Appendix B.
Initial ReMeS architecture

Contents

1 Introduction, key decisions and rationale **ii**
1.1 *Av1* (Measurement Database Failure) ii
1.2 *P1* (Timely closure of valves) ii
1.3 *M1* (Dynamic pricing) iii

2 Main component diagram **iv**

3 Key Decompositions **iv**
3.1 Decomposition diagram: AlarmHandler component iv

4 Deployment diagram **v**

5 Sequence diagrams **v**
5.1 Alarm handling: queuing v
5.2 Alarm handling: processing vi
5.3 Billing: invoice generation vi

6 Residual drivers **vi**

1 Introduction, key decisions and rationale

This document presents an initial ReMeS architecture, in which a number of key decompositions have been executed. This initial architecture already addresses the following drivers in the following way:

1.1 *Av1* (Measurement Database Failure)

Architectural decisions. The elements of *Av1* are tackled in the following manner:

1. To accomplish that a failure of the measurement database "*does not affect the availability of other types of persistent data*", a first key architectural decision is to **separate measurement data from all the other data**. This gives rise to the `MeasurementDB` component in the main component diagram (cf. Section 2).

2. To minimize the impact and likelihood of this scenario ("*guaranteed minimal uptime of 99,9%*"), the `MeasurementDB` is actively replicated (**active redundancy** tactic) on different nodes, which is visible in the deployment diagram (cf. Section 4). The replication factor will be determined later, based on the scale in which ReMeS will be deployed. A `ReplicationManager` component is placed in front of the replicas, visible in the component diagram (cf. Section 2).

3. The requirement that "*ReMeS Operators must be notified*" is in fact a new functional requirement (a use case called "*Notify ReMeS operators*", highly similar to *UC9*). This is handled by introducing the `Operator-NotificationHandler` component which has the ability to push messages and alerts to the ReMeS Operator.

4. Failure detection is done using **ping/echo** (done every 4 seconds). This responsibility is assigned to the `ReplicationManager.`

5. As a result of the active redundancy tactic we adopted, the database will not have to go into degraded modus when a database instance fails. As a consequence, temporarily caching the incoming measurement trames has become unnecessary to satisfy this requirement. If one node fails, the other node(s) will still store incoming measurement trames. When the failed node is operational again, the `ReplicationManager` will synchronize it with the other instances.

6. While this should not happen frequently, the `ReplicationManager` throws an exception when not enough replicas are available to process read or write requests in order to "*fail gracefully*". This should be shown in the interface documentation (not yet included in this document) and other components should be able to handle this exception appropriatly.

1.2 *P1* (Timely closure of valves)

Architectural decisions. The elements of *P1* are tackled in the following manner:

1. The flow to process alarm trames is separated from the flow to process measurement trames. The Remote Modules are configured to send alarm trames (*UC13*) directly to the `AlarmHandler` component (and they will send measurement trames to a different front-end component).

2. One can directly derive from the incoming alarm trames whether it involves gas or water (from the Device-Type field, at Byte 1, cf. Figure 2 of the domain description). As a consequence, no additional look-up of the type of utility that is measured by the source Remote Module is required.

3. The `AlarmHandler` component is responsible for (i) receiving alarm trames (the `AlarmFacade` sub-component), (ii) associating different priorities (**priority queue**) to these alarm trames `(the AlarmQueue` sub-component), (iii) performing the look-up of the associated Remote Actuator (the `AlarmProcessor` sub-component) and finally, (iii) sending actuation trames (*UC7*) to that Remote Actuator (the `AlarmProcessor` sub-component). The processing of Alarm Trames for gas is given priority over other alarm trames (this is done in a deadline-based fashion so as to avoid starvation).

4. To ensure that *"sending alarm notifications to the Customer and his Alarm Recipients does not affect the actual actuation of valves"*, sending out notifications is done by a separate component (the `NotificationHandler` component) and invoked by means of an asynchronous call to that component (no active wait). The `NotificationHandler` component is responsible for the look-up of the Customer and Alarm Recipient contact details.

1.3 *M1* (Dynamic pricing)

Architectural decisions. The elements of *M1* are tackled in the following way:

1. For communicating the current utility price to the customer directly, we expect to use the infrastructure available to realize the "Notify customer (*UC9*)" use case. Evidently, this will be implemented in a smart way (e.g., price thresholds that are configurable by the customer), so that customers will not be flooded with price update notifications.

2. For communicating changes to the current utility price to smart meters, smart appliances, or home automation systems, we expect to reuse the infrastructure available to realize the existing use case "Send trame to remote device (*UC7*)". These devices will be registered in the system as Remote Modules, and a new type of control trame will be created, encapsulating the pricing information. The design of the database and all trame-handling interfaces should allow new types of Remote Modules and trames to be added without change.

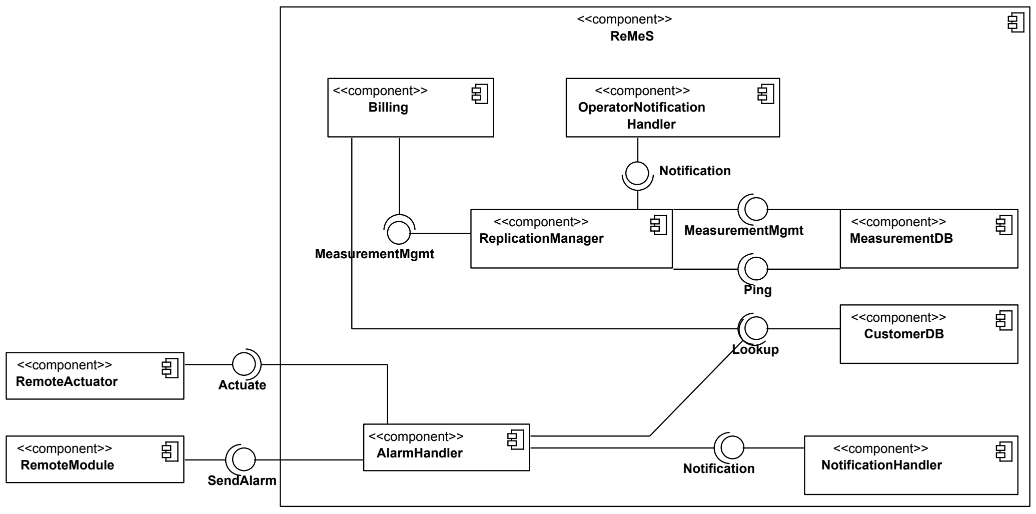
3. The details of how the smart meters, smart appliances and home automation systems (HAS) themselves deal with these price updates are out of scope of the ReMeS system.

4. For the price updates, we envision an extension of the (not yet existing) interfaces between the UIS and ReMeS that is based on a publish-subscribe mechanism on the basis of individual customers. This way, ReMeS can be notified every time the price changes for an individual customer.

5. Invoice generation (*UC15*) is handled by an autonomous component (the `Billing` component), which is also deployed on a separate node (visible in the deployment diagram). Therefore, incorporating dynamic pricing will *"not affect remote metering, actuation, leak and anomaly detection"*.

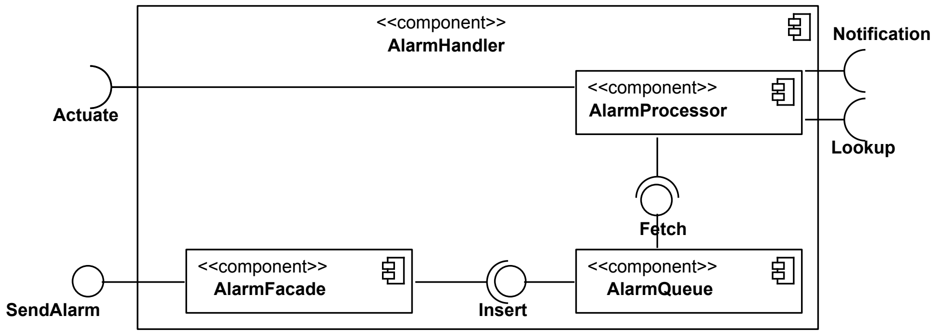
6. Within the `Billing` component, the invoice creation process is **encapsulated** using the Strategy design pattern (`InvoiceGenerationStrategy`). This way, we can use a generic and well-defined interface for triggering the invoice generation process, and hide the specific details are hidden. This will allow run-time extension of the `Billing` component with alternative invoice generation strategies, specifically because it allows dynamically introducing a new `InvoiceGenerationStrategy` implementation (without affecting the existing `StaticPricingInvoiceGenerationStrategy` implementation). The configuration per customer of which `InvoiceGenerationStrategy` applies to that customer is stored in the `CustomerDB`. In the current version, this configuration parameter is set to `the StaticPricingInvoiceGenerationStrategy` (the only available `InvoiceGenerationStrategy` for now) for all customers by default.

2 Main component diagram

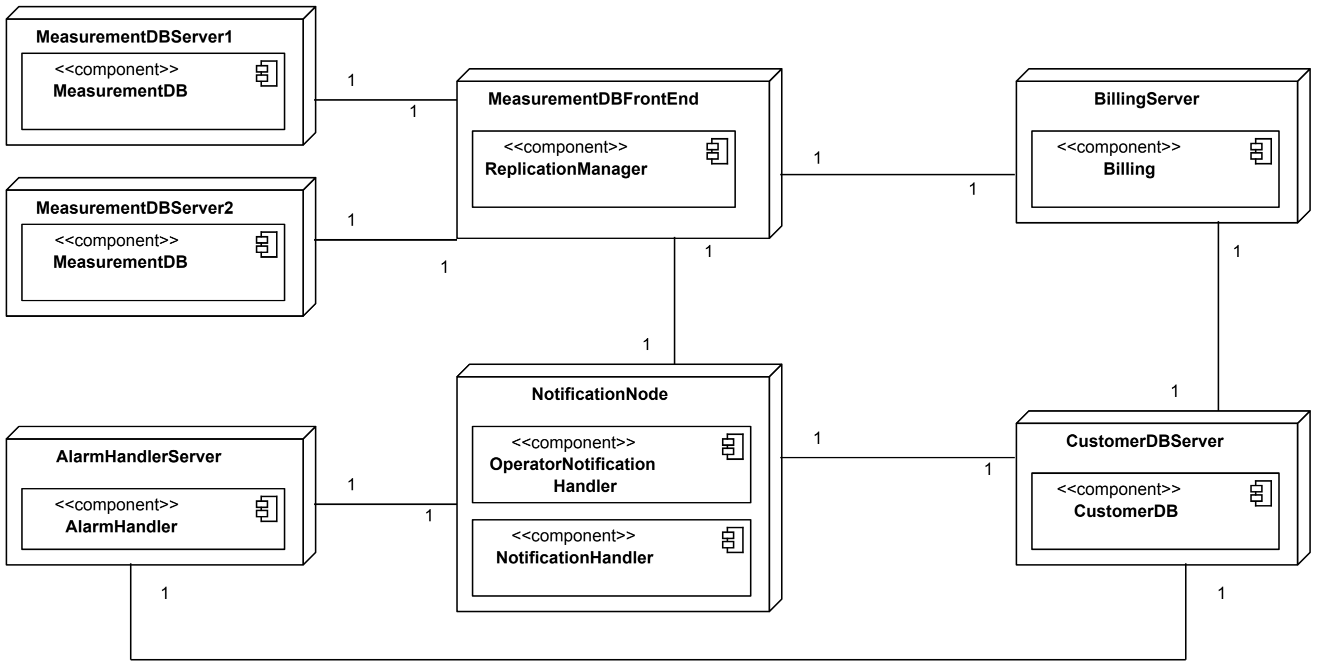


3 Key Decompositions

3.1 Decomposition diagram: AlarmHandler component

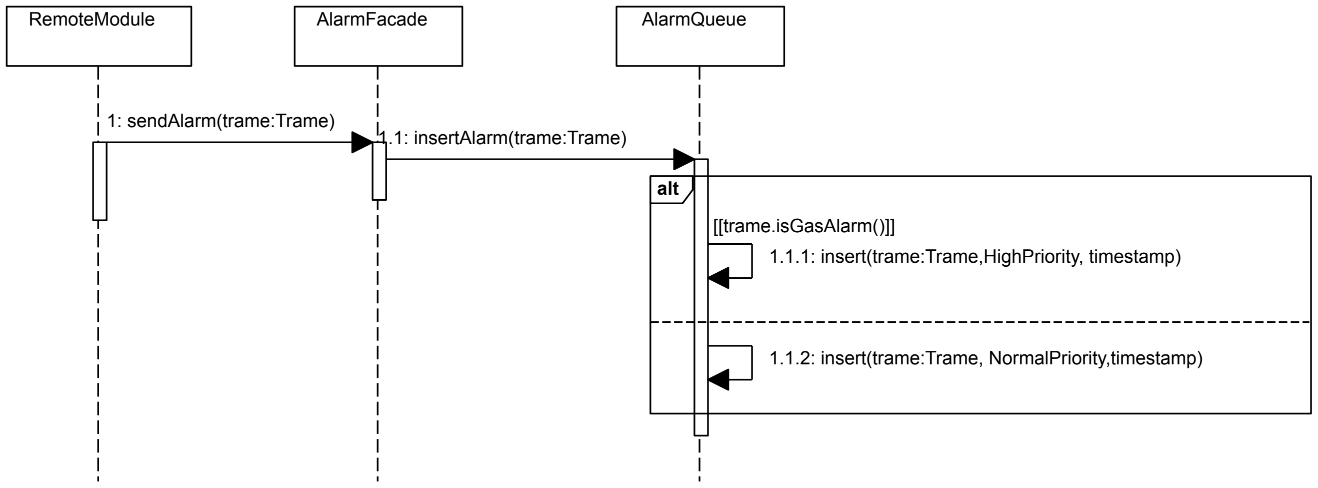


4 Deployment diagram



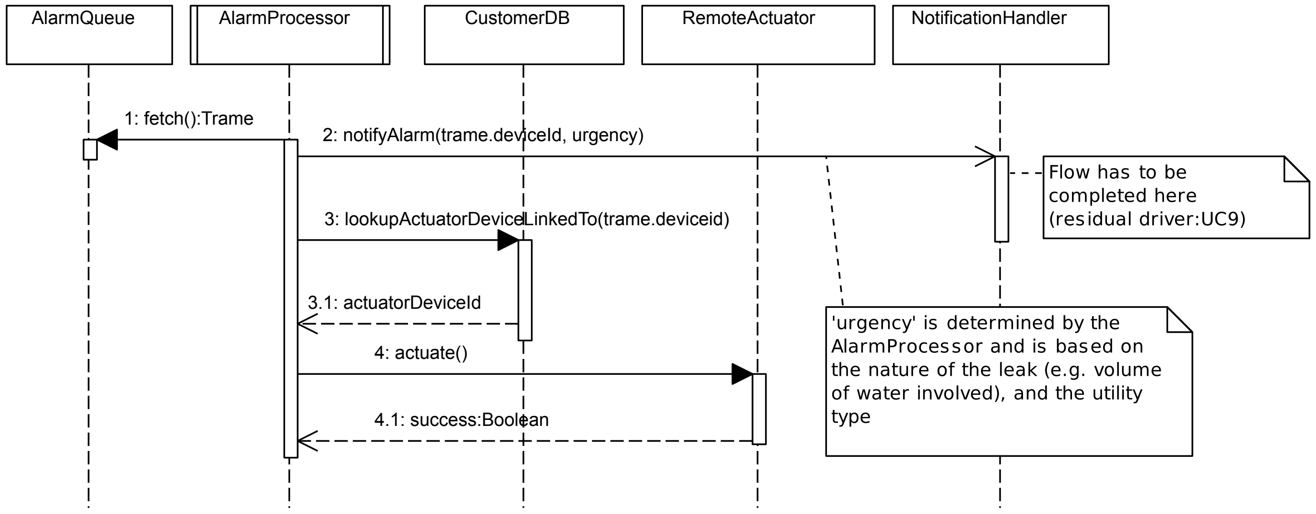
5 Sequence diagrams

5.1 Alarm handling: queuing

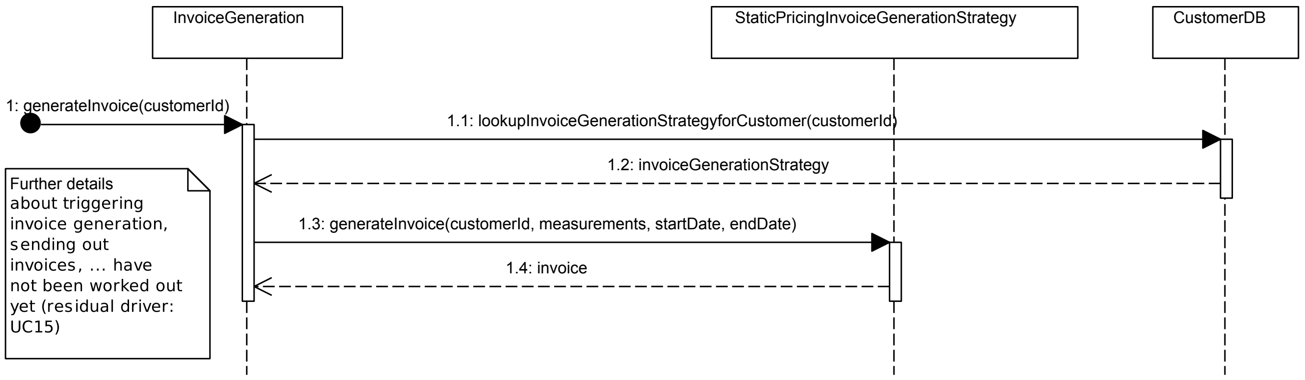


RemoteModule
AlarmFacade
AlarmQueue
1: sendAlarm(trame:Trame)
1.1: insertAlarm(trame:Trame)
alt
[[trame.isGasAlarm()]]
1.1.1: insert(trame:Trame,HighPriority, timestamp)
1.1.2: insert(trame:Trame, NormalPriority,timestamp)

5.2 Alarm handling: processing


AlarmQueue
AlarmProcessor
CustomerDB
RemoteActuator
NotificationHandler
1: fetch():Trame
2: notifyAlarm(trame.deviceId, urgency)
Flow has to be completed here (residual driver:UC9)
3: lookupActuatorDeviceLinkedTo(trame.deviceid)
3.1: actuatorDeviceId
4: actuate()
4.1: success:Boolean
'urgency' is determined by the AlarmProcessor and is based on the nature of the leak (e.g. volume of water involved), and the utility type

5.3 Billing: invoice generation


InvoiceGeneration
StaticPricingInvoiceGenerationStrategy
CustomerDB
1: generateInvoice(customerId)
1.1: lookupInvoiceGenerationStrategyforCustomer(customerId)
1.2: invoiceGenerationStrategy
1.3: generateInvoice(customerId, measurements, startDate, endDate)
1.4: invoice
Further details about triggering invoice generation, sending out invoices, ... have not been worked out yet (residual driver: UC15)

6 Residual drivers

All drivers not explicitly mentioned in this report have not been addressed yet. Of the selected use cases, the following elements have been tackled:

- *UC7*: sending of actuation trames to remote devices has been covered,
  - the residual driver *UC7a* therefore involves sending other trame types (reconfiguration trames).
- *UC9* has not been covered yet.
- *UC13* is covered entirely.
- *UC15:* while it has been assigned to the `Billing` component because of *M1*, *UC15* is currently not covered from a behavioral perspective.