diagram)

#### Figures

3.1	depl_context	8
3.2	depl_primary	9

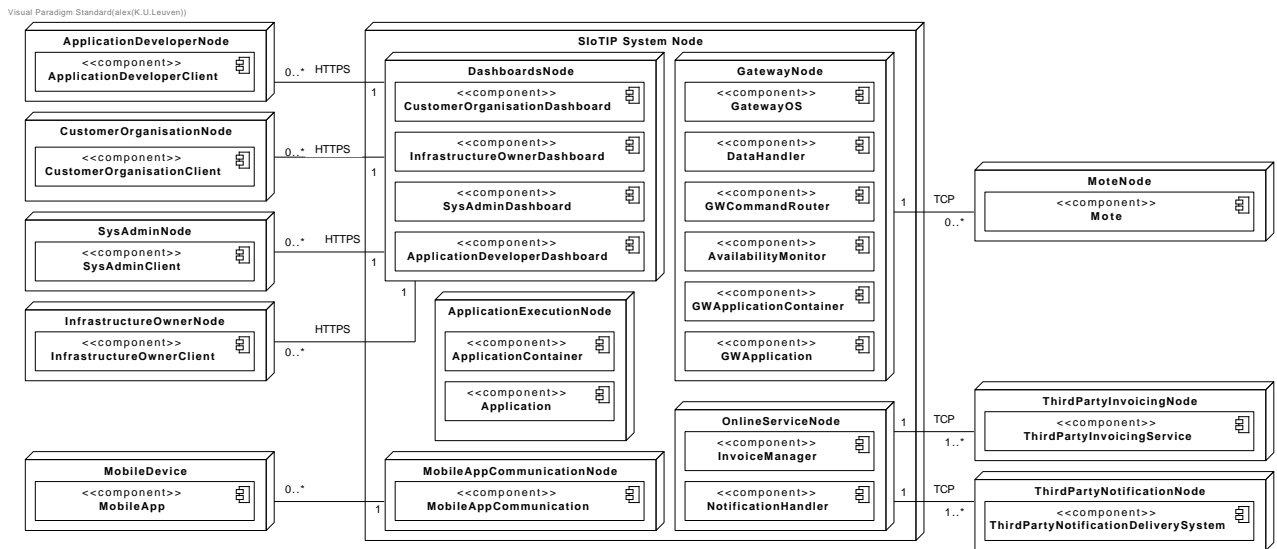


Figure 3.1: Context diagram for the deployment view. Note that TCP is used for communication with the mote since loss of packets (and thus device data) is unacceptable. The various clients connect to the system via a web browser and thus HTTPS is used for those connections. The rest of the connections are done over TCP to avoid packet loss.

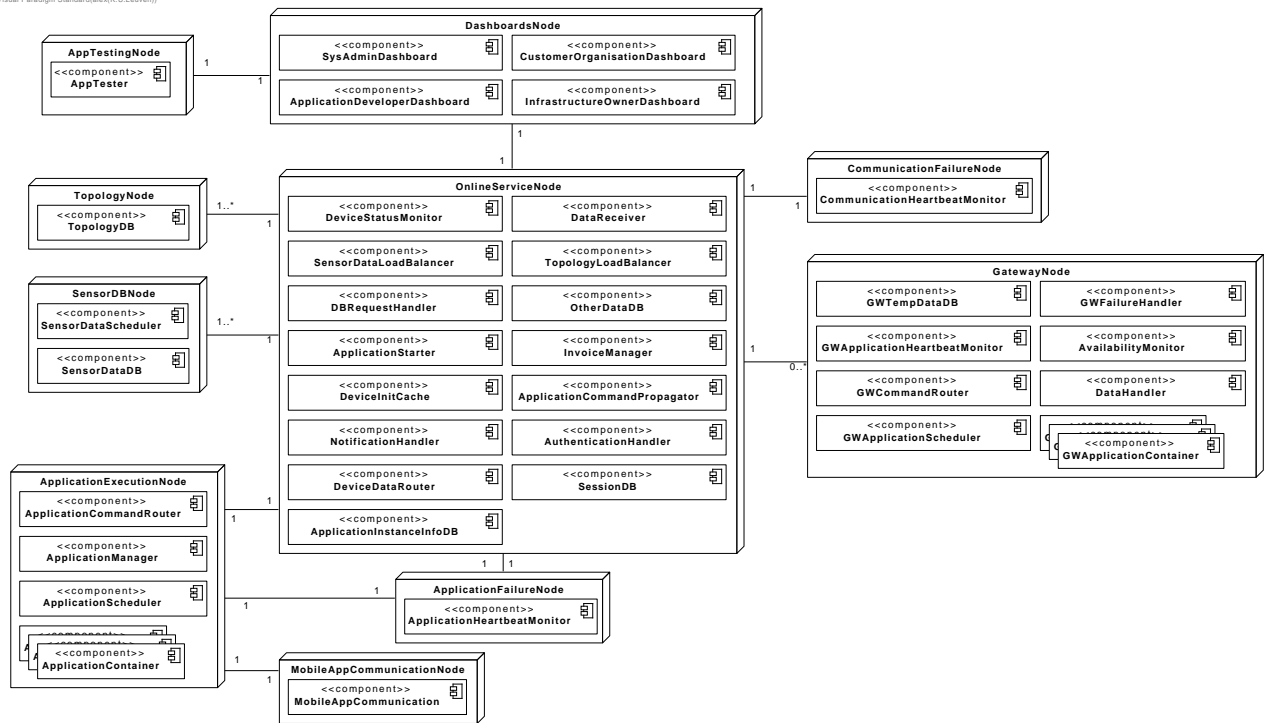


Figure 3.2: Primary diagram for the deployment view.

## 4. Scenarios (UML Sequence Diagram)

### Figures

---

4.1	ApplicationActivation . . . . .	11
4.2	ApplicationActuationCommand . . . . .	11
4.3	ApplicationProcedureCrash . . . . .	11
4.4	ApplicationUpload . . . . .	12
4.5	CommunicationChannelFailure(GW) . . . . .	12
4.6	CommunicationChannelFailure(OS) . . . . .	13
4.7	DeviceFailure . . . . .	13
4.8	KillContainer . . . . .	14
4.9	NewPluggableDevice . . . . .	14
4.10	ProcessSensorData . . . . .	14
4.11	RegisterEndUser . . . . .	15
4.12	RouteCommandToMobileApp . . . . .	15
4.13	RouteCommandToGatewayApplication . . . . .	15
4.14	RouteCommandToOnlineService . . . . .	16
4.15	RouteData . . . . .	16
4.16	RouteToGW . . . . .	16
4.17	RouteToOS . . . . .	16
4.18	ProcessActuationFromMobileApp . . . . .	17
4.19	RouteApplicationCommandOrMessage . . . . .	17
4.20	SendNotification . . . . .	18
4.21	Subscribe . . . . .	19
4.22	SuspendApplication . . . . .	19
4.23	Unsubscribe . . . . .	20
4.24	UpdateApplications . . . . .	20
4.25	UpdateApplications2 . . . . .	21
4.26	UpdateApplications3 . . . . .	22
4.27	WarnApplicationOfDeviceFailure . . . . .	22

---

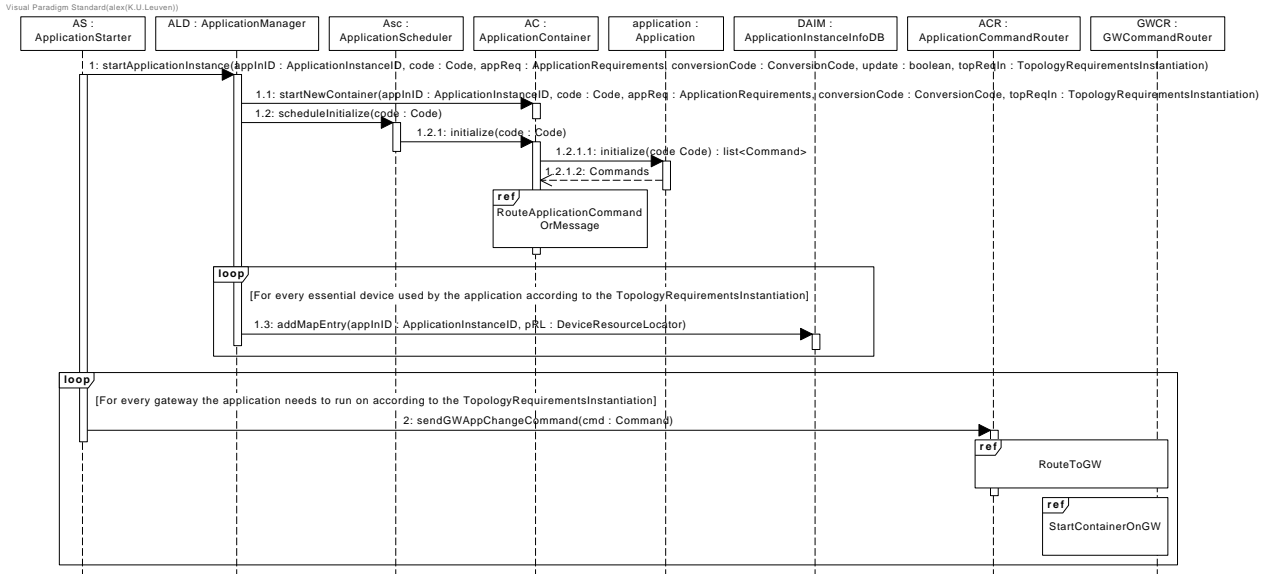


Figure 4.1: Note the addition of entries to a map used for routing sensor data to the correct application instances.

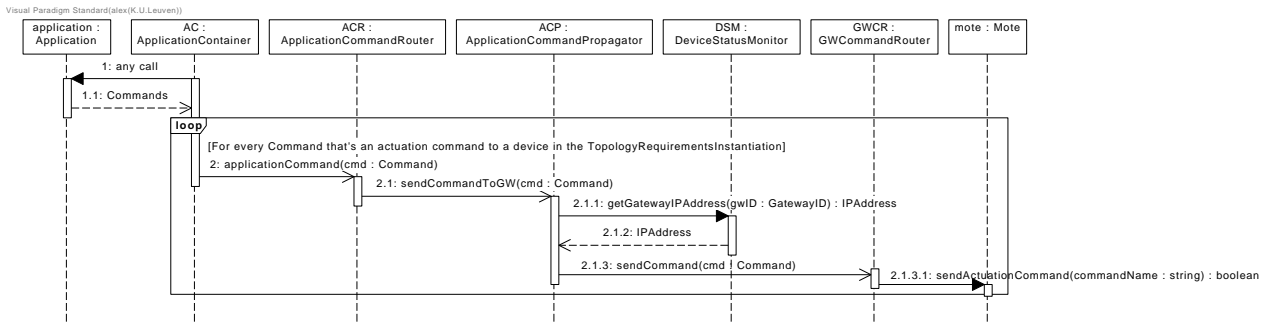


Figure 4.2: Make sure an application command for an actuator makes it to that actuator.

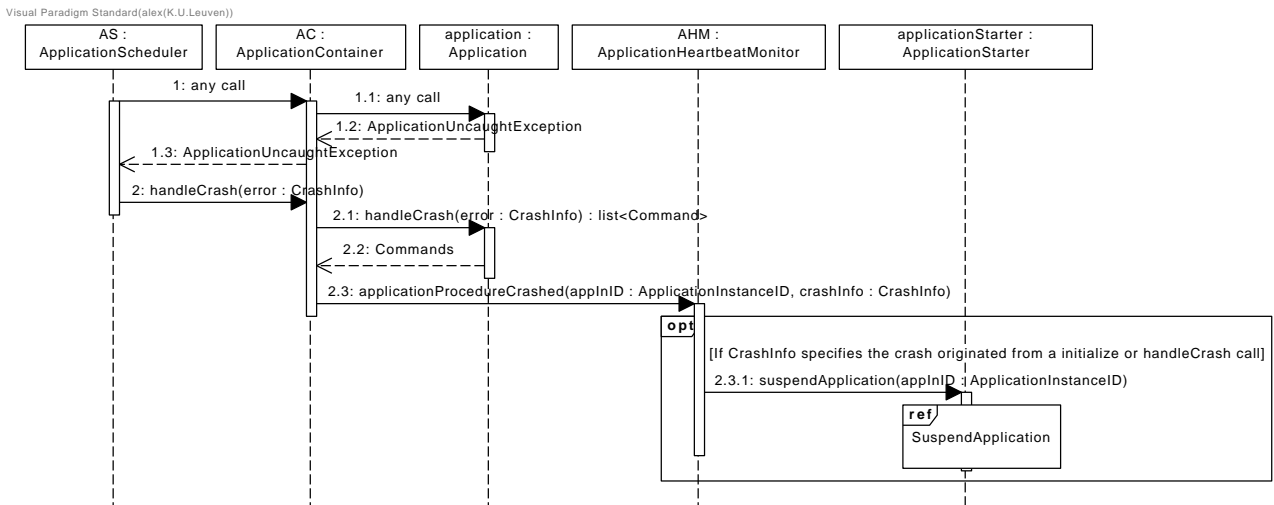


Figure 4.3: ApplicationProcedureCrash

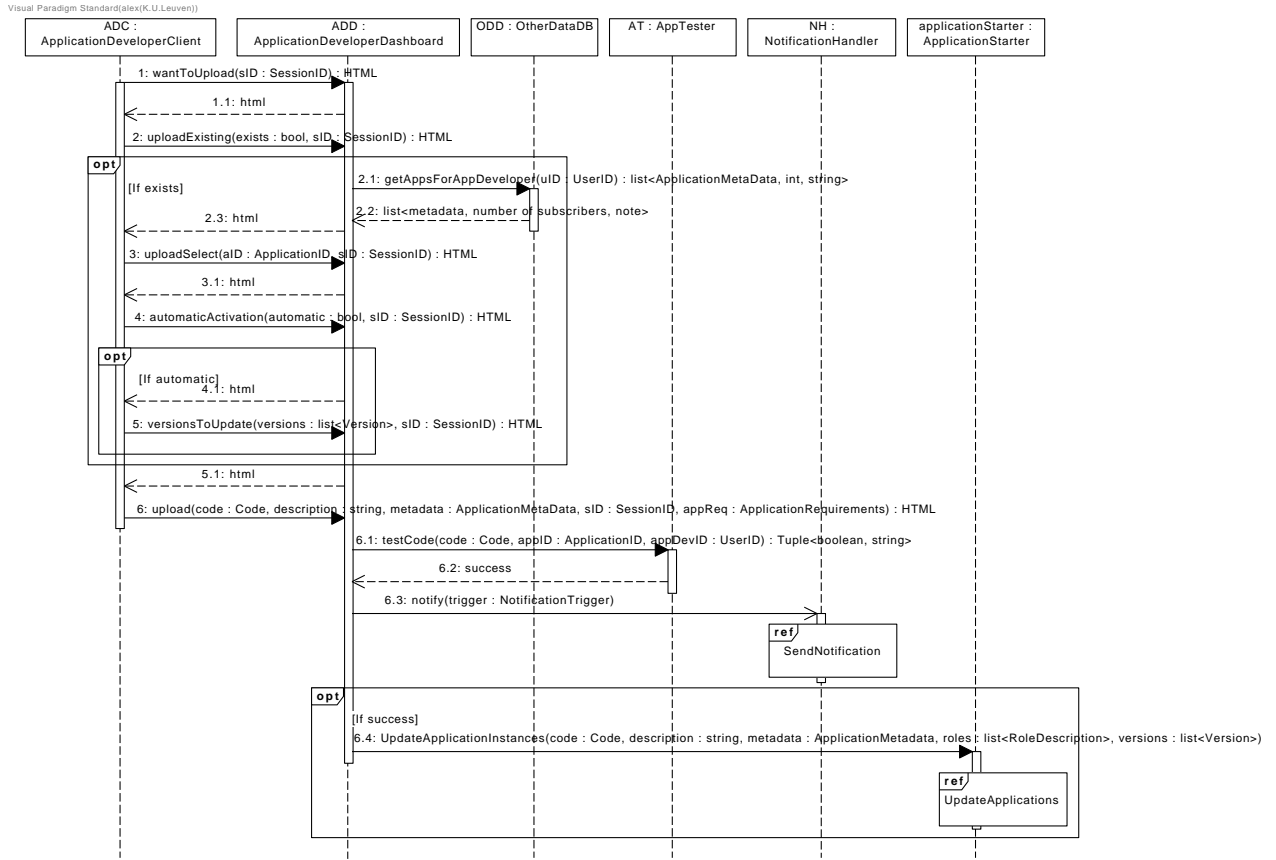


Figure 4.4: An application developer uploads his or her application, indicating in the process whether this is a new version of an existing application and, if so, which previous versions should be automatically updated.

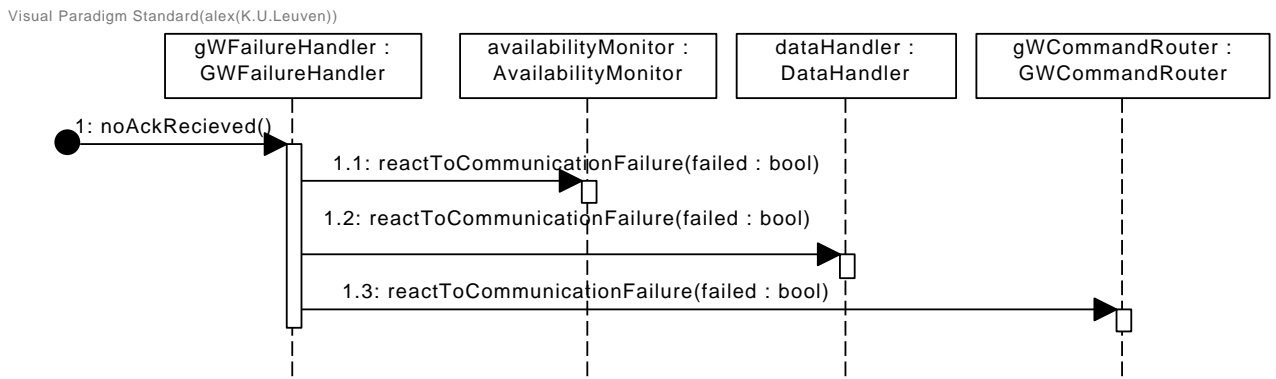


Figure 4.5: CommunicationChannelFailure(GW)

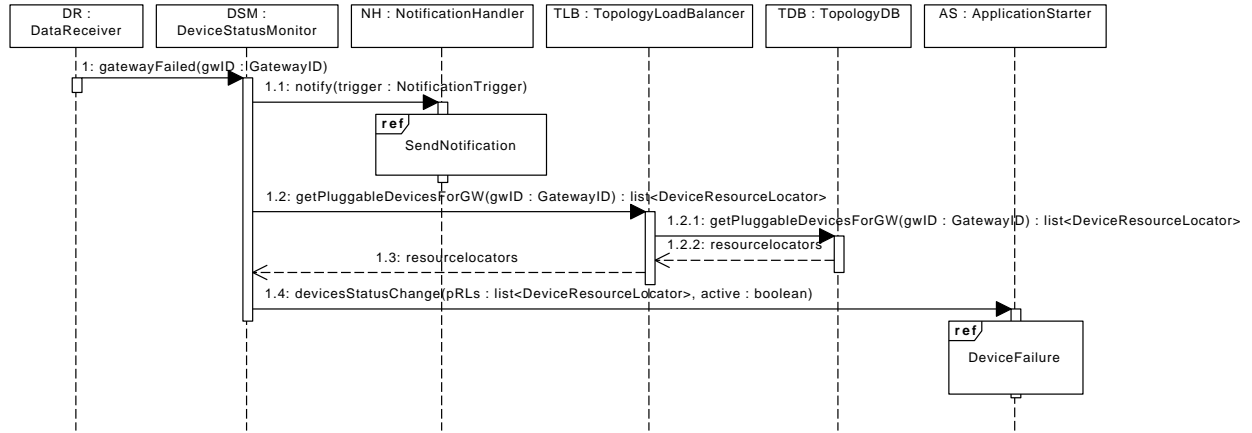


Figure 4.6: CommunicationChannelFailure(OS)

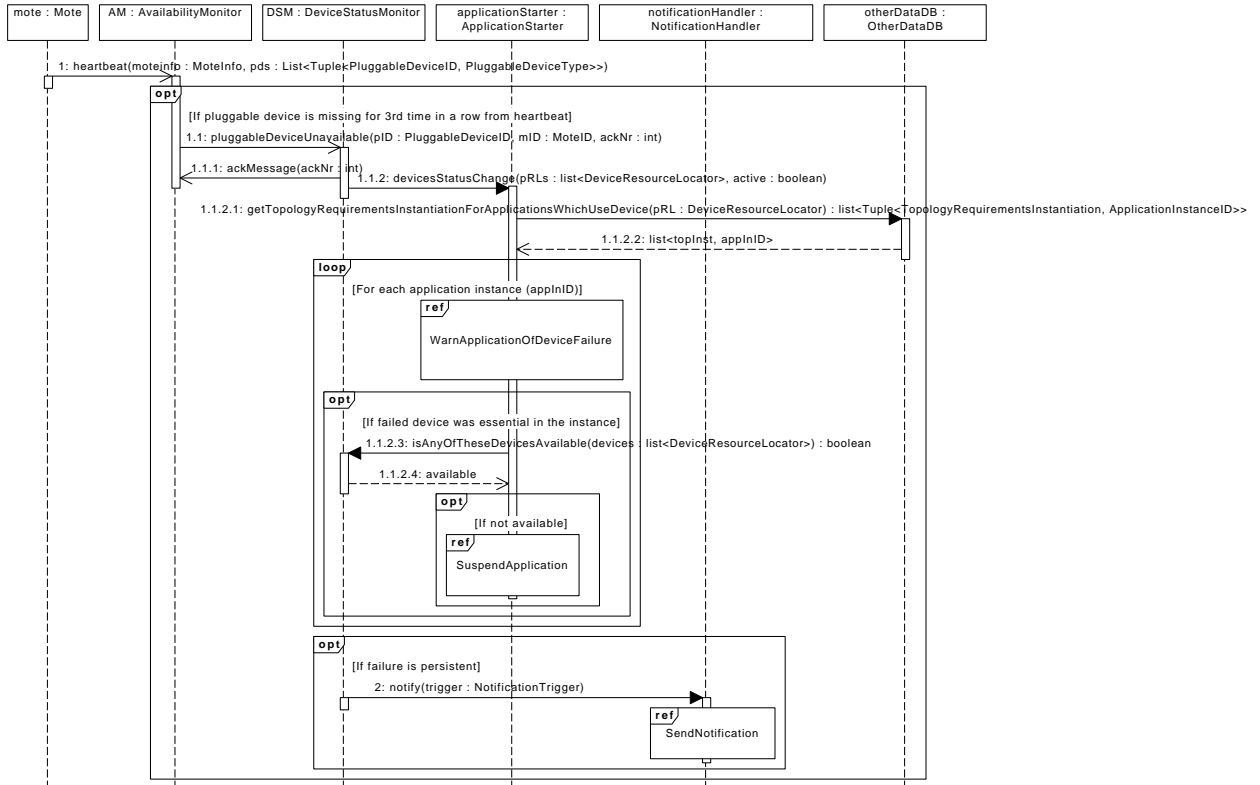


Figure 4.7: A device is no longer sending out heartbeats. The system responds by figuring out which applications are affected by this failure, checking whether they can still run without the failed device, and suspending them if they can't. If they can still run, the application is informed of the failure so it can leverage any redundancies if possible (by asking sensor data from other accessible pluggable devices).

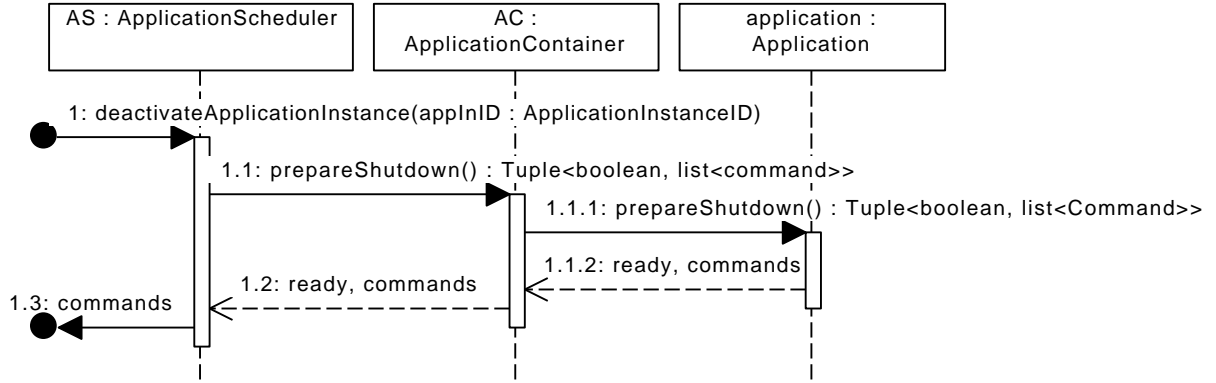


Figure 4.8: KillContainer

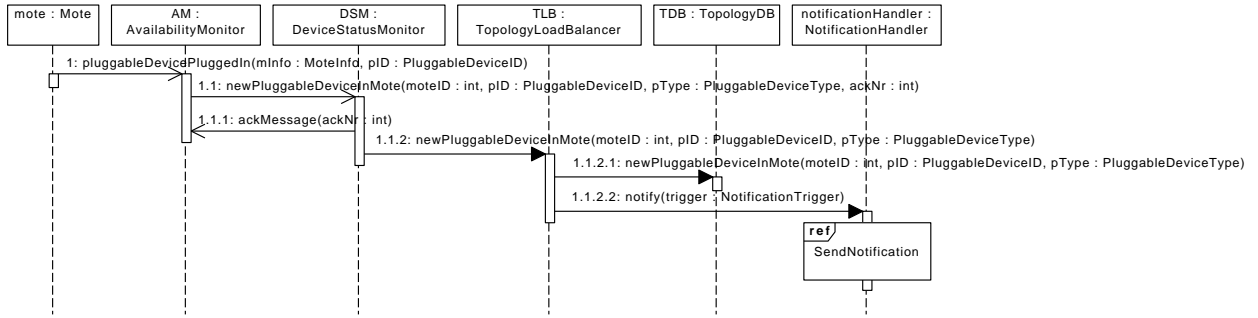


Figure 4.9: A new pluggable device is plugged into a mote.

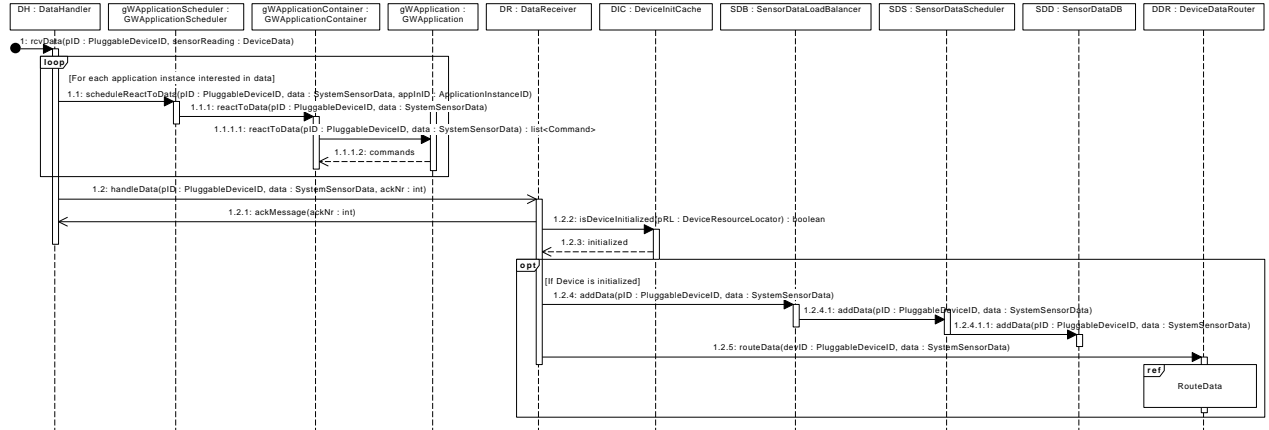


Figure 4.10: How a single pluggable device reading is handled. Checking whether a device is initialized is only needed at the Online Service as no application on the Gateway (or the Online Service) will be scheduled to receive data from an uninitialized device (cannot be part of a **TopologyRequirementsInstantiation**).

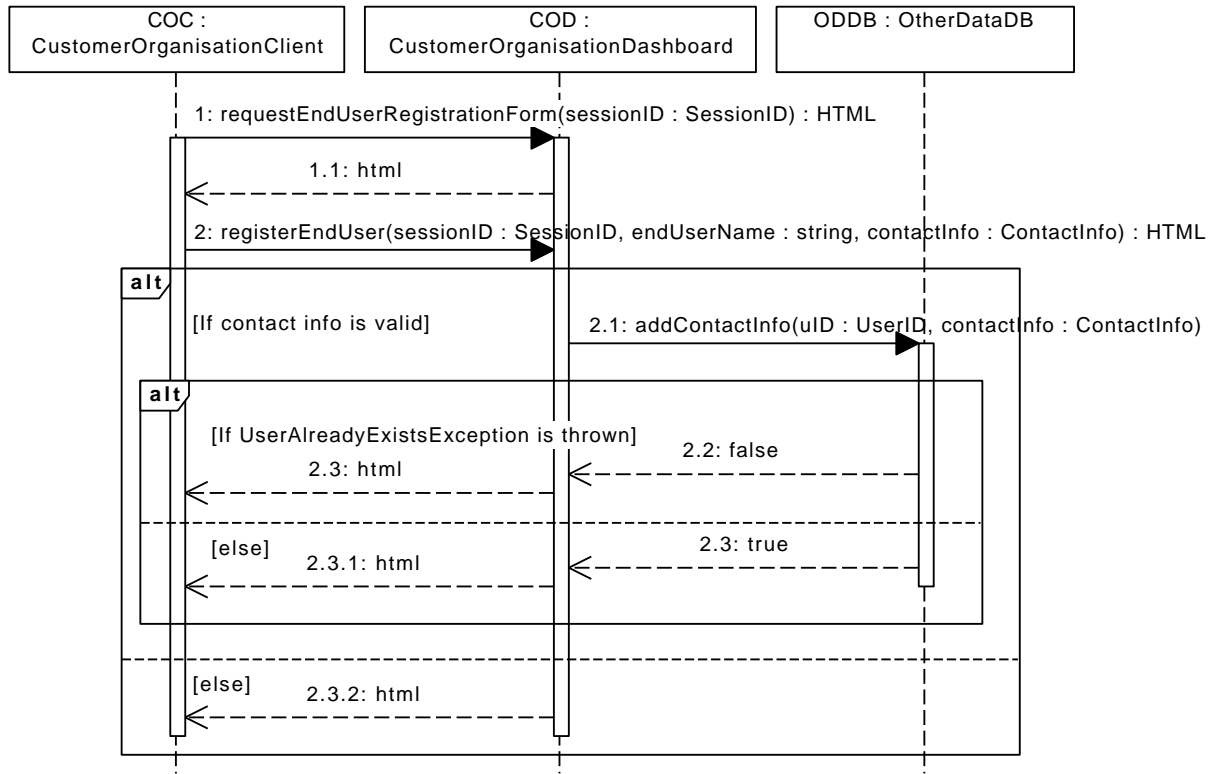


Figure 4.11: Register a possible end user for a Customer Organisation.

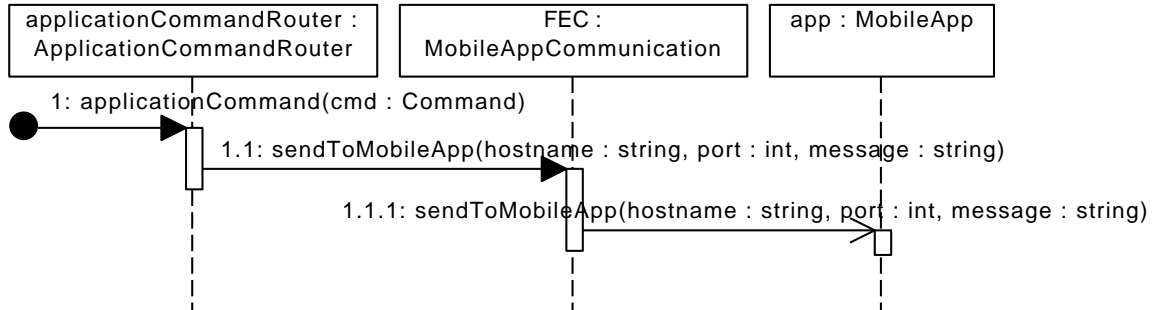


Figure 4.12: Send a command from an application instance on the Online Service to a smartphone app.

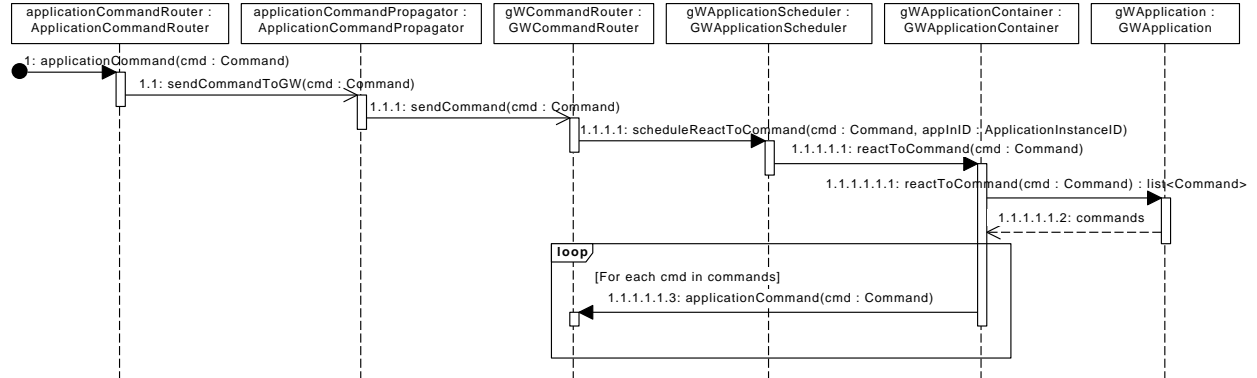


Figure 4.13: A command intended for the Gateway part of an application instance has arrived on the ApplicationCommandRouter, which forwards it to the correct Gateway.



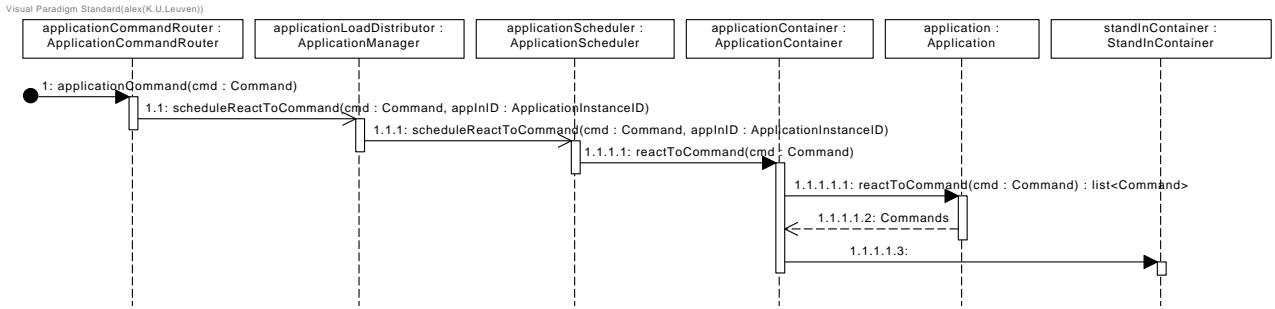


Figure 4.14: A command intended for an Online Service part of an **Application** instance has arrived on the **ApplicationCommandRouter** and is forwarded to the correct **Application** instance (via its container).

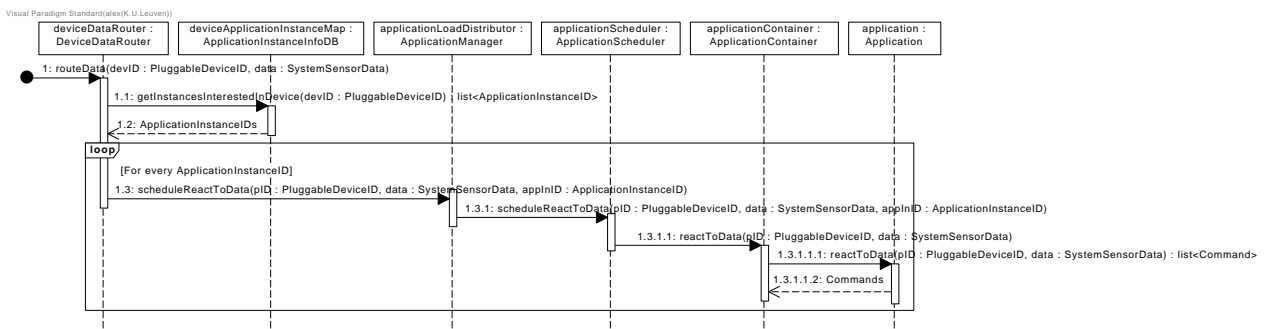


Figure 4.15: RouteData

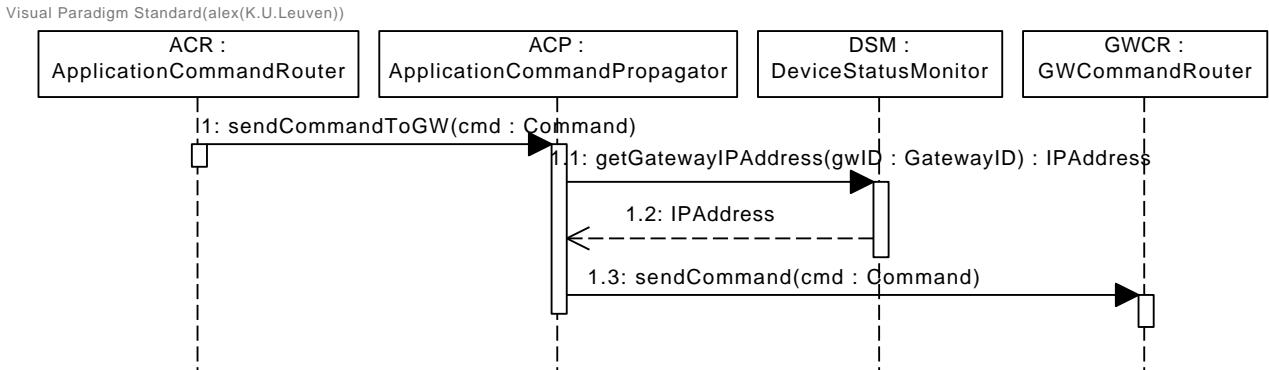


Figure 4.16: Forward a command to a **Gateway**, this can be an actuation command or a command for an **Gateway Application** instance.

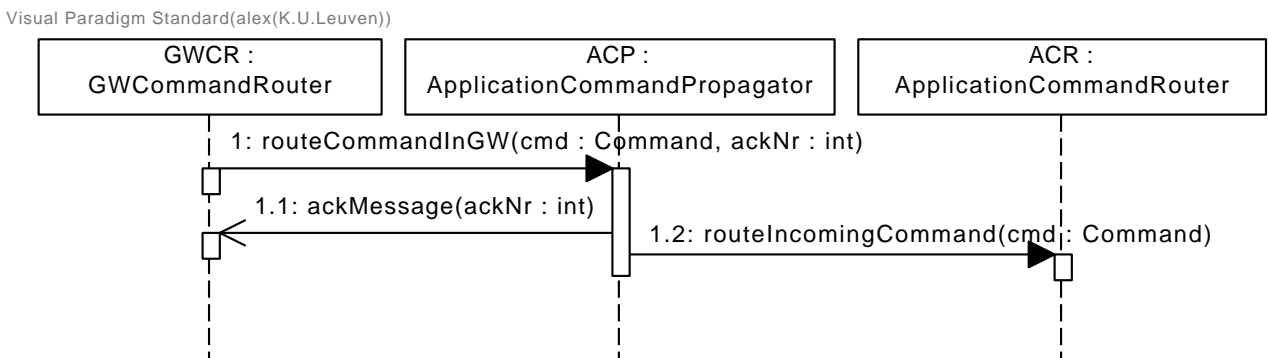


Figure 4.17: RouteToOS: Forward a command from the **Gateway** to the Online Service.

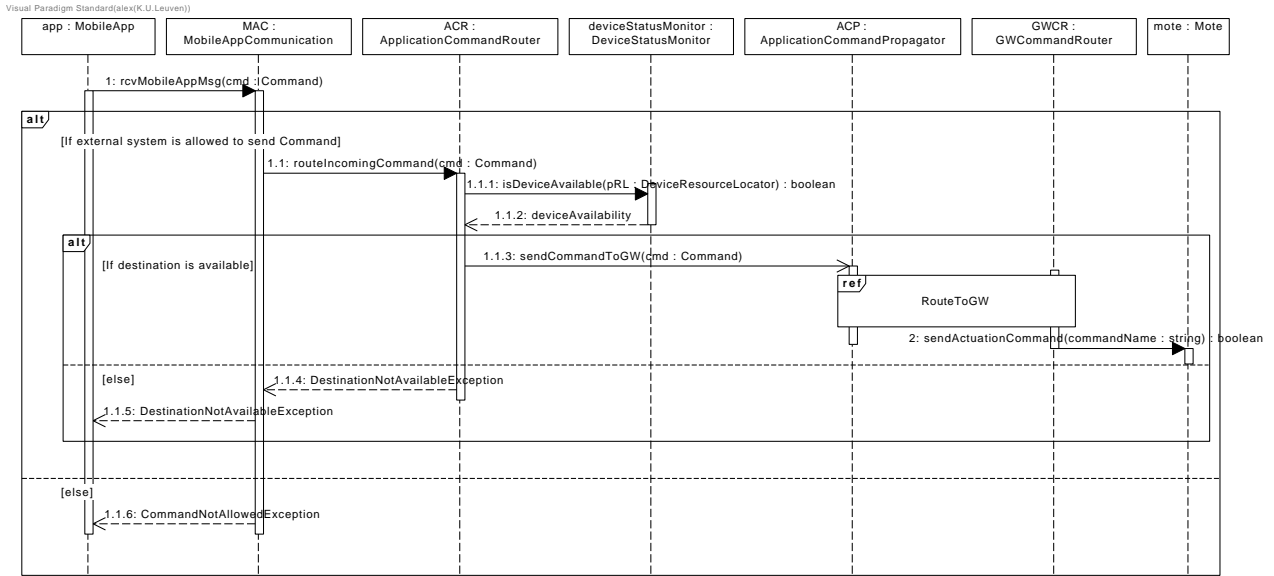


Figure 4.18: ProcessActuationFromMobileApp

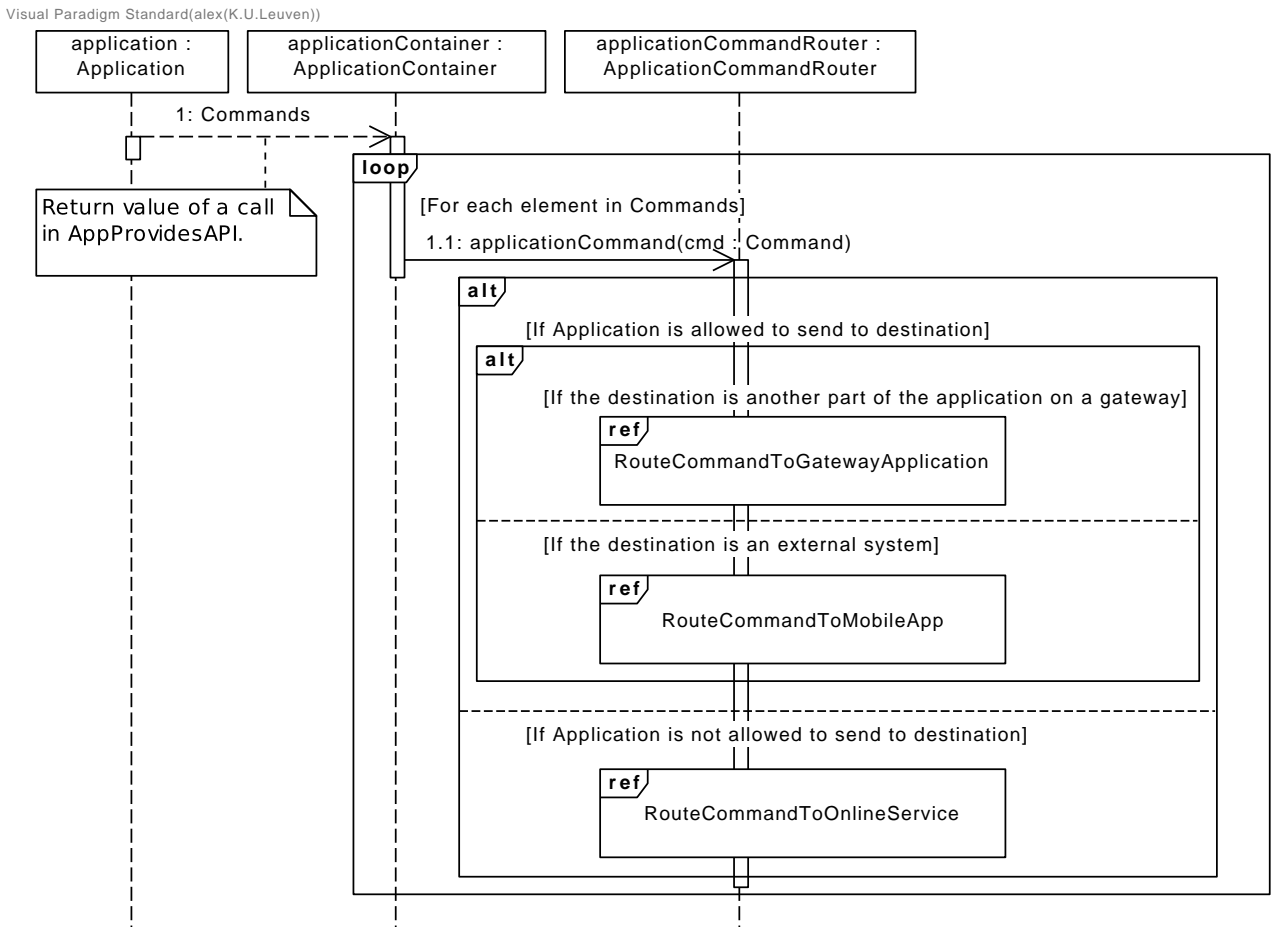


Figure 4.19: Forward commands received from an application instance on the Online Service to its intended recipient.

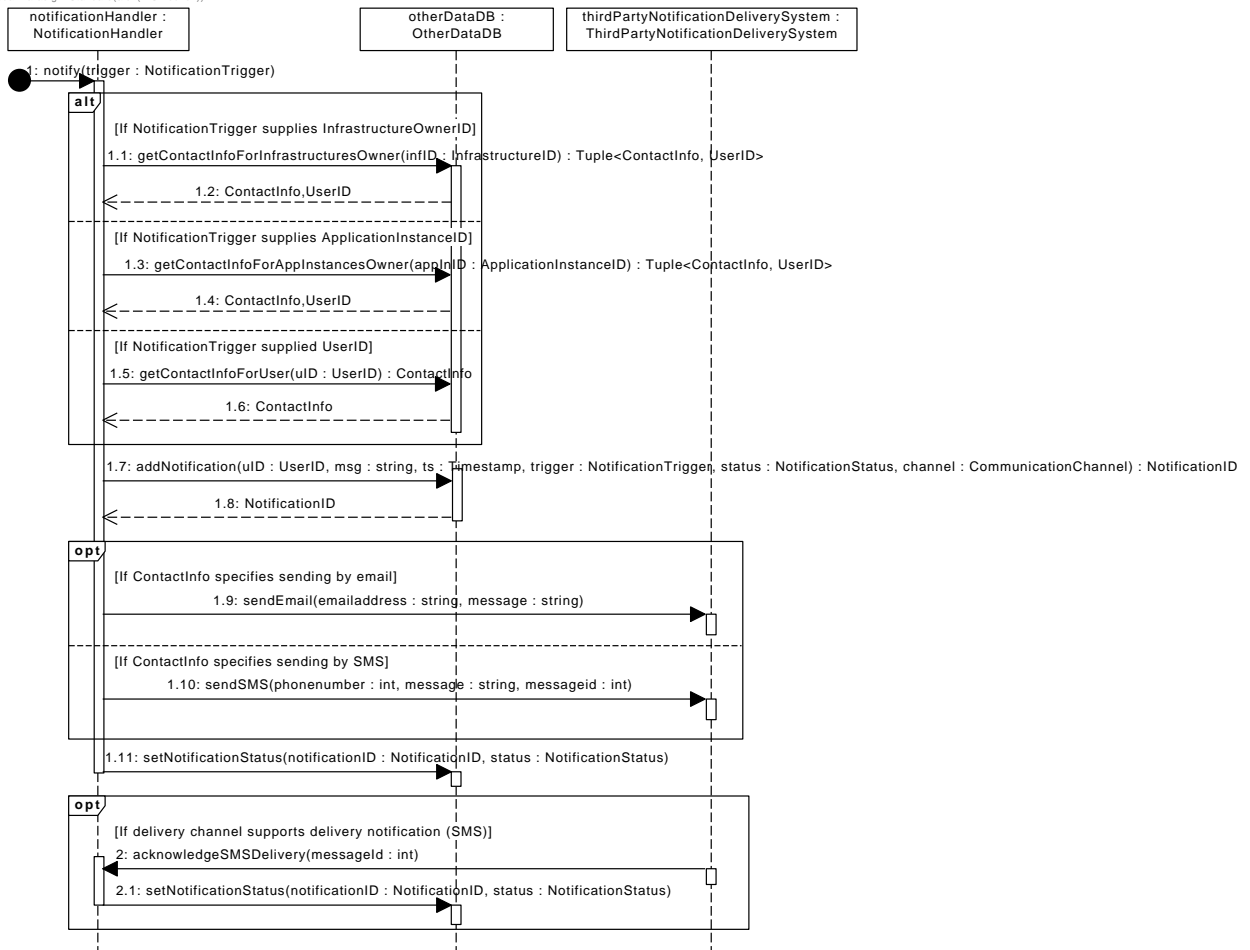


Figure 4.20: Sending a notification because one of the many possible triggers happened. Note that a **NotificationTrigger** can contain a variety of data causing different lookups to the **OtherDataDB**. The first **setNotification** call will mark a notification as “sent”, the optional second one as “delivered”.

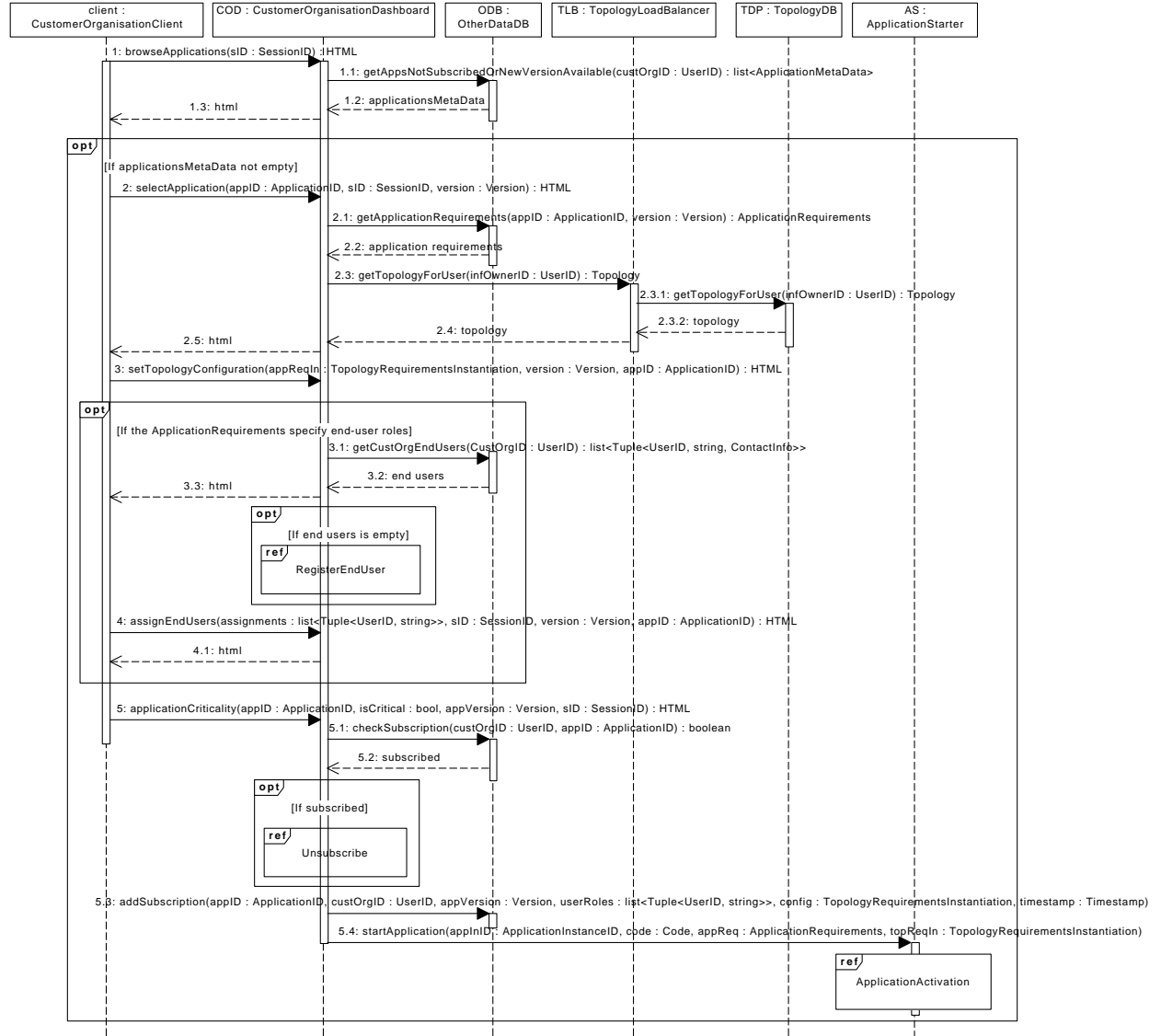


Figure 4.21: A Customer Organisation subscribes to an application.

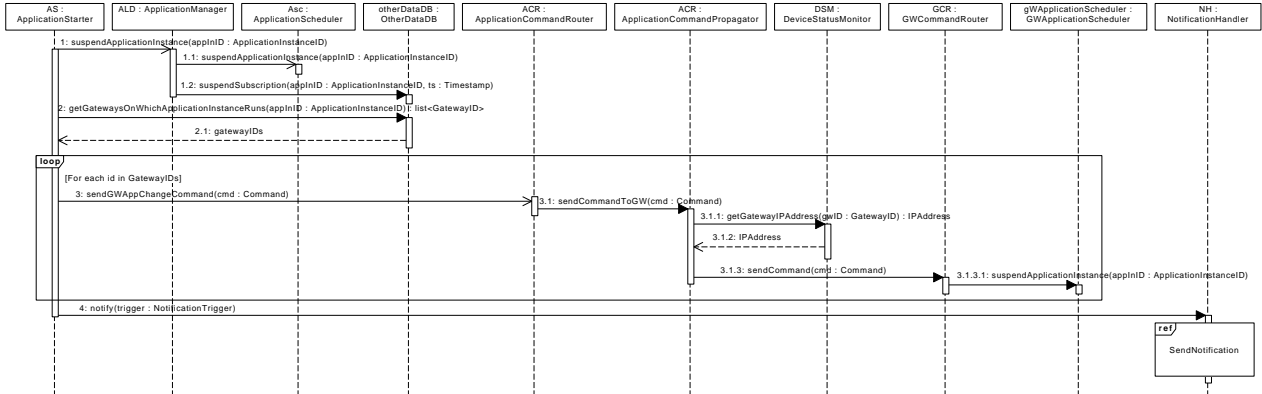


Figure 4.22: The sent "Command" in this case specifies the crashed application instance.

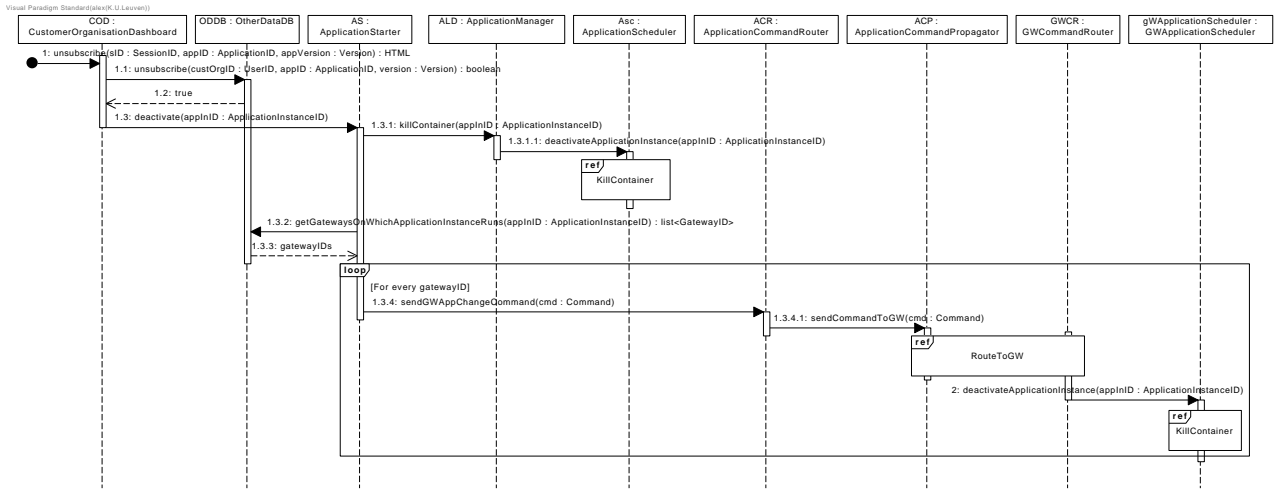


Figure 4.23: A Customer Organisation unsubscribes from an application. The **Command** being sent to the **Gateway** asks the **Gateway** to kill the container running the specified application instance.

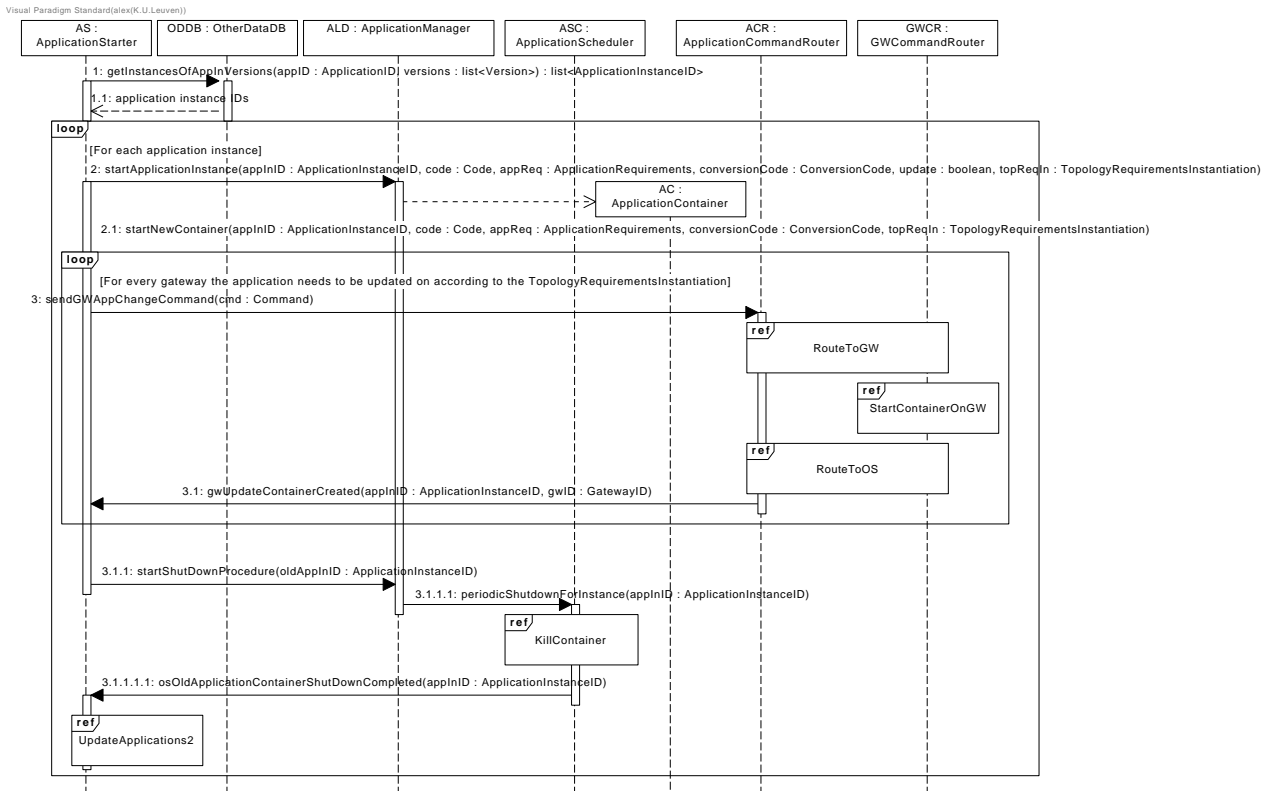


Figure 4.24: An application is eligible for an update. The old version is safely shut down and the new one booted up. Starting a container on the **Gateway** is similar as on the Online Service and is not explicitly shown.

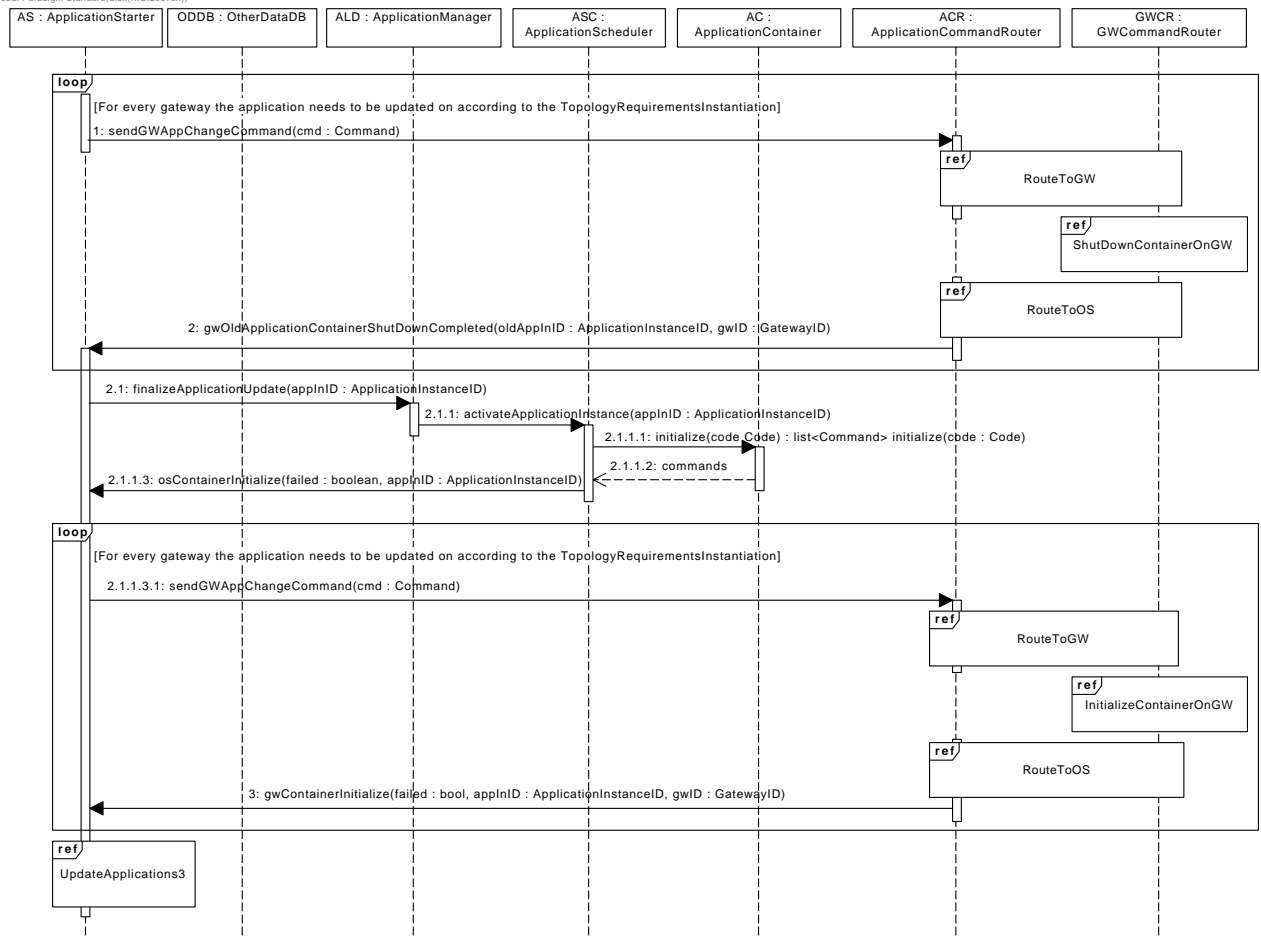


Figure 4.25: UpdateApplications2

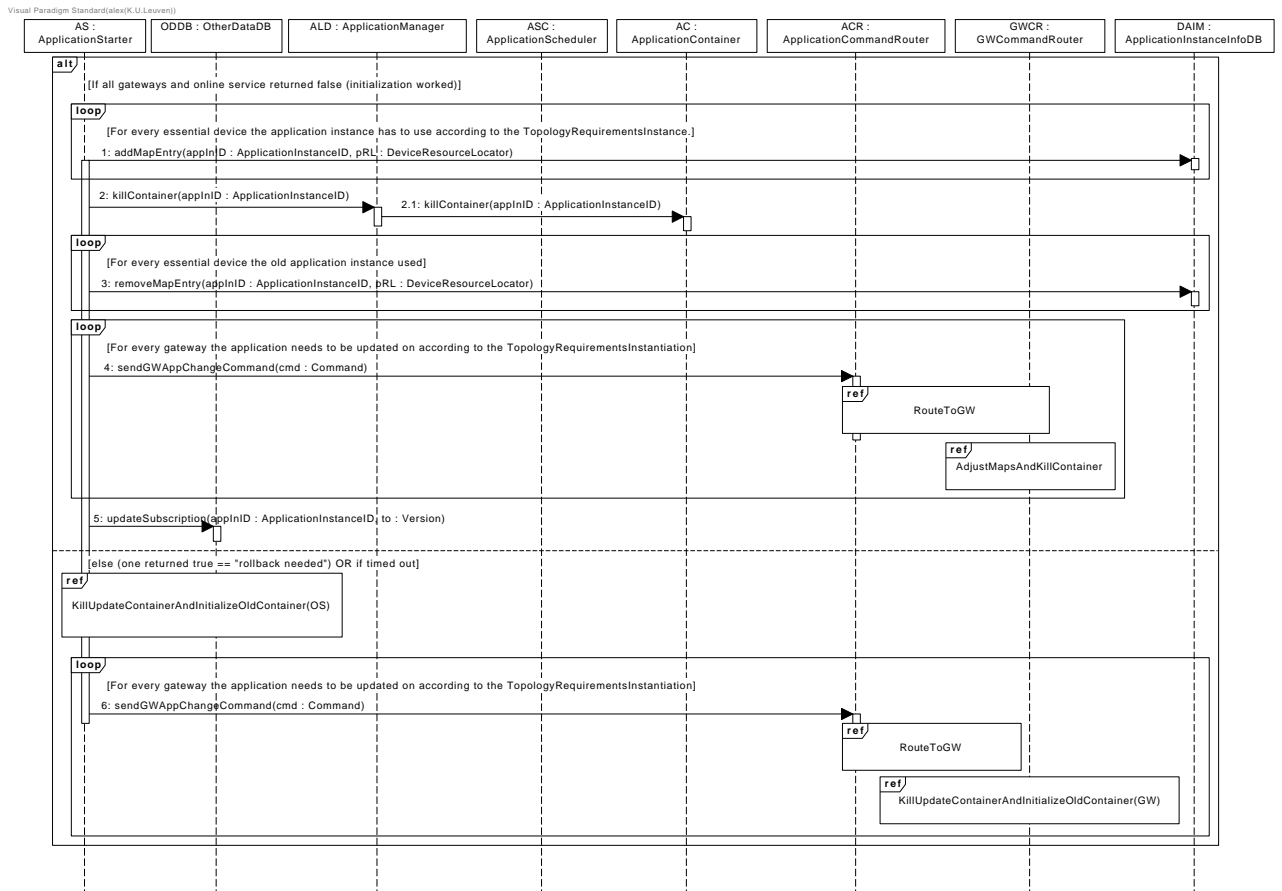


Figure 4.26: Add required pluggable devices to the new (updated) application instances, remove them from the previous, now outdated, instances. The containers executing the old application instances are killed.

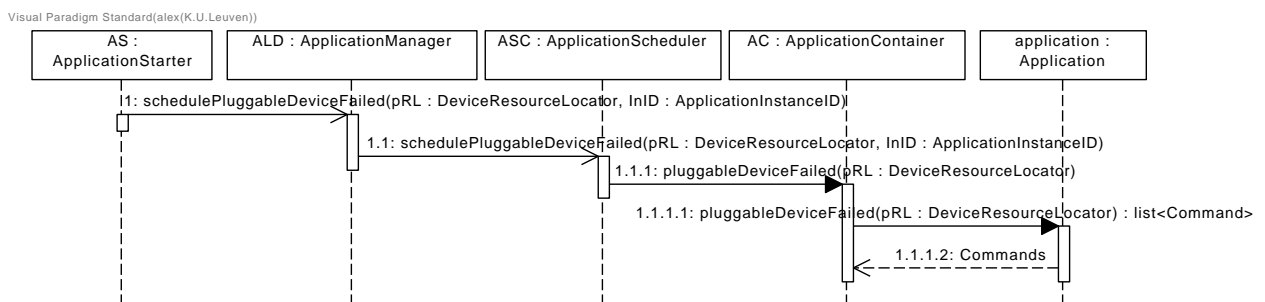


Figure 4.27: Inform an application that a pluggable device it used has failed.

# 5. Element catalog

## 5.1 Components

### 5.1.1 Application

**Responsibility:** Represents an application instance of a certain Customer Organisation for an application subscription. Although this physically runs on the **SIoTIP system** it is still communicated with as an external entity.

**Super-components:** None

**Sub-components:** None

**Provided interfaces:**  $\circ$  AppProvidesAPI

**Required interfaces:** None

**Deployed on:** ApplicationExecutionNode

**Visible on diagrams:** figs. 1.1, 3.1, 4.1, 4.2, 4.3, 4.8, 4.14, 4.15, 4.19 and 4.27

### 5.1.2 ApplicationCommandPropagator

**Responsibility:** The **ApplicationCommandPropagator** is responsible for propagating commands from the Online Service to the correct **Gateway**, as well as routing incoming commands to the correct subcomponent to process the command.

**Super-components:**  $\boxplus$  SIoTIP system  $\triangleright$   $\boxplus$  OSWithGWCommunication

**Sub-components:** None

**Provided interfaces:**  $\circ$  GWToOSCommand,  $\circ$  SendCommandToGW

**Required interfaces:**  $\prec$  AckMessage,  $\prec$  ChangeSyncTimer,  $\prec$  CommHeartbeat,  $\prec$  IncomingCommand,  $\prec$  IPLookup,  $\prec$  OSToGWCommand

**Deployed on:** OnlineServiceNode

**Visible on diagrams:** figs. 2.2, 3.2, 4.2, 4.13, 4.16, 4.17, 4.18, 4.22 and 4.23

### 5.1.3 ApplicationCommandRouter

**Responsibility:** The **ApplicationCommandRouter** is responsible for routing commands from application instances and mobile apps to the correct location in the system, i.e. another part of the same application instance, an actuator or database. It will also check whether the issued command is legitimate, i.e. is the application instance allowed to issue that command. See the AppCalls interface for more information on the routing.

**Super-components:**  $\boxplus$  SIoTIP system

**Sub-components:** None

**Provided interfaces:**  $\circ$  AppCalls,  $\circ$  GWAppChange,  $\circ$  IncomingCommand

**Required interfaces:**  $\prec$  AppFailure,  $\prec$  AppInstanceMapChange,  $\prec$  DeliverMsgMobileApp,  $\prec$  DeviceAvailabilityCheck,  $\prec$  HandleDBRequest,  $\prec$  OtherDataMgmt,  $\prec$  ScheduleApplication,  $\prec$  SendCommandToGW,  $\prec$  TopologyMgmt,  $\prec$  UpdateMessage

**Deployed on:** ApplicationExecutionNode

**Visible on diagrams:** figs. 1.2, 3.2, 4.1, 4.2, 4.12, 4.13, 4.14, 4.16, 4.17, 4.18, 4.19, 4.22, 4.23, 4.24, 4.25 and 4.26

### 5.1.4 ApplicationContainer

**Responsibility:** The **ApplicationContainer** is responsible for internally running and monitoring an application instance (or the part of it which is running on the Online Service). All interaction with the application instance is done through the **ApplicationContainer**. It will keep track of the local state of the instance and provide it whenever an interface of the container is used. An application uses the container, which has the requirements of the application, to check whether a configuration/other command is valid. If an application expects a sensor data unit different than the one provided (e.g. degrees Celsius instead of degrees Fahrenheit, found out by checking the



**ApplicationRequirements**), the container will convert the unit before handing it to the application by using **ConversionCode**.

**Super-components:**  $\boxplus$  **ApplicationHandler**  $\triangleright$   $\boxplus$  **SIOtIP system**

**Sub-components:** None

**Provided interfaces:**  $\circ$  **AppContainerMgmt**,  $\circ$  **CallThroughContainer**

**Required interfaces:**  $\prec$  **AppCalls**,  $\prec$  **AppFailure**,  $\prec$  **AppProvidesAPI**

**Deployed on:** **ApplicationExecutionNode**

**Visible on diagrams:** figs. 1.1, 2.4, 3.1, 3.2, 4.1, 4.2, 4.3, 4.8, 4.14, 4.15, 4.19, 4.24, 4.25, 4.26 and 4.27

### 5.1.5 ApplicationDeveloperClient

**Responsibility:** The **ApplicationDeveloperClient** is external to the **SIOtIP system** and represents the client of the application developer that communicates with the **SIOtIP system**.

**Super-components:** None

**Sub-components:** None

**Provided interfaces:** None

**Required interfaces:**  $\prec$  **ApplicationInfo**,  $\prec$  **ConsultNotifications**,  $\prec$  **UploadApplication**,  $\prec$  **UserAuthentication**

**Deployed on:** **ApplicationDeveloperNode**

**Visible on diagrams:** figs. 1.1, 3.1 and 4.4

### 5.1.6 ApplicationDeveloperDashboard

**Responsibility:** The **ApplicationDeveloperDashboard** provides the main interface to all application developers. It allows them to upload new applications as well as update old ones, gather statistical data and consult notifications.

**Super-components:**  $\boxplus$  **SIOtIP system**

**Sub-components:** None

**Provided interfaces:**  $\circ$  **ApplicationInfo**,  $\circ$  **ConsultNotifications**,  $\circ$  **UploadApplication**,  $\circ$  **UserAuthentication**

**Required interfaces:**  $\prec$  **ApplicationActivation**,  $\prec$  **NotifyRecipient**,  $\prec$  **OtherDataMgmt**,  $\prec$  **TestApplication**,  $\prec$  **UserAuthentication**,  $\prec$  **VerifySession**

**Deployed on:** **DashboardsNode**

**Visible on diagrams:** figs. 1.1, 1.2, 3.1, 3.2 and 4.4

### 5.1.7 ApplicationHandler

**Responsibility:** The **ApplicationHandler** is responsible for the execution of application instances on the Online Service.

**Super-components:**  $\boxplus$  **SIOtIP system**

**Sub-components:**  $\boxplus$  **ApplicationManager**,  $\boxplus$  **ApplicationScheduler**,  $\boxplus$  **ApplicationContainer**

**Provided interfaces:**  $\circ$  **AppInstanceChange**,  $\circ$  **Failure**,  $\circ$  **ScheduleApplication**

**Required interfaces:**  $\prec$  **AppCalls**,  $\prec$  **AppFailure**,  $\prec$  **AppInstanceMapChange**,  $\prec$  **AppProvidesAPI**,  $\prec$  **OtherDataMgmt**,  $\prec$  **UpdateMessage**

**Deployed on:** **ApplicationExecutionNode**

**Visible on diagrams:** figs. 1.2 and 2.4

### 5.1.8 ApplicationHeartbeatMonitor

**Responsibility:** The **ApplicationHeartbeatMonitor** is responsible for detecting failures of application execution components. When an **ApplicationContainer** on a server crashes, the application instance is restarted in a new container, when an **ApplicationScheduler** or **ApplicationCommandRouter** crashes, the entire server is reset and the **ApplicationInstanceInfoDB** lookup map and **DeviceDataRouter** can be reconstructed from the data in the database.

**Super-components:**  $\boxplus$  **SIOtIP system**




**Sub-components:** None

**Provided interfaces:**  $\circ$  **AppFailure**












**Required interfaces:**  $\prec$  **Failure**,  $\prec$  **InternalAppActivation**,  $\prec$  **NotifyRecipient**

**Deployed on:** ApplicationFailureNode  
**Visible on diagrams:** figs. 1.2, 3.2 and 4.3









### 5.1.9 ApplicationInstanceInfoDB

**Responsibility:** This component is responsible to keep track of which application instances are interested in which pluggable devices.  
**Super-components:**  SIoTIP system  
**Sub-components:** None  
**Provided interfaces:** ,   
**Required interfaces:** None  
**Deployed on:** OnlineServiceNode  
**Visible on diagrams:** figs. 1.2, 3.2, 4.1, 4.15 and 4.26













### 5.1.10 ApplicationManager

**Responsibility:** The ApplicationManager is responsible to manage calls concerning application instances. This includes starting new instances, stopping existing ones and forwarding incoming requests to the ApplicationScheduler.  
**Super-components:**  ApplicationHandler  SIoTIP system  
**Sub-components:** None  
**Provided interfaces:** , ,   
**Required interfaces:** , , , , ,   
**Deployed on:** ApplicationExecutionNode  
**Visible on diagrams:** figs. 2.4, 3.2, 4.1, 4.14, 4.15, 4.22, 4.23, 4.24, 4.25, 4.26 and 4.27

### 5.1.11 ApplicationScheduler


**Responsibility:** The ApplicationScheduler is responsible for scheduling all information/commands to the various application instances. This includes answers from databases, commands, synchronization data from Gateways and wake up calls in case the application wishes asynchronous updates. It will route any of those requests to the corresponding ApplicationContainer where they will be relayed to the Application instance.  
**Super-components:**  ApplicationHandler  SIoTIP system  
**Sub-components:** None  
**Provided interfaces:** ,   
**Required interfaces:** , , ,   
**Deployed on:** ApplicationExecutionNode  
**Visible on diagrams:** figs. 2.4, 3.2, 4.1, 4.3, 4.8, 4.14, 4.15, 4.22, 4.23, 4.24, 4.25, 4.26 and 4.27

### 5.1.12 ApplicationStarter


**Responsibility:** The ApplicationStarter is responsible for activating, deactivating and updating application instances. Whenever it gets a request to start a new application of some type, it will spread the required information to do so in a correct order to various other parts of the system (e.g. starting a new ApplicationContainer with the given code). Certain events (end-user role assignment, pluggable device failure, etc.) might trigger the ApplicationStarter to check whether any of its three responsibilities need to be executed.  
**Super-components:**  SIoTIP system  
**Sub-components:** None  
**Provided interfaces:** , ,   
**Required interfaces:** , , , , , , ,   
**Deployed on:** OnlineServiceNode  
**Visible on diagrams:** figs. 1.2, 3.2, 4.1, 4.3, 4.4, 4.6, 4.7, 4.21, 4.22, 4.23, 4.24, 4.25, 4.26 and 4.27

### 5.1.13 AppTester

**Responsibility:** Responsible for running various tests on uploaded applications. If the tests fail, notifications will have to be sent out to certain parties, if they succeed, the application is made available to Customer Organisations and (if needed) updates to running application instances are initiated. If the tests fail the application developer is notified.

**Super-components:**  SIO TIP system

**Sub-components:** None

**Provided interfaces:**  TestApplication


**Required interfaces:** None

**Deployed on:** AppTestingNode



**Visible on diagrams:** figs. 1.2, 3.2 and 4.4



### 5.1.14 AuthenticationHandler

**Responsibility:** The AuthenticationHandler is responsible for authenticating all end-users of the SIO TIP system dashboards. The authentication is done by username and password.

**Super-components:**  SIO TIP system

**Sub-components:** None

**Provided interfaces:**  UserAuthentication,  VerifySession



**Required interfaces:**  OtherDataMgmt,  SessionMgmt

**Deployed on:** OnlineServiceNode





**Visible on diagrams:** figs. 1.2 and 3.2



### 5.1.15 AvailabilityMonitor

**Responsibility:** The AvailabilityMonitor is responsible for keeping track of the motes connected to the gateway and their pluggable devices. Whenever the status of either of those changes (e.g. new pluggable added or a mote fails), it sends a message to the Online Service.

**Super-components:**  Gateway  SIO TIP system

**Sub-components:** None

**Provided interfaces:**  AckMessage,  AMReactToFailure,  DeviceManagement,  Heartbeat



**Required interfaces:**  DeviceAvailabilityMessage,  FailureDetection

**Deployed on:** GatewayNode


**Visible on diagrams:** figs. 1.1, 2.1, 3.1, 3.2, 4.5, 4.7 and 4.9

### 5.1.16 CommunicationHeartbeatMonitor

**Responsibility:** The CommunicationHeartbeatMonitor is responsible for heartbeating the various components on the Online Service which communicate with the Gateways to make sure they (the components) have not gone down.

**Super-components:**  SIO TIP system  OSWithGWCommunication

**Sub-components:** None

**Provided interfaces:**  CommHeartbeat

**Required interfaces:** None

**Deployed on:** CommunicationFailureNode

**Visible on diagrams:** figs. 2.2 and 3.2

### 5.1.17 CustomerOrganisationClient

**Responsibility:** The CustomerOrganisationClient is external to the SIO TIP system and represents the client of the Customer Organisation that communicates with the SIO TIP system.

**Super-components:** None

**Sub-components:** None

**Provided interfaces:** None


**Required interfaces:**  ConsultNotifications,  Registration,  Subscription,  UserAuthentication

**Deployed on:** CustomerOrganisationNode

**Visible on diagrams:** figs. 1.1, 3.1, 4.11 and 4.21

### 5.1.18 CustomerOrganisationDashboard

**Responsibility:** The `CustomerOrganisationDashboard` provides the main interface to Customer Organisations, it can be used to register a new organisation or by a registered organisation to subscribe to applications. Notifications for Customer Organisations can also be requested. Timers are used to timeout registration links.

**Super-components:**  `SIoTIP` system

**Sub-components:** None

**Provided interfaces:** `⊖ ConsultNotifications`, `⊖ Registration`, `⊖ Subscription`, `⊖ UserAuthentication`



**Required interfaces:** `⊢ ApplicationActivation`, `⊢ OtherDataMgmt`, `⊢ TopologyMgmt`, `⊢ UserAuthentication`, `⊢ VerifySession`

**Deployed on:** `DashboardsNode`

**Visible on diagrams:** figs. 1.1, 1.2, 3.1, 3.2, 4.11, 4.21 and 4.23

### 5.1.19 DataHandler

**Responsibility:** The `DataHandler` is responsible for translating between the data format the pluggable devices use and one used inside of the `SIoTIP` system (in both directions). It will propagate any incoming sensor data from the sensors to the Online Service (and if needed to a part of the application instance running on the Gateway).

**Super-components:**  Gateway  `SIoTIP` system

**Sub-components:** None

**Provided interfaces:** `⊖ AckMessage`, `⊖ AppInstanceMapChange`, `⊖ DHReactToFailure`, `⊖ SensorData`



**Required interfaces:** `⊢ FailureDetection`, `⊢ GWTempDataMgmt`, `⊢ RcvSensorData`, `⊢ RequestData`, `⊢ ScheduleApplication`

**Deployed on:** `GatewayNode`

**Visible on diagrams:** figs. 1.1, 2.1, 3.1, 3.2, 4.5 and 4.10

### 5.1.20 DataReceiver

**Responsibility:** The `DataReceiver` is responsible for propagating sensor data received from a gateway to the `OSSensorDataDB`, as well as route it to the application execution subsystem. It will warn the `DeviceStatusMonitor` if no data has been received for a while as these are used as heartbeats. It uses timers to detect whether a `Gateway` has failed or when a new gateway starts sending out heartbeats.

**Super-components:**  `SIoTIP` system  `OSWithGWCommunication`

**Sub-components:** None

**Provided interfaces:** `⊖ ChangeSyncTimer`, `⊖ RcvSensorData`


**Required interfaces:** `⊢ AckMessage`, `⊢ CommHeartbeat`, `⊢ DeviceInitCacheMgmt`, `⊢ GWAavailability`, `⊢ RouteDeviceData`, `⊢ SensorDataMgmt`

**Deployed on:** `OnlineServiceNode`

**Visible on diagrams:** figs. 2.2, 3.2, 4.6 and 4.10

### 5.1.21 DBRequestHandler

**Responsibility:** The `DBRequestHandler` will forward requests by application instances to the `OSSensorDataDB` and `TopologyLoadBalancer`. Once It receives an answer, it forwards that information to the `ApplicationScheduler`. These requests concern historical data and overview of the topology.

**Super-components:**  `SIoTIP` system

**Sub-components:** None

**Provided interfaces:** `⊖ DBRequestResults`, `⊖ HandleDBRequest`


**Required interfaces:** `⊢ ScheduleApplication`, `⊢ SensorDataMgmt`, `⊢ TopologyMgmt`

**Deployed on:** `OnlineServiceNode`


**Visible on diagrams:** figs. 1.2 and 3.2



### 5.1.22 DeviceDataRouter

**Responsibility:** Responsible for routing incoming data to the correct application instances by scheduling the receipt of this data by the interested applications via the `ApplicationManager`.

**Super-components:**  `SIoTIP` system

**Sub-components:** None

**Provided interfaces:**  `RouteDeviceData`


**Required interfaces:**  `ApplicationInstanceLookup`,  `ScheduleApplication`

**Deployed on:** `OnlineServiceNode`


**Visible on diagrams:** figs. 1.2, 3.2, 4.10 and 4.15

### 5.1.23 DeviceInitCache

**Responsibility:** The `DeviceInitCache` provides a cache of the status of pluggable devices which can be used to determine whether a pluggable device has been initialized yet. Useful to reduce load on the `TopologyDB` by avoiding doing a database call every time sensor data arrives.

**Super-components:**  `SIoTIP` system

**Sub-components:** None

**Provided interfaces:**  `DeviceInitCacheMgmt`



**Required interfaces:** None

**Deployed on:** `OnlineServiceNode`





**Visible on diagrams:** figs. 1.2, 3.2 and 4.10







### 5.1.24 DeviceStatusMonitor

**Responsibility:** The `DeviceStatusMonitor` is responsible for monitoring the status of the gateways, motes and their pluggable devices. It will receive both direct messages from the `Gateway` informing of failures as well as information from the `DataReceiver` whose messages act as heartbeats to detect `Gateway` failure. It will propagate which devices have become unavailable as a result to the `ApplicationStarter`. Other communication components use this component to lookup `Gateway` IP addresses based on `Gateway` IDs.

**Super-components:**  `SIoTIP` system  `OSWithGWCommunication`

**Sub-components:** None

**Provided interfaces:**  `DeviceAvailabilityCheck`,  `DeviceAvailabilityMessage`,  `GWAvailability`,  `IPLookup`


**Required interfaces:**  `AckMessage`,  `ApplicationActivation`,  `ChangeSyncTimer`,  `CommHeartbeat`,  `NotifyRecipient`,  `TopologyMgmt`









**Deployed on:** `OnlineServiceNode`






**Visible on diagrams:** figs. 2.2, 3.2, 4.2, 4.6, 4.7, 4.9, 4.16, 4.18 and 4.22









### 5.1.25 Gateway

**Responsibility:** The `Gateway` is responsible for all the parts of the system that are expected to be executed on a gateway. This includes running parts of application instances, communication with the Online Service and communication with motes and their pluggable devices.

**Super-components:**  `SIoTIP` system

**Sub-components:**  `GWApplicationContainer`,  `GWApplicationScheduler`,  `GWApplicationHeartbeatMonitor`,  `GWCommandRouter`,  `GWFailureHandler`,  `GWTempDataDB`,  `AvailabilityMonitor`,  `DataHandler`

**Provided interfaces:**  `AckMessage`,  `DeviceManagement`,  `Heartbeat`,  `OSToGWCommand`,  `SensorData`

**Required interfaces:**  `Actuate`,  `Config`,  `DeviceAvailabilityMessage`,  `GWAppProvidesAPI`,  `GWToOSCommand`,  `RcvSensorData`,  `RebootGW`,  `RequestData`

**Deployed on:** `GatewayNode`

**Visible on diagrams:** figs. 1.2 and 2.1

### 5.1.26 GatewayOS

**Responsibility:** Represents the operating system of the `Gateway`.

**Super-components:** None

**Sub-components:** None  
**Provided interfaces:**  $\neg$  RebootGW  
**Required interfaces:** None  
**Deployed on:** GatewayNode  
**Visible on diagrams:** figs. 1.1 and 3.1

### 5.1.27 GWApplication

**Responsibility:** Represents an application instance of a certain customer organisation for an application part running on the **Gateway**. This physically runs on the **SIoTIP system** but is still communicated with as an external entity.

**Super-components:** None  
**Sub-components:** None  
**Provided interfaces:**  $\neg$  GWAppProvidesAPI  
**Required interfaces:** None  
**Deployed on:** GatewayNode  
**Visible on diagrams:** figs. 1.1, 3.1, 4.10 and 4.13

### 5.1.28 GWApplicationContainer

**Responsibility:** The **GWApplicationContainer** is responsible for internally running and monitoring a **GWApplication** instance. All interaction with the application instance is done through the **GWApplicationContainer**. It will keep track of the local state of the instance and provide it whenever an interface of the container is used. An application uses the container (which has the applications' requirements) to check whether a configuration/other command is valid. If an application expects a sensor data unit different than the one provided (e.g. degrees Celsius instead of degrees Fahrenheit, found out by checking the **ApplicationRequirements**), the container will convert the unit before handing it to the application by using **ConversionCode**.

**Super-components:**  $\boxplus$  Gateway  $\triangleright$   $\boxplus$  SIoTIP system  
**Sub-components:** None  
**Provided interfaces:**  $\neg$  AppInstanceChange,  $\neg$  CallThroughContainer  
**Required interfaces:**  $\prec$  AppFailure,  $\prec$  GWAppCalls,  $\prec$  GWAppProvidesAPI  
**Deployed on:** GatewayNode  
**Visible on diagrams:** figs. 1.1, 2.1, 3.1, 3.2, 4.10 and 4.13

### 5.1.29 GWApplicationHeartbeatMonitor

**Responsibility:** This component monitors the application instances executing on a **Gateway** as well as the components running them by keeping of their heartbeats (or lack thereof).

**Super-components:**  $\boxplus$  Gateway  $\triangleright$   $\boxplus$  SIoTIP system  
**Sub-components:** None  
**Provided interfaces:**  $\neg$  AppFailure  
**Required interfaces:** None  
**Deployed on:** GatewayNode  
**Visible on diagrams:** figs. 2.1 and 3.2

### 5.1.30 GWApplicationScheduler

**Responsibility:** The **GWApplicationScheduler** is responsible for scheduling all information/commands, such as sensor data, to the various application instances on a **Gateway**.

**Super-components:**  $\boxplus$  Gateway  $\triangleright$   $\boxplus$  SIoTIP system  
**Sub-components:** None  
**Provided interfaces:**  $\neg$  AppInstanceSuspension,  $\neg$  ScheduleApplication  
**Required interfaces:**  $\prec$  AppFailure,  $\prec$  CallThroughContainer  
**Deployed on:** GatewayNode  
**Visible on diagrams:** figs. 2.1, 3.2, 4.10, 4.13, 4.22 and 4.23

### 5.1.31 GWCommandRouter

**Responsibility:** The `GWCommandRouter` is responsible for routing commands from the Online Service to the correct component on the `Gateway` and vice versa. Incoming actuation commands are forwarded to the correct actuators and application commands, including commands to (de)activate or suspend instances, are forwarded to the `GWApplicationScheduler`.

**Super-components:** `Gateway` ▷ `SIoTIP system`

**Sub-components:** None

**Provided interfaces:** `◊ AckMessage`, `◊ CRReactToFailure`, `◊ GWAppCalls`, `◊ OSToGWCommand`

**Required interfaces:** `◊ Actuate`, `◊ AppInstanceChange`, `◊ AppInstanceMapChange`, `◊ AppInstanceSuspension`, `◊ Config`, `◊ FailureDetection`, `◊ GWTempDataMgmt`, `◊ GWToOSCommand`, `◊ ScheduleApplication`

**Deployed on:** `GatewayNode`

**Visible on diagrams:** figs. 1.1, 2.1, 3.1, 3.2, 4.1, 4.2, 4.5, 4.13, 4.16, 4.17, 4.18, 4.22, 4.23, 4.24, 4.25 and 4.26

### 5.1.32 GWFailureHandler

**Responsibility:** The `GWFailureHandler` is responsible for reacting to and resolving various kinds of failures on the `Gateway` or the communication channel between the `Gateway` and the Online Service. This includes the failure of communication components running on the `Gateway` (monitored by using heartbeats) and the failure of the communication channel (reported by the three communication components when they do not receive an acknowledgement on a message from the Online Service).

**Super-components:** `Gateway` ▷ `SIoTIP system`

**Sub-components:** None

**Provided interfaces:** `◊ FailureDetection`

**Required interfaces:** `◊ AMReactToFailure`, `◊ CRReactToFailure`, `◊ DHReactToFailure`, `◊ RebootGW`

**Deployed on:** `GatewayNode`

**Visible on diagrams:** figs. 1.1, 2.1, 3.2 and 4.5

### 5.1.33 GWTempDataDB

**Responsibility:** The `GWTempDataDB` is responsible for storing (a limited amount of) sensor and actuation data when there is a communication failure with the Online System.

**Super-components:** `Gateway` ▷ `SIoTIP system`

**Sub-components:** None

**Provided interfaces:** `◊ GWTempDataMgmt`

**Required interfaces:** None

**Deployed on:** `GatewayNode`

**Visible on diagrams:** figs. 2.1 and 3.2

### 5.1.34 InfrastructureOwnerClient

**Responsibility:** The `InfrastructureOwnerClient` is external to the `SIoTIP system` and represents the client of the infrastructure owner that communicates with the `SIoTIP system`.

**Super-components:** None

**Sub-components:** None

**Provided interfaces:** None

**Required interfaces:** `◊ ConsultNotifications`, `◊ ManageDevices`, `◊ UserAuthentication`

**Deployed on:** `InfrastructureOwnerNode`

**Visible on diagrams:** figs. 1.1 and 3.1

### 5.1.35 InfrastructureOwnerDashboard

**Responsibility:** The `InfrastructureOwnerDashboard` provides the main interface to any Infrastructure Owners, it is used to configure devices as well as consulting notifications.

**Super-components:** `SIoTIP system`

**Sub-components:** None

**Provided interfaces:** `◊ ConsultNotifications`, `◊ ManageDevices`, `◊ UserAuthentication`


**Required interfaces:** `◊ TopologyMgmt`, `◊ UserAuthentication`, `◊ VerifySession`

**Deployed on:** DashboardsNode

**Visible on diagrams:** figs. 1.1, 1.2, 3.1 and 3.2

### 5.1.36 InvoiceManager

**Responsibility:** The `InvoiceManager` is responsible for sending invoices for used products to a third party invoicing service (e.g. Zoomit or a credit card company). At the end of each month the `InvoiceManager` will check which subscriptions that month have run and for how long, it will use the `OtherDataDB` to get information on both the subscriptions and the payment options. Invoices will also be sent whenever an unsubscription happens.

**Super-components:**  SIoTIP system

**Sub-components:** None

**Provided interfaces:** None

**Required interfaces:**  $\prec$  `OtherDataMgmt`,  $\prec$  `SendInvoice`

**Deployed on:** OnlineServiceNode

**Visible on diagrams:** figs. 1.1, 1.2, 3.1 and 3.2

### 5.1.37 MobileApp

**Responsibility:** A mobile application acting as front-end for an application.

**Super-components:** None

**Sub-components:** None

**Provided interfaces:**  $\prec$  `SendMobileAppMsg`


**Required interfaces:**  $\prec$  `RcvMobileAppMsg`

**Deployed on:** MobileDevice

**Visible on diagrams:** figs. 1.1, 1.2, 3.1, 4.12 and 4.18

### 5.1.38 MobileAppCommunication

**Responsibility:** This component is responsible for communication with mobile applications acting as front-ends. It will propagate any commands issued (if deemed valid) by the external actor to the `ApplicationCommandRouter` as well as send data back to any connected mobile applications.

**Super-components:**  SIoTIP system

**Sub-components:** None

**Provided interfaces:**  $\prec$  `DeliverMsgMobileApp`,  $\prec$  `RcvMobileAppMsg`

**Required interfaces:**  $\prec$  `IncomingCommand`,  $\prec$  `SendMobileAppMsg`

**Deployed on:** MobileAppCommunicationNode

**Visible on diagrams:** figs. 1.1, 1.2, 3.1, 3.2, 4.12 and 4.18

### 5.1.39 Mote

**Responsibility:** The `Mote` is external to the SIoTIP system and represents the hardware device in which the various pluggable devices the system uses are plugged in.

**Super-components:** None

**Sub-components:** None

**Provided interfaces:**  $\prec$  `Actuate`,  $\prec$  `Config`,  $\prec$  `RequestData`


**Required interfaces:**  $\prec$  `DeviceManagement`,  $\prec$  `Heartbeat`,  $\prec$  `SensorData`

**Deployed on:** MoteNode

**Visible on diagrams:** figs. 1.1, 3.1, 4.2, 4.7, 4.9 and 4.18

### 5.1.40 NotificationHandler

**Responsibility:** The `NotificationHandler` is responsible for sending notifications to the appropriate parties, e.g. system administrators, customer organisations and application developers. The `OtherDataDB` is used to determine in what way the notifications will be sent. When only the trigger is supplied, the `NotificationHandler` will look at end-roles and other relationships to determine the receiver. If applicable, it will act as an interface for confirming deliveries of messages.

**Super-components:**  SIoTIP system

**Sub-components:** None



**Provided interfaces:**  $\neg$  ConfirmDelivery,  $\neg$  NotifyRecipient  
**Required interfaces:**  $\prec$  DeliverNotification,  $\prec$  OtherDataMgmt  
**Deployed on:** OnlineServiceNode  
**Visible on diagrams:** figs. 1.1, 1.2, 3.1, 3.2, 4.4, 4.6, 4.7, 4.9, 4.20 and 4.22

#### 5.1.41 OSSensorDataDB

**Responsibility:** The OSSensorDataDB is responsible for storing sensor data sent by the Gateways. It employs a scheduling policy for database requests.  
**Super-components:**  $\boxplus$  SIoTIP system  
**Sub-components:**  $\boxplus$  SensorDataLoadBalancer,  $\boxplus$  SensorDataScheduler,  $\boxplus$  SensorDataDB  
**Provided interfaces:**  $\neg$  SensorDataMgmt  
**Required interfaces:**  $\prec$  DBRequestResults  
**Deployed on:** OnlineServiceNode, SensorDBNode  
**Visible on diagrams:** figs. 1.2 and 2.3

#### 5.1.42 OSWithGWCommunication

**Responsibility:** The OSWithGWCommunication is a supercomponent responsible for all communication with the Gateways (incoming and outgoing).  
**Super-components:**  $\boxplus$  SIoTIP system  
**Sub-components:**  $\boxplus$  DataReceiver,  $\boxplus$  DeviceStatusMonitor,  $\boxplus$  CommunicationHeartbeatMonitor,  $\boxplus$  ApplicationCommandPropagator  
**Provided interfaces:**  $\neg$  DeviceAvailabilityCheck,  $\neg$  DeviceAvailabilityMessage,  $\neg$  GWToOSCommand,  $\neg$  RcvSensorData,  $\neg$  SendCommandToGW  
**Required interfaces:**  $\prec$  AckMessage,  $\prec$  ApplicationActivation,  $\prec$  DeviceInitCacheMgmt,  $\prec$  IncomingCommand,  $\prec$  NotifyRecipient,  $\prec$  OStoGWCommand,  $\prec$  RouteDeviceData,  $\prec$  SensorDataMgmt,  $\prec$  TopologyMgmt  
**Deployed on:** OnlineServiceNode, CommunicationFailureNode  
**Visible on diagrams:** figs. 1.2 and 2.2

#### 5.1.43 OtherDataDB



**Responsibility:** The OtherDataDB is responsible for storing any data in the SIoTIP system aside from topology information, sensor and session data. For all users this includes account information such as name, credentials and contact information. For each subscription it includes a status, timeframe and which application. It also includes application instance info per application such as the configuration (end-user roles, topology assignment). The sent notifications and their status is also stored here. Any information about applications in the system is also stored (description, payment info, status, developer, default configuration, requirements) in the OtherDataDB. Lastly, hardware information such as the pairing between device models and types, conversion libraries and a listing of the various supported categories/models is also stored here.  
**Super-components:**  $\boxplus$  SIoTIP system  
**Sub-components:** None  
**Provided interfaces:**  $\neg$  OtherDataMgmt  
**Required interfaces:** None  
**Deployed on:** OnlineServiceNode  
**Visible on diagrams:** figs. 1.2, 3.2, 4.4, 4.7, 4.11, 4.20, 4.21, 4.22, 4.23, 4.24, 4.25 and 4.26

#### 5.1.44 SensorDataDB


**Responsibility:** The SensorDataDB is responsible for actually storing the sensor data.  
**Super-components:**  $\boxplus$  SIoTIP system  $\triangleright$   $\boxplus$  OSSensorDataDB  
**Sub-components:** None  
**Provided interfaces:**  $\neg$  SensorDataMgmt  
**Required interfaces:** None  
**Deployed on:** SensorDBNode  
**Visible on diagrams:** figs. 2.3, 3.2 and 4.10



### 5.1.45 SensorDataLoadBalancer

**Responsibility:** The `SensorDataLoadBalancer` is responsible for sending database queries (reads/writes) to the correct server on which part of the sensor data is located.

**Super-components:**  `SIoTIP system`  `OSSensorDataDB`

**Sub-components:** None

**Provided interfaces:**  `SensorDataMgmt`



**Required interfaces:**  `DBRequestResults`,  `SensorDataMgmt`

**Deployed on:** `OnlineServiceNode`


**Visible on diagrams:** figs. 2.3, 3.2 and 4.10


### 5.1.46 SensorDataScheduler

**Responsibility:** The `SensorDataScheduler` is responsible for handling all read/write requests to the `SensorDataDB`. It will monitor the completion times and execute them after scheduling them in an order based off the current modus of the database (normal or overloaded). In normal mode all requests are processed in a first-in-first-out order, in overload mode write requests are prioritised over specific lookup queries which are in turn prioritised over broad lookup queries. Furthermore, while in overload mode, requests from critical application instances have priority over non-critical application instances. Write requests should be handled within 500msec, read requests from critical application instances within 750msec and read requests from non-critical application instances within 1000msec.

**Super-components:**  `SIoTIP system`  `OSSensorDataDB`

**Sub-components:** None

**Provided interfaces:**  `SensorDataMgmt`


**Required interfaces:**  `SensorDataMgmt`

**Deployed on:** `SensorDBNode`

**Visible on diagrams:** figs. 2.3, 3.2 and 4.10

### 5.1.47 SessionDB

**Responsibility:** The `SessionDB` is responsible for storing any data on open sessions of users.

**Super-components:**  `SIoTIP system`

**Sub-components:** None

**Provided interfaces:**  `SessionMgmt`

**Required interfaces:** None








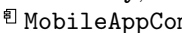
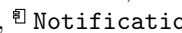
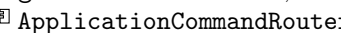
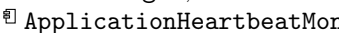
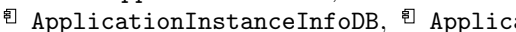
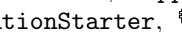
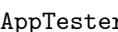
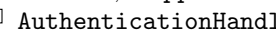
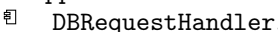
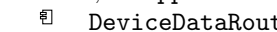


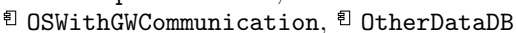
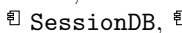
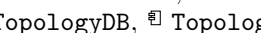
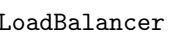

**Deployed on:** `OnlineServiceNode`




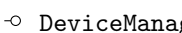
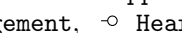
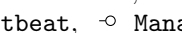
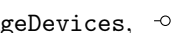
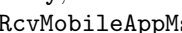

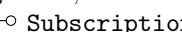
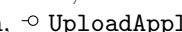
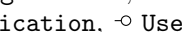
**Visible on diagrams:** figs. 1.2 and 3.2






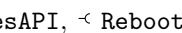

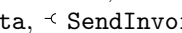
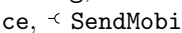
### 5.1.48 SIoTIP system

**Responsibility:** The `SIoTIP system` component is the highest level supercomponent representing the system as a whole.

**Super-components:** None

**Sub-components:**  `ApplicationDeveloperDashboard`,  `InfrastructureOwnerDashboard`,  `SysAdminDashboard`,  `Gateway`,  `ApplicationHandler`,  `CustomerOrganisationDashboard`,  `InvoiceManager`,  `MobileAppCommunication`,  `NotificationHandler`,  `ApplicationCommandRouter`,  `ApplicationHeartbeatMon`,  `ApplicationInstanceInfoDB`,  `ApplicationStarter`,  `AppTester`,  `AuthenticationHandler`,  `DBRequestHandler`,  `DeviceDataRouter`,  `DeviceInitCache`,  `OSSensorDataDB`,  `OSWithGWCommunication`,  `OtherDataDB`,  `SessionDB`,  `TopologyDB`,  `TopologyLoadBalancer`

**Provided interfaces:**  `ApplicationInfo`,  `ConfirmDelivery`,  `ConsultNotifications`,  `DeviceManagement`,  `Heartbeat`,  `ManageDevices`,  `RcvMobileAppMsg`,  `Registration`,  `SensorData`,  `Subscription`,  `UploadApplication`,  `UserAuthentication`

**Required interfaces:**  `Actuate`,  `AppProvidesAPI`,  `Config`,  `DeliverNotification`,  `GWAppProvidesAPI`,  `RebootGW`,  `RequestData`,  `SendInvoice`,  `SendMobileAppMsg`

**Deployed on:** `OnlineServiceNode`, `ApplicationExecutionNode`, `GatewayNode`, `TopologyNode`, `SensorDBNode`, `ApplicationFailureNode`, `CommunicationFailureNode`, `MobileAppCommunicationNode`, `AppTestingNode`, `DashboardsNode`

**Visible on diagrams:** fig. 1.1

### 5.1.49 SysAdminClient

**Responsibility:** The SysAdminClient is external to the SIoTIP system and represents the client of a SIoTIP system administrator that communicates with the SIoTIP system.

**Super-components:** None

**Sub-components:** None

**Provided interfaces:** None

**Required interfaces:** < ConsultNotifications, < UserAuthentication

**Deployed on:** SysAdminNode

**Visible on diagrams:** figs. 1.1 and 3.1

### 5.1.50 SysAdminDashboard

**Responsibility:** The SysAdminDashboard provides the main interface to all SIoTIP system administrators, it is currently only used to consult notifications.

**Super-components:** SIoTIP system

**Sub-components:** None

**Provided interfaces:** < ConsultNotifications, < UserAuthentication

**Required interfaces:** < OtherDataMgmt, < UserAuthentication, < VerifySession

**Deployed on:** DashboardsNode

**Visible on diagrams:** figs. 1.1, 1.2, 3.1 and 3.2

### 5.1.51 ThirdPartyInvoicingService

**Responsibility:** The ThirdPartyInvoicingService is external to the SIoTIP system and represents a service that allows SIoTIP to send invoices, for example Zoomit or a credit card company.

**Super-components:** None

**Sub-components:** None

**Provided interfaces:** < SendInvoice

**Required interfaces:** None

**Deployed on:** ThirdPartyInvoicingNode

**Visible on diagrams:** figs. 1.1 and 3.1

### 5.1.52 ThirdPartyNotificationDeliverySystem

**Responsibility:** The ThirdPartyNotificationDeliverySystem is external to the SIoTIP system and represents a system that allows SIoTIP to send notifications via email and SMS.

**Super-components:** None

**Sub-components:** None

**Provided interfaces:** < DeliverNotification

**Required interfaces:** < ConfirmDelivery

**Deployed on:** ThirdPartyNotificationNode

**Visible on diagrams:** figs. 1.1, 3.1 and 4.20

### 5.1.53 TopologyDB

**Responsibility:** Responsible for storing all data related to topologies, access rights and device configurations.

**Super-components:** SIoTIP system

**Sub-components:** None

**Provided interfaces:** < TopologyMgmt

**Required interfaces:** None

**Deployed on:** TopologyNode

**Visible on diagrams:** figs. 1.2, 3.2, 4.6, 4.9 and 4.21

### 5.1.54 TopologyLoadBalancer

**Responsibility:** The TopologyLoadBalancer is responsible for sending queries to the TopologyDB to the correct server, i.e. the server on which the relevant data is stored.

**Super-components:** `SIoTIP` system

**Sub-components:** None

**Provided interfaces:** `TopologyMgmt`

**Required interfaces:** `DBRequestResults`, `DeviceInitCacheMgmt`, `NotifyRecipient`,  
`TopologyMgmt`

**Deployed on:** `OnlineServiceNode`

**Visible on diagrams:** figs. 1.2, 3.2, 4.6, 4.9 and 4.21

## 5.2 Interfaces

### 5.2.1 AckMessage

**Provided by:** `AvailabilityMonitor`, `DataHandler`, `GWCommandRouter`, `Gateway`

**Required by:** `ApplicationCommandPropagator`, `DataReceiver`, `DeviceStatusMonitor`, `OSWithGWCommunication`

**Operations:**

- `void ackMessage(int ackNr)`
  - Effect: The caller will send an acknowledgement over to a communication component of the `Gateway`. This message carries an acknowledgement number which was also in the message which has to be acknowledged. This is to make sure the receiver knows which of its messages was acknowledged such that it can be thrown away.
  - Sequence Diagrams: figs. 4.7, 4.9, 4.10 and 4.17

**Diagrams:** figs. 1.2, 2.1 and 2.2

### 5.2.2 Actuate

**Provided by:** `Mote`

**Required by:** `GWCommandRouter`, `Gateway`, `SIoTIP` system

**Operations:**

- `boolean sendActuationCommand(string commandName)`
  - Effect: Send an actuation command to the actuator. The actuator acknowledges execution of the command. Sending an incorrect actuation command (e.g. ‘take picture’ to a ‘buzzer’) has no effect.
  - Sequence Diagrams: figs. 4.2 and 4.18
- `boolean sendAsyncActuationCommand(string commandName, int requestID)`
  - Effect: Send an actuation command to the actuator (Callback). The actuator acknowledges receipt of the command. Sending an incorrect actuation command (e.g. ‘take picture’ to a ‘buzzer’) has no effect.
  - Sequence Diagrams: None

**Diagrams:** figs. 1.1, 1.2 and 2.1

### 5.2.3 AMReactToFailure

**Provided by:** `AvailabilityMonitor`

**Required by:** `GWFailureHandler`



**Operations:**

- `void reactToCommunicationFailure(bool failed)`
  - Effect: The `AvailabilityMonitor` will start/stop sending messages to the Online System depending on the value of the failed parameter (false/true respectively) as this message signifies the state of the communication channel at that point.
  - Sequence Diagrams: fig. 4.5
- `void restart()`
  - Effect: Causes the `AvailabilityMonitor` component to restart.
  - Sequence Diagrams: None

**Diagrams:** fig. 2.1

### 5.2.4 AppCalls

**Provided by:** `ApplicationCommandRouter`


**Required by:**  `ApplicationContainer`,  `ApplicationHandler`



**Operations:**

- `void applicationCommand(Command cmd)`
  - Effect: The `ApplicationCommandRouter` will route the command based off of the type of command. Database requests will be sent to the `DBRequestHandler`, actuation commands to the communication components, messages to mobile apps to the `MobileAppCommunication` component and requests for device data to the `ApplicationInstanceInfoDB`. For application commands, the validity of the command will be checked (using the **ApplicationRequirements** of various application instances as well as the current configuration of the device) of the various and if reconfiguration is needed, the command is updated with the reconfiguration info.
  - Sequence Diagrams: figs. 4.2 and 4.19

**Diagrams:** figs. 1.2 and 2.4

### 5.2.5 AppContainerMgmt

**Provided by:**  `ApplicationContainer`



**Required by:**  `ApplicationManager`,  `ApplicationScheduler`








**Operations:**

- `void killContainer(ApplicationInstanceID applnID)`
  - Effect: Forces a container (and its stand-in) to shut down, killing the application executing inside. This method is not responsible for making sure the application is killed in a controlled or safe manner, the caller of this method is responsible for making sure of that.
  - Sequence Diagrams: fig. 4.26
- `void startNewContainer(ApplicationInstanceID applnID, Code code, ApplicationRequirements appReq, ConversionCode conversionCode, TopologyRequirementsInstantiation topReqIn)`
  - Effect: Starts a new container running the specified code. The provided application requirements will be used for validity checking of certain commands.
  - Sequence Diagrams: fig. 4.1

**Diagrams:** fig. 2.4

### 5.2.6 AppFailure

**Provided by:**  `ApplicationHeartbeatMonitor`,  `GWApplicationHeartbeatMonitor`




**Required by:**  `ApplicationCommandRouter`,  `ApplicationContainer`,  `ApplicationHandler`,  
 `ApplicationManager`,  `ApplicationScheduler`,  `GWApplicationContainer`,  `GWApplicationScheduler`



**Operations:**

- `void applicationProcedureCrashed(ApplicationInstanceID applnID, CrashInfo crashInfo)`
  - Effect: Informs the relevant scheduler that a procedure of a certain application instance threw an exception/error, implying a crash. The scheduler will inform the application itself of this by scheduling it to receive the info.
  - Sequence Diagrams: fig. 4.3
- `void heartbeat()`
  - Effect: This will cause the a reset of the time for the calling component. Used by the heartbeat monitors to check whether the requiring component is still available.
  - Sequence Diagrams: None

**Diagrams:** figs. 1.2, 2.1 and 2.4

### 5.2.7 AppInstanceChange

**Provided by:**  `ApplicationHandler`,  `ApplicationManager`,  `GWApplicationContainer`

**Required by:**  `ApplicationStarter`,  `GWCommandRouter`

**Operations:**



- `void finalizeApplicationUpdate(ApplicationInstanceID applnID)`
  - Effect: Finalizes an application update. The caller must ensure that the environment is in a state where the update can be finalized. More specifically, this method assumes that the old version of the application is ready to shut down and its containers are already dormant. This method kills the old version's container, adds the entries that are necessary for the new version to the `ApplicationInstanceInfoDB`, activates the new version's container and calls its initialize






procedure. The Online Service version of this method is also responsible for sending the message to the gateways to call the gateway version of this method.

- Sequence Diagrams: fig. 4.25
- **void killContainer(ApplicationInstanceID applnID)**
  - Effect: The called container is deleted along with its contents.
  - Sequence Diagrams: figs. 4.23 and 4.26
- **void startApplicationInstance(ApplicationInstanceID applnID, Code code, ApplicationRequirements appReq, ConversionCode conversionCode, boolean update, TopologyRequirementsInstantiation topReqIn)**
  - Effect: Start a new container for an application or reactivate an existing one (decided by whether there exists a container with the given appInID). The conversionCode is passed along to be used by the container. The update parameter signifies whether routing data should be activated already, if an update of the app is going on, it should not. (First the old container has to be killed).
  - Sequence Diagrams: figs. 4.1 and 4.24
- **void startShutDownProcedure(ApplicationInstanceID oldAppInID)**
  - Effect: Initializes the shut down procedure for an application in the context of an application update. The application is warned of its impending shutdown, but it can still act and postpone its own shutdown to prevent unsafe or inconsistent situations. Eventually, calling this method should result in the application being shut down everywhere, both on the Online Service and on the Gateways.
  - Sequence Diagrams: fig. 4.24
- **void suspendApplicationInstance(ApplicationInstanceID applnID)**
  - Effect: The ApplicationManager is asked to locate and suspend the application container corresponding to the given id.
  - Sequence Diagrams: fig. 4.22

**Diagrams:** figs. 1.2, 2.1 and 2.4

## 5.2.8 AppInstanceMapChange

**Provided by:**  ApplicationInstanceInfoDB,  DataHandler



**Required by:**  ApplicationCommandRouter,  ApplicationHandler,  ApplicationManager,  ApplicationStarter,  GWCommandRouter



**Operations:**

- **void addMapEntry(ApplicationInstanceID applnID, DeviceResourceLocator pRL)**
  - Effect: Adds the specified pluggable device to the list of devices the specified application instance is interested in (and wishes to receive data from).
  - Sequence Diagrams: figs. 4.1 and 4.26
- **void removeMapEntry(ApplicationInstanceID applnID, DeviceResourceLocator pRL)**
  - Effect: Removes the specified pluggable device to the list of devices the specified application instance is interested in (and wishes to receive data from). Nothing happens if this device was not in the list.
  - Sequence Diagrams: fig. 4.26

**Diagrams:** figs. 1.2, 2.1 and 2.4

## 5.2.9 AppInstanceSuspension

**Provided by:**  ApplicationScheduler,  GWApplicationScheduler

**Required by:**  ApplicationManager,  GWCommandRouter

**Operations:**





- **void activateApplicationInstance(ApplicationInstanceID applnID)**
  - Effect: Activates the specified application instance, causing the initialize procedure to be called on the appropriate container.
  - Sequence Diagrams: fig. 4.25
- **void deactivateApplicationInstance(ApplicationInstanceID applnID)**
  - Effect: Initiate the shutdown of an application by sending Commands to gateways and prepareShutdown() calls to application instance's containers.
  - Sequence Diagrams: fig. 4.23
- **void periodicShutdownForInstance(ApplicationInstanceID applnID)**

- Effect: Used during the update of an application, the scheduler will start calling the shutdown procedure on an application until it is ready to be killed and make way for the updated version of the application.
- Sequence Diagrams: fig. 4.24
- void suspendApplicationInstance(**ApplicationInstanceID** applnID)
  - Effect: Marks the specified application instance as suspended, meaning no more scheduling will be done for the instance. The suspend call is done to the application via the appropriate container running the instance.
  - Sequence Diagrams: fig. 4.22

**Diagrams:** figs. 2.1 and 2.4

### 5.2.10 ApplicationActivation

**Provided by:**  ApplicationStarter



**Required by:**  ApplicationDeveloperDashboard,  CustomerOrganisationDashboard,  DeviceStatusMonitor,  OSWithGWCommunication


**Operations:**

- void UpdateApplicationInstances(**Code** code, string description, **ApplicationMetaData** metadata, list<**RoleDescription**> roles, list<**Version**> versions)
  - Effect: Starts the update procedure for all active instances of the application with a version that's in the 'versions' list.
  - Sequence Diagrams: fig. 4.4
- void deactivate(**ApplicationInstanceID** applnID)
  - Effect: Starts the deactivation process of a certain subscription (appInstance). The **ApplicationStarter** will set the process of killing the old applications in motion.
  - Sequence Diagrams: fig. 4.23
- void devicesStatusChange(list<**DeviceResourceLocator**> pRLs, boolean active)
  - Effect: Informs the **ApplicationStarter** of a device that underwent a change that may cause application instances to activate/deactivate. Checks will be started to see if any application instances need to be suspended/reactivated (checks are limited to the instances which have one of the specified devices in their configuration). A list is used as some events (such as mote failure or a topology change) may cause multiple pluggable devices to be changed at once.
  - Sequence Diagrams: figs. 4.6 and 4.7
- void endUserRoleChange(**UserID** cudtOrgID, **UserID** uID, string role, **ApplicationID** applID)
  - Effect: The specified end user for a customer organisation will have a new role assigned to them for a certain app.
  - Sequence Diagrams: None
- void gwDown(**GatewayID** gwID)
  - Effect: The **ApplicationStarter** is responsible for coordinating the update process of applications. In that context, it needs to know about any gateways becoming unavailable. An update procedure for an application needs to be aborted if a gateway goes down that's running a component of the application which needs to be updated, otherwise we might end up in an inconsistent situation where most of the system is running the new version but some gateways are still running the old version. This method is responsible for checking whether any updates must be aborted when a gateway goes down, and initializing that process if it needs to be done.
  - Sequence Diagrams: None
- void startApplication(**ApplicationInstanceID** applnID, **Code** code, **ApplicationRequirements** appReq, **TopologyRequirementsInstantiation** topReqIn)
  - Effect: Will start the process of starting the various parts of an application on the Online Service and the gateways (if applicable).
  - Sequence Diagrams: fig. 4.21

**Diagrams:** figs. 1.2 and 2.2

### 5.2.11 ApplicationInfo

**Provided by:**  ApplicationDeveloperDashboard,  SIOtIP system

**Required by:**  ApplicationDeveloperClient

**Operations:**

- **HTML** `getInfoOnApprovedApp(SessionID sID, ApplicationID appID)`
  - Effect: Fetch detailed statistics on the specified application and will return these to the client.
  - Sequence Diagrams: None
- **HTML** `getOverviewOfUploadedApplications(SessionID sID)`
  - Effect: Provides the application provider with an overview of all their uploaded applications on their dashboard.
  - Sequence Diagrams: None

**Diagrams:** figs. 1.1 and 1.2

### 5.2.12 ApplicationInstanceLookup

**Provided by:** `ApplicationInstanceInfoDB`

**Required by:** `DeviceDataRouter`

**Operations:**

- `list<ApplicationInstanceID> getInstancesInterestedInDevice(PluggableDeviceID devID)`
  - Effect: Return the list of **ApplicationInstanceID**'s that correspond to application instances that are currently interested in getting updates from a device.
  - Sequence Diagrams: fig. 4.15

**Diagrams:** fig. 1.2

### 5.2.13 AppProvidesAPI

**Provided by:** `Application`

**Required by:** `ApplicationContainer`, `ApplicationHandler`, `SIoTIP system`



**Operations:**



- `list<Command> handleCrash(CrashInfo error)` throws *ApplicationUncaughtException*
  - Effect: When an application procedure crashes, its effects are not executed and the application should still be in a workable state. The application is notified of the fact that one of its procedures crashed by calling this procedure with information about the crash. This allows the application to respond to the event.
  - Sequence Diagrams: fig. 4.3
- `list<Command> initialize(Code code)`
  - Effect: The application is started with the given code.
  - Sequence Diagrams: None
- `list<Command> pluggableDeviceFailed(DeviceResourceLocator pRL)` throws *ApplicationUncaughtException*
  - Effect: Applications should be informed when a device they were using has crashed, this is done via a function call on the application.
  - Sequence Diagrams: fig. 4.27
- `Tuple<boolean, list<Command>> prepareShutdown()` throws *ApplicationUncaughtException*
  - Effect: When this procedure is called, the system requests the application to prepare itself for an impending shutdown. The application can return a list of commands it wants to see executed before it shuts down, and a boolean indicating whether it is ready to shut down.
  - Sequence Diagrams: fig. 4.8
- `list<Command> reactToCommand(Command cmd)` throws *ApplicationUncaughtException*
  - Effect: Pass the provided command to the application.
  - Sequence Diagrams: fig. 4.14
- `list<Command> reactToData(PluggableDeviceID pID, SystemSensorData data)` throws *ApplicationUncaughtException*
  - Effect: Pass the provided sensor data to the application so that it can process it.
  - Sequence Diagrams: fig. 4.15
- `list<Command> wakeUp()` throws *ApplicationUncaughtException*
  - Effect: Wake up the application.
  - Sequence Diagrams: None

**Diagrams:** figs. 1.1, 1.2 and 2.4



### 5.2.14 CallThroughContainer

**Provided by:**  ApplicationContainer,  GWApplicationContainer


**Required by:**  ApplicationScheduler,  GWApplicationScheduler



**Operations:**

- void handleCrash(**CrashInfo** error) throws *ApplicationUncaughtException*
  - Effect: When an application procedure crashes, its effects are not executed and the application should still be in a workable state. The application is notified of the fact that one of its procedures crashed by calling this procedure with information about the crash. This allows the application to respond to the event.
  - Sequence Diagrams: fig. 4.3
- void initialize(**Code** code)
  - Effect: The application starts to run with the code given to its container.
  - Sequence Diagrams: figs. 4.1 and 4.25
- void pluggableDeviceFailed(**DeviceResourceLocator** pRL) throws *ApplicationUncaughtException*
  - Effect: See AppProvidesAPI::pluggableDeviceFailed
  - Sequence Diagrams: fig. 4.27
- Tuple<boolean, list<**Command**>> prepareShutdown() throws *ApplicationUncaughtException*
  - Effect: When this procedure is called, the system requests the application to prepare itself for an impending shutdown. The application can return a list of commands it wants to see executed before it shuts down, and a boolean indicating whether it is ready to shut down.
  - Sequence Diagrams: fig. 4.8
- void reactToCommand(**Command** cmd) throws *ApplicationUncaughtException*
  - Effect: Pass the provided command to the application.
  - Sequence Diagrams: figs. 4.13 and 4.14
- void reactToData(**PluggableDeviceID** plD, **SystemSensorData** data) throws *ApplicationUncaughtException*
  - Effect: Pass the provided sensor data to the application so that it can process it.
  - Sequence Diagrams: figs. 4.10 and 4.15
- void wakeUp() throws *ApplicationUncaughtException*
  - Effect: Wake up the application.
  - Sequence Diagrams: None

**Diagrams:** figs. 2.1 and 2.4

### 5.2.15 ChangeSyncTimer

**Provided by:**  DataReceiver


**Required by:**  ApplicationCommandPropagator,  DeviceStatusMonitor




**Operations:**

- void checkGWSyncInterval()
  - Effect: Informs of a possible change of the expected synchronization interval for a gateway. This change may be needed due to an **Application** command changing the update frequency of a device, installation of a new device with a shorter synchronization period, or a failing of the previous device with the shortest synchronization period.
  - Sequence Diagrams: None

**Diagrams:** fig. 2.2

### 5.2.16 CommHeartbeat

**Provided by:**  CommunicationHeartbeatMonitor


**Required by:**  ApplicationCommandPropagator,  DataReceiver,  DeviceStatusMonitor




**Operations:**

- void heartbeat()
  - Effect: This will cause the **CommunicationHeartbeatMonitor** to reset the time for the calling component. Used by the **CommunicationHeartbeatMonitor** to check whether the requiring component is still available.
  - Sequence Diagrams: None

**Diagrams:** fig. 2.2

### 5.2.17 Config

**Provided by:**  Mote



**Required by:**  GWCommandRouter,  Gateway,  SIOtIP system

**Operations:**

- `Map<string, string> getConfig()`
  - Effect: Returns the current configuration of a PluggableDevice as a map.
  - Sequence Diagrams: None
- `void setConfig(Map<string, string> config)`
  - Effect: Changes to configuration of the PluggableDevice. Sending values to an incorrect PluggableDevice (e.g., ‘resolution’, ‘640x480’ to the buzzer) has no effect.
  - Sequence Diagrams: None

**Diagrams:** figs. 1.1, 1.2 and 2.1

### 5.2.18 ConfirmDelivery

**Provided by:**  NotificationHandler,  SIOtIP system






**Required by:**  ThirdPartyNotificationDeliverySystem





**Operations:**

- `void acknowledgeSMSDelivery(int messageId)`
  - Effect: Causes the NotificationHandler to mark the SMS message with the matching messageId as "sent" in the OtherDataDB.
  - Sequence Diagrams: fig. 4.20

**Diagrams:** figs. 1.1 and 1.2

### 5.2.19 ConsultNotifications

**Provided by:**  ApplicationDeveloperDashboard,  CustomerOrganisationDashboard,  InfrastructureOwnerDashboard,  SIOtIP system,  SysAdminDashboard

**Required by:**  ApplicationDeveloperClient,  CustomerOrganisationClient,  InfrastructureOwnerClient,  SysAdminClient


**Operations:**

- **HTML** `displayNotificationsForUser(SessionID sID)`
  - Effect: The dashboard will fetch the notifications for the user with the **UserID** contained in the **SessionID** from the OtherDataDB. It will ask the front-end to display these.
  - Sequence Diagrams: None
- **HTML** `giveNotificationDetails(SessionID sID, NotificationID nID)`
  - Effect: The dashboard will fetch the details of the identified notification from the OtherDataDB and will ask the front-end to display these.
  - Sequence Diagrams: None

**Diagrams:** figs. 1.1 and 1.2

### 5.2.20 CRReactToFailure

**Provided by:**  GWCommandRouter

**Required by:**  GWFailureHandler

**Operations:**

- `void reactToCommunicationFailure(bool failed)`
  - Effect: The component will start/stop sending messages to the Online System depending on the value of the failed parameter (false/true respectively) as this message signifies the state of the communication channel at that point.
  - Sequence Diagrams: fig. 4.5
- `void restart()`
  - Effect: Causes the GWCommandRouter component to restart.
  - Sequence Diagrams: None

**Diagrams:** fig. 2.1

### 5.2.21 DBRequestResults

**Provided by:** `DBRequestHandler`

**Required by:** `OSSensorDataDB`, `SensorDataLoadBalancer`, `TopologyLoadBalancer`

**Operations:**

- `void queryResults(Command cmd)`
  - Effect: The `DBRequestHandler` will schedule the results of a certain query to be sent to the application instance that requested them.
  - Sequence Diagrams: None

**Diagrams:** figs. 1.2 and 2.3

### 5.2.22 DeliverMsgMobileApp

**Provided by:** `MobileAppCommunication`

**Required by:** `ApplicationCommandRouter`

**Operations:**

- `void sendToMobileApp(string hostname, int port, string message)`
  - Effect: A message is sent to the specified hostname/port combination.
  - Sequence Diagrams: fig. 4.12

**Diagrams:** fig. 1.2

### 5.2.23 DeliverNotification

**Provided by:** `ThirdPartyNotificationDeliverySystem`

**Required by:** `NotificationHandler`, `SIoTIP system`

**Operations:**

- `void sendEmail(string emailaddress, string message)`
  - Effect: Will cause the third-party messaging system to send an email with the specified message to the specified email address.
  - Sequence Diagrams: fig. 4.20
- `void sendSMS(int phonenumber, string message, int messageid)`
  - Effect: Will cause the third-party messaging system to send an sms with the specified message to the specified number.
  - Sequence Diagrams: fig. 4.20

**Diagrams:** figs. 1.1 and 1.2

### 5.2.24 DeviceAvailabilityCheck

**Provided by:** `DeviceStatusMonitor`, `OSWithGWCommunication`

**Required by:** `ApplicationCommandRouter`, `ApplicationStarter`

**Operations:**

- `boolean isAnyOfTheseDevicesAvailable(list<DeviceResourceLocator> devices)`
  - Effect: Returns true if any of the specified devices is marked as available in the `DeviceStatusMonitor` and false otherwise.
  - Sequence Diagrams: fig. 4.7
- `boolean isDeviceAvailable(DeviceResourceLocator pRL)`
  - Effect: Returns true if the specified device is marked as available in the `DeviceStatusMonitor` and false otherwise.
  - Sequence Diagrams: fig. 4.18

**Diagrams:** figs. 1.2 and 2.2

### 5.2.25 DeviceAvailabilityMessage

**Provided by:** `DeviceStatusMonitor`, `OSWithGWCommunication`

**Required by:** `AvailabilityMonitor`, `Gateway`

**Operations:**

- `void moteAvailable(MoteInfo motelInfo, int ackNr)`
  - Effect: Message will be propagated to the `TopologyDB` together with the ID of the gateway this message came from. The returned list of pluggable devices will then be propagated to the

**ApplicationStarter** and a notification will be sent to the owner of the infrastructure this mote belongs.

– Sequence Diagrams: None

- **void moteUnavailable(MotefInfo motefInfo, int ackNr)**

– Effect: Message will be propagated to the **TopologyDB** together with the ID of the gateway this message came from. The returned list of pluggable devices will then be propagated to the **ApplicationStarter** and a notification will be sent to the infrastructure owner this mote belongs to.

– Sequence Diagrams: None

- **void newMote(MotefInfo motefInfo, int ackNr)**

– Effect: The message (along with information about the gateway/topology this message came from) will be propagated to the **TopologyDB**.

– Sequence Diagrams: None

- **void newPluggableDeviceInMote(int motefID, PluggableDeviceID pID, PluggableDeviceType pType, int ackNr)**

– Effect: Message will be propagated to the **TopologyDB** together with the ID of the gateway this message came from.

– Sequence Diagrams: fig. 4.9

- **void pluggableDeviceAvailable(PluggableDeviceID pID, MotefID mID, int ackNr)**

– Effect: Informs the **DeviceStatusMonitor** that a pluggable device on a certain mote has become available. This will be marked as such and then propagated to other components (such as the **ApplicationStarter**) which might be interested in this information.

– Sequence Diagrams: None

- **void pluggableDeviceUnavailable(PluggableDeviceID pID, MotefID mID, int ackNr)**




– Effect: Informs the **DeviceStatusMonitor** that a pluggable device on a certain mote has become unavailable. This will be marked as such and then propagated to other components (such as the **ApplicationStarter**) which might be interested in this information.

– Sequence Diagrams: fig. 4.7

**Diagrams:** figs. 1.2, 2.1 and 2.2

## 5.2.26 DeviceInitCacheMgmt

**Provided by:**  **DeviceInitCache**

**Required by:**  **DataReceiver**,  **OSWithGWCommunication**,  **TopologyLoadBalancer**

**Operations:**

- **boolean isDeviceInitialized(DeviceResourceLocator pRL)**

– Effect: The **DeviceInitCache** returns whether the given device is marked "initialized".

– Sequence Diagrams: fig. 4.10




- **void setDeviceInitialized(DeviceResourceLocator pRL)**


– Effect: The **DeviceInitCache** marks the given device as "initialized"

– Sequence Diagrams: None

**Diagrams:** figs. 1.2 and 2.2

## 5.2.27 DeviceManagement

**Provided by:**  **AvailabilityMonitor**,  **Gateway**,  **SIOtIP system**

**Required by:**  **Mote**

**Operations:**

- **void pluggableDevicePluggedIn(MotefInfo mInfo, PluggableDeviceID pID)**

– Effect: Notify the gateway that a new **PluggableDevice** is connected.

– Sequence Diagrams: fig. 4.9


- **void pluggableDeviceRemoved(PluggableDeviceID pID)**


– Effect: Notify the gateway that a **PluggableDevice** is removed.

– Sequence Diagrams: None

**Diagrams:** figs. 1.1, 1.2 and 2.1

### 5.2.28 DHReactToFailure

Provided by:  DataHandler



Required by:  GWFailureHandler


Operations:

- void reactToCommunicationFailure(bool failed)
  - Effect: The DataHandler will either start storing all messages for the gateway in the GWTempDataDB or stop doing so and clean it out depending on the value of the boolean.
  - Sequence Diagrams: fig. 4.5
- void restart()
  - Effect: Causes the DataHandler component to restart.
  - Sequence Diagrams: None

Diagrams: fig. 2.1

### 5.2.29 Failure

Provided by:  ApplicationHandler,  ApplicationManager


Required by:  ApplicationHeartbeatMonitor




Operations:

- void containerFailed(**ApplicationInstanceID** applnID)
  - Effect: Initiate the start of a new ApplicationContainer for the application instance associated with the crashed container.
  - Sequence Diagrams: None
- void serverFailed(list<**ApplicationInstanceID**> applnIDs)
  - Effect: Restart the crashed server and all active application instances.
  - Sequence Diagrams: None

Diagrams: figs. 1.2 and 2.4

### 5.2.30 FailureDetection

Provided by:  GWFailureHandler

Required by:  AvailabilityMonitor,  DataHandler,  GWCommandRouter

Operations:

- void internalHeartbeat()
  - Effect: The GWFailureHandler will use the heartbeats it receives to check whether the communication components are still available.
  - Sequence Diagrams: None
- void noAckReceived()
  - Effect: The GWFailureHandler will assume there is something wrong with either a communication component on the Online Service or the communication channel and will order the other Communication components to store their messages in the GWTempDataDB.
  - Sequence Diagrams: None

Diagrams: fig. 2.1

### 5.2.31 GWAppCalls

Provided by:  GWCommandRouter


Required by:  GWApplicationContainer

Operations:

- void applicationCommand(**Command** cmd)
  - Effect: Route the command based on its type. Actuation commands to the corresponding actuator, application commands to the required part of the application instance. For application commands, the validity of the command will be checked (using the **ApplicationRequirements** of various application instances as well as the current configuration of the device) of the various and if reconfiguration is needed, the command is updated with the reconfiguration info.
  - Sequence Diagrams: fig. 4.13

Diagrams: fig. 2.1

### 5.2.32 GWAppChange

Provided by:  ApplicationCommandRouter


Required by:  ApplicationStarter




Operations:

- void sendGWAppChangeCommand(**Command** cmd)
  - Effect: Sends a command to a gateway informing the application execution system that an application needs to be suspended/updated/started/needs to start shutdown of old container, along with the data required to do so (e.g. **ApplicationInstanceID**, **ApplicationRequirements**, **ConversionCode** and **Code** for starting an application). The update parameter is also passed here (see AppInstanceChanges' startApplicationContainer function).
  - Sequence Diagrams: figs. 4.1, 4.22, 4.23, 4.24, 4.25 and 4.26

Diagrams: fig. 1.2

### 5.2.33 GWAppProvidesAPI

Provided by:  GWApplication

Required by:  GWApplicationContainer,  Gateway,  SIoTIP system


Operations:

- list<**Command**> handleCrash(**CrashInfo** error) throws *ApplicationUncaughtException*
  - Effect: When an application procedure crashes, its effects are not executed and the application should still be in a workable state. The application is notified of the fact that one of its procedures crashed by calling this procedure with information about the crash. This allows the application to respond to the event.
  - Sequence Diagrams: None
- list<**Command**> initialize(**Code** code)
  - Effect: The application is started with the given code.
  - Sequence Diagrams: None
- list<**Command**> pluggableDeviceFailed(**DeviceResourceLocator** pRL) throws *ApplicationUncaughtException*
  - Effect: Applications should be informed when a device they were using has crashed, this is done via a function call on the application.
  - Sequence Diagrams: None
- Tuple<boolean, list<**Command**>> prepareShutdown() throws *ApplicationUncaughtException*
  - Effect: When this procedure is called, the system requests the application to prepare itself for an impending shutdown. The application can return a list of commands it wants to see executed before it shuts down, and a boolean indicating whether it is ready to shut down.
  - Sequence Diagrams: None
- list<**Command**> reactToCommand(**Command** cmd) throws *ApplicationUncaughtException*
  - Effect: Pass the provided command to the application.
  - Sequence Diagrams: fig. 4.13
- list<**Command**> reactToData(**PluggableDeviceID** plID, **SystemSensorData** data) throws *ApplicationUncaughtException*
  - Effect: Pass the provided sensor data to the application so that it can process it.
  - Sequence Diagrams: fig. 4.10
- list<**Command**> wakeUp() throws *ApplicationUncaughtException*
  - Effect: Wake up the application.
  - Sequence Diagrams: None

Diagrams: figs. 1.1, 1.2 and 2.1

### 5.2.34 GWAvailability

Provided by:  DeviceStatusMonitor

Required by:  DataReceiver


Operations:



- void gatewayFailed(**GatewayID** gwID)
  - Effect: Register the fact that a Gateway has failed in the DeviceStatusMonitor. The DeviceStatusMonitor will inform all interested parties of this event.
  - Sequence Diagrams: fig. 4.6

- void registerNewGateway(**GatewayID** gatewayID, **IPAddress** ipAddress)
  - Effect: The DeviceStatusMonitor will register the new gateway and its IP address internally.
  - Sequence Diagrams: None

Diagrams: fig. 2.2

### 5.2.35 GWTempDataMgmt

Provided by:  GWTempDataDB



Required by:  DataHandler,  GWCommandRouter



Operations:

- void addCommand(**Command** cmd)
  - Effect: Temporarily add a message/command to the gateway's storage. (This should only occur in case of a communication channel failure or failure of a communication component on the OS).
  - Sequence Diagrams: None
- void addData(**DeviceData** data, **PluggableDeviceID** pID)
  - Effect: Temporarily add a sensor reading to the gateway's storage. (This should only occur in case of a communication channel failure or failure of a communication component on the OS).
  - Sequence Diagrams: None
- list<**Command**> getCommand()
  - Effect: Return a list of all messages/commands stored in the GWTempDataDB and delete them from the GWTempDataDB.
  - Sequence Diagrams: None
- list<Tuple<**PluggableDeviceID**, **DeviceData**>> getData()
  - Effect: Return a list of all sensor data stored in the GWTempDataDB and delete them from the GWTempDataDB.
  - Sequence Diagrams: None

Diagrams: fig. 2.1

### 5.2.36 GWToOSCommand

Provided by:  ApplicationCommandPropagator,  OSWithGWCommunication


Required by:  GWCommandRouter,  Gateway


Operations:

- void routeCommandInGW(**Command** cmd, int ackNr)
  - Effect: The ApplicationCommandPropagator will propagate the command to the ApplicationCommandRouter and will then send an acknowledgement back to the sender.
  - Sequence Diagrams: fig. 4.17

Diagrams: figs. 1.2, 2.1 and 2.2

### 5.2.37 HandleDBRequest

Provided by:  DBRequestHandler




Required by:  ApplicationCommandRouter


Operations:

- void handleDBRequest(**Command** cmd)
  - Effect: Process the database request command.
  - Sequence Diagrams: None

Diagrams: fig. 1.2

### 5.2.38 Heartbeat

Provided by:  AvailabilityMonitor,  Gateway,  SIO TIP system

Required by:  Mote


Operations:




- void heartbeat(**MoteInfo** moteinfo, List<Tuple<**PluggableDeviceID**, **PluggableDeviceType**>> pds)
  - Effect: Periodic heartbeat from the mote to the gateway, including a list of the PluggableDevices and their DeviceTypes (i.e. those currently plugged into the mote)

- Sequence Diagrams: fig. 4.7

**Diagrams:** figs. 1.1, 1.2 and 2.1

### 5.2.39 IncomingCommand

**Provided by:**  ApplicationCommandRouter

**Required by:**  ApplicationCommandPropagator,  MobileAppCommunication,  OSWithGWCommunication

**Operations:**


- void routeIncomingCommand(**Command** cmd) throws *DestinationNotAvailableException*
  - Effect: The ApplicationCommandRouter determines which application instance or other component a command is intended for (from the **Command** information), and makes sure the scheduler schedules the application's reactToCommand procedure or the other component receives the correct call.

- Sequence Diagrams: figs. 4.17 and 4.18

**Diagrams:** figs. 1.2 and 2.2

### 5.2.40 InternalAppActivation

**Provided by:**  ApplicationStarter

**Required by:**  ApplicationHeartbeatMonitor


**Operations:**

- void suspendApplication(**ApplicationInstanceID** applnID)
  - Effect: Informs the ApplicationStarter that an application has crashed on a critical procedure and that suspension has to be initiated.
  - Sequence Diagrams: fig. 4.3

**Diagrams:** fig. 1.2

### 5.2.41 IPLookup

**Provided by:**  DeviceStatusMonitor

**Required by:**  ApplicationCommandPropagator


**Operations:**

- **IPAddress** getGatewayIPAddress(**GatewayID** gwID)
  - Effect: The DeviceStatusMonitor returns the IP address associated with the specified gateway.
  - Sequence Diagrams: figs. 4.2, 4.16 and 4.22

**Diagrams:** fig. 2.2

### 5.2.42 ManageDevices

**Provided by:**  InfrastructureOwnerDashboard,  SIOtIP system

**Required by:**  InfrastructureOwnerClient

**Operations:**

- **HTML** changeTopologyTo(**SessionID** sID, **Topology** newTopology)
  - Effect: Will ask the front end to display the new topology.
  - Sequence Diagrams: None
- **HTML** doneChangingTopology()
  - Effect: Will propagate the last topology submitted by this user for storage in the TopologyDB. If there are still unplaced motes and/or pluggable devices, the front end will be asked to inform the user of this.
  - Sequence Diagrams: None
- **HTML** doneConsultingDeviceOverview(**SessionID** sID)
  - Effect: The InfrastructureOwnerDashboard will ask the front end to display the **Topology** overview again.
  - Sequence Diagrams: None
- **HTML** getDetailedInfoAboutDevice(**PluggableDeviceID** pID)
  - Effect: The InfrastructureOwnerDashboard will get information about the status of the selected gateway/mote/pluggable device from the OtherDataDB.
  - Sequence Diagrams: None



- **HTML** `grantAccess(SessionID sID, UserID custOrgID)`
  - Effect: Forward the access right (customer organisation gains access to a specific device) to the `TopologyDB`.
  - Sequence Diagrams: None
- **HTML** `wantToConfigureAccessRights(SessionID sID)`
  - Effect: The `InfrastructureOwnerDashboard` will fetch the list of devices associated with the user (according to the `UserID`) and will ask the front end to present this list.
  - Sequence Diagrams: None
- **HTML** `wantToConfigureAccessRightsOfDevice(SessionID sID, PluggableDeviceID pID)`
  - Effect: The `InfrastructureOwnerDashboard` will fetch the list of customer organisations associated with the specified device, as well as their current access rights (based on a request to the `TopologyDB` about which of those currently have access). It will then ask the front-end to display this information.
  - Sequence Diagrams: None
- **HTML** `wantToConfigureTopology(SessionID sID)`
  - Effect: The `InfrastructureOwnerDashboard` will fetch the right (corresponding to the current user) topology from the `TopologyDB` and will ask the front end to display this information.
  - Sequence Diagrams: None

**Diagrams:** figs. 1.1 and 1.2

### 5.2.43 NotifyRecipient

**Provided by:** `NotificationHandler`

**Required by:** `ApplicationDeveloperDashboard`, `ApplicationHeartbeatMonitor`, `ApplicationStarter`, `DeviceStatusMonitor`, `OSWithGWCommunication`, `TopologyLoadBalancer`

**Operations:**

- `void notify(NotificationTrigger trigger)`
  - Effect: Places a notification as ‘unread’ in the inbox of the recipient, and may send out an SMS or e-mail alert depending on the recipient’s preferences. The notification configurations will be looked up based off the information in the `NotificationTrigger`.
  - Sequence Diagrams: figs. 4.4, 4.6, 4.7, 4.9 and 4.22

**Diagrams:** figs. 1.2 and 2.2

### 5.2.44 OStoGWCommand

**Provided by:** `GWCommandRouter`, `Gateway`

**Required by:** `ApplicationCommandPropagator`, `OSWithGWCommunication`

**Operations:**

- `void sendCommand(Command cmd)`
  - Effect: Sending a command (actuation or for another part of the application) to a gateway. The command will be routed to either the scheduler or a mote. If it’s a reconfiguration command, a new command directed to the sender of this command is created with the info that the reconfiguration worked.
  - Sequence Diagrams: figs. 4.2, 4.13, 4.16 and 4.22

**Diagrams:** figs. 1.2, 2.1 and 2.2

### 5.2.45 OtherDataMgmt

**Provided by:** `OtherDataDB`

**Required by:** `ApplicationCommandRouter`, `ApplicationDeveloperDashboard`, `ApplicationHandler`, `ApplicationManager`, `ApplicationStarter`, `AuthenticationHandler`, `CustomerOrganisationDashboard`, `InvoiceManager`, `NotificationHandler`, `SysAdminDashboard`

**Operations:**

- `ApplicationID addApplication(UserID appDevID, Code code, string description, Version version, ApplicationMetadata metadata)`
  - Effect: Add the new application to the store by putting it in the database and marking it as having been tested.
  - Sequence Diagrams: None



- **void addContactInfo(**UserID** uID, **ContactInfo** contactInfo)**
  - Effect: Associate the contactInfo to the user identified by userID.
  - Sequence Diagrams: fig. 4.11
- **UserID addCustomerOrganisation(string userName, string password, **ContactInfo** contactInfo, **PaymentInfo** paymentInfo)**
  - Effect: Add the information of the new customer organisation to the database, generates a new **UserID** and associate the new organisation with that id. The generated **UserID** is returned.
  - Sequence Diagrams: None
- **void addDeviceLogForDevice(**Timestamp** ts, **DeviceResourceLocator** pRL, string description)**
  - Effect: Add a log entry for a device describing an availability event (e.g. failed) for a specific pluggable device.
  - Sequence Diagrams: None
- **void addDeviceLogForGateway(**Timestamp** ts, **GatewayID** gwID, string description)**
  - Effect: Add a log entry for a device describing an availability event (e.g. failed) for a specific gateway.
  - Sequence Diagrams: None
- **void addDeviceLogForMote(**Timestamp** ts, **MotelID** motelID, string description)**
  - Effect: Adds a log entry for a device describing an availability event (e.g. failed) for a specific mote.
  - Sequence Diagrams: None
- **boolean addEndUser(**UserID** custOrgID, string userName, **ContactInfo** contactInfo)** throws *UserAlreadyExistsException*
  - Effect: Store the provided information as a new end user for the identified customer organisation. Return true if successful, false otherwise.
  - Sequence Diagrams: None
- **NotificationID addNotification(**UserID** uID, string msg, **Timestamp** ts, **NotificationTrigger** trigger, **NotificationStatus** status, **CommunicationChannel** channel)** throws *NoSuchUserException*
  - Effect: Adds a notification to the database with the given info for the specified user with uID.
  - Sequence Diagrams: fig. 4.20
- **void addSubscription(**ApplicationID** applID, **UserID** custOrgID, **Version** appVersion, list<Tuple<**UserID**, string>> userRoles, **TopologyRequirementsInstantiation** config, **Timestamp** timestamp)**
  - Effect: Store the Customer Organisations' subscription and its associated data.
  - Sequence Diagrams: fig. 4.21
- **boolean checkEmailAlreadyInUseByOrg(string email)**
  - Effect: Return true if the email address is already registered for a Customer Organisation in the database, and false otherwise.
  - Sequence Diagrams: None
- **boolean checkSubscription(**UserID** custOrgID, **ApplicationID** applID)**
  - Effect: Return true if there is already a subscription of a Customer Organisation for an application (no matter which version)
  - Sequence Diagrams: fig. 4.21
- **Version checkSubscriptionVersion(**UserID** custOrgID, **ApplicationID** applID)** throws *NoSuchSubscription*
  - Effect: Return the version of the current subscription for a Customer Organisation for an application. If there is no such subscription, an error is thrown.
  - Sequence Diagrams: None
- **boolean checkUserNameExists(string userName)**
  - Effect: Return true if there already exists a Customer Organisation using the given string as username, false otherwise.
  - Sequence Diagrams: None
- **ApplicationStatistics detailedStatisticsForApplication(**ApplicationID** applID)**
  - Effect: Return statistics related to a certain application, e.g. number of subscriptions.
  - Sequence Diagrams: None
- **list<Tuple<**Timestamp**, string>> getAllLogsForDevice(**DeviceResourceLocator** pRL)**
  - Effect: Return all availability log entries for the specified pluggable device.
  - Sequence Diagrams: None
- **list<Tuple<**Timestamp**, string>> getAllLogsForGateway(**GatewayID** gwID)**
  - Effect: Return all availability log entries for the specified Gateway.
  - Sequence Diagrams: None


- `list<Tuple<Timestamp, string>> getAllLogsForMote(MotelID mID)`
  - Effect: Return all availability log entries for the specified mote.
  - Sequence Diagrams: None
- `Tuple<UserID, Code, string, Version, ApplicationMetaData> getApplicationInfo(ApplicationID applID)` throws *NoSuchApplication*
  - Effect: Return info on the specified application.
  - Sequence Diagrams: None
- `ApplicationRequirements getApplicationRequirements(ApplicationID applID, Version version)`
  - Effect: Return the application requirements for a certain version of an application.
  - Sequence Diagrams: fig. 4.21
- `list<ApplicationMetaData, int, string> getAppsForAppDeveloper(UserID uID)` throws *NoSuchUserException*
  - Effect: Return the information on all of the applications of a certain application developer, along with the amount of subscribers and a possible note for each of those applications.
  - Sequence Diagrams: fig. 4.4
- `list<ApplicationMetaData> getAppsNotSubscribedOrNewVersionAvailable(UserID custOrgID)`
  - Effect: Return the **ApplicationMetaData** of all application for which the given user does not have an active subscription or for which the user has a subscription to an older version.
  - Sequence Diagrams: fig. 4.21
- `list<ApplicationMetaData> getAppsSubscribed(UserID custOrgID)` throws *NoSuchUserException*
  - Effect: Return the **ApplicationMetaData** of all applications for which the given user has an active subscription.
  - Sequence Diagrams: None
- `Tuple<ContactInfo, UserID> getContactInfoForAppInstancesOwner(ApplicationInstanceID applnID)` throws *NoSuchApplicationInstance*
  - Effect: Return the **ContactInfo** and **UserID** associated with the Customer Organisation which owns the given application instance.
  - Sequence Diagrams: fig. 4.20
- `Tuple<ContactInfo, UserID> getContactInfoForInfrastructuresOwner(InfrastructureID inflID)` throws *NoSuchInfrastructure*
  - Effect: Return the **ContactInfo** and **UserID** associated with the owner of the given infrastructure.
  - Sequence Diagrams: fig. 4.20
- `ContactInfo getContactInfoForUser(UserID uID)` throws *NoSuchUserException*
  - Effect: Return the **ContactInfo** associated with the given user.
  - Sequence Diagrams: fig. 4.20
- `Tuple<Timestamp, TopologyRequirementsInstantiation, list<Tuple<UserID, string>>> getCurrentSubscriptionInfo(ApplicationInstanceID applnID)` throws *NoSuchApplicationInstance*
  - Effect: Get the subscription info from a currently active subscription with the given id.
  - Sequence Diagrams: None
- `list<Tuple<UserID, string, ContactInfo>> getCustOrgEndUsers(UserID CustOrgID)` throws *NoSuchUserException*
  - Effect: Return a list of tuples containing information on the various end users of a Customer Organisation.
  - Sequence Diagrams: fig. 4.21
- `list<UserID> getCustomerOrganisationsAssociatedWithInfrastructureOwner()` throws *NoSuchUserException*
  - Effect: Return all Customer Organisations who are associated with the specified Infrastructure Owner, i.e. the organisations have an application running in the infrastructure of the owner.
  - Sequence Diagrams: None
- `void getDeviceSpecifications(PluggableDeviceType type)` throws *NoSuchPluggableDeviceType*
  - Effect: Return the Specifications for the specified device type. This includes information on the category of the device, the limitations (e.g. max sampling frequency).
  - Sequence Diagrams: None
- `list<GatewayID> getGatewaysOnWhichApplicationInstanceRuns(ApplicationInstanceID applnID)`
  - Effect: Returns all the gateways on which, according to the subscriptions **TopologyRequirementsInstantiation**, an application instance of the subscription runs.
  - Sequence Diagrams: figs. 4.22 and 4.23
- `list<ApplicationInstanceID> getInstancesOfApplnVersions(ApplicationID applID, list<Version> versions)`
  - Effect: Return the application identifiers of all application instances of a certain application for one of several versions.

- Sequence Diagrams: fig. 4.24
- list<**NotificationID**, string> getNotificationsForUser(**UserID** uID) throws *NoSuchUserException*
  - Effect: Returns the IDs and descriptions of all the Notifications that belong to the user with id uID.
  - Sequence Diagrams: None
- list<Tuple<**TopologyRequirementsInstantiation**, **ApplicationInstanceID**>> getTopologyRequirementsInstantiationForApplicationsWhichUseDevice(**DeviceResourceLocator** pRL)
  - Effect: Return the **TopologyRequirementsInstantiation** and **ApplicationInstanceID** for all the application instances whose current configuration includes the device with id pID.
  - Sequence Diagrams: fig. 4.7
- Tuple<string, **Timestamp**, **NotificationTrigger**, **NotificationStatus**, **CommunicationChannel**, **NotificationStatus**> readNotification(**NotificationID** nID)
  - Effect: Return information on the specified notification and sets the status as read.
  - Sequence Diagrams: None
- list<**ApplicationID**> removeEndUser(**UserID** custOrgID, **UserID** uID) throws *NoSuchUserException*
  - Effect: Marks the specified end user profile as inactive and will remove the end user with **UserID** uID from any application instance configurations. The ID of any application for which this user was assigned a mandatory role will be returned.
  - Sequence Diagrams: None
- void setApplInstanceStatus(string status)
  - Effect: Set the current status of an application instance, e.g. suspended or running.
  - Sequence Diagrams: None
- void setApplicationNote(**ApplicationID** applID, string note)
  - Effect: A short description of the current state of the application is attached to that application. Usually this will be no more than "running" but a reason of failure (runtime or testing) may also be attached here.
  - Sequence Diagrams: None
- void setNotificationStatus(**NotificationID** notificationID, **NotificationStatus** status)
  - Effect: Set the status of the notification (sent, delivered, stored).
  - Sequence Diagrams: fig. 4.20
- void subscriptionActivationChange(**ApplicationInstanceID** applnID, **Timestamp** ts, boolean active)
  - Effect: Adds an (in)activation timestamp to the subscription and marks it as (in)active depending on the boolean value.
  - Sequence Diagrams: None
- void suspendSubscription(**ApplicationInstanceID** applnID, **Timestamp** ts)
  - Effect: **Timestamp** the end of the subscription and mark it as "suspended".
  - Sequence Diagrams: fig. 4.22
- boolean unsubscribe(**UserID** custOrgID, **ApplicationID** applID, **Version** version)
  - Effect: Removes the subscription entry of the Customer Organisation with custOrgID for the application with **ApplicationID**. If there was no such subscription, false is returned, true if there was.
  - Sequence Diagrams: fig. 4.23
- void updateSubscription(**ApplicationInstanceID** applnID, **Version** to)
  - Effect: The old subscription (application instance) of the given application will be timestamped as "ended". A new subscription will be started (along with a timestamp) of that application (for the new version).
  - Sequence Diagrams: fig. 4.26

**Diagrams:** figs. 1.2 and 2.4

## 5.2.46 RcvMobileAppMsg

**Provided by:**  MobileAppCommunication,  SIOtIP system

**Required by:**  MobileApp



**Operations:**



- void rcvMobileAppMsg(**Command** cmd) throws *CommandNotAllowedException*, *DestinationNotAvailableException*
  - Effect: A command is received from a mobile app, the component makes sure the command destination is an **Application** container and then it will ask the **ApplicationCommandRouter** to

- send it to the correct container.
- Sequence Diagrams: fig. 4.18

**Diagrams:** figs. 1.1 and 1.2

### 5.2.47 RcvSensorData

**Provided by:**  DataReceiver,  OSWithGWCommunication


**Required by:**  DataHandler,  Gateway




**Operations:**

- void handleData(**PluggableDeviceID** pID, **SystemSensorData** data, int ackNr)
  - Effect: Sensor data is sent from the gateway to the online service.
  - Sequence Diagrams: fig. 4.10

**Diagrams:** figs. 1.2, 2.1 and 2.2

### 5.2.48 RebootGW

**Provided by:**  GatewayOS



**Required by:**  GWFailureHandler,  Gateway,  SIIoTIP system


**Operations:**

- void reboot()
  - Effect: The operating system of the gateway will reboot the gateway.
  - Sequence Diagrams: None

**Diagrams:** figs. 1.1, 1.2 and 2.1

### 5.2.49 Registration

**Provided by:**  CustomerOrganisationDashboard,  SIIoTIP system

**Required by:**  CustomerOrganisationClient


**Operations:**




- **HTML** ChangeCustOrgInfo(**SessionID** sessionID, string userName, string password, **PaymentInfo** paymentInfo, **ContactInfo** contactInfo, string companyName, **Address** companyAddress)
  - Effect: The CustomerOrganisationDashboard will propagate this data to the OtherDataDB and thus change customer organisation info in the system.
  - Sequence Diagrams: None
- **HTML** completeCustOrgRegistration(string userName, string password, **PaymentInfo** paymentInfo, **ContactInfo** contactInfo, string companyName, **Address** companyAddress)
  - Effect: The CustomerOrganisationDashboard will do validity checks on the contactInfo, address and paymentInfo. If these are valid, the user will be registered in the system. If not, the CustomerOrganisationDashboard will request that the invalid options be corrected.
  - Sequence Diagrams: None
- **HTML** confirmCustOrgRegistration(string link)
  - Effect: This method is called when the representative of an unregistered customer organization follows the confirmation link they received via e-mail. It triggers the CustomerOrganisationDashboard to send out the rest of the registration form.
  - Sequence Diagrams: None
- **HTML** enterCompanyEmailAddress(string email)
  - Effect: The CustomerOrganisationDashboard checks the validity of the email address. If it is valid, an activation email will be sent. If it is not valid, the user is notified.
  - Sequence Diagrams: None
- **HTML** registerEndUser(**SessionID** sessionID, string endUserName, **ContactInfo** contactInfo)
  - Effect: The CustomerOrganisationDashboard will validate the end-user details (end-user already exists or badly formatted contact info). If the information is correct, it is stored in the OtherDataDB and the user is informed of this. If not the form needs to be resubmitted.
  - Sequence Diagrams: fig. 4.11
- **HTML** requestCustOrgRegistrationForm()
  - Effect: The CustomerOrganisationDashboard will make the front end display the first steps of the registration process.
  - Sequence Diagrams: None

- **HTML requestEndUserRegistrationForm(SessionID sessionID)**
  - Effect: The **CustomerOrganisationDashboard** will make the front end display the first steps of the end-user registration process.
  - Sequence Diagrams: fig. 4.11
- **HTML unregisterCustOrg(SessionID sessionID)**
  - Effect: The **CustomerOrganisationDashboard** will ask the **OtherDataDB** to delete any subscriptions this user has, as well as the profile itself. Invoice creation will be triggered before the profile is fully deleted. **ApplicationContainers** are asked to shut the corresponding application instances down.
  - Sequence Diagrams: None
- **HTML unregisterEndUser(SessionID sessionID, string endUserName, ContactInfo contactInfo)**
  - Effect: The entry for the specified user for the customer organisation with the **UserID** in the **SessionID** will be removed from the database and from any application instance configurations. Checks will be done to see if those configurations are still valid and whether application instances will have to be suspended.
  - Sequence Diagrams: None
- **HTML wantToUnregisterEndUser(SessionID sessionID)**
  - Effect: The customer organisation with the **UserID** in the **SessionID** wishes to receive a list of end users such that they can select which ones they wish to unregister.
  - Sequence Diagrams: None

**Diagrams:** figs. 1.1 and 1.2

### 5.2.50 RequestData

**Provided by:**  **Mote**


**Required by:**  **DataHandler**,  **Gateway**,  **SIO TIP system**



**Operations:**

- **DeviceData getData()**
  - Effect: Synchronously retrieve the current value of the sensor.
  - Sequence Diagrams: None
- **boolean getData(int requestID)**
  - Effect: Asynchronously retrieve the current value of the sensor (Callback).
  - Sequence Diagrams: None

**Diagrams:** figs. 1.1, 1.2 and 2.1

### 5.2.51 RouteDeviceData

**Provided by:**  **DeviceDataRouter**





**Required by:**  **DataReceiver**,  **OSWithGWCommunication**








**Operations:**

- **void routeData(PluggableDeviceID devID, SystemSensorData data)**
  - Effect: The **DeviceDataRouter** will use the **ApplicationInstanceInfoDB** to decide which application instances are interested in the given sensor data. It will then call the **ApplicationManager** once per instance, asking them to schedule the receiving of the data by that instance.
  - Sequence Diagrams: fig. 4.10

**Diagrams:** figs. 1.2 and 2.2

### 5.2.52 ScheduleApplication

**Provided by:**  **ApplicationHandler**,  **ApplicationManager**,  **ApplicationScheduler**,  **GWApplicationScheduler**

**Required by:**  **ApplicationCommandRouter**,  **ApplicationManager**,  **ApplicationStarter**,  **DBRequestHandler**,  **DataHandler**,  **DeviceDataRouter**,  **GWCommandRouter**



**Operations:**


- **void scheduleHandleCrash(CrashInfo crashInfo, ApplicationInstanceID applnID)**
  - Effect: Schedule to notify application instance of an occurred crash.
  - Sequence Diagrams: None
- **void scheduleInitialize(Code code)**

- Effect: Schedule to initialize a new application instance.
- Sequence Diagrams: fig. 4.1
- void schedulePluggableDeviceFailed(**DeviceResourceLocator** pRL, **ApplicationInstanceID** lnID)
  - Effect: Schedule to notify an application instance that a pluggable device has failed.
  - Sequence Diagrams: fig. 4.27
- void scheduleReactToCommand(**Command** cmd, **ApplicationInstanceID** applnID)
  - Effect: Schedule to pass the provided command to the application.
  - Sequence Diagrams: figs. 4.13 and 4.14
- void scheduleReactToData(**PluggableDeviceID** plD, **SystemSensorData** data, **ApplicationInstanceID** applnID)
  - Effect: Schedule to pass the provided sensor data to the application so that it can process it.
  - Sequence Diagrams: figs. 4.10 and 4.15
- void scheduleWakeUp(**ApplicationInstanceID** applnID)
  - Effect: Schedule to wake up the application.
  - Sequence Diagrams: None

**Diagrams:** figs. 1.2, 2.1 and 2.4

### 5.2.53 SendCommandToGW

**Provided by:**  ApplicationCommandPropagator,  OSWithGWCommunication


**Required by:**  ApplicationCommandRouter

**Operations:**

- void sendCommandToGW(**Command** cmd)
  - Effect: Route the command to the correct gateway according to the **DeviceResourceLocator** in the **Command** (contains the **GatewayID**).
  - Sequence Diagrams: figs. 4.2, 4.13, 4.16, 4.18, 4.22 and 4.23

**Diagrams:** figs. 1.2 and 2.2

### 5.2.54 SendInvoice

**Provided by:**  ThirdPartyInvoicingService


**Required by:**  InvoiceManager,  SIOtIP system



**Operations:**

- void sendInvoice(string service, float amount, **Date** dueData, int id)
  - Effect: The external invoicing service is requested to handle the payment.
  - Sequence Diagrams: None

**Diagrams:** figs. 1.1 and 1.2

### 5.2.55 SendMobileAppMsg

**Provided by:**  MobileApp




**Required by:**  MobileAppCommunication,  SIOtIP system


**Operations:**

- void sendToMobileApp(string hostname, int port, string message)
  - Effect: A message is sent to the specified hostname/port combination.
  - Sequence Diagrams: fig. 4.12

**Diagrams:** figs. 1.1 and 1.2

### 5.2.56 SensorData

**Provided by:**  DataHandler,  Gateway,  SIOtIP system

**Required by:**  Mote

**Operations:**

- void rcvActuationCallback(int requestID)
  - Effect: Confirm execution of an actuation command.
  - Sequence Diagrams: None
- void rcvData(**PluggableDeviceID** plD, **DeviceData** sensorReading)
  - Effect: Provide sensor data to the gateway (Periodic).

- Sequence Diagrams: None
- void rcvData(**PluggableDeviceID** plD, **DeviceData** sensorReading, int requestID)
  - Effect: Provide requested sensor data to the gateway (Callback).
  - Sequence Diagrams: None

**Diagrams:** figs. 1.1, 1.2 and 2.1

### 5.2.57 SensorDataMgmt

**Provided by:** [OSSensorDataDB](#), [SensorDataDB](#), [SensorDataLoadBalancer](#), [SensorDataScheduler](#)

**Required by:** [DBRequestHandler](#), [DataReceiver](#), [OSWithGWCommunication](#), [SensorDataLoadBalancer](#), [SensorDataScheduler](#)

**Operations:**

- void addData(**PluggableDeviceID** plD, **SystemSensorData** data)
  - Effect: Adds sensor data to the database.
  - Sequence Diagrams: fig. 4.10
- void getDataFromTimeframe(**ApplicationInstanceID** applnID, list<**DeviceResourceLocator**> devices, **Timestamp** from, **Timestamp** to)
  - Effect: All sensor data for the given pluggable devices within the given timeframe will be sent as a **Command** to the application instance with the given id.
  - Sequence Diagrams: None

**Diagrams:** figs. 1.2, 2.2 and 2.3

### 5.2.58 SessionMgmt

**Provided by:** [SessionDB](#)

**Required by:** [AuthenticationHandler](#)

**Operations:**

- Map<**SessionAttributeKey**, **SessionAttributeValue**> isValidSession(**SessionID** sID) throws *NoSuchSessionException*
  - Effect: Returns the session attributes corresponding to the session with the supplied **SessionID** if it is a valid session.
  - Sequence Diagrams: None
- void deleteSession(**SessionID** sID) throws *NoSuchSessionException*
  - Effect: Deletes the session associated with the given **SessionID** from the database.
  - Sequence Diagrams: None
- **SessionID** newSession(**UserID** uID)
  - Effect: Generates a new **SessionID** for the given **UserID** and stores this as an active session.
  - Sequence Diagrams: None

**Diagrams:** fig. 1.2

### 5.2.59 Subscription

**Provided by:** [CustomerOrganisationDashboard](#), [SIOtIP system](#)

**Required by:** [CustomerOrganisationClient](#)

**Operations:**


- **HTML** applicationCriticality(**ApplicationID** applD, bool isCritical, **Version** appVersion, **SessionID** sID)
  - Effect: The user to which the session belongs selects the criticality a certain application has to them.
  - Sequence Diagrams: fig. 4.21
- **HTML** assignEndUsers(list<Tuple<**UserID**, string>> assignments, **SessionID** sID, **Version** version, **ApplicationID** applD)
  - Effect: The user to which the session belongs has assigned the end user roles, next, the user is asked the criticality of the application. After selecting it, all information regarding the subscription is stored in the **OtherDataDB** and the customer organisation is unsubscribed from previous versions of that application, if any. The **ApplicationStarter** is then asked to activate the application.
  - Sequence Diagrams: fig. 4.21
- **HTML** browseApplications(**SessionID** sID)




- Effect: The primary actor indicates they want to subscribe to an application. The customer organisation can be identifier based on the provides session information.
- Sequence Diagrams: fig. 4.21
- **HTML selectApplication(ApplicationID appID, SessionID sID, Version version)**
  - Effect: The user to which the session belongs selects an application to subscribe to, the CustomerOrganisationDashboard will get fetch the **Topology** corresponding to the user and will match it with the **ApplicationRequirements**. The configuration that still needs to be done is presented to the user by the front end.
  - Sequence Diagrams: fig. 4.21
- **HTML setTopologyConfiguration(TopologyRequirementsInstantiation appReqIn, Version version, ApplicationID appID)**
  - Effect: The user to which the session belongs is finished with configuring the **Topology**. The CustomerOrganisationDashboard will look up the required end-user roles and the end-users associated with the customer organisation in the OtherDataDB and will present them together to the user via the front end such that they may assign the roles.
  - Sequence Diagrams: fig. 4.21
- **HTML unsubscribe(SessionID sID, ApplicationID appID, Version appVersion)**
  - Effect: The user to which the session belongs wishes to cancel a subscription. The subscription matching the **UserID** and **ApplicationID** is removed from the database, the InvoiceManager is asked to create an invoice and the application is deactivated by the ApplicationStarter.
  - Sequence Diagrams: None
- **HTML wantToUnsubscribe(SessionID sID)**
  - Effect: The user to which the session belongs indicates they want to unsubscribe from an application. The CustomerOrganisationDashboard will fetch all applications the user is currently subscribed to from the OtherDataDB and will ask the front-end to present these.
  - Sequence Diagrams: None

**Diagrams:** figs. 1.1 and 1.2

### 5.2.60 TestApplication

**Provided by:**  AppTester

**Required by:**  ApplicationDeveloperDashboard









**Operations:**

- **Tuple<boolean, string> testCode(Code code, ApplicationID appID, UserID appDevID)**
  - Effect: Run tests on the provided code and return true if they all pass and false if they don't, together with a string witch contains a small description of the failure (empty if successful).
  - Sequence Diagrams: fig. 4.4

**Diagrams:** fig. 1.2

### 5.2.61 TopologyMgmt

**Provided by:**  TopologyDB,  TopologyLoadBalancer

**Required by:**  ApplicationCommandRouter,  ApplicationStarter,  CustomerOrganisationDashboard,  DBRequestHandler,  DeviceStatusMonitor,  InfrastructureOwnerDashboard,  OSWithGWCommunication,  TopologyLoadBalancer

**Operations:**



- **list<UserID> getCostumerOrganisationsWithAccessToDevice(PluggableDeviceID pID)**
  - Effect: Returns all UserIDs where an access right entry exists for the Pluggable Device identified by pID.
  - Sequence Diagrams: None
- **Map<String, String> getDeviceConfiguration(PluggableDeviceID pID)**
  - Effect: Fetch the current configuration of the specified pluggable device.
  - Sequence Diagrams: None
- **list<Tuple<PluggableDeviceID, PluggableDeviceType, GatewayID>> getDevicesForInfrastructureOwner(UserID infOwnerID)**
  - Effect: Will return the IDs of all devices which belong to a topology of which the user with the specified **UserID** is the owner. **PluggableDeviceType** and **GatewayID** are also returned to give context about the nature of the device.

- Sequence Diagrams: None
- **list<DeviceResourceLocator> getPluggableDevicesForGW(GatewayID gwID)**
  - Effect: Return the DeviceResourceLocators of the pluggable devices associated with the given gateway.
  - Sequence Diagrams: fig. 4.6
- **Topology getTopologyForUser(UserID infOwnerId)**
  - Effect: Retrieve the topology of the infrastructure owned by the infrastructure owner with the specified UserID.
  - Sequence Diagrams: fig. 4.21
- **void grantAccess(UserID uID, PluggableDeviceID pID)**
  - Effect: A new access right (customer organisation gaining access to a specific device) is stored in the TopologyDB.
  - Sequence Diagrams: None
- **UserID newMote(MoteInfo moteInfo, GatewayID gwID)**
  - Effect: A new mote is added as unplaced to the topology to which the gateway with the specified ID belongs to. The UserID of the user which is the infrastructure owner of that topology is returned.
  - Sequence Diagrams: None
- **void newPluggableDeviceInMote(int moteID, PluggableDeviceID pID, PluggableDeviceType pType)**
  - Effect: The Pluggable Device will be associated with the specified mote and will be marked as uninitialized.
  - Sequence Diagrams: fig. 4.9
- **void removePluggableDevice(DeviceResourceLocator devRL)**
  - Effect: Removes the entry for the pluggable device with the matching DeviceResourceLocator from the TopologyDB.
  - Sequence Diagrams: None
- **void setDeviceConfiguration(PluggableDeviceID pID, Map<string, string> config)**
  - Effect: Store the current configuration of the specified pluggable device.
  - Sequence Diagrams: None
- **void setTopologyForUser(Topology newTopology)**
  - Effect: Sets the topology for a certain user (infrastructure owner) to newTopology. If any devices (identified by PluggableDeviceID) who were previously not included yet are now included, they are marked as active and that change will be propagated to the DeviceInitCache.
  - Sequence Diagrams: None

**Diagrams:** figs. 1.2 and 2.2

## 5.2.62 UpdateMessage

**Provided by:**  ApplicationStarter

**Required by:**  ApplicationCommandRouter,  ApplicationHandler,  ApplicationScheduler



**Operations:**


- **void gwContainerInitialize(bool failed, ApplicationInstanceID applnID, GatewayID gwID)**
  - Effect: A call informing the ApplicationStarter that a (new) container on the gateway has (un)successfully been initialized and whether the old container has been killed.
  - Sequence Diagrams: fig. 4.25
- **void gwOldApplicationContainerShutDownCompleted(ApplicationInstanceID oldApplnID, GatewayID gwID)**
  - Effect: A call informing the ApplicationStarter that the old GWApplicationContainer was successfully shut down and that the new one has been activated.
  - Sequence Diagrams: fig. 4.25
- **void gwUpdateContainerCreated(ApplicationInstanceID applnID, GatewayID gwID)**
  - Effect: A call informing the ApplicationStarter that the parallel update GWApplicationContainer was successfully created on a certain gateway.
  - Sequence Diagrams: fig. 4.24
- **void osContainerInitialize(boolean failed, ApplicationInstanceID applnID)**
  - Effect: A call informing the ApplicationStarter that the new container on the Online Service has (un)successfully been initialized and whether the old container has been killed.
  - Sequence Diagrams: fig. 4.25
- **void osOldApplicationContainerShutDownCompleted(ApplicationInstanceID applnID)**

- Effect: A call informing the `ApplicationStarter` that the old `ApplicationContainer` was successfully shut down on a certain gateway and that the new one has been activated.
- Sequence Diagrams: fig. 4.24

**Diagrams:** figs. 1.2 and 2.4

### 5.2.63 UploadApplication

**Provided by:**  `ApplicationDeveloperDashboard`,  `SIoTIP` system







**Required by:**  `ApplicationDeveloperClient`









**Operations:**

- **HTML** `automaticActivation(bool automatic, SessionID sID)`
  - Effect: The `ApplicationDeveloperDashboard` will remember the fact that that developer wants the update to be either automatic or not. Depending on the answer either the previous versions of the application will be fetched and asked to be displayed by the front-end (in case of automatic), or the form for the application specifications will be displayed.
  - Sequence Diagrams: fig. 4.4
- **HTML** `upload(Code code, string description, ApplicationMetaData metadata, SessionID sID, ApplicationRequirements appReq)`
  - Effect: The provided application specifications are first be stored in the `OtherDataDB`, then they are tested by the `AppTester` and if the tests pass the application is made available in the store. If they fail the user is notified. In case an automatic update was requested, the appropriate update data is passed to the `ApplicationStarter`.
  - Sequence Diagrams: fig. 4.4
- **HTML** `uploadExisting(bool exists, SessionID sID)`
  - Effect: The `ApplicationDeveloperDashboard` will either ask the front end to show the form used for entering the application specifications (when the boolean is false) or make a list of the applications of that application developer and ask the front-end to show that list.
  - Sequence Diagrams: fig. 4.4
- **HTML** `uploadSelect(ApplicationID aID, SessionID sID)`
  - Effect: The `ApplicationDeveloperDashboard` will remember the fact that that the user wants to update the specified application and will ask the front end to display the question of whether or not the update should be automatic.
  - Sequence Diagrams: fig. 4.4
- **HTML** `versionsToUpdate(list<Version> versions, SessionID sID)`
  - Effect: The `ApplicationDeveloperDashboard` will remember the fact that that user wants the specified versions to be automatically updated and will ask the front-end to display the application specification form.
  - Sequence Diagrams: fig. 4.4
- **HTML** `wantToUpload(SessionID sID)`
  - Effect: The `ApplicationDeveloperDashboard` will start the application uploading procedure by making the front-end display the question of whether the developer wants to upload an existing application.
  - Sequence Diagrams: fig. 4.4

**Diagrams:** figs. 1.1 and 1.2

### 5.2.64 UserAuthentication

**Provided by:**  `ApplicationDeveloperDashboard`,  `AuthenticationHandler`,  `CustomerOrganisationDashboard`,  `InfrastructureOwnerDashboard`,  `SIoTIP` system,  `SysAdminDashboard`

**Required by:**  `ApplicationDeveloperClient`,  `ApplicationDeveloperDashboard`,  `CustomerOrganisationClient`,  `CustomerOrganisationDashboard`,  `InfrastructureOwnerClient`,  `InfrastructureOwnerDashboard`,  `SysAdminClient`,  `SysAdminDashboard`

**Operations:**

- **SessionID** `login(Credentials cred)` throws *IncorrectCredentialsException*
  - Effect: Verify the credentials with the `OtherDataDB`. If valid, a new session is created (with a new ID) which is stored in the `SessionDB`. The users `UserID` will be added to this session as an attribute.
  - Sequence Diagrams: None

- **bool** `logout(SessionID sID)`
  - Effect: The session identified by the provided id `sID` is removed from the `SessionDB` if it exists. Otherwise nothing happens.
  - Sequence Diagrams: None

**Diagrams:** figs. 1.1 and 1.2

### 5.2.65 VerifySession

**Provided by:** `AuthenticationHandler`

**Required by:** `ApplicationDeveloperDashboard`, `CustomerOrganisationDashboard`, `InfrastructureOwnerDashboard`, `SysAdminDashboard`

**Operations:**

- `Map<SessionAttributeKey, SessionAttributeValue> verifySession(SessionID sID)` throws *NoSuchSessionException*
  - Effect: Verify whether the provided session identifier `sID` corresponds to an existing session and, if so, return the session attributes.
  - Sequence Diagrams: None

**Diagrams:** fig. 1.2

## 5.3 Exceptions

- *ApplicationUncaughtException* Thrown when an application throws an exception which it does not catch itself.
- *CommandNotAllowedException* Thrown when an external system tries to execute a command that is not allowed (e.g. Commands which are not actuation commands or messages to applications).
- *DestinationNotAvailableException* Thrown when the destination for which the command was meant is not available.
- *IncorrectCredentialsException* Thrown if the provided credentials are incorrect.
- *NoSuchApplication* Thrown if no application with the provided identifier exists.
- *NoSuchApplicationInstance* Thrown when no application instance with the given identifier exists.
- *NoSuchInfrastructure* Thrown when no infrastructure owner exists with the provided identifier.
- *NoSuchPluggableDeviceType* Thrown when the given pluggable device type does not exist in the system.
- *NoSuchSessionException* Thrown if no session exists with the provided identifier
- *NoSuchSubscription* Thrown when no subscription exists between the given `UserID` and `ApplicationID`.
- *NoSuchUserException* Thrown if no user with the provided identifier exists.
- *UserAlreadyExistsException* Thrown when a user already exists for the specified identifier and/or contact information.

## 5.4 Data types

- **Address:**  
A data structure representing a physical address in the world. May contain info such as Country, city, postal code, street name and street number.
- **ApplicationConfigurationInstantiation:**  
Data structure representing the assigned pluggable devices (as `DeviceResourceLocators`) to an application instance.
- **ApplicationID:**  
Uniquely identifies an application. Note that there may be multiple versions of this application (different Version in the database).
- **ApplicationInstanceID:**  
Attributes: **UserID** `custOrgID`, **Version** `version`  
Uniquely identifies an application instance.
- **ApplicationMetaData:**  
All metadata attached to an application such as an ID, a name, a version and a description.
- **ApplicationRequirements:**

Attributes: **TopologyRequirements** topologyRequirements, list<**RoleDescription**> endUserRequirements

A data structure representing the various requirements which need to be fulfilled before an application instance of that application is able to run. This includes topology requirements and end-user role requirements.

- **ApplicationStatistics:**

A data structure representing detailed statistics of a certain application. Contains, for example, the number of customer organisations subscribed to the application.

- **Code:**

Represents the code/libraries/other dependencies needed to run an instance of an application. This includes the location the code needs to run (Gateway/Online Service).

- **Command:**

A data structure representing a command of any type in the system. This includes the type (sort of request to database, actuation command, message between parts of an application instance, communication from and to external mobile apps, reconfiguration command, application update messages). Also includes extra information depending on the type of command (e.g. what the actuation command is, timeframe for sensor database request, the ResourceLocator of the actuator, the name of the actuation command, etc.). It also includes the ApplicationInstanceID of the application instance that sent it. Most of the internal communication is done by sending Commands around.

- **CommunicationChannel:**

An enum representing the available channels for communication with external parties, initially only email and SMS.

- **ContactInfo:**

Describes the way a certain user should be contacted in case they need to be notified. Includes a CommunicationChannel and any additional information needed for the message to be sent (e.g. email address/phone number).

- **ConversionCode:**

A library (or part of) used to convert sensor data from one unit to another, for example Celsius to Fahrenheit.

- **CrashInfo:**

Contains information on how/why a function call of an application failed. An application can use this information to perhaps avoid such failures in the future.

- **Credentials:**

The authentication credentials of a user of the SIIoTIP system. The credentials include the identifier of the user and a proof of their identity, for example a username and password.

- **Date:**

Represents the Date type the external invoice components use.

- **DeviceCategory:**

A datatype denoting the category of a device, for example a thermometer or a smoke detector.

- **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.

- **DeviceResourceLocator:**

Can be used to locate a specific pluggable device since it includes info about what infrastructure/gateway/mote it belongs to.

- **GatewayID:**

Uniquely identifies a gateway.

- **HTML:**

Hypertext Markup Language file, possibly stored as a string.

- **InfrastructureID:**

Uniquely identifies an infrastructure.

- **IPAddress:**

Represents an IP address of some device.

- **MoteID:**

Uniquely identifies a mote.

- **MoteInfo:**  
Attributes: int moteID, int manufacturerID, int iproductID, 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.
- **NotificationID:**  
Uniquely identifies a notification.
- **NotificationStatus:**  
An enum describing the various states a notification which needs to be sent to a user can be in (sent, delivered or read).
- **NotificationTrigger:**  
A data structure containing information about the reason a notification has to be sent. Depending on the reason, an ID of some sort may be attached to find the correct notification configuration. Examples of this are: Gateway failure (with UserID of Infrastructure Owner) New Mote is available (with UserID of Infrastructure Owner) An application instance crashed (with ApplicationInstanceID) Application tests failed/succeeded (with UserID of application developer)
- **PaymentInfo:**  
Contains Information on the method of payment (Credit Card or Zoomit), along with extra information needed to send invoices (such as credit card information).
- **PluggableDeviceID:**  
A unique identifier of the Pluggable Device.
- **PluggableDeviceType:**  
Denotes the type of the Pluggable Device.
- **RoleDescription:**  
Attributes: string name, int required, int optional  
A data structure representing the need for a specific role for an application. The name of the role is a string, and the amount of required and optional user which need to fulfill this role for the application to work are given as integers.
- **SessionAttributeKey:**  
The key of an attribute attached to a session.
- **SessionAttributeValue:**  
The value of an attribute attached to a session.
- **SessionID:**  
A piece of information uniquely identifying a user session in the SIoTIP system. Contains at least the user's UserID and the time the session was initiated (TimeStamp).
- **SystemSensorData:**  
A data structure used to represent different kinds of device data in a generic manner. Contains the converted JSON map (which held sensor values or actuator state) as well as a TimeStamp representing the time the DataHandler received the data.
- **Timestamp:**  
The representation of a time (e.g. year/month/day/time) in the system
- **Topology:**  
The topology shows a representation of the gateways, and the motes and pluggable devices associated with each gateway, and specifies key relationships between these devices, such as physical location information, device model/category information, device status (e.g. uninitialized). There is one such topology per infrastructure.
- **TopologyRequirements:**  
Specifies the topological requirements for an application: mandatory and optional devices, device configurations, device relations and redundancy information. Device information entail a DeviceCategory, along with the physical requirements of the device (update speed, needed configuration settings). By using this format, multiple device models can be flagged as useable for a certain application.
- **TopologyRequirementsInstantiation:**  
The instantiation of the requirements regarding topology for an application. These are always linked to a description and include information such as the devices/gateways associated with the instance and information on the devices themselves such as model/category.
- **UserID:**

A piece of data uniquely identifying a user (Application Developer, SIoTIP system administrator, Customer Organisation or Infrastructure Owner) in the SIoTIP system.

- **Version:**

Uniquely identifies a version of an application. Includes metadata such as the version number.