Software Architecture

Remote Measurement, Monitoring and Control System: Phase 2 - Architectural Design

Professoren: Wouter Joossen,
Riccardo Scandariato,
Dimitri Van Landuyt

Maarten Allard - s0199048
Kristof Coninx - s0199831

April 25, 2012

Contents

List of Figures

1 Preface

For the creation and documentation of this system, it was recommended to use Visual Paradigm as the design editor of choice. However due to some compatibility issues that arose from trying to use Visual Paradigm as the editor between different members of the design team, Another software suite was used for creating the design diagrams. We did however consult the didactical team about using another diagram design tool than Visual Paradigm. We also made sure that the software program we used adhered to the same uml specifications as Visual Paradigm such that the diagram contents would be readable and understandable by anyone who is used to Visual Paradigm.

The design tool that was eventually used to document the development of the ReMeS system is called UMLet, a free, open-source uml tool. We took the time to confirm that the diagrams actually are similar to diagrams generated from Visual Paradigm. The only noticeable difference is that in Visual Paradigm, when drawing components, both the component symbol and stereotype (¡¡component¿¿) are shown. However in our diagrams, components are only represented by rectangles with the component-symbol in the top right corner. These are not to be confused with single blank rectangles which are used to indicate class elements. Our specification is still conform to the uml specification about representing components, as is Visual Paradigm's.

2 Attribute Driven Design (ADD)

2.1 Iteration 1: Level 0

2.1.1 ReMeS

In the first iteration of the ADD runs, the full ReMeS system was used as a component to decompose. For this decomposition, several architectural drivers were chosen.

Architectural drivers

The primary decomposition for the ReMeS system started from certain key architectural drivers. The quality attributes chosen as key architectural drivers were the performance QAS and some Availability QAS. These chosen QAS also related to some functional requirements. To ease the documentation of the ADD runs, the related Use Cases are also mentioned. The architectural drivers that were chosen for this decomposition are:

- **P1: Timely closure of valves**
  - UC7: Send trame to remote device
  - UC13: Send alarm

  Performance 1 has certain qualities it demands. All the demands are about how certain alarm trames need to be processed. In case of a gas alarm trame, the incoming alarm trame must be processed with the highest priority. The outgoing control trame to close the gas valve also must be treated with the highest possible priority. This to ensure a response measure of less than 10 seconds between the alarm trame arriving and the control trame being sent. In case of a water alarm trame, the incoming alarm trame must be processed with normal priority. The outgoing control trame must be treated with the highest priority, like the outgoing control trames to close gas valves. The response time between an incoming water alarm trame and the corresponding outgoing control trame must be less than two minutes. Finally, in case of a power alarm trame, ReMeS must process incoming and outgoing with normal priority. The response time must be less than 3 minutes. We must also ensure a mechanism that avoids starvation of trame processing jobs.

- **P2: Anomaly detection**
  - UC10: Detect anomaly
  - UC9: Notify customer

  Incoming measurement trames will be processed by ReMeS to detect anomalies. It is possible that ReMeS receives new measurement trames at a faster rate than ReMeS can process them. In this case, ReMeS goes into overload modus. This modus will be activated when the throughput is greater than 50 anomaly detections per minute. In normal modus,

ReMeS will process the incoming measurements in first-in, first-out order (FIFO). When ReMeS goes into overload modus, customers subscribed to the premium service level will get priority over the normal service level. Even while in overload modus, there should be no starvation of jobs and 98 percent of the measurements should be handled within 10 minutes after arriving. The load is also balanced over multiple instances of each sub-system.

- **AV1: Measurement database failure**
  - UC8: Send measurement

  It is possible that the internal database for storing measurements fails. This does not affect the availability of other types of persistent data. The database should be at least 99.9 percent be up and running. If it fails, detection of the failure should happen within 5 seconds and the operators are notified within 1 minute. In case of a failure, the database goes in degraded modus. This means that incoming measurements are temporarily stored elsewhere and are processed when the database returns operational. This buffer should be able to store at least 3 hours of measurement trames. No measurements are lost when ReMeS switches from normal to degraded modus. The user interface should also display clearly that the measurement data is temporarily unavailable.

- **AV2: Missing measurements**
  - UC8: Send measurement

  It is possible that certain measurements are missing because of a external communication failure, a remote module failure or an internal subsystem failure. To detect this, ReMeS must be able to detect a single missing measurement. Remote devices and ReMeS must acknowledge received trames so they can detect failed trames. ReMeS should be able to detect the failure of an internal subsystem autonomously within 1 minute.

- **P3: Requests to the measurement database** In normal modus, the database processes the incoming requests first-in, first-out. Measurements for premium service level are handled within 500ms, other measurements are handled within 1500ms. If the system fails to comply to these deadlines, it goes into overload modus. This means that requests are handled in the order that returns the system to normal modus the fastest, keeping in mind that requests from premium service level customers are handled before other requests and history queries for anomaly detection are prioritized over research purpose history queries. In In overload modus, consumer history queries are allowed to return a stale cached version.

- **M1: Dynamic Pricing**
  - UC15: Generate invoice

  In the near future, utility prices will change dynamically on a minute-to-minute basis. Since ReMeS already has the necessary infrastructure in place to remotely communicate with many customers, providing the current price to them seems like a logical extention. ReMeS will ensure

that this modification takes less than 250 man man months to implement. The costs of this implementation will cost less than 2Mio Euro.

Tactics

To address these drivers, certain tactics have been used. Mainly, performance tactics have been used.

- Performance
  - Resource management
    For considering resource management, the introduction of concurrency and the increase of available resources are the chosen tactics. Increasing resource management is, however, out of the scope of these add runs.
  - Resource arbitration
    For considering resource arbitration, different scheduling policies can be used.
- Availability
  - Fault Detection
    The tactic that was considered to address the most important drivers (for this level of decomposition), was the use of a heartbeat mechanism. The eventual decision and explanation will be given later on.

Architectural Patterns

The architectural patterns used to effect the tactics chosen in the previous section will be explained in this section.

- Active Object
  Concurrency is an important requisite for the system that is being developed. Being able to execute operations of components within their own threads of control can help achieve this goal. The use of an active object architecture also improves the ability to issue requests on components without blocking until the requests execute.

  It should also be possible to schedule the execution of requests according to specific criteria, such as customer type based priorities and load based priorities.
- Command processor
  The command processor pattern will be used for the implementation of different schedulers. Often the functionality of the scheduler, as described in the command processor pattern, will be split up into two separate components. These components are the buffer where the service requests are stored and the actual scheduler that empties the buffer and schedules the commands for execution. This shows a more explicit view of the workings of such a scheduler without going into another deeper level of ADD design.

- Server Request Handler
  To be able to easily receive messages from the remote modules, we opted for a server request handler pattern.

  It is also possible to use strategies in the aforementioned schedulers. But this will be explained in further iterations of the ADD process when needed.

- Heartbeat
  In order to conform to the driver stating that the system should know about failing components, a heartbeat mechanism will be implemented. Every component that is important enough to be registered, sends heartbeats to the watchdog component at frequent intervals. When the first heartbeat arrives, the watchdog keeps track of that components uptime and notifies the event channel if certain heartbeats are missing. This way components can be dynamically added to the systems uptime monitor (or in this case watchdog).

  The choice was made not to use ping/echo because that would assume a one to many relationship of dependency between the Watchdog and the outside components. In terms of scalability that would lead to a poorer design than the many to one relationship that is in place by using heartbeats. By using heartbeats, the watchdog does not need any knowledge of the system layout or structure in order to function. It is almost completely passive in it's functionality. The responsibility for deciding on the importance of failure detection lies completely with the other components.

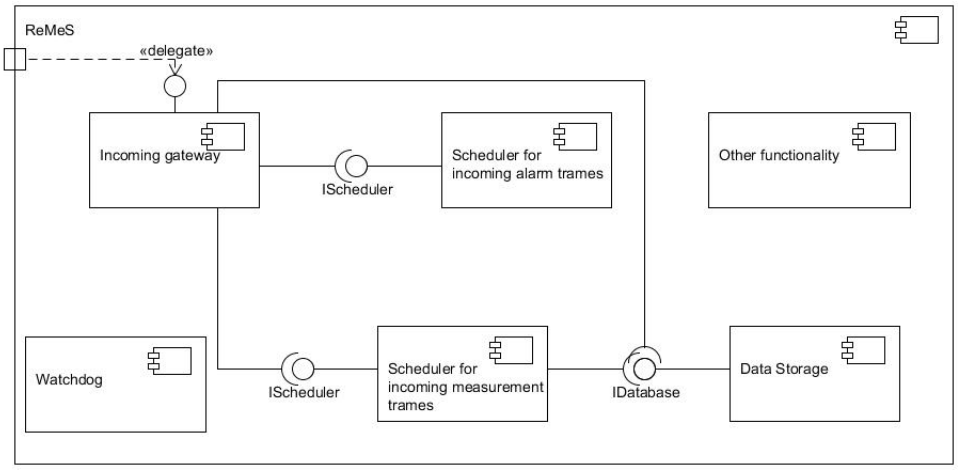


Figure 1: The ReMeS component diagram first decomposition.

The result of this iteration of the ADD process can be shown in figure 1

Verification and refinement of drivers

After the initial decomposition no quality attributes have been completed yet. All considered drivers will be delegated to child components to act as drivers for their individual decomposition.

Incoming gateway

- **UCx:** Know the type of the trame
- **UCz:** Send acknowledgement
- **UC8':** Send Measurement
- **UC13':** Send alarm
- **P1':** Timely closure of valves

Scheduler for incoming measurement trames

- **UC8':** Send measurement
- **AV1:** Measurement database failure
- **AV2:** Missing measurements
- **P2:** Anomaly detection
- **UCy:** Know the modus
- **UCw:** Retrieve module checking schedule

Scheduler for incoming alarm trames

- **UC13':** Send alarm
- **P1':** Timely closure of valves

Data storage

- **UC8':** Send measurement
- **UC10:** Detect anomaly
- **AV1:** Measurement database failure
- **P2:** Anomaly detection
- **P3:** Requests to the measurement database
- **UCy:** Know the modus

Watchdog

- **AV1:** Measurement database failure
- **AV2:** Missing measurements

Other functionality

- **P1':** Timely closure of valves
- **UC7:** Send trame to remote device
- **UC13':** Send alarm

- **UC9:** Notify customer
- **M1:** Dynamic pricing
- **UC15:** Generate invoice
- **UC16:** Mark invoice paid

2.2 Iteration 1: Level 1

2.2.1 Incoming gateway

In this section, we will decompose the "Incoming Gateway" component.

Architectural drivers

The main purpose of the incoming gateway is separating the measurement from the alarm trames and other types of trames in the future, if need be. Therefore the chosen architectural drivers are the following:

- **UCx: Know the type of the trame**
  This use case is a new use case. It is not described in the assignment. What this use case will do is decompose the incoming trame to find out what type of trame it is.
- **UCz: Send acknowledgement**
  This use case is a new use case. If a measurement or alarm trame is received by ReMeS, ReMeS will send an acknowledgement of receiving the trame back to the sending device.
- **UC8': Send measurement**
  The incoming gateway will partially solve the use case "Send measurement" because it will make sure that the measurement trame is sent to the correct component of the system, after receiving the measurement trame and acknowledging it.
- **UC13': Send alarm**
  The incoming gateway will partially solve the use case "Send alarm" because it will make sure that the alarm trame is sent to the correct component of the system, after receiving the alarm trame and acknowledging it.
- **P1': Timely closure of valves**
  A mechanism should be in place to avoid starvation of (alarm) trame processing jobs. Alarm trames should be handled as soon as possible and ultimately within a hard deadline.

Tactics

To efficiently solve these drivers, the following tactics were used.

- Performance

– Reduce computational overhead
In order to maximize the performance of the system, it is important to minimize the computational overhead caused by switching between functionality based on the type of incoming trame.

Architectural Patterns

The architectural patterns that are used to effect the tactics are the following:

- Message Router
The message router pattern localizes the router logic for the incoming trames. By using a message router implementation in the incoming gateway, it is possible to cleanly separate different functionalities from each other. It is also has a high impact on modifiability aspects of the system. Although this is not one of the main QAS, it is considered an important factor in good quality architectures to be extensible and minimize the ripple effect of adding new changes. By using this pattern, adding new functionality based on new types of trames is made easier.

  Message routers consume messages from one message channel (the channel where all external messages are received) and reinsert them into different message channels, depending on a set of conditions (alarm or measurement trames).

- Invoker
To be able to easily receive messages from the remote modules, we opted for an invoker pattern.

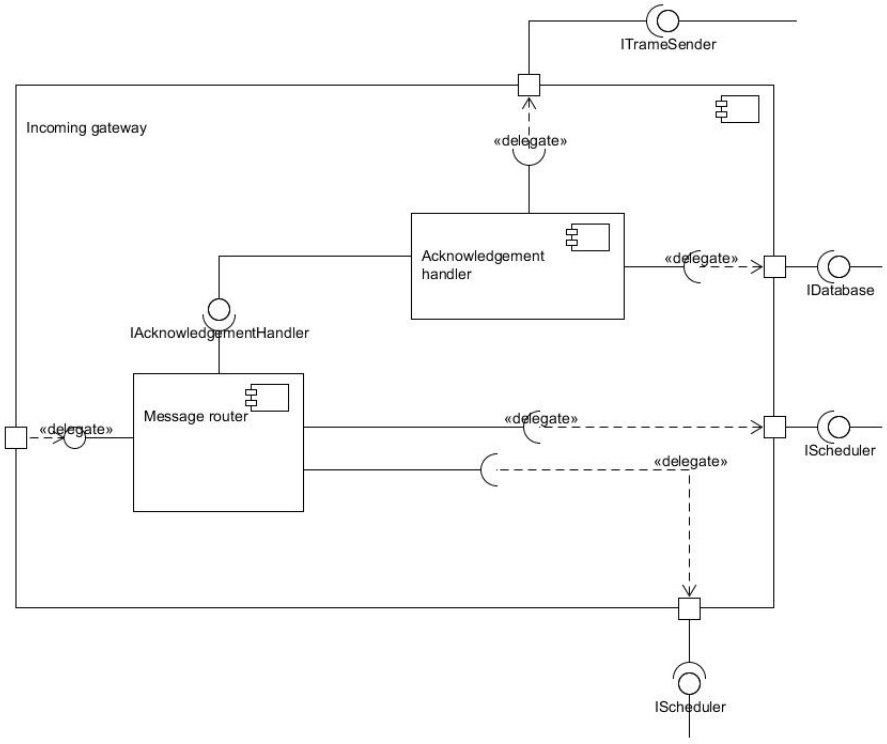


Figure 2: The Incoming Gateway after decomposition

Verification and refinement of drivers

After this decomposition, UCx (Know the type of trame) will be satisfied by the Message Router component. The partial functionalities and quality attributes described in UC8'(Send measurement), UC13'(Send alarm) and P1'(Timely closure of valves) are all handled and satisfied in the Message Router component. The sending of acknowledgements on receiving certain trames, which is described in UCz(Send Acknowledgement) is handled by the component Acknowledgement Handler.

Message Router

- **UCx:** Know the type of trame
- **UC8':** Send measurement
- **UC13':** Send alarm
- **P1':** Timely closure of valves

Acknowledgement Handler

- **UCz:** Send Acknowledgement

2.2.2 Scheduler for incoming measurement trames

This section will explain our reasons behind the decomposition of the "Scheduler for incoming measurement trames" component.

Architectural drivers

This component has to ensure that the incoming measurement trames are handed in the correct order to the data storage component. It also must temporary store the incoming measurement trames in case that the measurement database fails. Lastly, this component detects missing measurements.

- **UC8': Send measurement** If a trame from a remote device is the first device sent, the system will mark the device as active.
- **AV1: Measurement database failure** The scheduler for incoming measurement trames contains a buffer that can store at least 3 hours of data in case of a measurement database failure.
- **AV2: Missing measurements** ReMeS must be able to detect missing measurement updates from remote modules and notify an operator.
- **P2': Anomaly detection** In normal modes ReMeS processes the incoming measurements in a first-in, first-out order. In overload modus, ReMeS gives priority based on the SLA with the customer.
- **UCy: Know the modus** For the system to know what order to hand the measurement trames to the data storage component, it must know the current modus of the system.

Tactics

- Performance
  - Scheduling Policy
    To ensure the correct order or the incoming measurement trames, keeping in mind the modus of the system and the subscription type of the customers we chose for a scheduling policy

Architectural Patterns

- Active Object
  The active object pattern allows us to insert new measurement trames in a scheduler. The scheduler will then decide itself which trames should be processed first. The active object will be implemented as a command and will be handled by a command processor structure in this case. The buffer that is used by the scheduler should also be large enough to hold at least 3 hours of data in case of a measurement database failure. This way, the availability driver will be addressed.

- Publish-Subscribe
  This pattern allows us to efficiently notify all interested parties about the modus of the system and more importantly, to be notified of changes in modus caused by other components. We will also use this pattern to notify an operator through the user interface by this publisher-subscriber pattern. The reason for notifying operators will be addressed in a further decomposition.

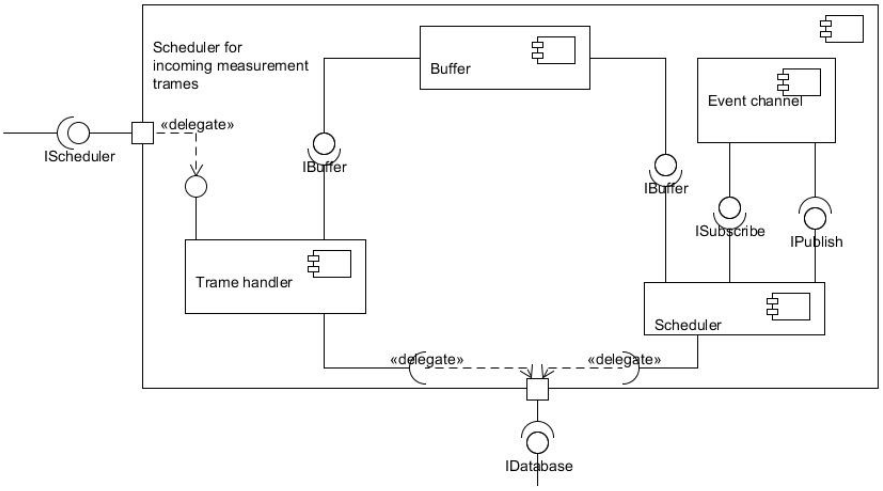


Figure 3: The Scheduler for Incoming Measurement Trames after decomposition

Verification and refinement of drivers

After the decomposition, UCy and P2 are fully satisfied by the scheduler. AV1 will be partially handled by the buffer of the scheduler component. This buffer will be large enough to contain 3 hours worth of data. AV1 and AV2 will be (at least partially) delegated to child nodes Trame Handler and Buffer for further decomposition. The Decomposition of Scheduler could show the implementation of a strategy pattern to effect the different policies of scheduling.

Trame Handler

- **AV2: Missing measurements**

Buffer

- **AV1: Measurement database failure**

2.2.3 Scheduler for incoming alarm trames

This section will explain our reasons behind the decomposition of the "Scheduler for incoming alarm trames" component.

Architectural drivers

The main purpose of this component is making sure that the trames are processed in the correct order, according to their priority. Incoming gas alarm trames must be treated with the highest priority and the incoming water and power alarm trames with normal priority. This component will make sure that this happens. This component will handle the following drivers:

- **UC13': Send alarm** The scheduler for incoming alarm trames will partially help in solving this use case since it will notify the correct system component to send a control trame to the remote valve.
- **P1': Timely closure of valves** ReMeS will process incoming gas alarm trames with a higher priority as water and power alarm trames. ReMeS must also make sure that starvation of alarm trames is not possible.

Tactics

- Performance
  - Scheduling Policy
    To ensure that certain incoming alarm trames such as gas alarm trames are treated with the highest priority, we chose for a scheduling policy. The scheduling policy will also make sure that starvation is not possible.

Architectural Patterns

- Active Object
  The active object pattern allows us, as in the Scheduler for incoming measurement trames, to insert new alarm trames in a scheduler. The scheduler will then decide itself which trames need to be processed first. The active object will be implemented in the same way as in the Scheduler for incoming measurement trames. The buffer that is used by the scheduler shouldn't be as big as with measurement trames.
- Proxy
  To increase the modifiability of the internal working of this component we chose to use the explicit interface pattern. The interface that we see outside the component will simply delegate it's methods to an internal component.

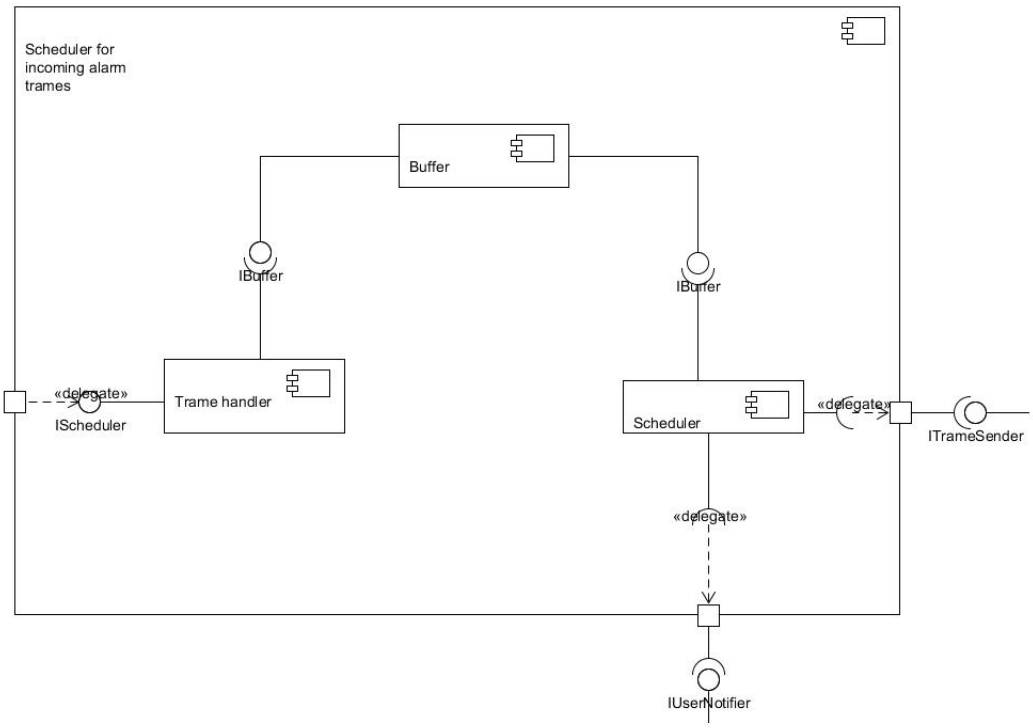


Figure 4: The Scheduler for Incoming Alarm Trames after decomposition

Verification and refinement of drivers

After the decomposition, P1 will be partially satisfied by the scheduler. Some incoming alarm trames will be prioritized while starvation of certain incoming alarm trames is not possible. UC13 will be delegated to another component in our system.

2.2.4 Data Storage

In this section, the data storage component will be decomposed.

Architectural drivers

- **UC8':** Send measurement
  ReMeS looks up the customer associated with the device sending the measurement, processes and stores the measurement in his consumption record.
- **UC10:** Detect anomaly
  ReMeS checks measurements for anomalies. If anomalies are found, the

appropriate recipients need to be notified and possibly actuators need to be operated remotely. For the detection of anomalies, also all of the measurement history needs to be fetched from the database.

- **AV1:** Measurement database failure
  If the database fails, it should do so gracefully and not impact other services for the time being. All concerning components need to be notified when going down.
- **P2:** Anomaly Detection
  There are different modi to be considered: A normal mode and an overload mode. Multiple instances need to handle the load that has to be processed for load balancing purposes. All requests need to be handled eventually. No starvation can occur while processing requests.
- **P3:** Requests to the measurement database
  In normal modus, the database processes the incoming requests first-in, first-out. If the system fails to comply to the specific deadlines listed below, it goes into overload modus in which requests are handled in the order that returns the system to normal modus the fastest, thereby taking into account the SLA with the customer and the origin of the request. It should be possible to return a stale cached version when in overloaded modus.
- **UCy:** Know the modus
  As previously indicated this module needs to be aware of the modus of operation, and possibly be able to change that modus.

Tactics

- Performance
  - Resource arbitration
    A scheduling policy will need to be applied in order to address the case of scheduling requests while keeping in mind the different priorities and origins of the requests.
  - Resource Management
    In order to address the case of load balancing, at least some form of concurrency needs to be effected. These last two tactics take precedent over any other tactics that may conflict because of the priority assigned to the quality attributes from which these tactics where chosen.
- Availability
  - Fault detection
    In order to address the graceful failure of different components in the system, said failure needs to be detected before it can be addressed. This however will be done by the watchdog component described in higher levels. It is merely mentioned here as a tactic because this component will probably be heavily influenced.

Architectural Patterns

- Active object
  In order to implement and use the different tactics to decompose this module, certain patterns were (re)used. This architecture relies quite heavily on the active object pattern. These active objects will, again, be implemented as commands which will be scheduled by a command processor. The choice of reusing the active object implementation makes it possible to pass along commands as service requests in a more easy and general way, improving performance (not needed to convert messages all the time) and improving extendability.
- Publish/Subscribe
  The publish/Subscribe (or Pub/Sub) pattern is reused to provide components of a mechanism to know and if needed, let other components know of a change in modus. For ease of explanation, the event channel is shown as a new component, but actually, this component is the same component as shown in other decompositions. Each component in the ReMeS system that uses an event channel, actually uses the same event channel.

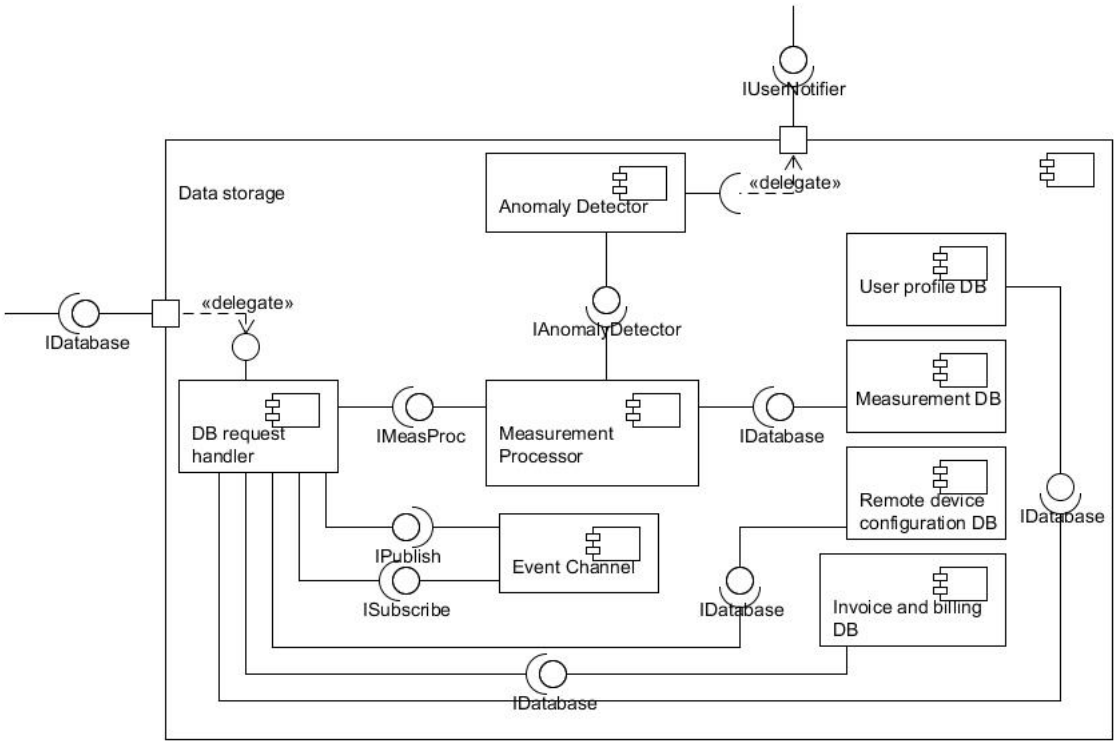


Figure 5: The Data Storage component after decomposition

Verification and refinement of drivers

For this decomposition, UC8(Send Measurement) has been completely addressed. UC10(Detect anomaly) has been addressed but the functionality of notifying alarm recipients has been delegated outside this component. UCy has also been completely addressed. As for the detection of failures as described in Av1, this functionality has been delegated to an outside component (watchdog). Only P2 and P3 (the major drivers) are only partially addressed and need to be explained further in a lower level decomposition although even in this level the structures for addressing these drivers have already been implemented.

DB Request Handler

- **P3:** Requests to the measurement database

Anomaly detector

- **P2:** Anomaly detection
- **P3:** Requests to the measurement database

2.2.5 Watchdog

Architectural drivers

The main drivers for this component are the quality attributes that are related to the detection of failing components. These events should be detected and operators should be notified within a certain time interval. In this case the time interval is 5 seconds.

- **AV1:** Measurement database failure The detection of failed components happens within 5 seconds.
- **AV2:** Missing measurements The failure of any component in the system should be detected and an operator should be notified.

Tactics

The tactics used to effect these drivers are the following.

- Availability
  - Failure detection
    The use of a heartbeat system as described in the upper layer decomposition of the ReMeS system will be used.

Architectural Patterns

- Heartbeat
  Each component for which failures should be detected, should be instantiated with a reference to the watchdog component. Every component

should send at least one heartbeat within a 5 second deadline of the former heartbeat sent.

The watchdog component monitors these heartbeat and starts tracking a component from the moment a first heartbeat arrives. When the watchdog detects missing heartbeats, a notification is sent to the event channel for all interested parties to see.

Although not one of the main drivers, the modifiability attribute will be affected in a positive manner because of this form of runtime registration. Deferring the binding time in this manner causes a lower coupling between this component and the others in the system.

- Publish/Subscribe
  The publish/subscribe pattern which is used in a few other decompositions all over the system, is also used in this component to notify interested parties of a change in state of a component. This change in state is determined by the absence of heartbeats. The event channel depicted, is the same event channel used throughout other components in the system.

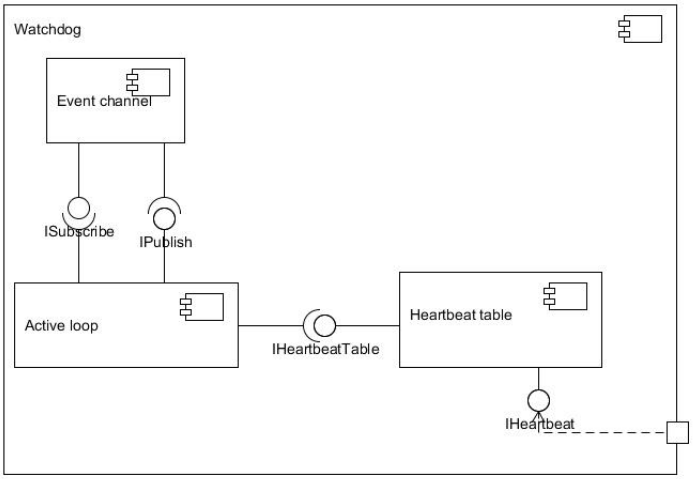


Figure 6: The Watchdog after decomposition

Verification and refinement of drivers

All drivers for this decomposition have been addressed. No further decomposition of this module is needed.

2.3 Iteration 1: Level 2

2.3.1 Trame Handler

Architectural drivers

- **UCw: Retrieve module checking schedule**
  For the scheduler for incoming measurement trames to check whether or not some measurements are missing, it must be able to retrieve the rate that the remote module will be sending measurements. It can then compare the timestamps of the latest measurement trames with this rate to see if some measurements are missing.
- **AV2: Missing measurements**
  ReMeS should be able to detect a single missing measurement by monitoring the measurement schedule Operator should be notified after missing measurements.

Tactics

- Availability
  - Fault detection
    An implementation of a Heartbeat system is used to check whether all devices send their measurements in time.

Architectural Patterns

- Heartbeat
  The device's measurement trames are used as heartbeats in this implementation. An active loop component keeps track of the new entries in the heartbeat table and monitors them for absent measurements. The active loop component also has the capability of querying the data storage for the send schedule of the different devices. This component can also mark devices as active in the data storage component when a device sends it's first trame.
- Active Object
  The active object pattern is used to provide a way of making an object run independently in its own thread of control for monitoring the heartbeat table for missing measurements. The active loop component represents an active object in this decomposition.

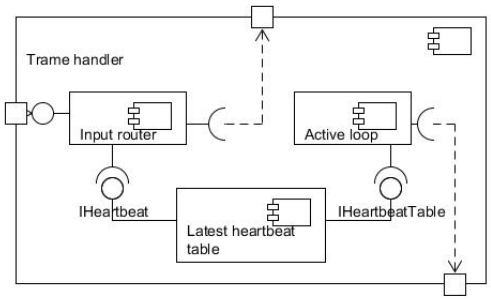


Figure 7: The Trame Handler after decomposition

Verification and refinement of drivers

All drivers for this module have been met and addressed. No further delegation to child nodes is needed.

2.3.2 Buffer

This section handles the decomposition of the Buffer component inside the Scheduler for incoming measurement trames.

Architectural drivers

- **AV1: Measurement database failure**
  When switching between normal and degraded modus, no new measurements are lost. When the database component fails, incoming measurements should be stored elsewhere and it should be possible to store at least 3 hours of measurement messages.

Tactics

- Availability
  - Recovery Preparation and repair
    In order to make sure that 3 hours worth of messages can be stored in case something goes wrong, and to make this storage more prepared against failures, some form of active redundancy needs to be implemented.

Architectural Patterns

- Replicated Component group
  The replicated component group pattern ensures that the buffer holding

the measurement trames is in fact distributed over different locations making this buffer more redundant. In case of failure no new measurements will be lost since there will always be a buffer instance ready to store this data. The buffer should be able to store 3 hours worth of data.

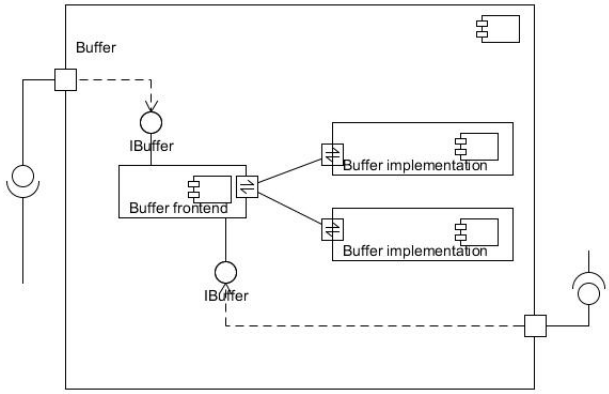


Figure 8: The Buffer after decomposition

Verification and refinement of drivers

All drivers for this module have been addressed and fulfilled.

2.3.3 DB Request Handler

This section handles the decomposition of the DB Request Handler inside the Data Storage component.

Architectural drivers

- **P2:** Requests to the measurement database
  In case of a database failure, a stale cache version should be returned when requests can't be handled.

Tactics

- Performance
  - Resource Management
    In order to conform to the drivers, a form of cache should be used to maintain multiple copies of data.
- Modifiability
  - Localize Changes
    Since this module is the main entry point for requests to the data

storage unit, it is important to offer a general interface to the outside components. Generalizing the module will address this.

Architectural Patterns

- Proxy
  A proxy is used to offer the generic data storage component interface to the other components in the system. This improves the simplicity in making requests to the database. This also unifies the different types of database services offered into one accessible interface.
- Active Object
  Again the active object is used and implemented as a command alongside a command processor structure. Data transfer objects will be used to transfer data to and from the databases itself.
- Chain of responsibility
  In order to accommodate for the caching of data, the chain of responsibility can be used to give the parallel processor and the Request Cache the opportunity to decide themselves if they can handle a request or if they should forward it. The Request Cache component can forward the request to the request process in cache a request is not available in cache and a request to the actual database needs to be made.

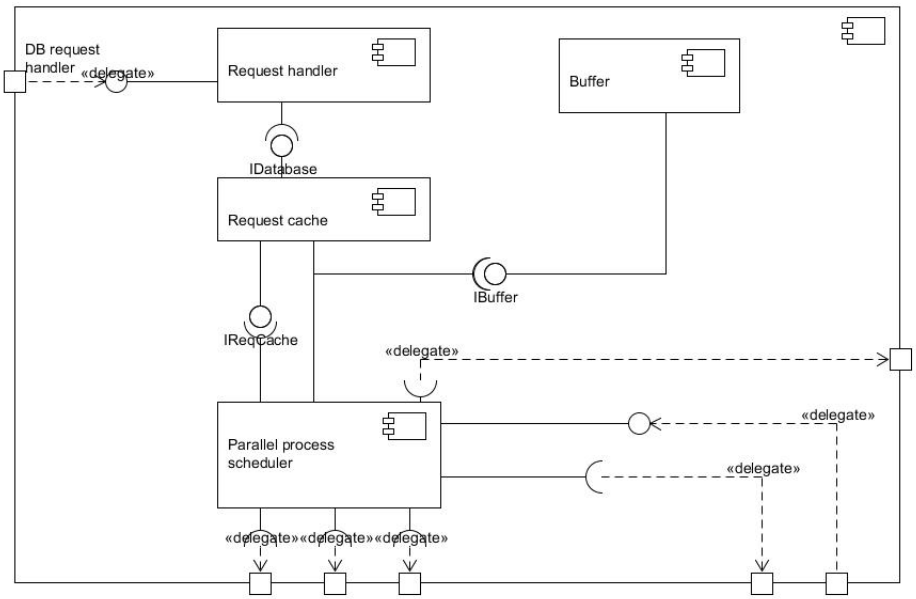


Figure 9: The DB Request Handler after decomposition

In the diagram of the DB request handler, the ports that are shown to connect this component to the outside components are placed in the same location as the

connections of this component in a diagram representing a level higher. Such a higher level diagram can be found in figure 5. The triplet of outgoing ports connect to the different database instances and the duo of connectors to the side interface with the event channel.

Verification and refinement of drivers

Using these patterns, the drivers for this decomposition have been fully addressed. There is no real need to decompose this module further.

2.4 Iteration 2: Level 1

2.4.1 Other functionality

Decomposing this module actually represents doing another iteration of ADD. In further diagrams, the components that will arise from this decomposition will not be subcomponents of the other functionality module since this was only a pseudo module for ADD-process purposes.

Solely the drivers from the first decomposition that were delegated to this pseudo-component will be handled. If another iteration is needed for quality attributes or functionality that is not covered by these drivers, then another iteration will follow after this one.

Architectural drivers

- **P1': Timely closure of valves**
  - UC7: Send trame to remote device
  - UC13': Send alarm
  - UC9: Notify customer

  Performance 1 has only been fulfilled partially in the first iteration. Our current ReMeS decomposition is unable to send outgoing messages or trames. But to close the valves we obviously need outgoing control trames. These outgoing control trames also must be treated with different priorities. Again, starvation of certain outgoing messages cannot happen.
- **M1': Dynamic pricing**
  - UC15: Generate invoice

  In the near future, utility prices will change dynamically on a minute-to-minute basis. Since ReMeS already has the necessary infrastructure in place to remotely communicate with many customers, providing the current price to them seems like a logical extention. ReMeS will ensure that this modification takes less than 250 man man months to implement. The costs of this implementation will cost less than 2Mio Euro.

Tactics

- Performance
  - Resource arbitration
    For considering resource arbitration, different scheduling policies can be used.

Architectural Patterns

We will be using mainly the same patterns as we used in the first decomposition of ReMeS. This is because the motivation and the reasoning behind the first decomposition on this level hasn't changed.

- Active Object
  Concurrency is an important requisite for the system that is being developed. Being able to execute operations of components within their own threads of control can help achieve this goal. The use of an active object architecture also improves the ability to issue requests on components without blocking until the requests execute.

  It should also be possible to schedule the execution of requests according to specific criteria, such as customer type based priorities and load based priorities.

- Command processor
  The command processor pattern will be used for the implementation of different schedulers. Often the functionality of the scheduler, as described in the command processor pattern, will be split up into two separate components. These components are the buffer where the service requests are stored and the actual scheduler that empties the buffer and schedules the commands for execution. This shows a more explicit view of the workings of such a scheduler without going into another deeper level of ADD design.

- Client Request Handler
  To send messages across the internet, through sms or gprs, we opted for a client request handler pattern.

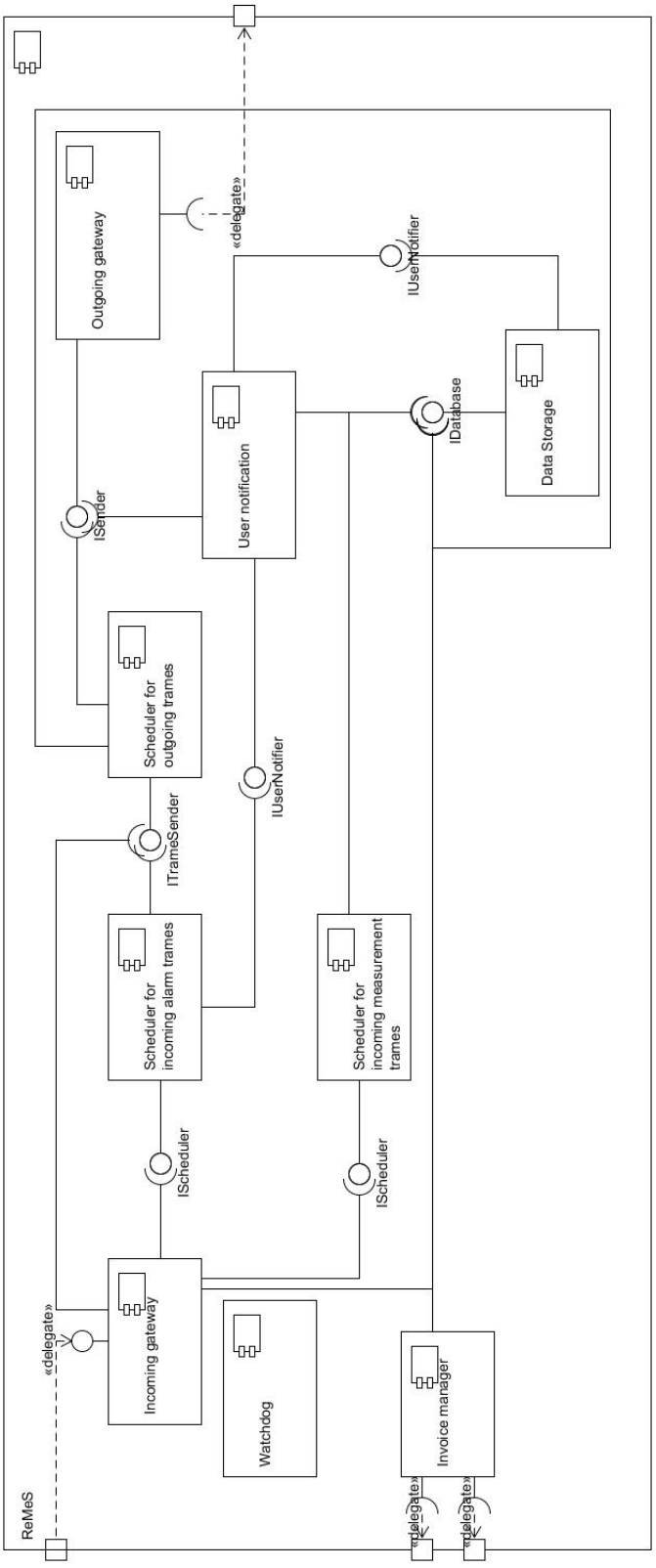


Figure 10: The decomposition of ReMeS after the second iteration

Verification and refinement of drivers

No quality attributes have been completed. All drivers are delegated to child components:

Scheduler for outgoing trames

- **P1':** Timely closure of valves
- **UC13':** Send alarm
- **UC7':** Send trame to remote device

User notification

- **UC9':** Notify customer
- **UC13':** Send alarm

Outgoing gateway

- **UC7':** Send trame to remote device
- **UC9':** Notify customer

Invoice Manager

- **M1:** Dynamic pricing
- **UC15:** Generate invoice

2.4.2 Scheduler for outgoing trames

Architectural drivers

This component will manage outgoing control trames. It will prioritize important trames over others without having starvation. This component also has to construct the outgoing control trames, so that the receiver can correctly read all the contained information. Finally, this component will determine the communication channel to reach the remote device with.

- **P1': Timely closure of valves** Outgoing valve control trames have a certain priority. Gas and Water valve control trames have the highest priority. Others have normal priority.
- **UC7': Send trame to remote device** ReMeS wants to send trames to remote devices to control those devices.
- **UC13': Send alarm** ReMeS has to send control trames to the remote actuators to close them in case that a leak is detected.

Tactics

- Performance

– Scheduling Policy
To ensure the correct order of outgoing trames to remote devices, we chose for a scheduling policy.

Architectural Patterns

- Message Translator
To construct the trame to send to the remote device, we opted for a message translator pattern
- Active Object
The active object pattern allows us to insert new outgoing trames (or atleast requests for sending outgoing trames) in a scheduler. The scheduler can then decide which requests to handle first.
- Proxy
To increase the modifiability of the internal working of this component we chose to use the explicit interface pattern. The interface that we see outside the component will simply delegate it's methods to an internal component.

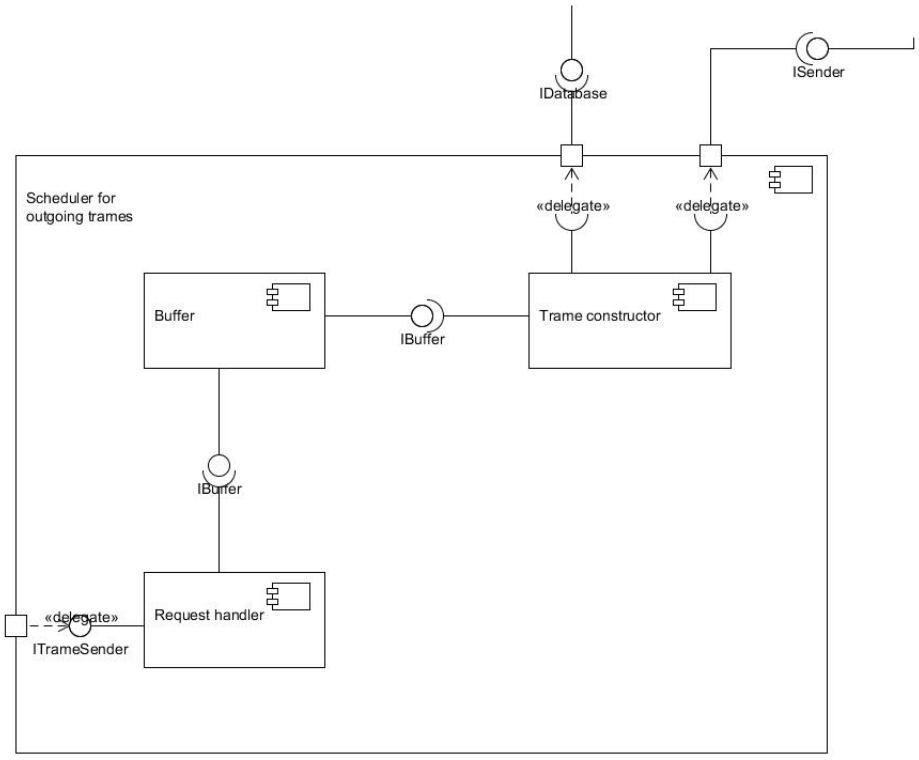


Figure 11: The Scheduler for Outgoing Trames after decomposition

Verification and refinement of drivers

The final part of P1 has been handled by this component. UC13' also was fulfilled by this component. Only UC7 is partially finished. The trames to send have been constructed, but not yet sent. But that part of UC7 will be done by another component in the system.

2.4.3 User notification

Architectural drivers

This component will manage outgoing customer notifications. It wont prioritize any requests. The component handles the request First-in-First-out (FIFO). The component will construct the messages and determine which communication channel to use.

- **UC9': Notify customer** Whenever ReMeS wants to send a notification to a customer, ReMeS should address this component.
- **UC13': Send alarm** When ReMeS receives an alarm trame, the customer needs to be notified (if the customer has indicated this in his profile).

Tactics

- Performance
  - Resource arbitration
    Different scheduling policies can be used. Currently the FIFO scheduling policy is used in this component.

Architectural Patterns

- Message Translator
  To construct the notification to send to the remote device, we opted for a message translator pattern. This pattern allows us to create standard user notifications such as "Gas leak detected in your home."
- Active Object
  The active object pattern allows us to insert new user notification requests in a scheduler. The scheduler can then decide which requests to handle first. Currently the scheduler will handle the requests FIFO.
- Proxy
  To increase the modifiability of the internal working of this component we chose to use the explicit interface pattern. The interface that we see outside the component will simply delegate it's methods to an internal component.

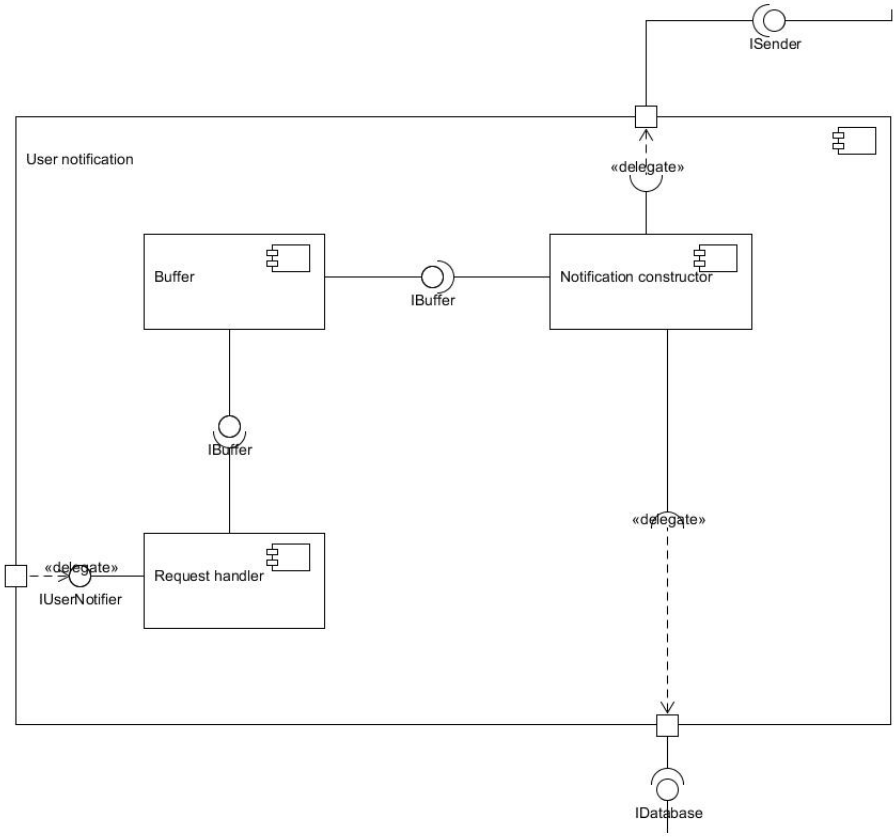


Figure 12: The User Notification component after decomposition

Verification and refinement of drivers

Both UC9' and UC13' have been handled by this component. The second part of UC9 (sending the message) will be handled by the outgoing gateway.

2.4.4 Outgoing gateway

Architectural drivers

This component can construct SMS, gprs or internet messages and send them.

- **UC7': Send trame to remote device** The final part of this usecase is sending the trame through the selected communication channel.
- **UC9': Notify customer** ReMeS will send the constructed message to the recipient.

Tactics

- Performance
  - Resource arbitration
    Different scheduling policies can be used. Currently the FIFO scheduling policy is used in this component.

Architectural Patterns

- Active Object
  The active object pattern allows us to insert new user notification requests in a scheduler. The scheduler can then decide which requests to handle first. Currently the scheduler will handle the requests FIFO.
- Proxy
  To increase the modifiability of the internal working of this component we chose to use the explicit interface pattern. The interface that we see outside the component will simply delegate it's methods to an internal component.
- Requestor
  To send messages across the internet, through sms or gprs, we opted for a requestor pattern.

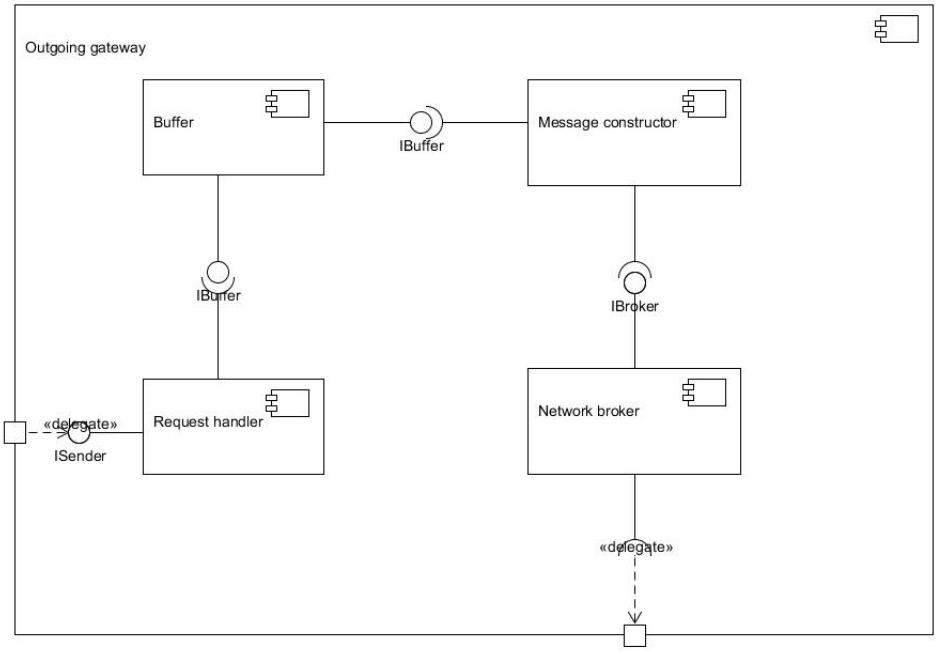


Figure 13: The Outgoing Gateway after decomposition

Verification and refinement of drivers

All the architectural drivers have been satisfied.

2.4.5 Invoice manager

Architectural drivers

This decomposition starts solely with a functional requirement

- **M1:** Dynamic pricing
  Room should be left for extending the functionality of the ReMeS system to incorporate a smart billing service. Communication between UIS and ReMeS should be made possible. Billing and invoicing generation and handling should be provided. All this does not affect remote metering, actuation, leak and anomaly detection.
- **UC15:** Generate invoice
  ReMeS processes the measurements since the last bill and constructs the invoice. ReMeS contacts the Utility Provider to get contract information and pricing information. The invoices are stored in ReMeS and marked as unpaid.

Tactics

- Modifiability
  - Defer Binding Time
  - Prevention of ripple effect

Architectural Patterns

- Explicit Interface
  The use of an explicit interface allows this component to change it's internal functionality and components independently from the interface it provides to the outside components.

  In combination with different ways of triggering the functionality, depending on the final implementation, the components that will be affected by a change, will be minimized. The different ways of triggering have been addressed in the ADD assumptions in section 2.6

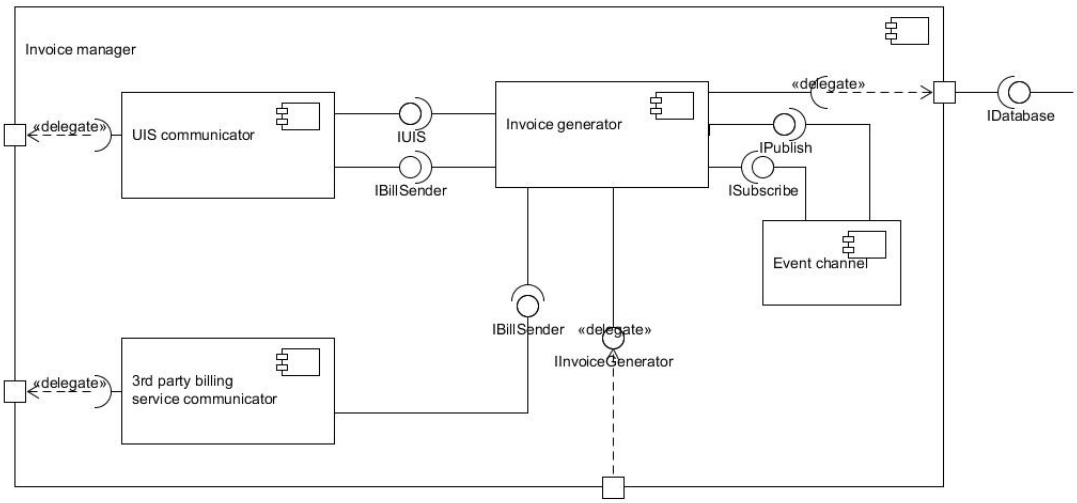


Figure 14: The Invoice Manager after decomposition

Verification and refinement of drivers

It is possible to refine some components further since not all functionality has been addressed yet.

Invoice generator

- **UC15:** Generate invoice

UIS communication handler

- **M1:** Dynamic pricing

2.5 Iteration 3: Level 1

2.5.1 ReMeS

This decomposition will manage all use cases that are not related to any QAS.

Architectural drivers

- **UC1:** Log in
- **UC2:** Log off
- **UC3:** Register customer
- **UC4:** Unregister customer
- **UC5:** Associate device to customer
- **UC6:** Customize customer profile

- **UC11:** Operate actuator remotely
- **UC12:** Set alarm recipients
- **UC14:** Request consumption predictions
- **UC16:** Mark invoice paid
- **UC17:** Perform research

Most of these use cases are dependent on User Interaction. Therefor we will introduce a new component: User Interaction.

Tactics

- Performance
  - Resource arbitration
    Because consumption predictions are a computational intensive task, we will foresee a scheduler for these requests

Architectural Patterns

- Command processor
  The command processor pattern will be used for the implementation of the consumption prediction scheduler. Often the functionality of the scheduler, as described in the command processor pattern, will be split up into two separate components. These components are the buffer where the requests are stored and the actual scheduler that empties the buffer and schedules the consumption predictions. This shows a more explicit view of the workings of such a scheduler without going into another deeper level of ADD design.

Verification and refinement of drivers

None of the architectural drivers have been satisfied. Most of them are delegated to the user interaction component. Consumption predictions are delegated to the scheduler. **User Interaction**

- **UC1:** Log in
- **UC2:** Log off
- **UC3:** Register customer
- **UC4:** Unregister customer
- **UC5:** Associate device to customer
- **UC6:** Customize customer profile
- **UC11:** Operate actuator remotely
- **UC12:** Set alarm recipients
- **UC14':** Request consumption predictions

- **UC16:** Mark invoice paid
- **UC17:** Perform research

Scheduler for consumption prediction requests

- **UC14':** Request consumption predictions

Computation of consumption prediction

- **UC14':** Request consumption predictions

2.5.2 User Interaction

Architectural drivers

- **AV1':** Measurement database failure If the measurement database fails, this must be clearly indicated in the user interface (which will work through this user interaction component).
- **UC1:** Log in
- **UC2:** Log off
- **UC3:** Register customer
- **UC4:** Unregister customer
- **UC5:** Associate device to customer
- **UC6:** Customize customer profile
- **UC11:** Operate actuator remotely
- **UC12:** Set alarm recipients
- **UC14':** Request consumption predictions
- **UC16:** Mark invoice paid
- **UC17:** Perform research

Tactics

- Security
  - Authenticate users
    We will want to ensure that a user is actually who it purports to be. For this we will use a password to authenticate the users.
  - Authorize users
    Authorization of users ensures that authenticated users have the correct rights to access and/or edit certain information. These access rights may differ between different users. This access control can be by user or by user class.

Architectural Patterns

- Authorization
  We must ensure that only specific clients can access functionality of a subsystem. Therefor we will assign access rights to each client that can send service requests. to the security-sensitive subsystem and check these rights before executing any requests on the subsystem.
- Publish-Subscribe
  This pattern allows us to efficiently notify all interested parties about the modus of the system and more importantly, to be notified of changes in modus caused by other components. We will also use this pattern to notify an operator through the user interface by this publisher-subscriber pattern. The reason for notifying operators will be addressed in a further decomposition.

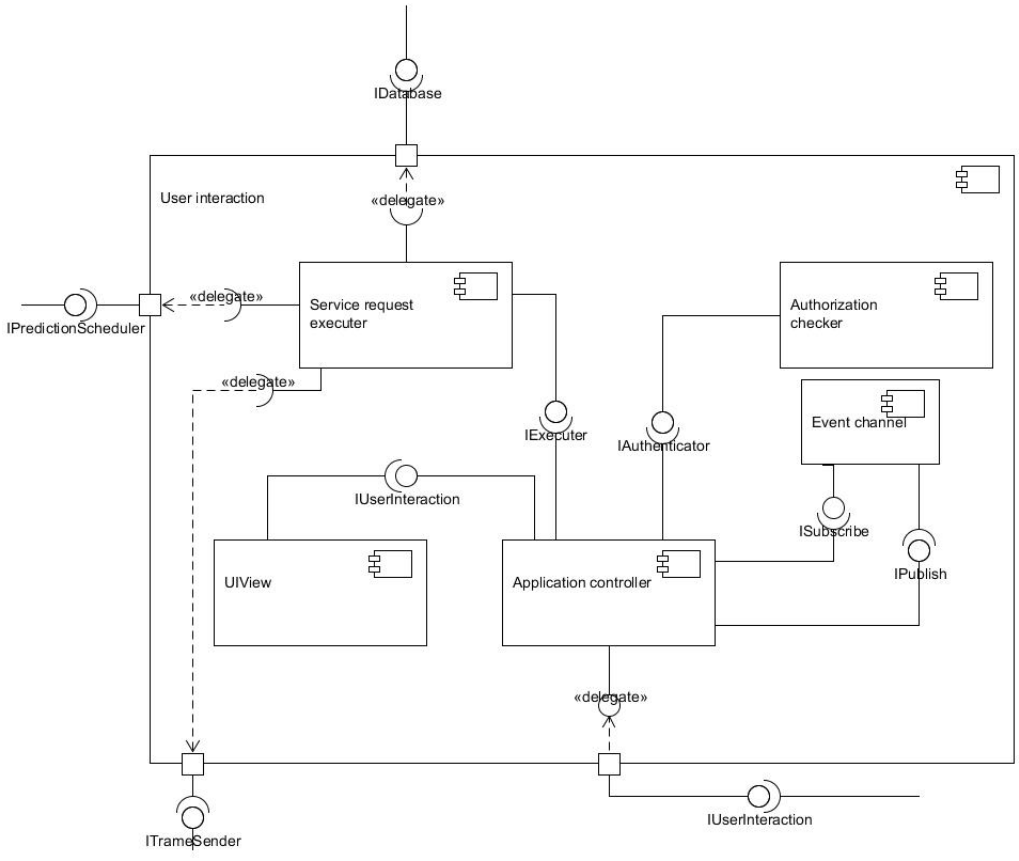


Figure 15: The User Interaction after decomposition

Verification and refinement of drivers

All drivers have been dealt with. Each of these operations is available in the user interaction interface. Before executing any of these operations, the access rights of the client requesting the service will be checked (except for UC1: Log in, here the client will be assigned certain access rights). If the measurement database is unavailable, the user interaction component will be notified of this through the event channel and will update its status appropriately.

2.5.3 Scheduler for consumption prediction requests

Architectural drivers

The main purpose of this component is making sure that new requests for consumption prediction can be saved and scheduled while other computations are still running. When a new computation can be made, the scheduler will acquire the required data from the data storage and hand this information to the computation of consumption prediction component.

- **UC14': Request consumption prediction**

Tactics

- Performance
  - Scheduling Policy
    To ensure that incoming consumption predictions are not lost and handed to the computation of consumption prediction component in FIFO order.

Architectural Patterns

- Proxy
  To increase the modifiability of the internal working of this component we chose to use the explicit interface pattern. The interface that we see outside the component will simply delegate it's methods to an internal component.
- Active Object
  The active object pattern allows us to insert new consumption prediction requests in a buffer which in its turn can be emptied by a scheduler. The scheduler can decide which requests need to be treated first. At the moment, this happens in FIFO order.

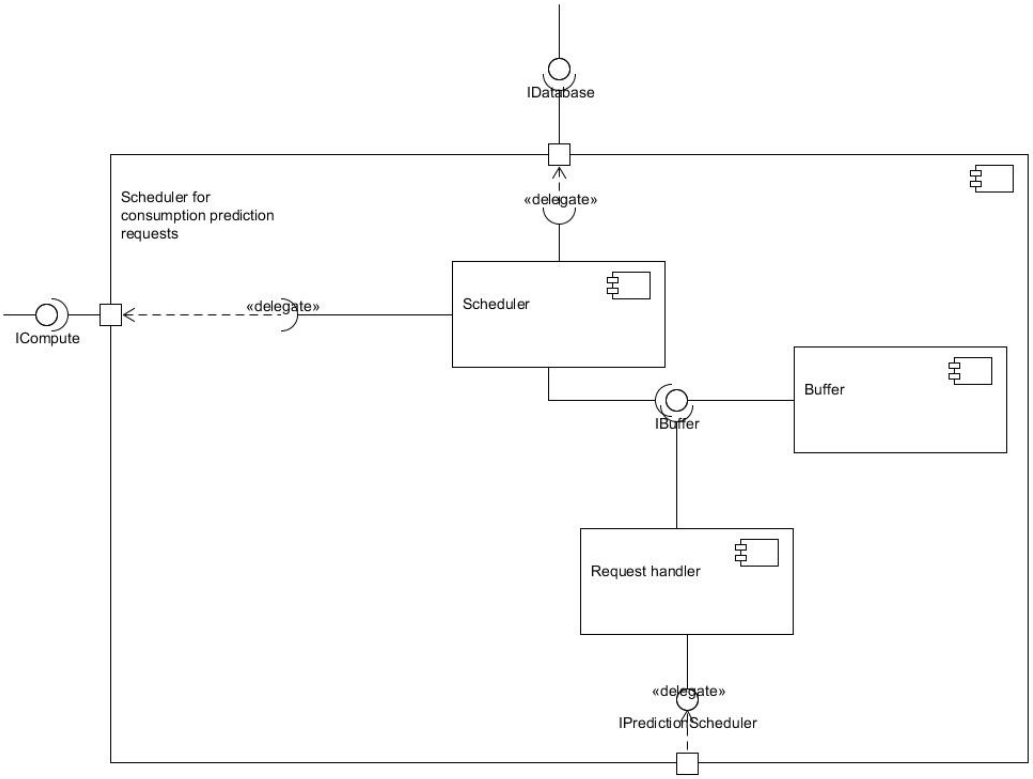


Figure 16: The Scheduler for Consumption Prediction Requests after decomposition

Verification and refinement of drivers

All the required parts of UC14' we needed to address in this component have been addressed. No further delegation is needed.

2.5.4 Computation of consumption prediction

Architectural drivers

The only purpose of this component is computing requested consumption predictions.

- **UC14': Request consumption prediction**

No further decomposition of this component is required.

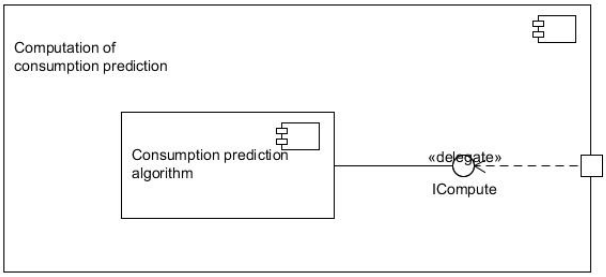


Figure 17: The Computation of Consumption Prediction component after decomposition

Verification and refinement of drivers

The processing of the data part of UC14 has been fulfilled by this component.

2.6 Assumptions

The assumption is made that a guaranteed lack of starvation is infeasible in relation to the processing of alarm trames. There are scenarios possible wherein simply cannot avoid theoretical starvation. The effects will be mitigated as well as possible but other than that, the requested properties cannot be guaranteed.

In relation to the 3 hours of measurement that need to be stored and retained when the database component should fail, the assumption is made that the buffers used, to accommodate for this measure is able to hold 3 hours worth of measurements. The size of the buffer that is needed, has to be determined further and is outside the scope of this document.

In relation to requests for the database component, the assumption is made that each request has a priority, albeit default if not otherwise explicitly stated.

Related to UC16 (Mark invoice paid), the assumption is made that the main actor indicates that the invoice has been received through the user interface. The actor will have this option in his personal portal. This action will drive the functionality for this use case.

To conform to UC15 (Generate invoice), the requirements do not indicate how and or when the generation of invoices is initiated. The assumption is made that only the client can specify the business rules for this case and it is not possible to provide an implementation without consent from the stakeholders. In the mean time, the functionality can be started from calling the generateInvoice() operation. Whichever component calls this to initiate the process, has to be specified later on. It is also possible to start this process in an event-driven manner. By monitoring the event channel, an indication that said process should be initiated, can be received.

This way of working is promoted throughout the system. It is an assumption that incoming requests are processed through a UI portal while the outgoing

results are distributed directly through other channels.

In most, if not all of the significant software projects today, logging technologies are used. The assumption is made that also for ReMeS there will be a component which handles the logging information for different software components in the system. To avoid cluttering the diagrams and explanations, we omit the logger as a physical entity in our decompositions, although it actually is a useful component to have. This Logging component will not be discussed further as it is considered to be out of scope for this assignment.

When concerning the incoming gateway, the assumption is made that a physical device for receiving trames from different networks is installed already and that this device delivers the incoming trames to the incoming gateway. Normally the same boundaries should be assumed as in the outgoing gateway, but due to unforeseen circumstances, this wasn't addressed in the decomposition of the incoming gateway component.

In the case that emergency services need to be notified because of a gas leak, it was not quite clear how this notification should occur. The assumption is made that an sms will be sent to the emergency systems to notify them of the problem.

2.7 Remarks

When using the ADD process to create an architecture for the ReMeS system, the dept of the iterations was limited to two levels in most cases. In some cases, however, another decomposition (level 3) was needed to explain some concepts a bit better.

The concept of iterations was used to describe different phases of the ADD project. An iteration is seen as a complete decomposition to the lowest level starting from the highest level. An iteration ends when there are no more decompositions to be done on a lower level while still dealing with the original top level components. Each time the component called Other functionality gets decomposed to address some previously unresolved quality attributes and/or use cases, a new iteration begins.

It is however the case in this assignment that a limited amount of quality attributes have been supplied to use as main drivers for the add process. Due to this, some functional requirements can not be linked or deduced from the supplied quality attributes. When this is the case, those functional requirements will not be used to instantiate components in the architecture by using the ADD process alone, since ADD stands for Attribute Driven Design. Most of these use cases are related to the user interface. There are no given quality attributes addressing any of the user interface functionalities. These functionalities will be added (albeit in a lower level of detail) in the last iteration and in the final architecture described in section 3. It is however not a strict ADD-methodology that is used for those remaining functions.

In some of the lowest level decompositions, the decomposition seems to be driven solely by functional requirements without quality attributes. Sometimes it is

possible that the non functional drivers for an iteration are already met but another decomposition for child components is needed to illustrate some remaining functional requirements. This is situation is however different from what is discussed in the previous paragraph where there were no non-functional drivers to begin with.

3 Final Architecture Design

3.1 Context diagram

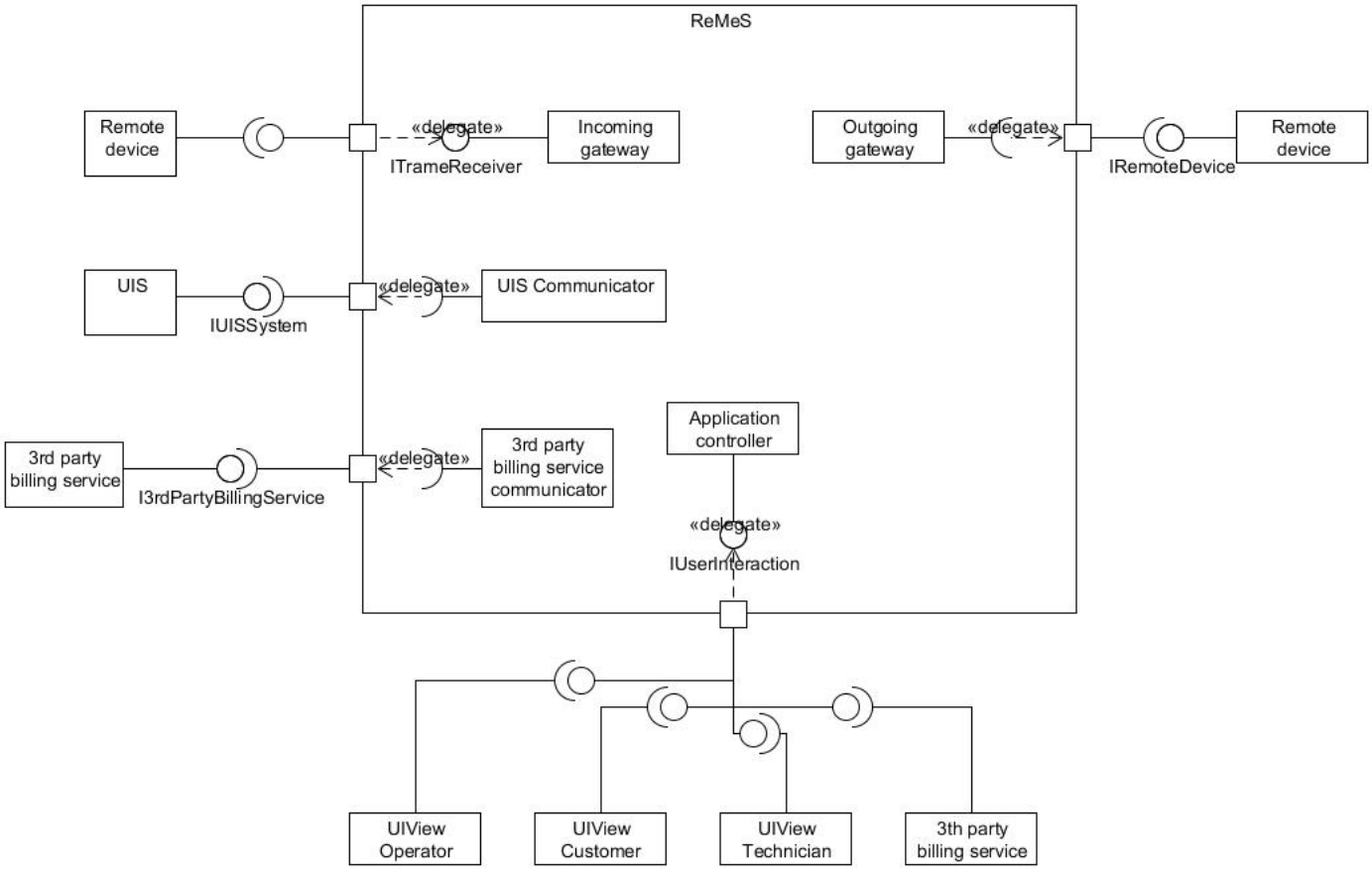


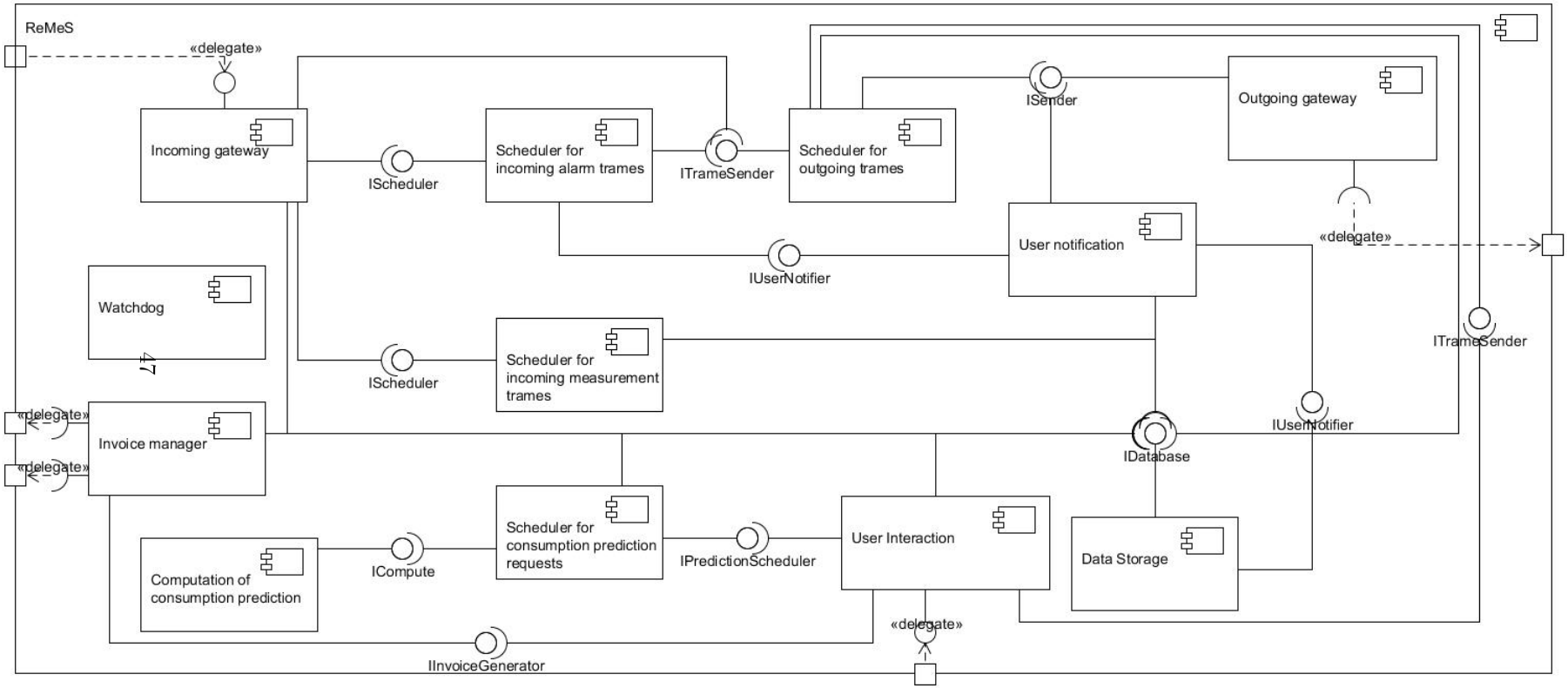
Figure 18: The context diagram for ReMeS.

The context diagram is used to describe the most important external factors that interact with the system. In this case the most important system to view is the ReMeS system itself. This is the system that has been designing thus far.

The most important external actors are the remote device, the users and the UIS (User Information System). The diagram is in fact quite self-explanatory as to how these components use the border components in the ReMeS system in question. The interfaces through which these forms of communication happen, are also included in the context diagram, albeit in name only. The detailed version of these interfaces can be found in section 5.2

3.2 Overall component diagram

The overall component diagram provides a good depiction of how the ReMeS is divided into components that together handle the functional and non-functional requirements.


ReMeS
«delegate»
Incoming gateway
IScheduler
Scheduler for incoming alarm trames
ITrameSender
Scheduler for outgoing trames
ISender
Outgoing gateway
User notification
«delegate»
IUserNotifier
Watchdog
IScheduler
Scheduler for incoming measurement trames
ITrameSender
«delegate»
«delegate»
Invoice manager
IUserNotifier
IDatabase
Computation of consumption prediction
ICompute
Scheduler for consumption prediction requests
IPredictionScheduler
User Interaction
Data Storage
«delegate»
IInvoiceGenerator

This section will explain how the core functionality is supported by the provided architecture. The diagram shown in figure 19 is a depiction of the architecture that came forth out of the previously discussed ADD runs (with a few minor modifications afterwards).

There are a couple of major components that play a vital role in this architecture. These components will be explained further in the following section about the decompositions. One of the most important components shown in the diagram is the data storage component. Most of the components use this data storage component in one way or another. That is why the interface the data storage component provides, is a requirement for a lot of different components. This also shows in the diagram if one follows the lines going out from the IDatabase.

The Watchdog component that is present in the overall component diagram doesn't seem to interact with the rest of the components. The watchdog component however has interaction and communication functionality with every other component. All components for which failure needs to be detected, will send heartbeat updates to this central watchdog. We omitted drawing these interface dependencies because of the fact that the whole diagram would have become practically unreadable. Keep in mind though that this is not some ghost component with no interaction with any component.

It is also important to note that an event channel is used throughout the system. This event channel is used for the dispatching of different types of messages such as failure indications and component state updates. This component is actually not shown in the overall component diagram because it interacts with a lot of different components but on a lower level than the overall component diagram is meant to show. This event channel will be shown and explained in both the further decompositions in section 3.3 and the element catalog in section 5.1

3.2.1 Core functionality

The core functionality of the system can be described as the ability to handle measurement information utility networks in the form of trames sent by remote devices. Also the control of remote devices installed on valves, is an important core functionality. Also the management of and reporting of this data by and to the user, is an important aspect of the functionality this ReMeS system should offer.

The proposed architecture uses an incoming and an outgoing gateway to handle functionality of receiving and sending the data trames from and to the remote devices. The incoming gateway is able to route different types of trames to different schedulers. These schedulers decide when, where and how these trames should be processed. Eventually the information is extracted from the trames and stored in a data storage component.

The different schedulers and processors also have the ability to schedule the notification of users (under certain circumstances such as leaks or failures) through a user notification module. The next section will handle the different components in a little bit more detail.

After the processing and storage of an incoming trame it may be the case that control trames should be sent automatically to effect certain effect in the remote devices. The components responsible for the processing of the incoming trames have the option of scheduling these outgoing trames when necessary. These outgoing trames will be sent through the outgoing gateway.

The former describes the main data flow for trames throughout the system. Another important aspect is the user interaction. A user interaction component handles all user input and commands to the system. This component provides views of the ReMeS state based on the type of user interacting with the component and handles the interaction of this user with the system. When a user issues a command, it can interact with the data storage component, but also the effectuation of sending control trames will be handled by the system through a manner of dispatching the commands to the right component.

A last important aspect is the use of an invoice manager which is in charge for the generation of invoices based on triggers by either the UI (for an operator) or certain internal events.

3.3 Decomposition component diagrams

This section will handle the decompositions of the major components in the overall component diagram. However a lot of decompositions are illustrated, discussed and explained in the ADD sections. To avoid overly repeating ourselves, the modules that have not changed since the last ADD iteration, will only be referred to.

The incoming gateway is shown in figure 2. This is the gateway which receives the data trames from remote devices. This component also handles the acknowledgements of incoming trames and the callback for expected acknowledgement from sent out trames.

The scheduler for incoming measurement trames is shown in figure 3. This component handles the incoming measurement trames and schedules them for processing. This component also handles the checking for missing measurements in the trame handler. The buffer in this components handles the storage of measurement data in case of a central database failure. This buffer is made redundant by using multiple buffering instances separated on a network. The scheduler component can use different modes of scheduling based on the operation mode and workload that is being advertised on the event channel.

The data storage component is shown in figure 5. This component handles the central data storage and effective processing of measurements. The DB request handler handles the requests for access to the database and also incorporates a caching mechanism for availability and performance. This component internally uses different physical databases for different types and natures of data. Anomaly detection is also provided by this module. These physical database are internally shielded by a database access layer for object oriented interaction with the databases.

The watchdog component is shown in figure 6. This component provides the availability monitoring service for the system. This module uses the event chan-

nel to post messages about the uptime status and possible failures to all interested (or subscribed) parties. Each component that wants or needs its status monitored sends repeated measurements to the watchdog to notify this component of it's activity (in some circles described as petting the watchdog). A missing heartbeat will be flagged by the active loop and in that manner the failure of a component can be detected.

The user interaction component is shown in figure 15. The user interaction component is the component that provides user interaction for this system. It is realised by using a MVC (model-view-controller) structure. Each user category has its own type of view on the system. These views will interact with the controller layer to perform actions and get information from the core system. The application can request data by directly querying the database component or by monitoring the event channel of the system.

The outgoing gateway is the only component that has really changed since our last iteration of ADD. During the ADD process, the different manners of sending data was made abstraction of. Eventually, for the final architecture discussion, it was decided to elaborate on the different subcomponents that are used to send trames over different technology networks (such as sms, grps or ethernet). The final structure of the outgoing gateway is shown in the following figure.

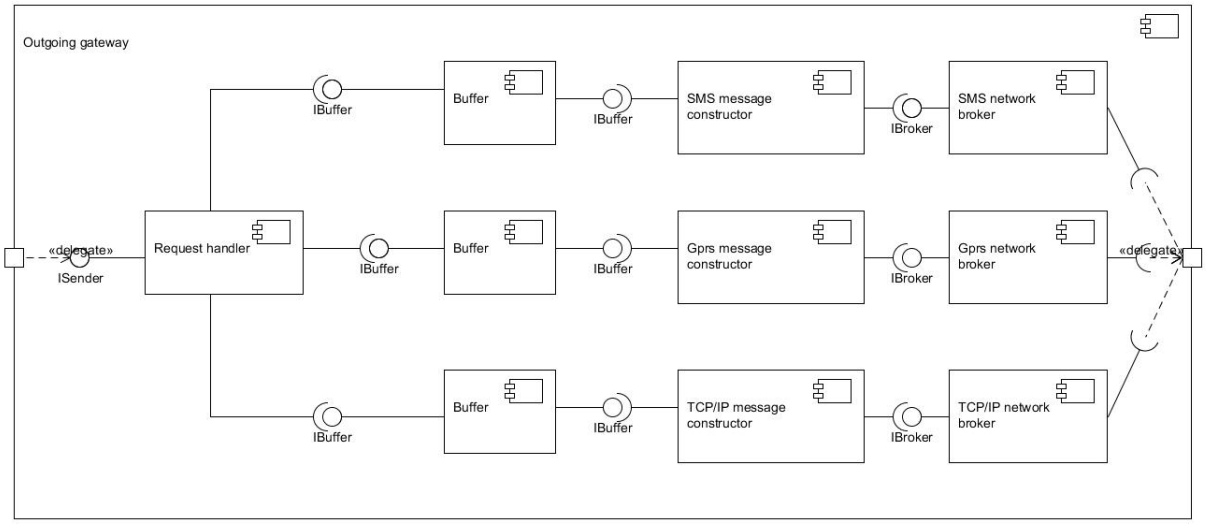


Figure 20: The component diagram of the final version of the outgoing gateway component.

The rest of the components are of a more trivial nature and the explanation and discussion can be found directly in the ADD section.

3.4 Deployment diagram

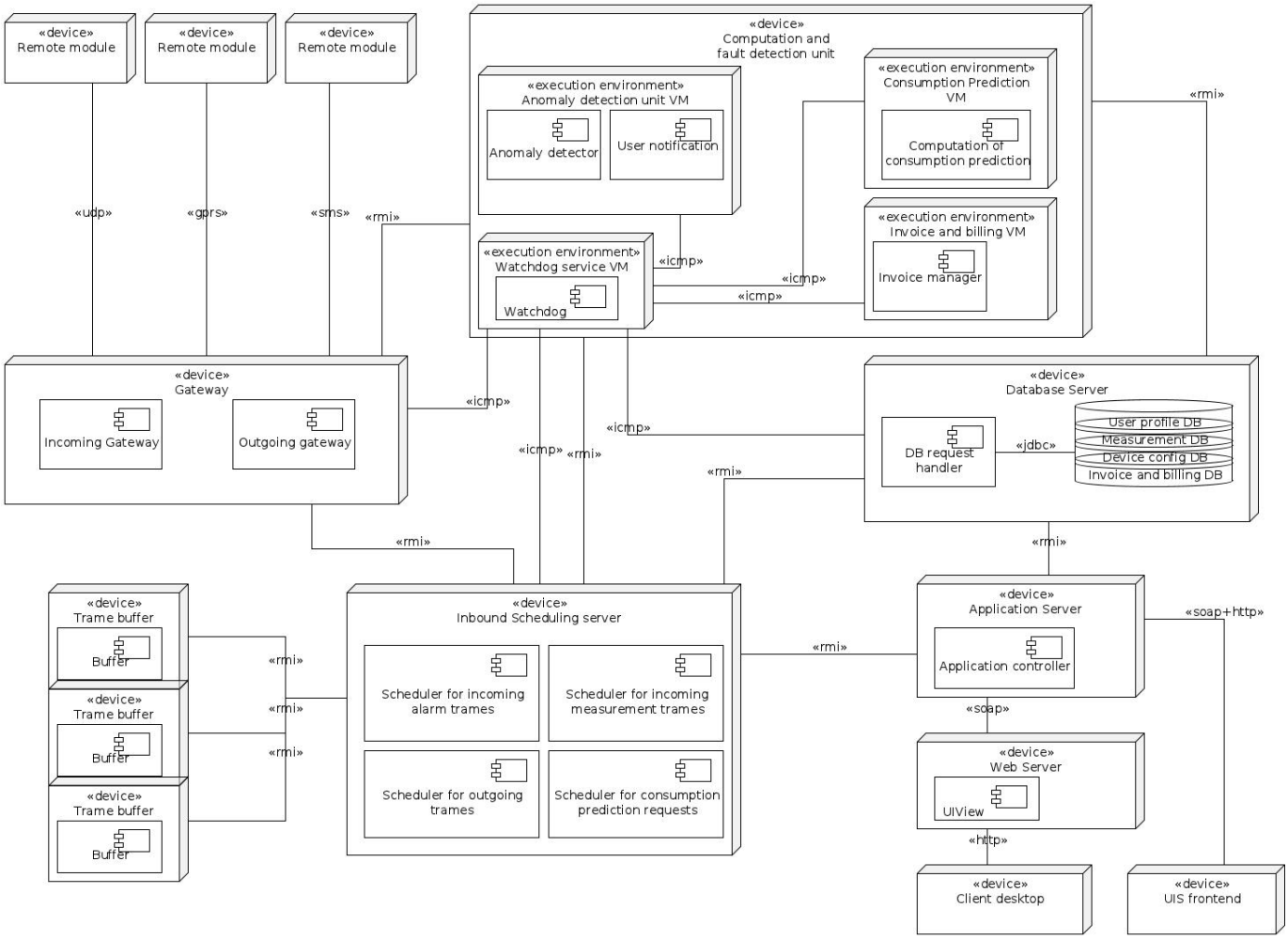


Figure 21: The final version of the ReMeS system deployment diagram.

The deployment diagram shows how different architectural components are to be deployed in a real world scenario. There are 4 major devices that make up the ReMeS system, not counting the application server and web server while two major communication protocols are used internally.

3.4.1 Device Nodes

These devices are the Gateway which forms the physical link to the outside world when speaking about trames. This device handles all communication to and from the remote modules, be it over an IP channel, a gprs channel or an sms channel.

The next component in line is the inbound scheduling server. This device handles the scheduling of trames coming in from the gateway or the trames pending to be sent out to remote devices. Also the scheduling for processing the measurement trames and the scheduling of consumption prediction requests gets handled by this device.

This device also has a connection to three standalone buffering devices. These buffers are very important for redundancy and availability purposes.

The next component is the consumption prediction server. This device handles the generation of consumption predictions based on the user history and measurement data.

The next component in line is the Database server. This server is an important device in the ReMeS system because this device will hold just about all the data albeit stored in different databases. Request to the database are handled through the request handler component which communicates with the database instances themselves using the jdbc protocol. This device is also the device with the most coupling to other components in the system. This is quite normal because the whole system is built for the core functionality of storing, transforming and using this data in one way or another.

The next component is the Computation and fault detection unit. This device is actually a device which hosts and supports multiple execution environments. In reality this could be realised by hosting multiple virtual machines layered on top of the host operation system running on this device. The first execution environment has as responsibility the detection of anomalies in the incoming user data and the notification of users when such anomalies are detected. The second execution environment provides the watchdog service that is being used to monitor the uptime of different components in the ReMeS system. Heartbeats are sent to the watchdog service by different components to inform it that they are still up and running. The heartbeats are sent using the icmp protocol. The third execution environment provides the computation of consumption predictions. The fact that consumption prediction component resides inside a virtual execution environment allows the replication of this component for load balancing purposes. If more consumption prediction modules are needed, it just suffices to replicate the virtual machine on different physical computation nodes that are capable of running virtual machines. This improves the scalability and load distribution aspects for the computation of consumption prediction. The fourth and final execution environment hosts the invoice manager. This component is capable of generating and dispatching invoices for customers and clients alike.

Beside these major devices, also an application server and a web server will be deployed in order to provide the human users and the User Information System with an easy to use interface to use the functionality provided by the ReMeS system. The web server will use the SOAP protocol to communicate with the application server and outside users can visit the web pages provided by this web server by using the http protocol.

3.4.2 Communication protocols

For the main method of communication between the different devices, the rmi protocol will be used. This form of communication will be made possible by connecting the different devices to a switched network based on the IP protocol. On top of this switched network, the rmi protocol will provide a means of communication for the internal components of the devices. Besides rmi, the icmp protocol will also be used by a lot of devices, to conform to the watchdog service specifications. The icmp protocol will be used to send heartbeats to the watchdog service for uptime monitoring purposes.

Furthermore two protocols for accessing the service from outside the ReMeS system will be used. The soap and the http protocol are both able to be transfered on the previously discussed switched network.

Finally, the remote devices have different options for communicating with the ReMeS system. They can use a simple udp, but also die gprs and sms protocol can be used to send and receive trames to and from the ReMeS system.

4 Scenarios

This section will handle the behavioral discussion for the ReMeS system. This section will illustrate, using a set of preset scenarios, how the system and its architecture support the fulfillment of these scenarios.

4.1 Notes

In a number of diagrams the auth() function call is used to authorise a user for an operation. However, in the diagrams the auth() function call is passed only an authentication token object. In reality this function call would need an authentication token and an object representing the operation that an actor wishes to execute in order to successfully authenticate that actor.

Another point of attention are the representation of the function calls made to the data storage instance. These calls are described and shown on a very high level of abstraction. Representing these interaction in a more detailed manner would very quickly lead to unnecessarily complex diagrams. Therefor this more simple approach is used. In reality also a query would be passed as an argument to execute operations on a database in stead of just the information that is needed for the operation (which is done in the sequence diagrams here).

4.2 User profile creation

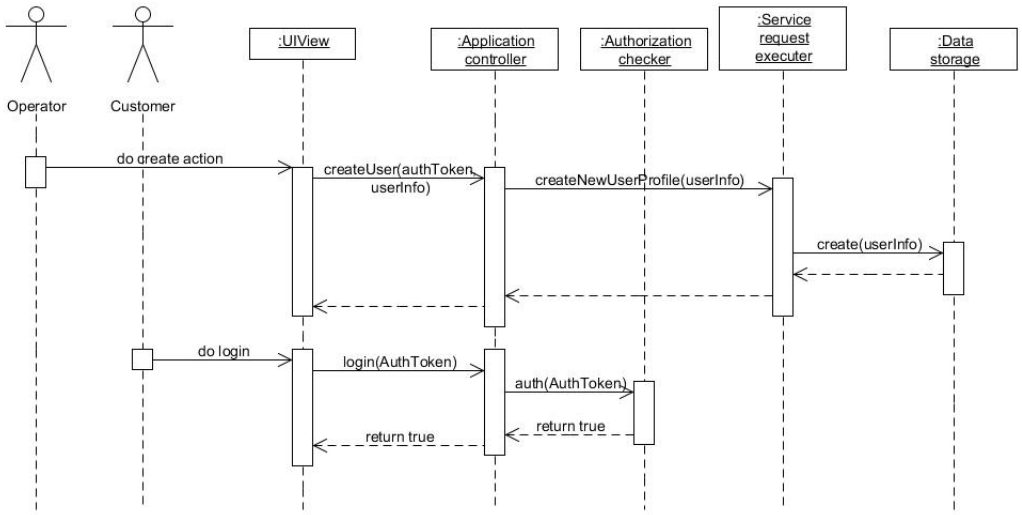


Figure 22: The sequence diagram for the user profile creation scenario.

The user profile creation scenario is mainly handled by the user interaction component. In the sequence diagram only the UIview component is shown. In

a real scenario, this component would be divided in separate views for normal users and operators. Further decomposition of this module was not warranted in the ADD phase. Even though it is not shown in the sequence diagram, the operator must first login into the application controller. It is also not shown that when the operator calls the createUser() method, the application controller will first check if the current user is authorized to execute this action.

4.3 User profile association with remote monitoring module

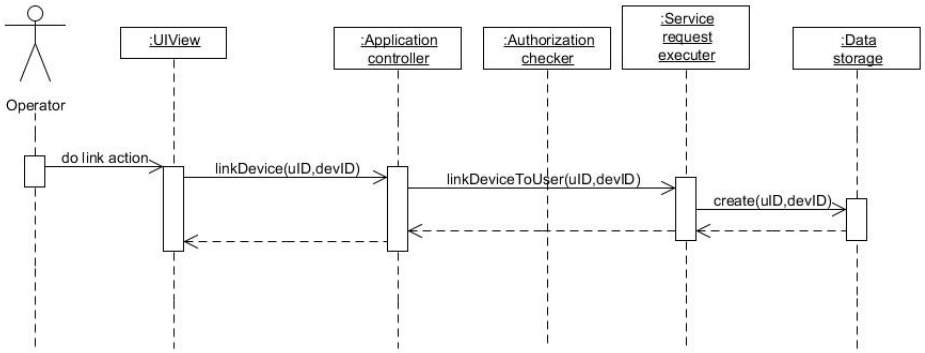


Figure 23: The sequence diagram for the association of the device to the profile.

This scenario illustrates that the functionality for association of a device with a user profile is provided. However the notification of the UIS is not foreseen. At least not in an elegant manner. Due to the late addition of the components in charge for communication with the UIS, this functional requirement was overseen. Again the UIView an actor can use will be tailored to the actual type of actor.

4.4 Installation and initialization

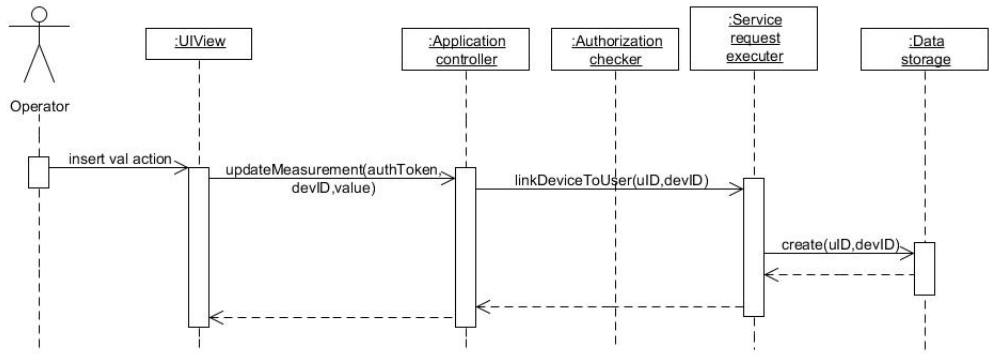


Figure 24: The sequence diagram for manually setting the meter value for a device.

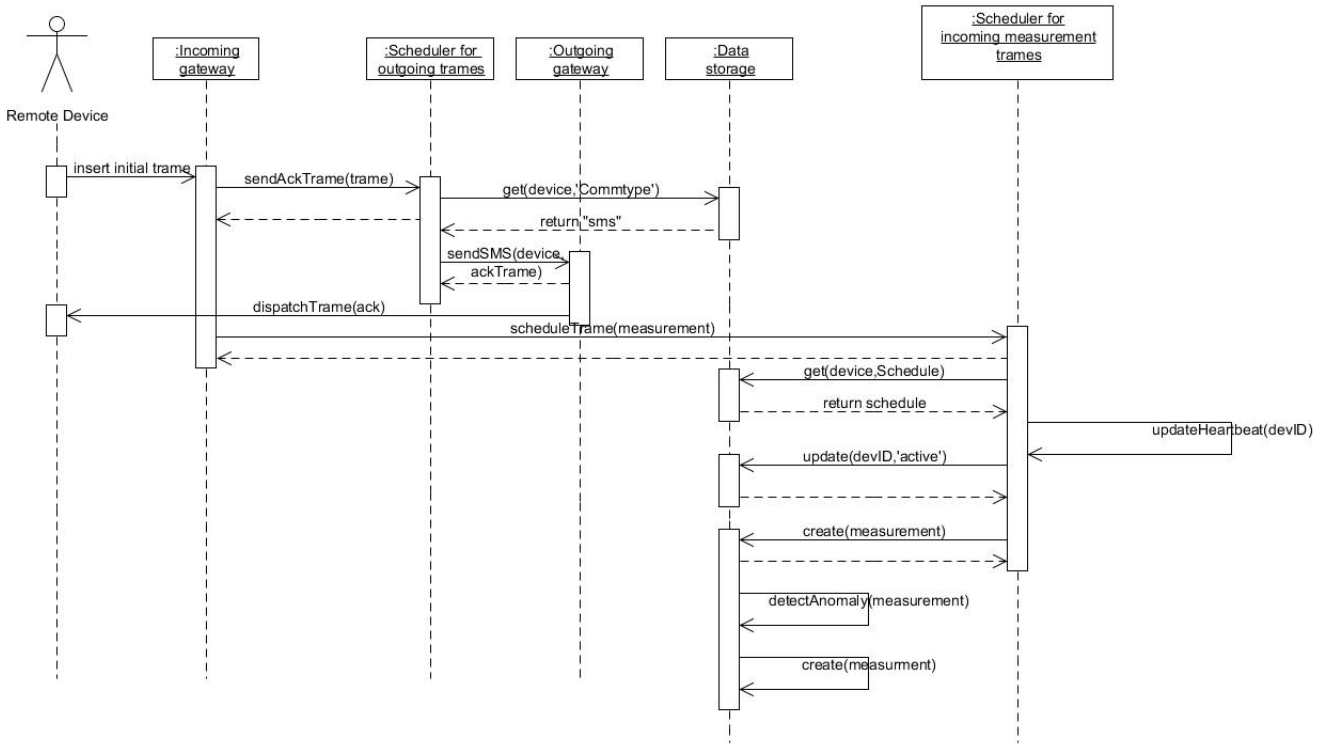


Figure 25: The sequence diagram for sending the initial measurement to ReMeS.

The scenario of installing a remote module at a remote location can be split up into two parts in order to discuss this behavior. The first functionality lies with the operator actually manually setting the initial meter setting through the user interface he presumably access through the web browser of his mobile unit. Again, in this sequence diagram it is not shown that the operator has to login into the application controller and that the checkRights() method is called by the Application controller.

The second behavior, which is described in the second sequence diagram, shows how the initial measurement is sent to the ReMeS system and how it is passed along the system until it is stored and until the device is marked as active. The sequence diagram also show the function calls to some components returning immediately. This is done on purpose to indicate that messages are being passed (through the use of a buffering mechanism) to other components in a way such that the sender doesn't have to wait for other operations to complete. Each component has it's own responsibilities and thus, only adhere to their own operations.

4.5 Transmission frequency reconfiguration

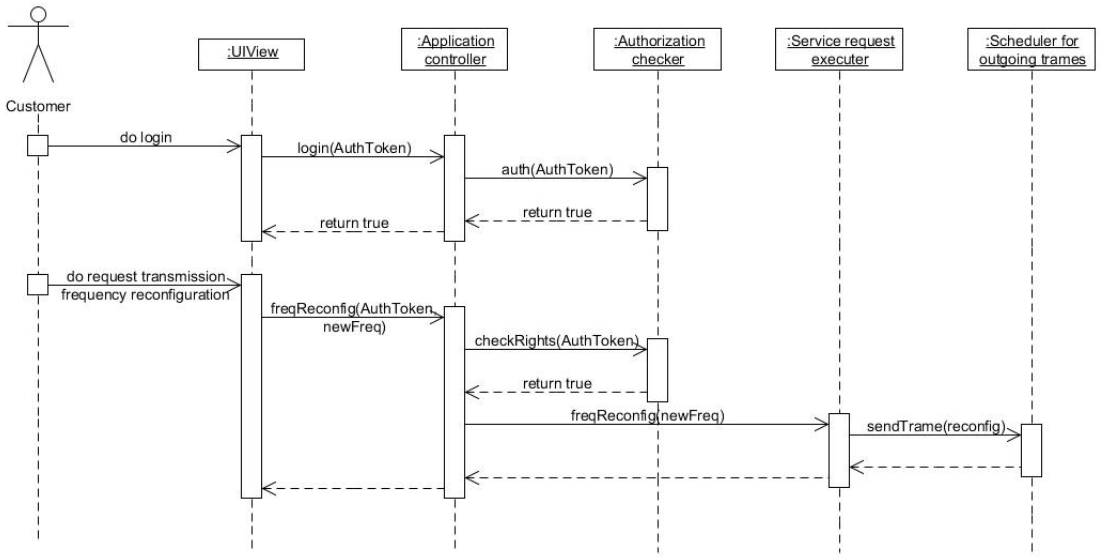


Figure 26: The sequence diagram for reconfiguring the transmission frequency.

The transmission frequency reconfiguration scenario is almost completely handled by the User interaction component of the system. This system diagram shows that the complete functionality of the scenario is provided by our system.

4.6 Troubleshooting

The scenario described in this section mainly describes interaction between actors in the problem domain. For the ReMeS system under development, this scenario does not bring much new functionality to describe. Therefor we didn't explicitly work out this example since the result would be too similar too other scenarios.

4.7 Alarm notification recipient configuration

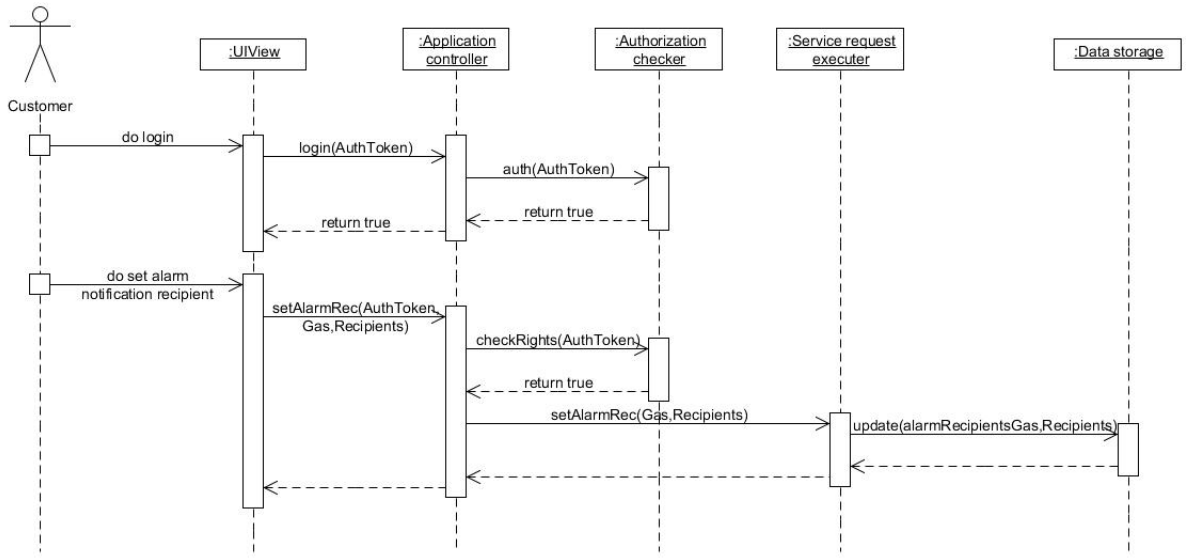


Figure 27: The sequence diagram for configuring the alarm notification recipient.

This scenario is very similar to the transmission frequency reconfiguration scenario. It is also almost completely handled by the User interaction component of the system. The only difference is that the service request executer will communicate with the Data storage component instead of the Scheduler for outgoing trames. The full functionality of the scenario is provided by the system.

4.8 Remote control

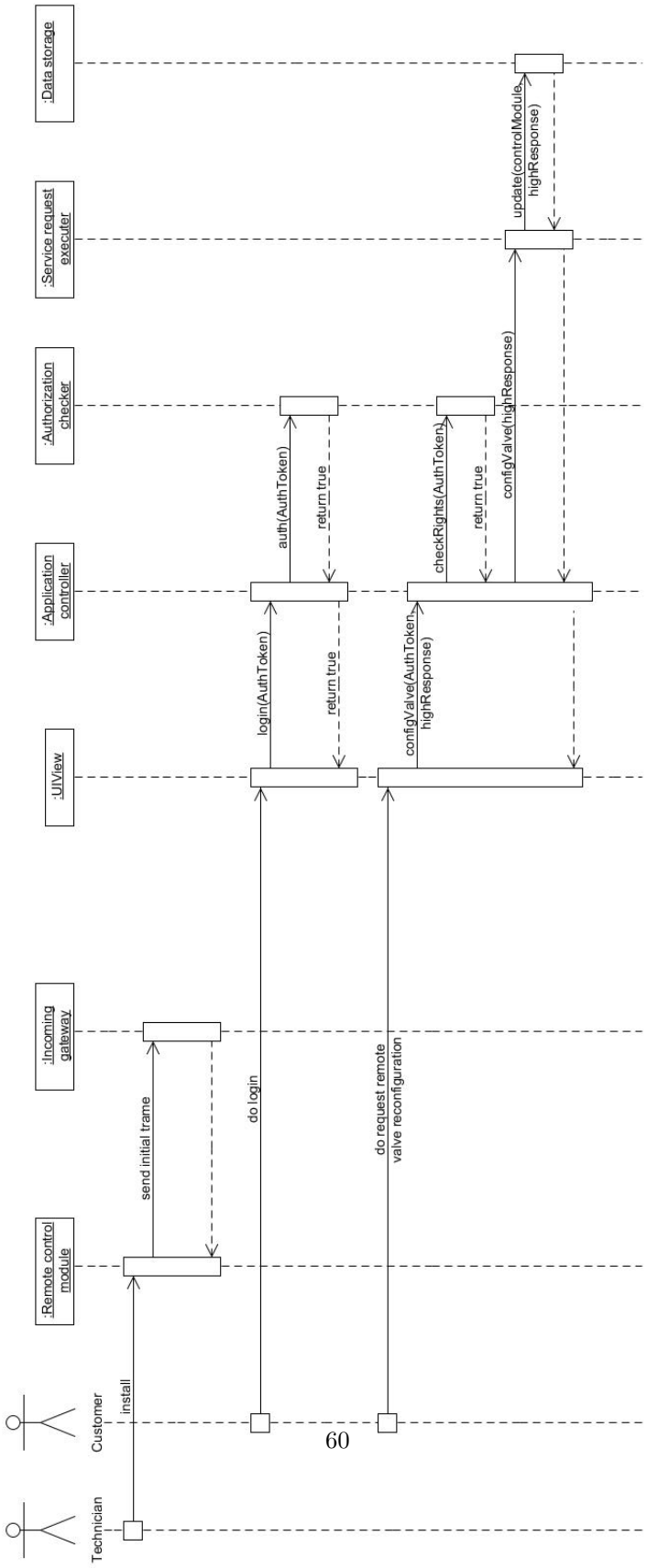


Figure 28: The sequence diagram for remotely configuring the remote device..

The first part of the scenario Remote control is the installation and activation of a remote control module. Unfortunately there is no functionality in the current ReMeS system to activate a remote control module. Only remote monitoring modules are activated automatically. It is possible to activate a remote control module through the User interaction component. That way, an operator can do this. The second part of the scenario is configuring the water control module. This part is very similar to the scenario Alarm notification recipient configuration. A customer must use the Application controller to select the desired configuration profile.

4.9 Normal measurement data transmission

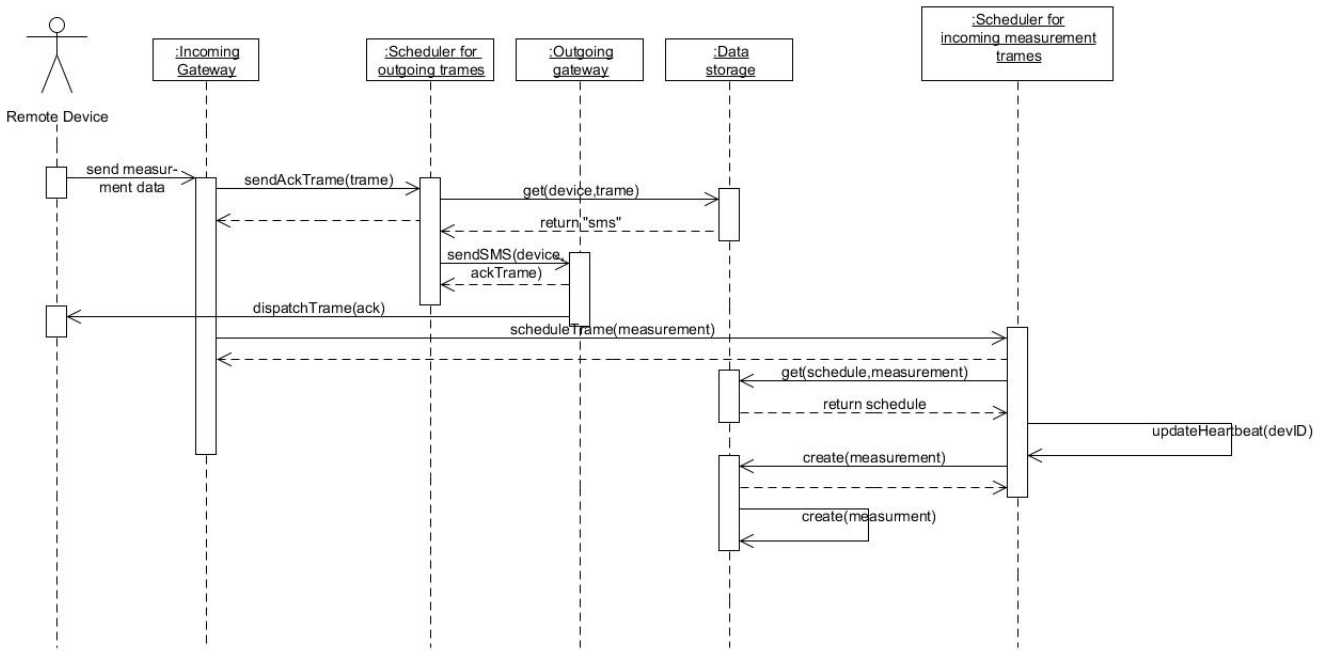


Figure 29: The sequence diagram for sending a measurement trame under normal circumstances.

This sequence diagram shows that the Incoming gateway will send an acknowledgement through the Scheduler for outgoing trames to the remote device. The Incoming gateway will forward the measurement trame to the Scheduler for incoming measurement trames. This component will do all required tasks to store the measurement in the Data storage. The Data storage component will also call the Anomaly detection component. This is not shown in the sequence diagram because it isn't in this scenario.

4.10 Individual data analysis

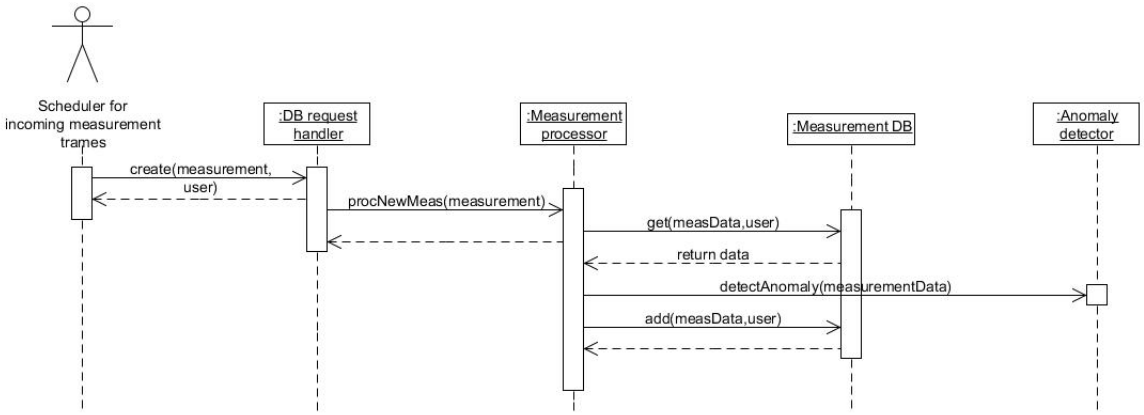


Figure 30: The sequence diagram for performing individual data analysis requests.

When the Data storage component receives a new measurement to store, it will call the anomaly detector to compute individual consumption profiles. These profiles can be used to determine potential leaks or anomalies. The complete scenario is covered by the design of ReMeS.

4.11 Utility production planning analysis

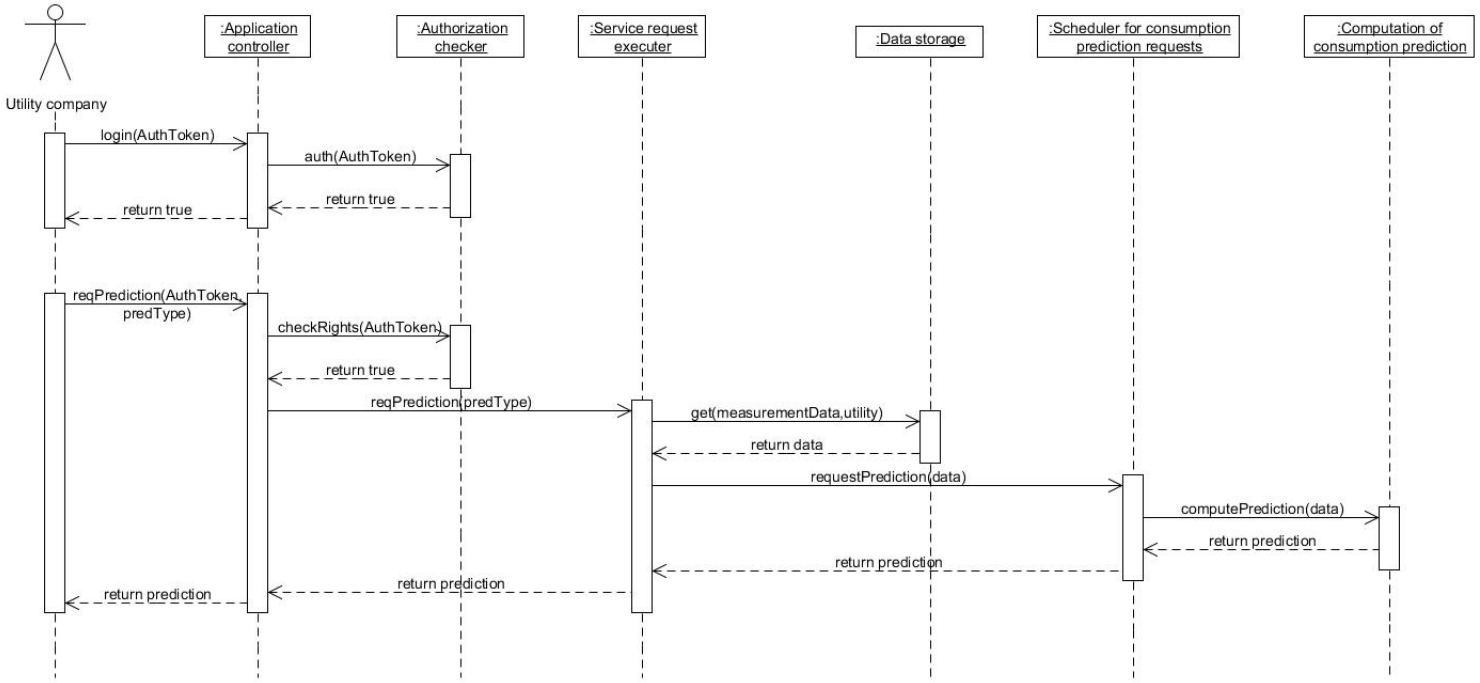


Figure 31: The sequence diagram for performing production planning analysis.

Utility production plannings can be requested through the Application control Component (or through the UIView component). Since utility companies will probably want to automate these requests, we provided an interface they can use without going through the UIView component. If a utility production planning is requested, this request will be added to the Scheduler for consumption prediction requests. When the computation of the prediction is finished, it will be returned to the caller. This scenario is also completely covered by our design.

4.12 Information exchange towards the UIS

This scenario indicates the regular information exchange between UIS and ReMeS. This functionality is available in the architectural design but only at such an high level so that creating sequence diagrams to illustrate this, would result in a fairly trivial diagram. These modules were not decomposed very deeply and that is why it is not possible at the moment to show a much more detailed view of this behavioral aspect of the design.

4.13 Alarm data transmission: remote monitoring module

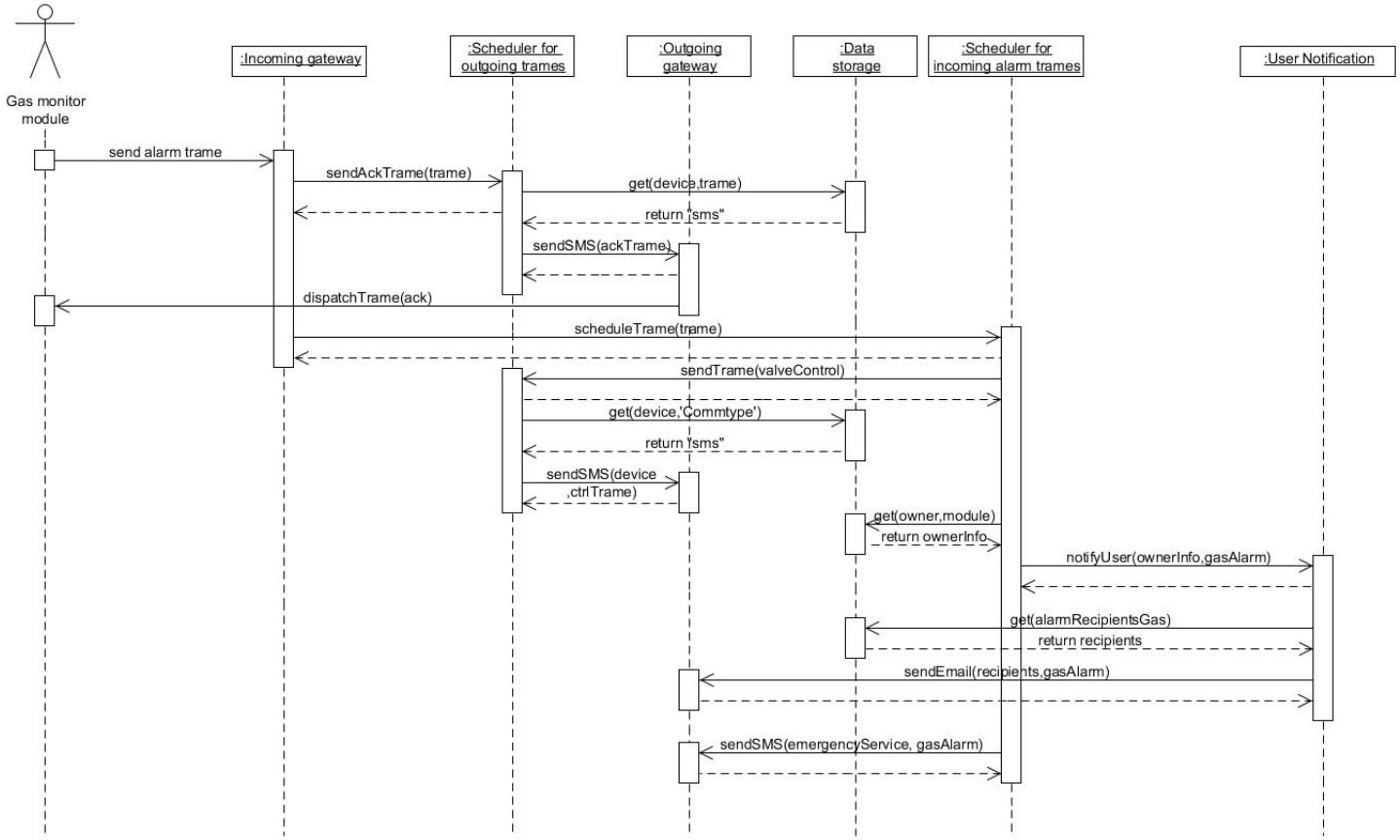


Figure 32: The sequence diagram for receiving an alarm trame.

When an alarm trame arrives at the Incoming gateway component. This trame will be delegated to the Scheduler for incoming alarm trames. This component will send the required notifications to the correct modules, people and emergency services. Since we didn't know how we should contact the emergency services through an automated method, we used the SMS service to notify the emergency services. Keep in mind that in the sequence diagram, all the delegations made by the Scheduler for incoming alarm trames are executed concurrently.

4.14 Alarm data transmission: ReMeS

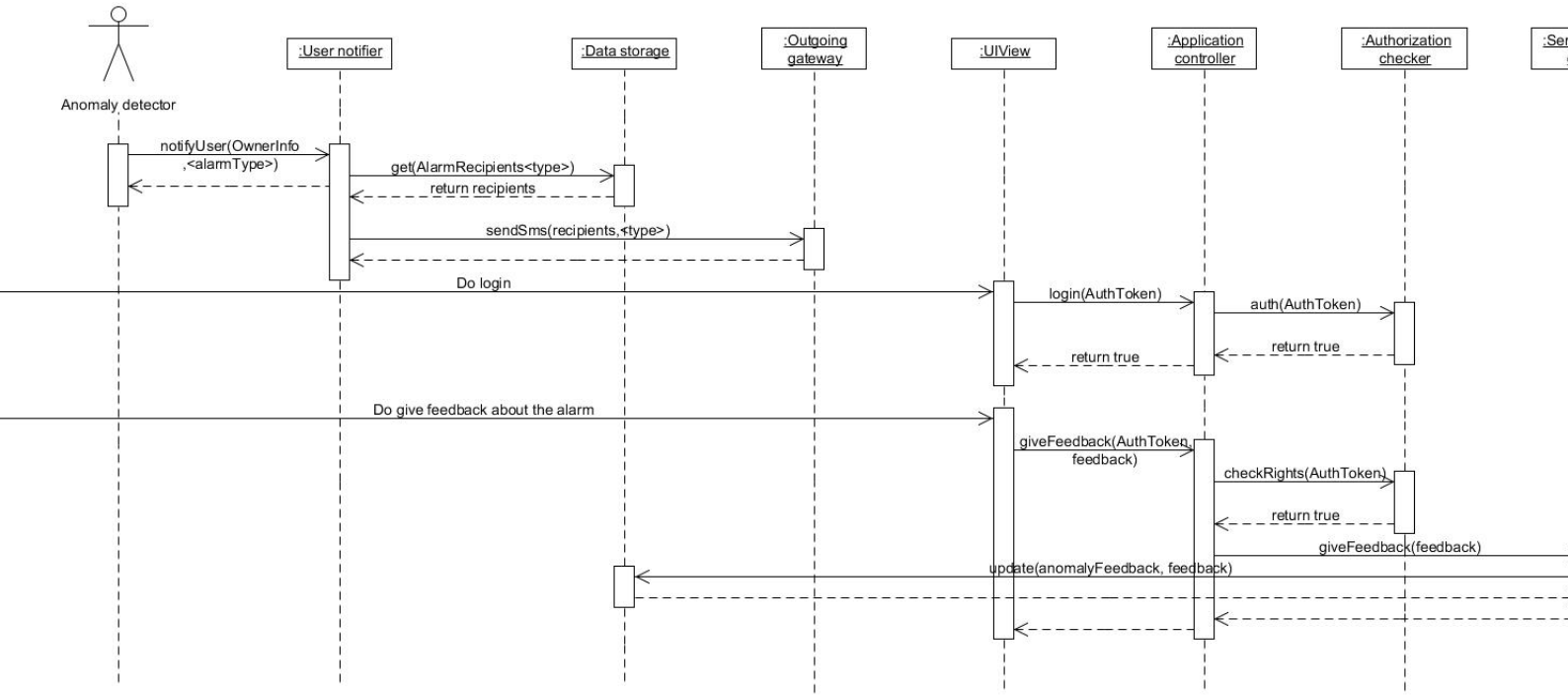


Figure 33: The sequence diagram for detecting an anomaly.

This scenario is also completely covered by the ReMeS system. When the anomaly detector detects an anomaly, it will call the User notifier component to send a message to the appropriate customer. The customer can also login into the UIView to give feedback about the detected anomaly. The customer can give positive feedback (the anomaly was indeed a leak) or negative feedback (the anomaly was a false positive).

4.15 Remote control module de-activation

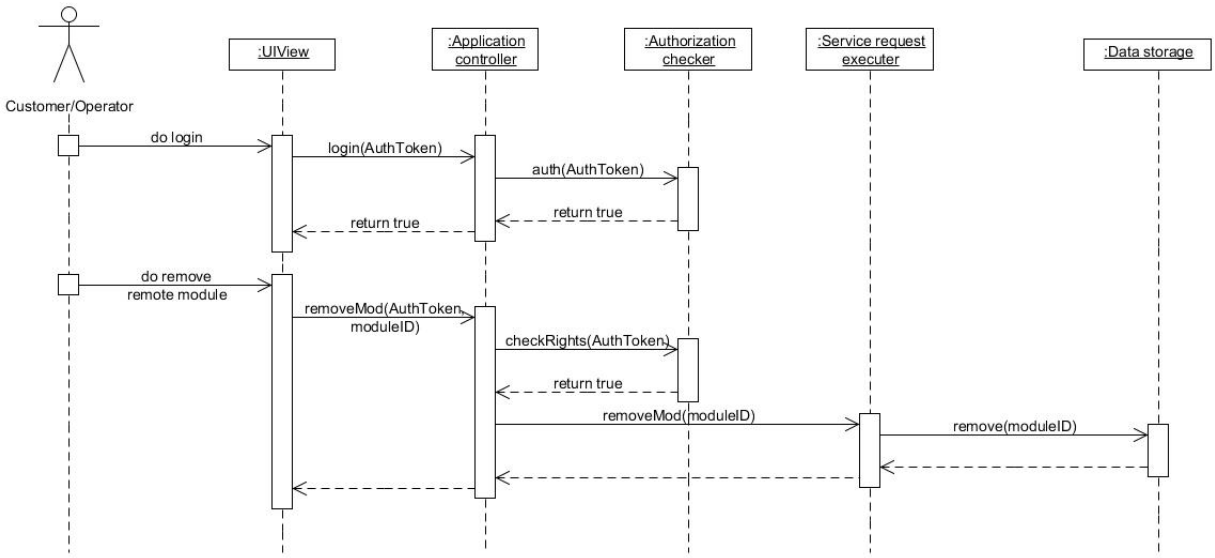


Figure 34: The sequence diagram for removing a remote module from a customer account.

This scenario is again very similar to the Alarm notification recipient configuration scenario. This action is available through the Application controller. One needs to login first before removing the remote module from the Data storage.

4.16 New bill creation

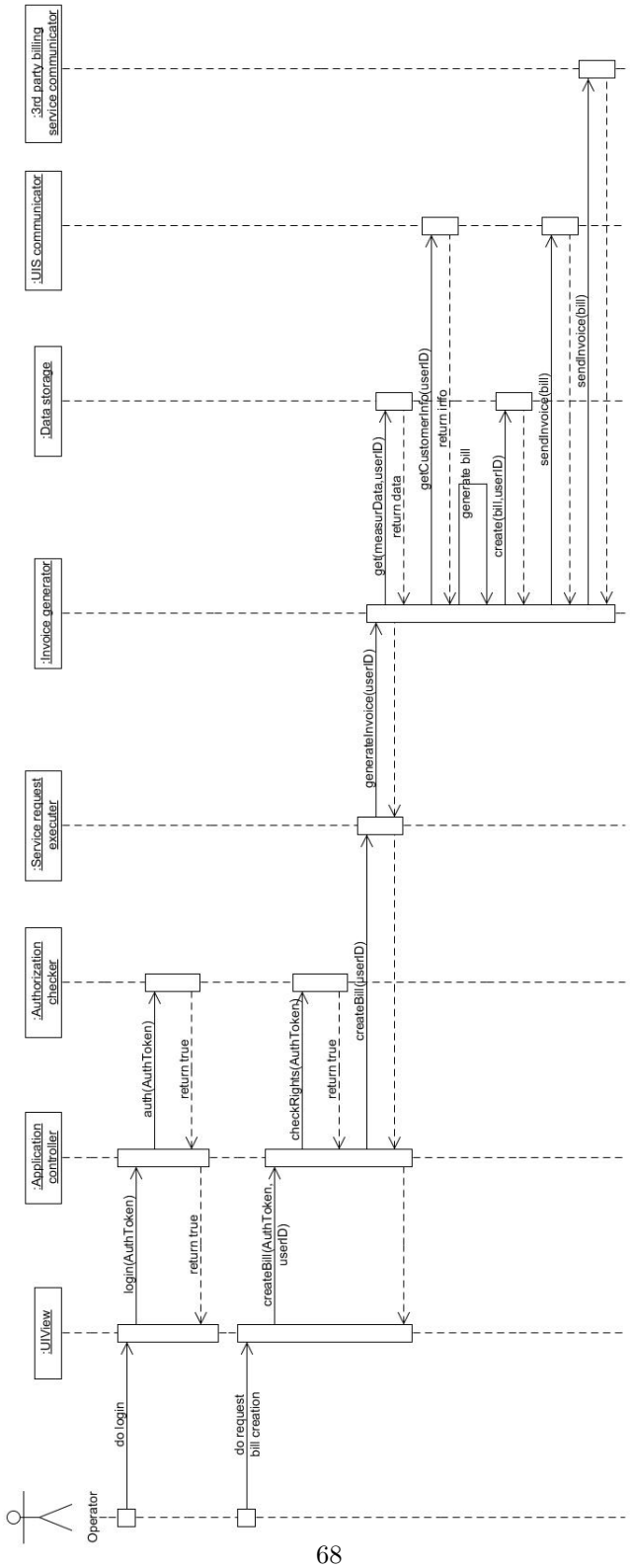


Figure 35: The sequence diagram for creating a new invoice.

Creating a bill is fully implemented in our ReMeS system. The only part that is different from the Bill Creation scenario is that ReMeS does not automatically creates a bill, but that an operator must initiate the operation. The created bill will be sent to a third party billing service and to the utility providing company. The created bill will also be stored in the Data storage component.

4.17 Bill payment is received

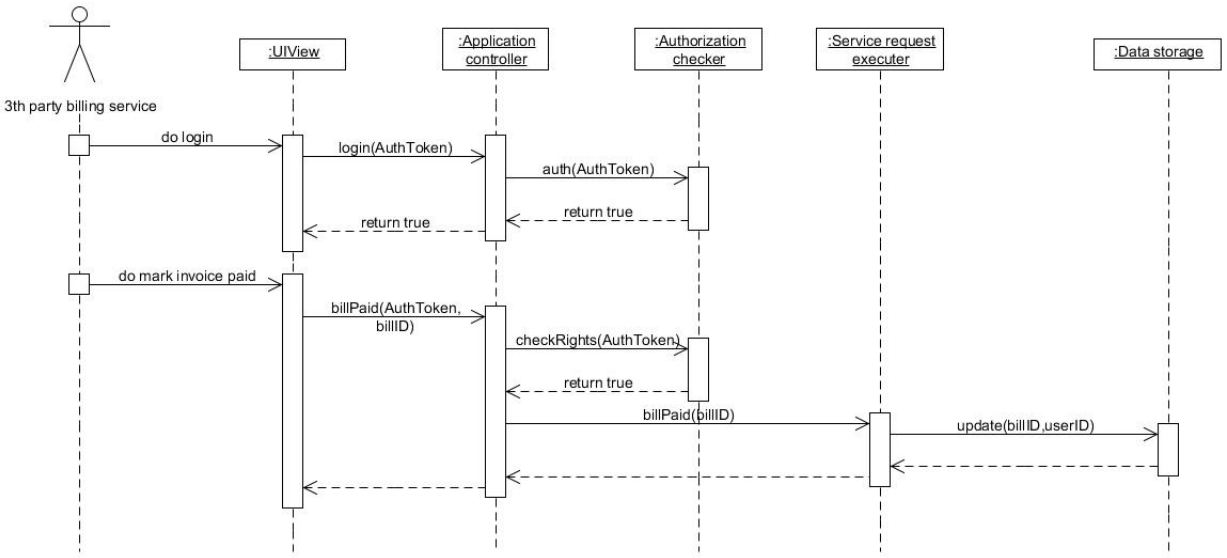


Figure 36: The sequence diagram for receiving a confirmation that a bill was paid.

Finally, this scenario is again very similar to the Alarm notification recipient configuration scenario. This action is available through the Application controller which provides an interface that the 3rd party billing service can use to notify ReMeS that a certain bill is paid.

5 Appendix

5.1 Element Catalog

5.1.1 Incoming gateway

The incoming gateway is the entry point for trames sent by a remote device, in the ReMeS system. This module dispatches the incoming trames to the components in the system that are fit to handle them. The incoming gateway also handles the incoming acknowledgement trames and provide a callback event for components waiting on such an acknowledgement.

Message router

The message router is the subcomponent that handles the dispatching of the trames to the modules that are capable of processing them.

Acknowledgement handler

If the type of trame is an acknowledgement trame, the acknowledgement handler will process this incoming acknowledgement. Also, other incoming trames are passed to the acknowledgement to make sure those incoming trames are acknowledged by ReMeS itself.

5.1.2 Scheduler for incoming alarm trames

The Scheduler for incoming alarm trames handles, as the name indicates, the incoming alarm trames. These trames are temporarily stored in a prioritized queue, which makes it easy to prioritize incoming trames. The scheduler component than empties the queue at as fast a pace he can manage and dispatches the commands to send an emergency control trame and/or send a notification to the alarm recipients.

Trame handler

A proxy instance which handles the external interface of this component and fills up the priority queue.

Input router

The input router component is a subcomponent visible in the decomposed view of the trame handler. This component handles the passing of incoming trame messages to the next supercomponent (in this case being the buffer). While passing the regular trame, it also stores part of the trame with the trame header (containing the time stamp, device id and such) in the latest heartbeat table.

Latest heartbeat table

The latest heartbeat table is a subcomponent of the trame handler component. This component stores incoming measurement trame headers containing time stamp information and device id's. This table is used to monitor the incoming measurements and possibly detect missing measurements.

Active loop

The active loop component is a subcomponent of the trame handler. This components gets its name from the fact that this component works in a continuous loop in it's own thread of control. This component iterates over the entries in the latest heartbeat table and monitors for missing measurements. This component also queries the data storage unit for the measurement schedules of the remote devices. These are needed to check for missing measurements.

Buffer

This is a buffer in which data is stored based on priority. The implementation of this element could be achieved by using a priority queue implementation.

Buffer frontend

The buffer frontend component is a subcomponent of the buffer component. The components in this decomposed view illustrate the redundancy measures that have been taken to effect the availability quality attribute scenarios. The Buffer frontend is an simple component offering an external interface and delegating the requests to add an element to each buffer instance. For retrieving data, a request is delegated to the buffer instances and the first response is used.

Buffer implementation

The buffer implementation component is a child component of the buffer component. This component represents the actual physical buffer. The instances manifesting this component will store and hold the actual data. These components will use the same interface as the previously discussed buffer frontend component.

Scheduler

This scheduler empties the queue and handles the incoming trames based on priority. This scheduler handles this responsibility with fairness, such that eventually all trames will be handled. This element will use a dynamic scheduling policy to avoid starvation.

5.1.3 Scheduler for outgoing trames

The scheduler for outgoing trames handle the requests for trames that need to be sent out from the ReMeS system. This component also uses a prioritized queue to sort and schedule those requests based on priority. This module has the capability of querying the central data storage unit for information on how a recipient needs to receive a trame or through which network technology.

Request handler

The request handler is a proxy for offering the external interface to this subcomponent. All received requests are stored into the prioritized queue.

Buffer

The prioritized queue is the same component as previously discussed in section 5.1.3.

Trame constructor

The trame constructor has the responsibility to empty the queue and process the incoming requests in such a manner that they are processed fairly, but based on their priorities. This component can also query the data storage unit for information on the recipient. This information will contain the technology that needs to be used to transfer the trame and the trame constructor will construct and prepare the outgoing trames in a manner compatible with that technology.

5.1.4 Outgoing gateway

The outgoing gateway component handles the physical dispatching of outgoing messages. These messages can be trames, but also email messages are possible. This gateway has the capability of sending those messages using a number of different technologies such as GPRS/3G, sms or just plain over tcp/ip.

Request handler

The request handler serves as a proxy for offering the external interface of this component. This proxy dispatches the outgoing messages based on the type of network technology that needs to be used.

Buffer

For each network technology a buffer exists which holds the yet to be sent messages.

Message constructor

For each network technology there exists a message constructor component which handles the construction of messages or packets for that specific network technology. It sort of wraps the data that has to be sent with the adequate packet data that the network can accept.

Network broker

For each network technology, there is a network broker component in place which handles the effective communication with the network. It is this component that will handle the final dispatching of messages over the network.

5.1.5 User notification

The user notification component handles the request for notifying users. This component also has the capability of polling the data storage unit for information about the user that needs to be notified.

Request handler

The request handler is a proxy for providing an external interface for the internal functionality.

Buffer

This buffer is a simple component for temporary storing the requests until they can be handled.

Notification constructor

The notification constructor handles the construction of the notification messages (email structure for example). These notification messages form the actual content that will be provided to the outgoing gateway.

5.1.6 Scheduler for incoming measurement trames

This component handles the scheduling of incoming measurement trames to the measurement processing service. When a measurement trame arrives, it will also be checked if the trame arrives in time or if a scheduled check in has been missed. This scheduling component also monitors the event channel for information on the load of other heavy duty components and considers this when scheduling new requests to the data storage unit (for example).

Trame handler

The trame handler functions as the entry point for this scheduler component and handles the dispatch to the buffer used for final scheduling. In this trame handler the regularity of the device, when it comes to sending measurement updates, is checked against the intended check in schedule. The moment a device sends a measurement trame, it is entered in the heartbeat table. The active loop request the device configuration from the data storage unit and uses that schedule to detect missing check ins.

Buffer

This buffer is a very important buffer in the system. In case of a database failure, this buffer will hold the measurement trames until the other components return to normal operation.

Event channel

The event channel in this component is the event channel used throughout the ReMeS system. This component is used to publish notifications to and to receive notifications from about the state of load on the system.

Scheduler

This scheduler component, schedules in a fair manner, the measurement trames to the data storage unit for processing. This component uses the event channel to monitor the operation mode of some heavy duty components.

5.1.7 Watchdog

The watchdog component signifies the capability of the system to monitor the operational state of different components. Different important components in the system can post heartbeat updates to this watchdog component. This component handles the monitoring and storage of these heartbeats.

Active loop

The active loops continuously checks the heartbeat table for new entries. If an anomaly is found, it posts a message in the event channel.

Event channel

This event channel component is the same component as the other event channels.

Heartbeat table

This heartbeat table registers and stores the heartbeats until the active loop can process data from them.

5.1.8 Invoice manager

The invoice manager component handles the creation and dispatching of invoices to the third party billing service. The creation of invoices can be triggered by synchronous and asynchronous calls to the invoice generation.

UIS communicator

The UIS communicator will handle the communication with the Utility Information System. This component will use the external interface provided by the Utility Information System. The details of how this communication happens, depend on the interface provided by the UIS and is outside the scope of this assignment.

3rd party billing service communicator

The 3rd party billing service communicator component handles the communication with the 3rd party billing service for sending them the generated invoices. The same limitations apply as in the previously discussed section.

Invoice generator

The invoice generator handles the responsibility of creating the invoices based on the stored consumption history and measurement data. The generation of these invoices is triggered either by direct method call or by a notification in the event channel.

Event channel

The event channel component has been amply discussed in other components throughout the system.

5.1.9 Computation of consumption prediction

The computation of consumption prediction component represents the internal component that will actually be computing the consumption predictions.

Consumption prediction algorithm

The consumption prediction algorithm component represents an algorithm to be used for performing consumption prediction. This component however is

not fully decomposed or designed to specification due to timing constraints. A strategy pattern could be used to organise this component further if the need arises.

5.1.10 Scheduler for consumption prediction requests

This scheduler component handles the scheduling of consumption prediction data. The structure of this component is fairly similar to previously discussed scheduling components. The elements responsible for initiating requests is primarily the user interaction component discussed later on.

Request handler

This component is a proxy which provides an external interface to the internal components. This component stores the requests for consumption prediction in a temporary buffer.

Buffer

This buffer stores the request before they can be scheduled and handled for processing.

Scheduler

The scheduler actually schedules and triggers the computation of consumption predictions for the ReMeS system.

5.1.11 User interaction

The user interaction component handles the interaction of different types of users, with the system. This module represents the user interface and the handling of user initiated requests for data processing or other stuff. This form of user interaction makes use of the MVC pattern as described in earlier sections.

Application controller

The application controller represents an application layer for handling the communication between the user interface controls and views and the core of the ReMeS system. External applications and user interfaces can make requests to this controller layer so they don't need to interact with the core itself.

UIView

The UIView component is an component representing the effective user interface that will be represented to clients of the ReMeS system. This user interface is related to the view component in an MVC architecture. It only directly

communicates with the controller layer which is, in this case, represented by the Application controller.

Service request executer

The Service request executor component delegates service requests to components in the core of the system.

Authorization checker

The Authorization checker is a module that handles the authorisation of users which are using the application controller or the user interface directly. This component has the responsibility of providing authentication services when needed.

Event channel

This event channel has been discussed a few times already. One extra function for usage in this components can be found in the fact that some operator user interfaces use this event channel to monitor the uptime states of different components. If a component of some sort fails and this failure is detected, this event will be communicated to an operator interface through this event channel.

5.1.12 Data storage

The data storage component represents the central data repository that will be used by a lot of components in the ReMeS System.

DB request handler

The DB request handler is a proxy for the database functionality representing a unified external interface to the database functionality. This handler component internally uses a caching module, which improves performance and availability within given boundaries. This component is also capable of delegating queries to the database instances that are capable of handling them.

Request handler

The request handler component is a child component of the DB request handler component. This component handles the same functionality of the DB request handler, without the functionality added by decomposing said element any further. This request handler offers the external interface to the outside components to shield the implementation details from the rest of the system.

Buffer

This buffer component is a subcomponent of the DB request handler component. This component is nothing more than a simple buffer instance for storing requests before they can be handled.

Request cache

The Request cache component is a subcomponent of the DB request handler. This component handles the requests to the database and temporarily stores the results to offer a caching service.

When a request is pulled from the buffer, this cache component will first check if a result for this query already resides in the internal cache. If this is the case, and the result is fresh enough, the cache component will return this cached result without querying the database for the information. If an answer for the query is not present in the internal cache, or the version of the result is too old, the request will be passed along to the next component in line which will query the actual database for the results.

Parallel process scheduler

The parallel process scheduler is a subcomponent of the DB request handler component. This component schedules the incoming requests that couldn't be handled by the caching instance for actual processing. This component is capable of scheduling and dispatching multiple queries at once (hence the component name).

Measurement processor

The measurement processor accepts request from the request handler for incoming measurement requests and processes those measurements by extracting data, querying the anomaly detection unit on this data, and inserting the measurements at the correct location in the measurement database.

Anomaly detector

The anomaly detector actually runs various anomaly detection algorithms on the measurement data in order to try and detect anomalies in the consumption behavior of a certain user, based on reading of a certain device.

Event channel

The event channel used here has been amply discussed in previous sections.

User profile DB

This is the physical database component storing the actual user profiles.

Measurement DB

This is the physical database component storing the actual measurements

Remote device configuration DB

This is the physical database component storing the actual device configurations.

Invoice and billing DB

This is the physical database component storing the actual invoice and billing data.

5.2 Interface descriptions

5.2.1 IAcknowledgementHandler

Interface Identity

The only component that provides this interface is the Acknowledgement handler.

Resources Provided

- void acknowledge(Trame incomingTrame)
  This method should be called when for a certain incoming trame, an acknowledgement should be returned to the sender. This method will be used to send acknowledgements of measurement and alarm trames.
- void handleAck()
  This method should be called when an acknowledgement is received. The Acknowledgement handler will store this acknowledgement in the Data storage.

Data Type Definitions

- Trame
  The incoming trame that should be acknowledged.

Exception Definitions

5.2.2 IAnomalyDetector

Interface Identity

This interface is provided by the Anomaly Detector in the Data storage component.

Resources Provided

- void detectAnomaly(MeasurementData data)
  This method will invoke the anomaly detector to detect anomalies in the given data.

Data Type Definitions

- MeasurementData
  A set of measurement data needed to detect an anomaly.

Exception Definitions

5.2.3 IAuthenticator

Interface Identity

This interface is provided by the Authorization checker component.

Resources Provided

- boolean auth(AuthToken authToken)
  This method is used when a user wants to login into the Application controller. The method returns true if the authToken is valid.
- boolean checkRights(AuthToken authToken, Service service)
  This method is used by the Application controller to check wether or not a user is authorized to request a certain service. This method will return true if the user is authorized.

Data Type Definitions

- AuthToken
  The specific piece of information that can authenticate a user.
- Service
  A data type that represents a certain service of the User interaction component.

Exception Definitions

5.2.4 IBillSender

Interface Identity

This interface is provided by all components that can send an invoice to an external source. Currently these are the UIS communicator and the 3rd party billing service communicator components.

Resources Provided

- void sendInvoice(Invoice invoice)
  This method will send an invoice to the utility providing company.

Data Type Definitions

- Invoice
  A data type containing the full information of an invoice.

Exception Definitions

- CommunicationException
  This exception can be thrown if the component fails to establish a connection with the external source.

5.2.5 IBroker

Interface Identity

The IBroker interface is provided by all network brokers.

Resources Provided

- void send(Packet packet)
  Method to send a packet over a network (SMS, gprs, TCP/IP).

Data Type Definitions

- Packet
  A packet that can be sent over a network.

Exception Definitions

5.2.6 IBuffer

Interface Identity

This interface is used by all buffers in our system. This interface uses a generic type T.

Resources Provided

- void addToBuffer(T data)
  This method will add data to the buffer.
- T getNext()
  This method will get the next data from the buffer and return it.
- T getFromBuffer(Integer index)
  This method will get data from the buffer at a given position.

Data Type Definitions

- T
  This is a generic type.

Exception Definitions

- IndexOutOfBoundsException
  This exception can be thrown by the getFromBuffer(Integer index) method if there is no data at the given index.

5.2.7 ICompute

Interface Identity

This interface is provided by components who can compute a consumption prediction. At the moment the only component with this functionality is the Computation of consumption prediction component.

Resources Provided

- ConsumptionPrediction computePrediction(MeasurementData data)
  This method will request a consumption prediction with the given data. The method returns the prediction.

Data Type Definitions

- MeasurementData
  A set of measurement data needed to compute a consumption prediction.
- ConsumptionPrediction
  A computed consumption prediction.

Exception Definitions

5.2.8 IDataBase

Interface Identity

This interface is provided by the Data storage component. Internally, each separate database instance in the Data storage component also provides this interface.

Resources provided

- void create(CreateQuery q)
  This is one of the four basic database access methods provided. This method signifies the creation of a record based on a given query q.
- void update(UpdateQuery q)
  This is one of the four basic database access methods provided. This method signifies updating a record based on a given query q.
- DataTransferObject get(GetQuery q)
  This is one of the four basic database access methods provided. This method signifies retrieving a record based on a given query q. The information contained in this record will be returned as a DataTransferObject.
- void remove(RemoveQuery q)
  This is one of the four basic database access methods provided. This method signifies removing a record based on a given query q.

Data Type Definitions

- CreateQuery
  This data type represents a database query of the creation category. This CreateQuery type is a subtype of the general Query data type.
- UpdateQuery
  This data type represents a database query of the update category. This UpdateQuery type is a subtype of the general Query data type.
- GetQuery
  This data type represents a database query of the retrieval category. This GetQuery type is a subtype of the general Query data type.

- RemoveQuery
  This data type represents a database query of the remove category. This RemoveQuery type is a subtype of the general Query data type.
- DataTransferObject
  A data object that contains the information of a record in the Data storage.

Exception Definitions

- QueryFormatException
  This exception can be thrown when an argument representing a query to be executed, does not follow the expected formatting rules.
- DatabaseUnavailableException
  This exception can be thrown when the implementing component is unresponsive for any reason.

5.2.9 IExecuter

Interface Identity

This interface is provided by the Service request executer component. The methods of this interface are very similar to those of the IUserInteraction interface. All the methods are the same, except that they don't have an AuthToken. Every method of IUserInteraction is also provided by this interface (except for the login method). For further information we refer to section 5.2.22.

5.2.10 IHeartBeat

Interface Identity

This interface is provided by all heartbeat tables in the system.

Resources Provided

- updateHeartbeat(ID id)
  This method will update the latest heartbeat received of the given id to the current system time.

Data Type Definitions

- ID
  An identification id.

Exception Definitions

5.2.11 IHeartBeatTable

Interface Identity

This interface is provided by all heartbeat tables in the system.

Resources Provided

- Iterator iterate()
  A method to get the iterator of a heartbeat table. This iterator can be used to iterate over all elements in the heartbeat table and check if there are any that are overdue.

Data Type Definitions

- Iterator
  An iterator consisting of all elements found in a heartbeat table.

Exception Definitions

5.2.12 IInvoiceGenerator

Interface Identity

This interface is provided by components that can generate an invoice. Currently this is only the Invoice manager component.

Resources Provided

- Invoice generateInvoice(UserInfo userInfo)
  This method will generate an invoice for the user specified by the userInfo argument.

Data Type Definitions

- Invoice
  A data type containing the full information of an invoice.
- UserInfo
  Data type that contains information of a user.

Exception Definitions

- IncompleteDataException
  This exception can be thrown by generateInvoice(UserInfo userInfo). It

is thrown when the UserInfo object does not contain sufficient data to correctly identify the customer.

5.2.13 IMeasProc

Interface Identity

This interface is provided by the Measurement processor.

Resources Provided

- void procNewMeas(Measurement meas)
  This method will feed new measurement data to the Measurement processor.

Data Type Definitions

- Measurement
  Data type containing measurement data.

Exception Definitions

5.2.14 IPredictionScheduler

Interface Identity

This interface is provided by the Scheduler for consumption prediction requests. This interface is different from IScheduler because it requires more information.

Resources Provided

- ConsumptionPrediction requestPrediction(PredictionType type)
  This method will schedule a consumption prediction. When the requested prediction is computed, this method will also return the prediction.

Data Type Definitions

- ConsumptionPrediction
  A computed consumption prediction.
- PredictionType
  The type of the requested consumption prediction. This structure contains the utility and the time frame.

Exception Definitions

5.2.15 IPublishable

Interface Identity

This interface is provided by every component that needs to receive messages from the event channel. These components are scattered throughout the complete system. This because every component that needs to receive data from the event channel needs such an interface.

Resources provided

- void publishMessage(EventMessage message)
  This method is used to receive data from the event channel.

Data Type Definitions

- EventMessage
  The message received from the event channel.

Exception Definitions

5.2.16 IReqCache

Interface Identity

Resources Provided

- void updateCache(DataTransferObject data)
  Method to store data in the cache.

Data Type Definitions

- DataTransferObject
  A data object that contains the information of a record in the Data storage.

Exception Definitions

5.2.17 IScheduler

Interface Identity

This interface is provided by the Scheduler for incoming alarm trames and the Scheduler for incoming measurement trames.

Resources provided

- void scheduleTrame(Trame incomingtrame)
  This method should be called to add a new trame to the scheduler.

Data Type Definitions

- Trame
  The trame that needs to be added to the scheduler for processing.

Exception Definitions

5.2.18 ISender

Interface Identity

This interface is provided by the Outgoing gateway. At the moment it is the only component providing this interface.

Resources Provided

- void sendSms(UserInfo userInfo, MessageContent content)
  This method will send a message using SMS with the given arguments.
- void sendGprs(UserInfo userInfo, MessageContent content)
  This method will send a message using gprs with the given arguments.
- void sendTcpIp(UserInfo userInfo, MessageContent content)
  This method will send a message using TCP/IP with the given arguments.

Data Type Definitions

- UserInfo
  Data type that contains information of a user.
- MessageContent
  The content of the message.

Exception Definitions

- IncompleteDataException This exception can be thrown by all three methods of the ISender. It is thrown when the UserInfo object does not contain sufficient data to send the message.

5.2.19 ISubscribable

Interface Identity

This interface is provided by the Event channel component. These components are scattered throughout the whole system. This is because several components need this channel to receive updates about the status of the system.

Resources provided

- void subscribeToTopic(EventDescription topic)
  This method can be used to subscribe to certain events that are sent through the event channel.
- void sendEvent(EventMessage event)
  This method allows components to send events through the event channel.

Data Type Definitions

- EventDescription
  The specific type of a message.
- EventMessage
  The message sent through the event channel.

Exception Definitions

5.2.20 ITrameSender

Interface Identity

This interface is provided by the Scheduler for outgoing trames.

Resources provided

- void sendTrame(TrameContent content)
  This method is used to add content to the queue that needs to be sent as a trame.

Data Type Definitions

- TrameContent
  The content that the trame must contain.

Exception Definitions

5.2.21 IUIS

Interface Identity

This interface is provided by the component that is responsible for the connection with the UIS.

Resources Provided

- UserInfo getCustomerInfo(UserInfo userInfo)
  This method will request all info that the UIS has of a customer. The method will pass a UserInfo object (that not necessarily needs to contain all information of a customer, only a predefined set of info which the UIS needs to identify the correct customer) and return a UserInfo object (which can contain more or different information of a customer).

Data Type Definitions

- UserInfo
  Data type that contains information of a user.

Exception Definitions

- IncompleteDataException
  This exception can be thrown by getCustomerInfo(UserInfo userInfo). It is thrown when the UserInfo object does not contain sufficient data to correctly identify the customer.
- CommunicationException
  This exception can be thrown if the component fails to establish a connection with the external source.

5.2.22 IUserInteraction

Interface Identity

This interface is provided by the Application controller component. The main use of this interface is interaction with users outside the ReMeS system.

Resources Provided

This interface has a wide variety of methods. Some of them are listed below.

- boolean login(AuthToken authToken)
  A method to login into the Application controller.

- void createBill(AuthToken authToken, UserInfo userInfo)
  A method to request a new bill creation.
- void giveFeedback(AuthToken authToken, Feedback feedback)
  A method the customer can use to give feedback to a detected anomy by ReMeS.
- void setAlarmRec(AuthToken authToken, UtilityType type, Recipient[] recipients)
  A method the customer can use to set the recipients for certain alarms.
- ConsumptionPrediction reqPrediction(AuthToken authToken, PredictionType type)
  A method to request a new consumption prediction.
- void createUser(AuthToken authToken, UserInfo userInfo)
  A method to create a new user in the system.
- void freqReconfig(AuthToken authToken, FreqProfile newFreq)
  A method to reconfigure the frequency that a remote measurement module sends trames with.

This list of methods is incomplete. The IUserInteraction interface contains all kinds of methods for a user to interact with the system.

Data Type Definitions

- AuthToken
  The specific piece of information that can authenticate a user.
- UserInfo
  Data type that contains information of a user.
- Feedback
  Data type that contains the feedback of a customer. This feedback can be positive or negative.
- UtilityType
  The type of utility (gas, water, power).
- Recipient
  Data type that contains information on who to contact and how to contact that person.
- ConsumptionPrediction
  A computed consumption prediction.
- PredictionType
  The type of the requested consumption prediction. This structure contains the utility and the timeframe.
- FreqProfile
  Data type that contains information on the frequency that a remote measurement module sends trames with.

Exception Definitions

- UnauthorizedException
  This exception can be thrown by all methods of this interface (except for the login method). This exception is thrown when the requested method cannot be executed by the user.

5.2.23 IUserNotifier

Interface Identity

This interface is only provided by the User notification component.

Resources provided

- void notifyUser(UserInfo userInfo, MessageContent notificiation)
  Use this method to add content to the queue that needs to be sent to a certain user.

Data Type Definitions

- UserInfo
  Data type that contains the information of a user.
- MessageContent
  The content of the message.

Exception Definitions