Remote Measurement, Monitoring and Control System

ADD application

Software Architecture (H09B5a) - Part 2a

Toon Nolten (r0258654)
Nele Rober (r0262954)

Contents

1 Introduction

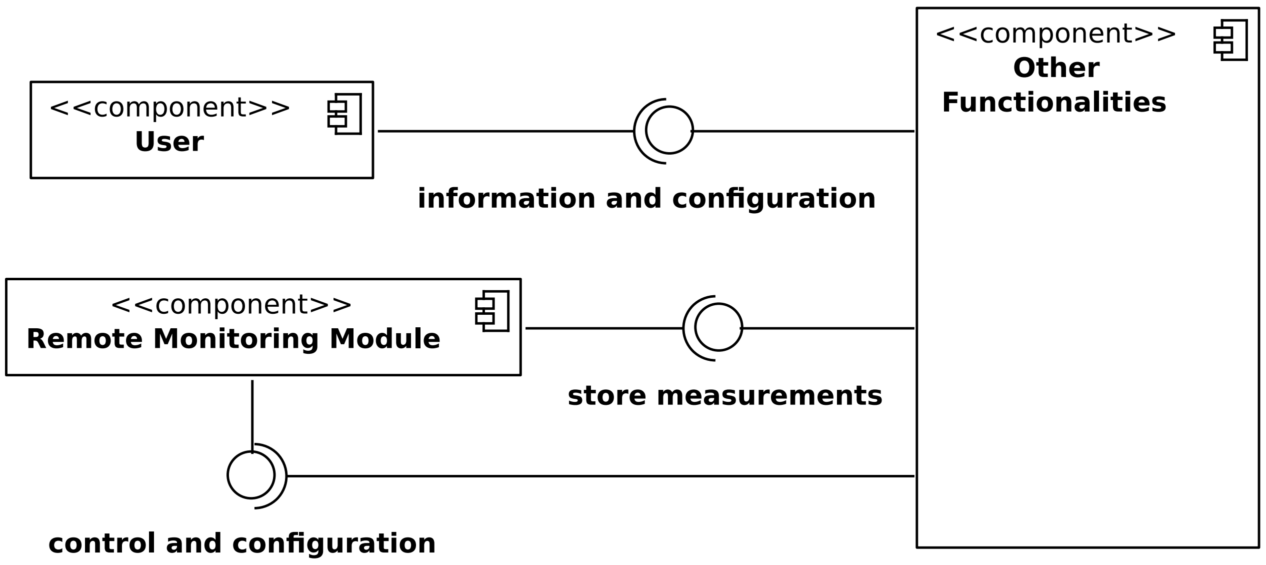

Figure 1: Component-and-connector diagram of *ReMeS*.

2 Attribute-driven design documentation

2.1 Iteration 1: *ReMeS* (Av1, P3, UC8)

2.1.1 Module to decompose

In this iteration we decompose *ReMeS*.

2.1.2 Selected architectural drivers

The non-functional drivers for this iteration are:

- *Av1*: Measurement database failure
- *P3*: Requests to the measurement database

The related functional drivers are:

- *UC8*: Send measurement

Rationale The core business of *ReMeS* is to collect measurements. Availability of the database is essential to support this. Related to this are the requests to the database and the sending of measurements.

2.1.3 Architectural design

To make the data as available as possible, it is important to be able to replace the database as soon as possible (Recovery Preperation). Replication is a general solution to this problem. We chose Passive Replication where the Replica Manager monitors and updates the Primary Database and the Primary Database synchronises with the Secondary Databases.

To resolve a failure in the database system, it is necessary to detect it as soon as possible. As the database is such a critical component of the system, heartbeats are an appropriate solution for detection. The Replica Manager sends a heartbeat to the Buffer every four seconds. Normally, the Buffer remembers all measurements for more than five seconds. However, when no heartbeat is received, up to three hours of measurements are stored and the Buffer notifies the operators within a minute. When the hearbeat returns, the Buffer sends the temporarily stored measuements to the Replica Manager via the Request Scheduler.

Requests to the database are normally processed in FIFO order. Only in overloaded modus arbitration is in order. The Request Scheduler should then adapt the schedule to minimise the violation of the SLAs, taking the priorities of the requests into account.

Alternatives considered

With *Active Replication* the Replica Manager updates all databases. This decreases the downtime considerably when one of the Databases fails or crashes. However, the availability requirement is not that strict and Passive Replication suffices.

In stead of keeping replicas, a *spare* database can be kept. However, it takes a lot of time to initialise the spare database. Moreover, data may be lost as the spare is updated infrequently.

An alternative to heartbeats lies in the use of *pings/echoes*. However, because the database is crucial to the system, pings would need to be sent frequently. It is more efficient to use heartbeats instead.

Finally, scheduling could be avoided by using *more powerful hardware*. This is, however, very costly because the capacity must be fitted to maximal throughput which rarely occurs.

2.1.4 Instantiation and allocation of functionality

Decomposition Main aspects of the resulting decomposition.

Primary Database The Primary Database stores the incoming measurements and keeps the other databases up to date. The Primary Database also handles requests from the system.

Secondary Databases The Secondary Databases are kept up to date by the Primary Database. The Secondary Databases do not handle requests as they may not be up to date. When the Primary Databse goes down, one of the Secondary databases is promoted to Primary.

Replica Manager The Replica Manager controls the Measurement Databases. When an incoming measurement arrives, the Replica Manager updates the Primary Database. When the Primary Database fails, the Replica Manager skips heartbeats until he has promoted one of the Secondary Databasas to the Primary Database. This makes the Buffer set the degraded mode. The Buffer will make sure that the missed measurements are resent after the heartbeats return. There should be enough databases to ensure the required $99,9\%$ availability. The problem of availability now shifts to the Replica Manager. This is solved by multicasting the measurements to both the Request Scheduler and the Buffer.

Measurement Multicast Both the Request Scheduler and the Buffer need to record all incoming measurements. The Measurement Multicast makes sure that both of them receive the measurements.

Buffer During replacement of the Replica Manager or during the switching of roles between the Databases, the incoming measurements should be stored in a temporary place, the Buffer. The Buffer receives a heartbeat from the Replica Manager. When the heartbeat does not arrive, the Buffer sets the Request Scheduler to degraded modus. When the heartbeat returns, the buffer will send all recorded measurements to the Replica Manager via the request Scheduler. Afterwards, degraded modus is disabled.

Request Scheduler Can be run in three modes: normal mode, degraded mode and overloaded mode. In normal mode, incoming measurements and requests are scheduled in FIFO order. In degraded mode, request are cached and not sent. Only the incoming measurements are passed to the Replica Manager. In overloaded mode, the requests and measurements are scheduled based on the SLA's of the customers. The degraded modus is set/unset by the Buffer. The overloaded modus is activated by the Request Scheduler when too many requests are pending. The Request Scheduler may keep his own cache of results.

Customer Database The Customer Database holds all information regarding the curstomers like the profiles, configurations and SLA's. The Request Scheduler uses the Customer Database to request SLA's in overloaded modus.

Notification System The Notification System notifies the operators.

Other functionality This component provides all other functionalities of *ReMeS*, e.g. the leak detection and invoicing.

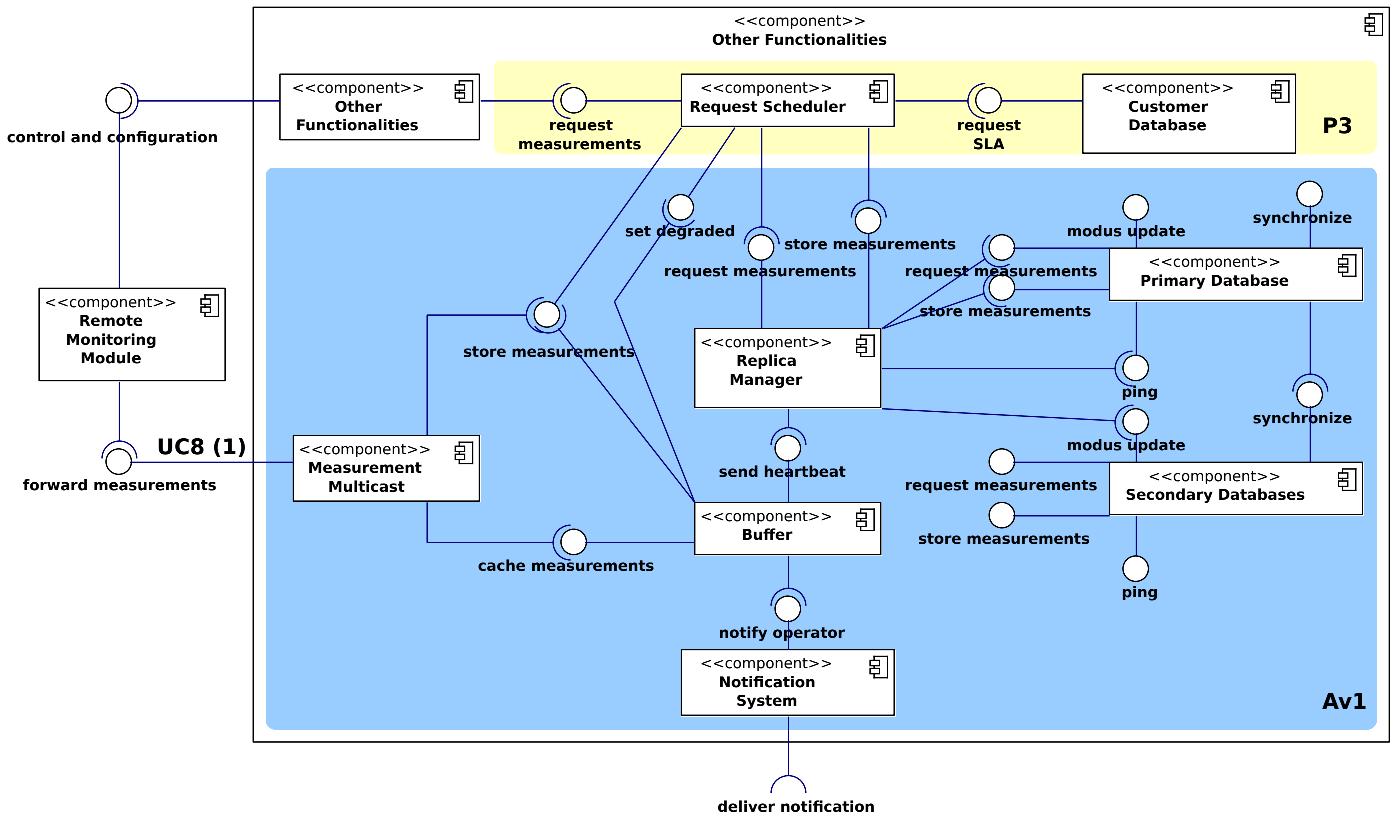

Figure 2: Component-and-connector diagram of iteration 1.

Behaviour

Deployment

2.1.5 Interfaces for child modules

Primary Database and Secondary Databases

- Store measurements
  - `void storeMeasurement(Measurement measurement)`
    - Effect: The given measurement is stored in the database.
    - Exceptions: None
- Request measurements
  - `Measurement requestMeasurement(Query query)` throws InvalidQueryException
    - Effect: The result of the query is returned.
    - Exceptions:
      - InvalidQueryException: thrown when the given query is invalid.
- Synchronize
  - `void synchronize(Database primary)`
    - Effect: The database starts synchronizing to the given primary.
    - Exceptions: None
- Ping
- Modus update

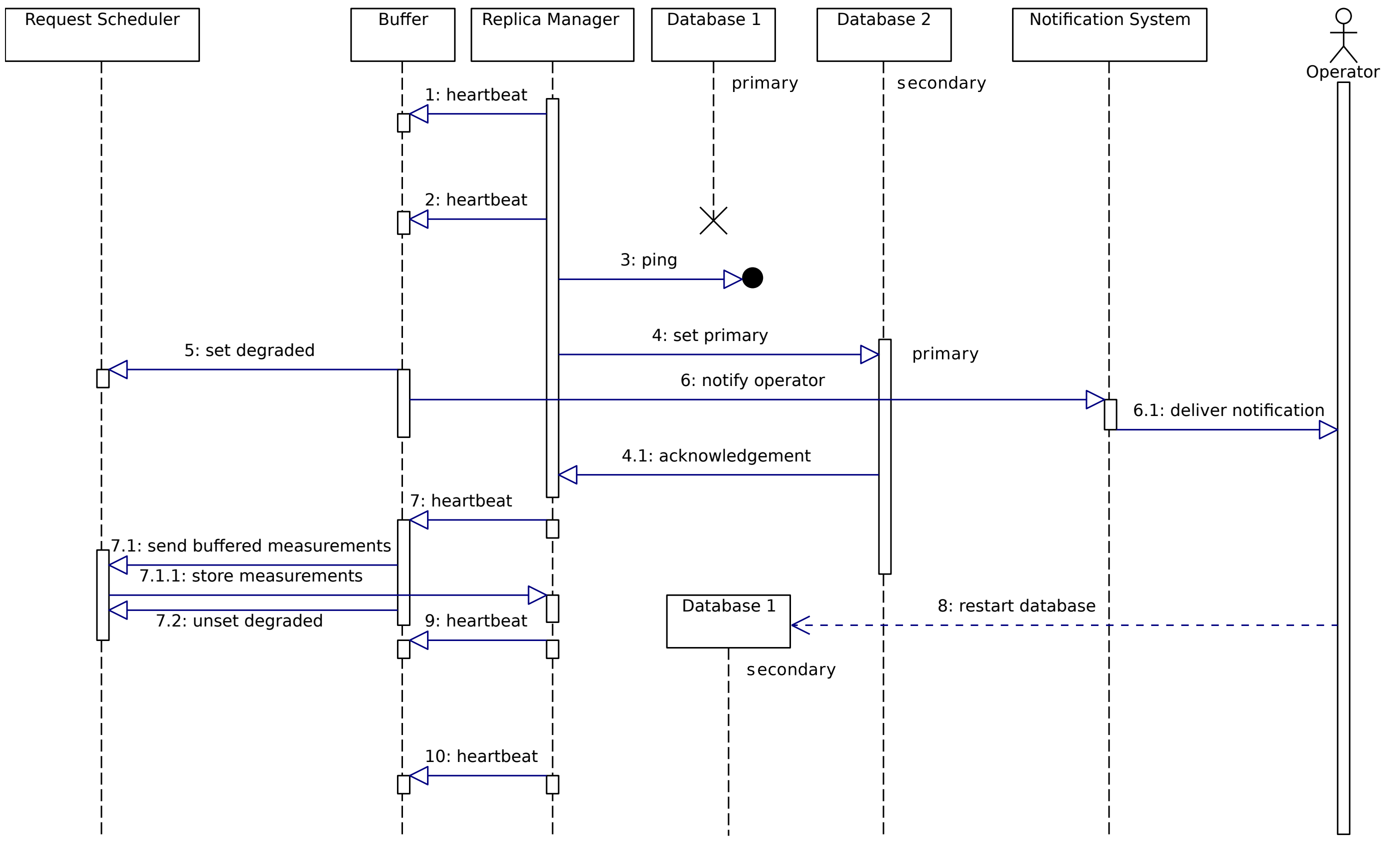

Figure 3: Sequence diagram illustrating the failure of the Measurement Database.

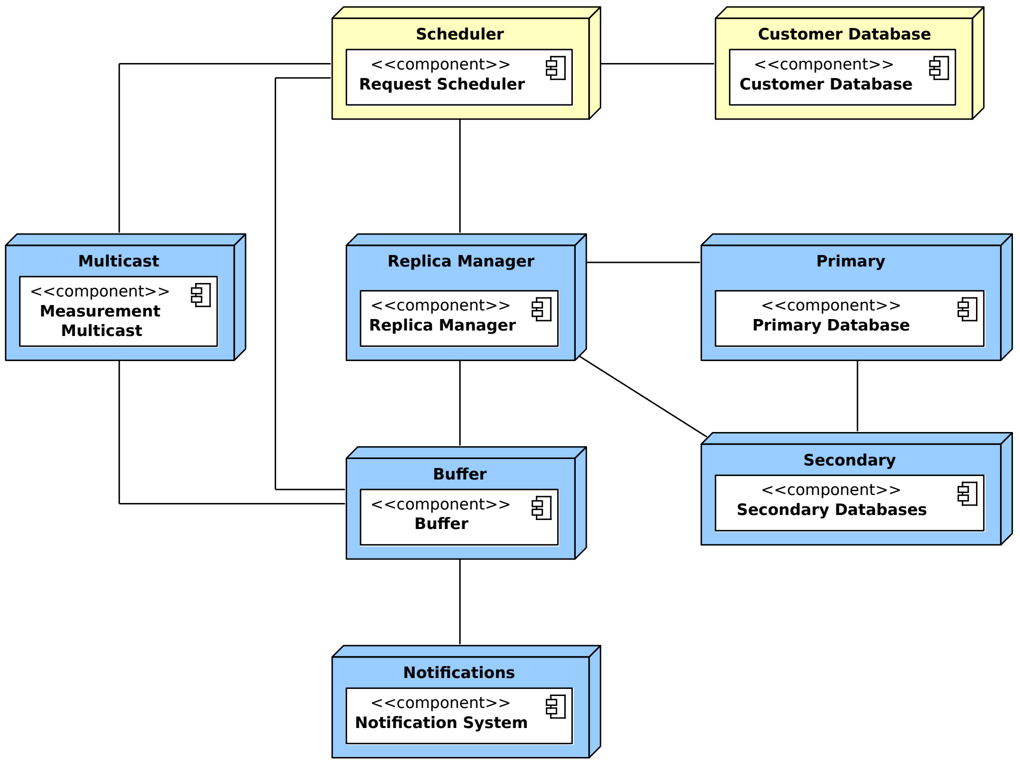

Figure 4: Deployment diagram of iteration 1.

  - `void setPrimary(Databases secondaries)`
    - Effect: The database is set to primary and will update all given secondaries from now on.
    - Exceptions: None
  - `Date lastUpdate()`
    - Effect: The database returns the date of the last stored measurement.
    - Exceptions: None

Replica Manager

- Store measurements
  - `void storeMeasurement(Measurement measurement)`
    - Effect: The given measurement is stored in the Primary Database.
    - Exceptions: None
- Request measurements
  - `Measurement requestMeasurement(Query query` throws InvalidQueryException
    - Effect: The result of the query is returned.
    - Exceptions:
      - InvalidQueryException: thrown when the given query is invalid.

Measurement Multicast

- Forward measurements
  - `void forwardMeasurement(Measurement measurement)`
    - Effect: The given measurement is passed to the Request Scheduler and the Buffer.
    - Exceptions: None

Buffer

- Send heartbeat
  - `void heartbeat()`
    - Effect: The buffer knows that the Replica Manager is still alive.
    - Exceptions: None
- Cache measurement
  - `void cacheMeasurement(Measurement measurement)`
    - Effect: The given measurement is temporarily stored.
    - Exceptions: None

Request Scheduler

- Set degraded
  - `void setDegraded()`
    - Effect: The Request Scheduler goes into degraded mode.
    - Exceptions: None
  - `void endDegraded()`
    - Effect: The Request Scheduler ends degraded mode.
    - Exceptions: None
- Request measurements

    - `Measurement requestMeasurement(Query query)` throws InvalidQueryException, DataUnavailableException
        - Effect: The request is scheduled and handled. The result of the query is returned.
        - Exceptions:
            - InvalidQueryException: thrown when the given query is invalid.
            - DataUnavailableException: thrown when the data storage system is currently unavailable (degraded mode).

Customer Database

- Request SLA
    - `SLA requestSLA(CustomerID customer)` throws NoSuchUserException
        - Effect: The SLA of the given customer is returned.
        - Exceptions:
            - NoSuchUserException: thrown when the given userId does not map to a registered user.

Notification System

- Notify operator
    - `void notifyOperator(Operator databaseAdmin)`
        - Effect: The databaseAdmin is notified.
        - Exceptions: None

2.1.6 Data type definitions

A short description of the occurring datatypes.

Measurement This data element represents a measurement.

Query This data element represents a query to the database.

Database This data element represents a database.

Databases This data element represents a collection of databases.

SLA This data element represents an SLA.

Operator This data element represents an operator.

2.1.7 Verify and refine

This section describes per component which (parts of) the remaining requirements it is responsible for.

Primary Database and Secondary Database

- *Av1b*: The database should have a guaranteed minimal up-time.
- *UC8.2b*: Store the measurement.

Replica Manager

- *Av1b*: The database should have a guaranteed minimal up-time.
- *UC8.2b*: Store the measurement.

Measurement Multicast

- *Av1e*: Incoming measurements are temporarily stored elsewhere - no new measurements are lost.

Buffer

- *Av1d*: Problems with the database are detected - within five seconds.
- *Av1e*: Incoming measurements are temporarily stored elsewhere - no new measurements are lost.

Request Scheduler

- *Av1f*: Fail gracefully: scheduler sends a DataUnavailableException.
- *P3a*: In normal modus, requests and incoming measurments are processed FIFO.
- *P3b*: Go to overloaded modus if necessary and schedule accordingly - premium withing 500msec and normal withing 1500msec.
- *P3d*: In overloaded modus, a stale cached version may be returned.
- *UC8.2b*: Store the measurement.

Customer Database

- *P3c*: Return the SLA's.
- *UC1.4*: Verifies the credentials.
- *UC3.8*: Profile is stored.
- *UC4.5*: Marks the profile as inactive.
- *UC6.5*: Stores the new customer profile.
- *UC8.2a*: Lookup customer id.
- *UC8.3*: Mark RMM as active.
- *UC9.1*: Lookup communication channel of user.
- *UC12.7*: Stores the new alarm recepients.
- *UC13.2c*: Lookup alarm recipients.

Notification System

- *Av1c*: The operators are notified of the problem with the database - within one minute.
- *M1c*: Customers are provided with price updates.
- *UC9.2*: Construct message for specific communication channel.
- *UC9.3*: Send the message to the customer.
- *UC10.3* Notifies the appropriate alarm recipients.
- *UC12.5*: Sends a validation message to the appropriate alarm recipients.
- *UC13.2b*: Notifies the emergency system.
- *UC13.2c*: Notifies all alarm recipients.

Other Functionality

- *Av1a*: The availability of the database does not affect the availability of other types of persistent data.
- *Av2*: Missing measurements.
- *Av3*: Third party billing service failure.
- *P1*: Timely closure of valves.
- *P2*: Anomaly detection.
- *M1*: Dynamic pricing.
- *M2*: Fine-grained metering for enterprises
- *M3*: Decentralized electricity generation.
- *UC1*: Log in.
- *UC2*: Log off.
- *UC3*: Register customer.
- *UC4*: Unregister customer.
- *UC5*: Associate device to customer.
- *UC6*: Customize customer profile.
- *UC7*: Send trame to remote device.
- *UC8.1*: RMM sends measurement.
- *UC8.2c*: Process all data in the trame.
- *UC10.1*: Detect anomaly.
- *UC10.2*: Detect anomaly.
- *UC11*: Operate actuator remotely.
- *UC12.1-4,6*: Set alarm recipients.
- *UC13.1*: Send alarm trame.
- *UC13.2ab*: Store alarm trame.
- *UC14*: Request consumption predictions.
- *UC15*: Generate invoice.
- *UC16*: Mark invoce paid.
- *UC17*: Perform research

2.2 Iteration 2: Other Functionalities (P1, P2, UC10, UC13)

2.2.1 Module to decompose

In this iteration we decompose Other Functionalities.

2.2.2 Selected architectural drivers

The non-functional drivers for this iteration are:

- *P1*: Timely closure of valves
- *P2*: Anomaly detection

The related functional drivers are:

- *UC10*: Detect anomaly
- *UC13*: Send alarm

Rationale Monitoring alarm situations is an important service *ReMeS* offers. Part of this service is the avoidance of disasters by quickly closing valves of damaged pipelines.

2.2.3 Architectural design

Alarms of different types have different priorities. This requires an RMM Communication Scheduler who processes the incoming alarms in the prescribed order. Outgoing control messages are also scheduled by the RMM Communication Scheduler before being passed to the RMM Communication. The RMM Communication looks up in the Device Database by which communication channel the RMM can be contacted and sends the control message to initiate valve control or configure the RMM. The RMM Control composes the control trames for the RMMs.

The RMM Communication Scheduler passes incoming alarms to the Consumption Monitor via the Monitor Scheduler. The Consumption Monitor validates the alarm based on the history of measurements which it gets from the History Database.

Alternatives considered

There are no alternatives to scheduling.

2.2.4 Instantiation and allocation of functionality

Decomposition Main aspects of the resulting decomposition.

Device Database The Device Database stores information on registered devices - for instance the communication channel they use.

RMM Communication The RMM Communication receives the incoming measurements from the RMMs - which it forwards to the Measurement Multicast - and the incoming alarms - which it forwards to the RMM Communication Scheduler. The RMM Communication also receives control and configuration trames. It looks up in the Device Database how to send these trames and sends them to the corresponding RMMs.

RMM Communication Scheduler The RMM Communication Scheduler receives incoming alarms. These are scheduled and passed along to the Monitor Scheduler. The RMM Communication Scheduler also receives control and configuration trames from the RMM Control. These trames are scheduled before they are passed on to the RMM Communication.

RMM Control The RMM Control composes the approriate trames for the control and configuration messages.

Monitor Scheduler The Monitor Scheduler schedules requests for alarm validation and incoming measurements for anomaly detection. These measurements are normaly scheduled in FIFO order. Only in overload mode, they are scheduled according to the SLA of the relevant customer.

Consumption Monitor The Consumption Monitor compares incoming measurements with historical usage of the customer. If this new measurement deviates from the usual consumption, an alarm is raised: the appropriate Remote Actuators are activated and the emergency service and the alarm recipients are notified.
When an alarm from an RMM arrives, the Consumption Monitor checks whether this alarm is justified by checking the history and potentialy asking the RMM for more measurements. However, in case of a gas leak alarm, these checks are bypassed.
The load on the Consumption Monitor can be too high for one instance. To prevent this, multiple instances can be provisioned. The Monitor Scheduler appoints tasks to the individual instances. If there are too many instances, the History Database and the Monitor Scheduler might become bottlenecks.

History Database The History Database stores all data that is needed by the Consumption Monitor to perform anomaly detection.

Other Functionalities 2 This component provides all other functionalities of *ReMeS*, e.g. the invoicing and the generating of statistics.

Deployment

2.2.5 Interfaces for child modules

Device Databases

- Request Device Information
  - `ChannelType requestChannel(DeviceId device) throws NoSuchDeviceException`
    - Effect: The ChannelType of the given device is returned.
    - Exceptions:
      - NoSuchDeviceException: the given device does not exist.

RMM Communication

- Measurements and alarms
  - `void sendMeasurement(Measurement measurement)`
    - Effect: The measurement trame is sent to *ReMeS*.
    - Exceptions: None
  - `void sendAlarm(Alarm alarm)`

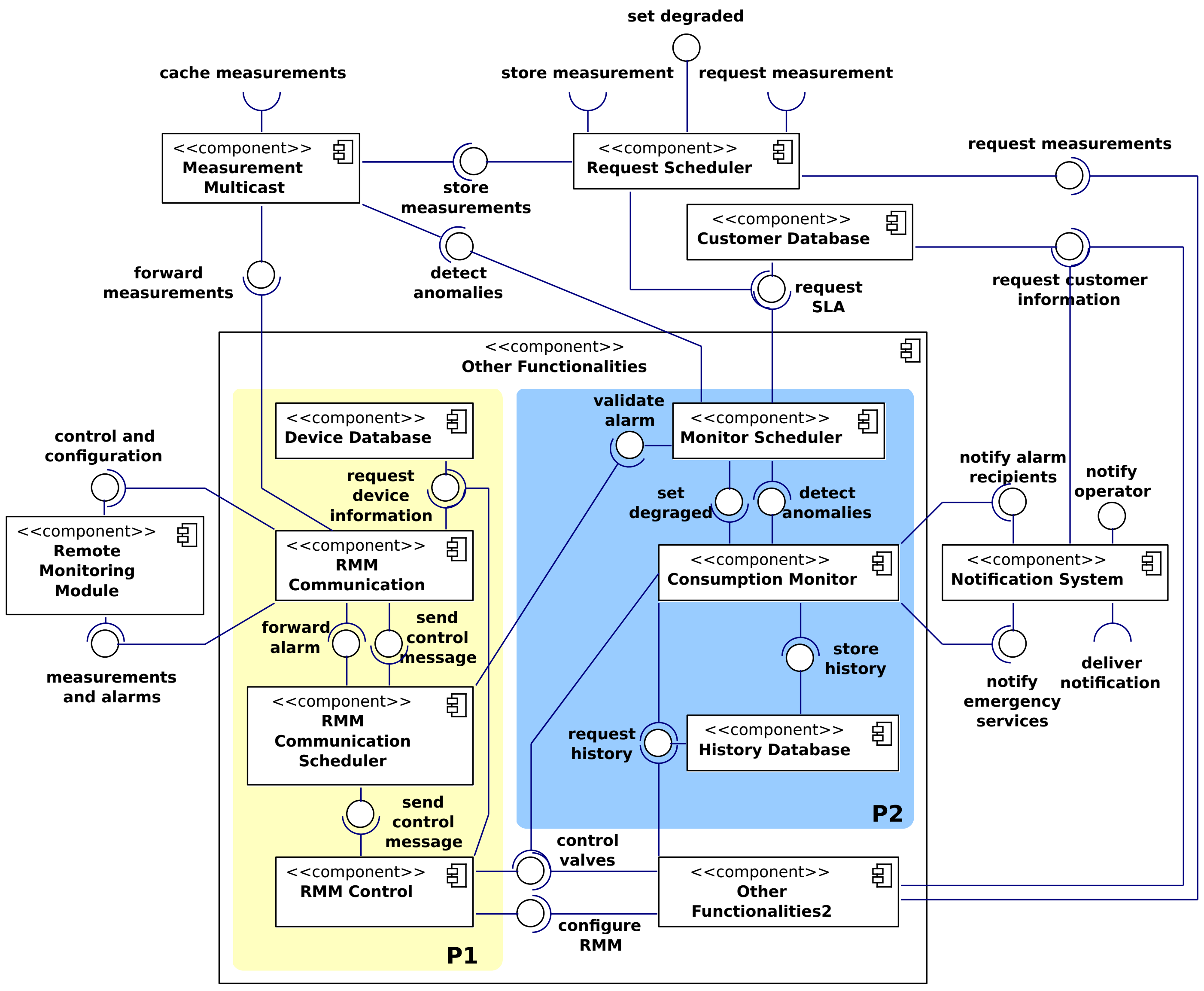
set degraded
cache measurements
store measurement
request measurement
<<component>> Measurement Multicast
<<component>> Request Scheduler
request measurements
store measurements
<<component>> Customer Database
forward measurements
detect anomalies
request SLA
request customer information
<<component>> Other Functionalities
validate alarm
<<component>> Device Database
<<component>> Monitor Scheduler
control and configuration
request device information
set degraged
detect anomalies
notify alarm recipients
notify operator
<<component>> Remote Monitoring Module
<<component>> RMM Communication
<<component>> Consumption Monitor
<<component>> Notification System
forward alarm
send control message
store history
measurements and alarms
deliver notification
notify emergency services
<<component>> RMM Communication Scheduler
request history
<<component>> History Database
send control message
P2
control valves
<<component>> RMM Control
<<component>> Other Functionalities2
configure RMM
P1

Figure 5: Component-and-connector diagram of iteration 2.

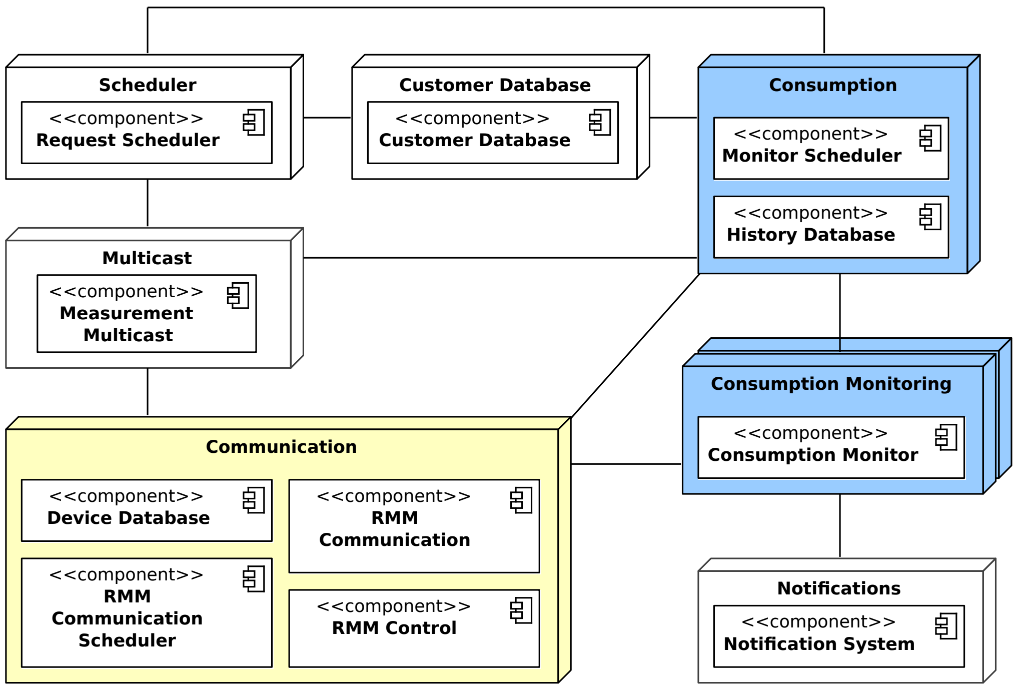

Figure 6: Deployment diagram of iteration 2.

    * Effect: The alarm trame is sent to *ReMeS*.
    * Exceptions: None

- Send control message
  - `void sendControlMessage(DeviceId device, Control control) throws NoSuchDeviceException, UnreachableDeviceException`
    * Effect: The control trame is sent to the the indicated RMM.
    * Exceptions:
      · NoSuchDeviceException: the given device does not exist.
      · UnreachableDeviceException: the given device is not reachable.

RMM Communication Scheduler

- Send control message
  - `void sendControlMessage(DeviceId device, Control control) throws NoSuchDeviceException, CommandNotExecutedException`
    * Effect: The control trame is sent to the indicated RMM.
    * Exceptions:
      · NoSuchDeviceException: the given device does not exist.
      · CommandNotExecutedException: the given command cannot be executed.

RMM Control

- Control valves
  - `void closeActuator(DeviceId measurementDevice) throws NoSuchDeviceException, CommandNotExecut`
    * Effect: The actuators linked to the given device are closed.
    * Exceptions:
      · NoSuchDeviceException: the given device does not exist.
      · CommandNotExecutedException: the given command cannot be executed.

- `void openActuator(DeviceId measurementDevice) throws NoSuchDeviceException, CommandNotExecute`
  - Effect: The actuators linked to the given device are opened.
  - Exceptions:
    - NoSuchDeviceException: the given device does not exist.
    - CommandNotExecutedException: the given command cannot be executed.
- `ActuatorStatus getActuatorStatus(DeviceId measurementDevice) throws NoSuchDeviceException`
  - Effect: The status of the actuators linked to the given device are returned.
  - Exceptions:
    - NoSuchDeviceException: the given device does not exist.

- Configure RMM
  - `void sendConfigurationMessage(DeviceId device, ConfigurationMessage configuration) throws NoSuchDeviceException`
    - Effect: The configuration trame is sent to the indicated RMM.
    - Exceptions:
      - NoSuchDeviceException: the given device does not exist.

History Database

- Store history
  - `void storeHistory(ConsumptionHistory history)`
    - Effect: The history is stored in the database.
    - Exceptions: None
- Request history
  - `ConsumptionHistory requestHistory(DeviceId device) throws NoSuchDeviceException, InvalidQuery`
    - Effect: The history of the device is returned.
    - Exceptions:
      - NoSuchDeviceException: the given device does not exist.
      - InvalidQueryException: thrown when the given query is invalid.

Consumption Monitor

- Detect anomalies
  - `void validateAlarm(Alarm alarm)`
    - Effect: High priority alarms are always assumed to be valid, lower priority alarms are first validated: the history is compared to the current measurement and potentially more measurements are asked. When the alarm is considered valid, the apropriate recipients are notified.
    - Exceptions: None
  - `void processMeasurement(Measurement measurement)`
    - Effect: The incoming measurement is compared to the existing history of the device. When necessary, an alarm is raised.
    - Exceptions: None

Monitor Scheduler

- Set degraded
  - `void setDegraded(MonitorId monitor)`
    - Effect: The Monitor Scheduler goes into degraded mode for this particular monitor.
    - Exceptions: None
  - `void endDegraded(MonitorId monitor)`

    * Effect: The Monitor Scheduler ends degraded mode for this particular monitor.
    * Exceptions: None

- Validate alarm
  - `void validateAlarm(Alarm alarm)`
    * Effect: The given alarm is scheduled.
    * Exceptions: None

Notification System

- Notify alarm recipients
  - `void notifyAlarmRecipients(Alarm alarm)`
    * Effect: The alarm recipients are notified of the given alarm. The recipients are retrieved from the Customer Database.
    * Exceptions: None
- Notify emergency services
  - `void notifyEmergencyServices(Alarm alarm)`
    * Effect: The emergency services are notified of the given alarm.
    * Exceptions: None
- Notify operator (same as iteration 1)

2.2.6 Data type definitions

A short description of the occurring datatypes.

DeviceId This data element represents a device.

ChannelType This data element represents a type of transmission channel.

Measurement This data element represents a measurement trame.

Alarm This data element represents an alarm trame.

Control This data element represents a control trame.

ActuatorStatus This data element represents the status of an actuator.

ConsumptionHistory This data element represents the consumption history of a device.

MonitorId This data element represents a specific Consumption Monitor.

2.2.7 Verify and refine

This section describes per component which (parts of) the remaining requirements it is responsible for.

Device Database

- *Av1a*: The availability of the database does not affect the availability of other types of persistent data.
- *UC7.2*: The communication channel for each device.

RMM Communication

- *Av2*: Missing measurement.
- *P1*: Splits the alarms from the measurements.
- *UC7.2*: Determines the communication channel.
- *UC7.3*: Sends the trame to the RMM.
- *UC7.6*: Receives the confirmation trame.

RMM Communication Scheduler

- *Av1b*: The database should have a guaranteed minimal up-time.
- *P1*: Gives precedence on urgent alarms, such as gas alarms and makes sure no trames starve.

RMM Control

- *Av1b*: The database should have a guaranteed minimal up-time.
- *UC7.1*: Constructs the control and configuration messages for the RMM.
- *UC11.6*: Changes the state of the actuator.
- *UC13.2a*: Determines which Remote Actuator is linked to which alarm.

Monitor Scheduler

- *Av1b*: The database should have a guaranteed minimal up-time.
- *P1*: Gives precedence on urgent alarms, such as gas alarms and makes sure no trames starve.
- *P2*: When more than 50 anomaly detections arrive per minute, the Monitor Scheduler goed into overloaded modus. In that case, 98% of the measurements should still be handled within their individual deadline of 10 minutes of arrival.

Consumption Monitor

- *Av1b*: The database should have a guaranteed minimal up-time.
- *P2*: Sets the Monitor Scheduler to degraded mode.
- *UC8.2c*: Processes the incoming trames.
- *UC10.2*: Applies anomaly detection.
- *UC13.2b*: Determines whether the emergency system should be notified.

History Database

- *Av1a*: The availability of the database does not affect the availability of other types of persistent data.
- *UC10.1*: Provides the history needed for the anomaly detection.
- *UC14.2*: Provides the consumption histories.

Other Functionalities 2

- *Av1a*: The availability of the database does not affect the availability of other types of persistent data.
- *Av3*: Third party billing service failure.
- *M1*: Dynamic pricing.
- *M2*: Fine-grained metering for enterprises.
- *M3*: Decentralized electricty generation.
- *UC1*: Log in.
- *UC2*: Log off.
- *UC3*: Register customer.
- *UC4*: Unregister customer.
- *UC5*: Associate device to customer.

- *UC6*: Customize custmer profile.
- *UC7.4*: The RMM receives and processes the trame.
- *UC7.5*: The RMM sends a confirmation trame.
- *UC8.1*: RMM sends measurement.
- *UC11.1-5,7*: Operate actuator remotely.
- *UC12.1-4,6*: Set alarm recipients.
- *UC13.1*: The RMM sends an alarm trame to *ReMeS*.
- *UC14.1,3*: Request consumption predicion.
- *UC15*: Generate invoice.
- *UC16*: Mark invoice paid.
- *UC17*: Perform research.

2.3 Iteration 3: Other functionalities 2 (M1, UC15)

2.3.1 Module to decompose

In this iteration we decompose Other Functionalities 2.

2.3.2 Selected architectural drivers

The non-functional drivers for this iteration are:

- *M1*: Dynamic pricing

The related functional drivers are:

- *UC15*: Generate invoice

Rationale With the advent of smart-grids, utility prices in the near future will change dynamically on a minute-to-minute basis. *ReMeS* should help to make this possible.

2.3.3 Architectural design

To allow dynamic pricing in the future, prices per customer or customer groups should be stored in a Pricing Database. To allow for modifiability intermediate components are used for the communication with external services. The invoices are generated from current prices in the Pricing Database by the Invoice Generator.

2.3.4 Instantiation and allocation of functionality

Decomposition Main aspects of the resulting decomposition.

Payment Communication The Payment Communication communicates with the Payment Service. This localizes changes to the Payment Services. If the Payment Service is unreachable, the invoices are resent with exponential backoff, after five tries an operator is notified.

Invoice Generator The Invoice Generator retrieves the consumption from the Measurement Database and the price from the Pricing Database and calculates the total dues of a customer. Together with the customer's information an invoice is generated and sent to the Payment Service. Invoices are retained untill they can be marked as paid.

Pricing Database The Pricing Database stores all pricing information received from the UIS. The prices are linked to a time period and to customers or customer groups.

UIS Communication The UIS Communication communicates with the UIS and notifies the appropriate customers of price updates. This localizes changes to the UIS.

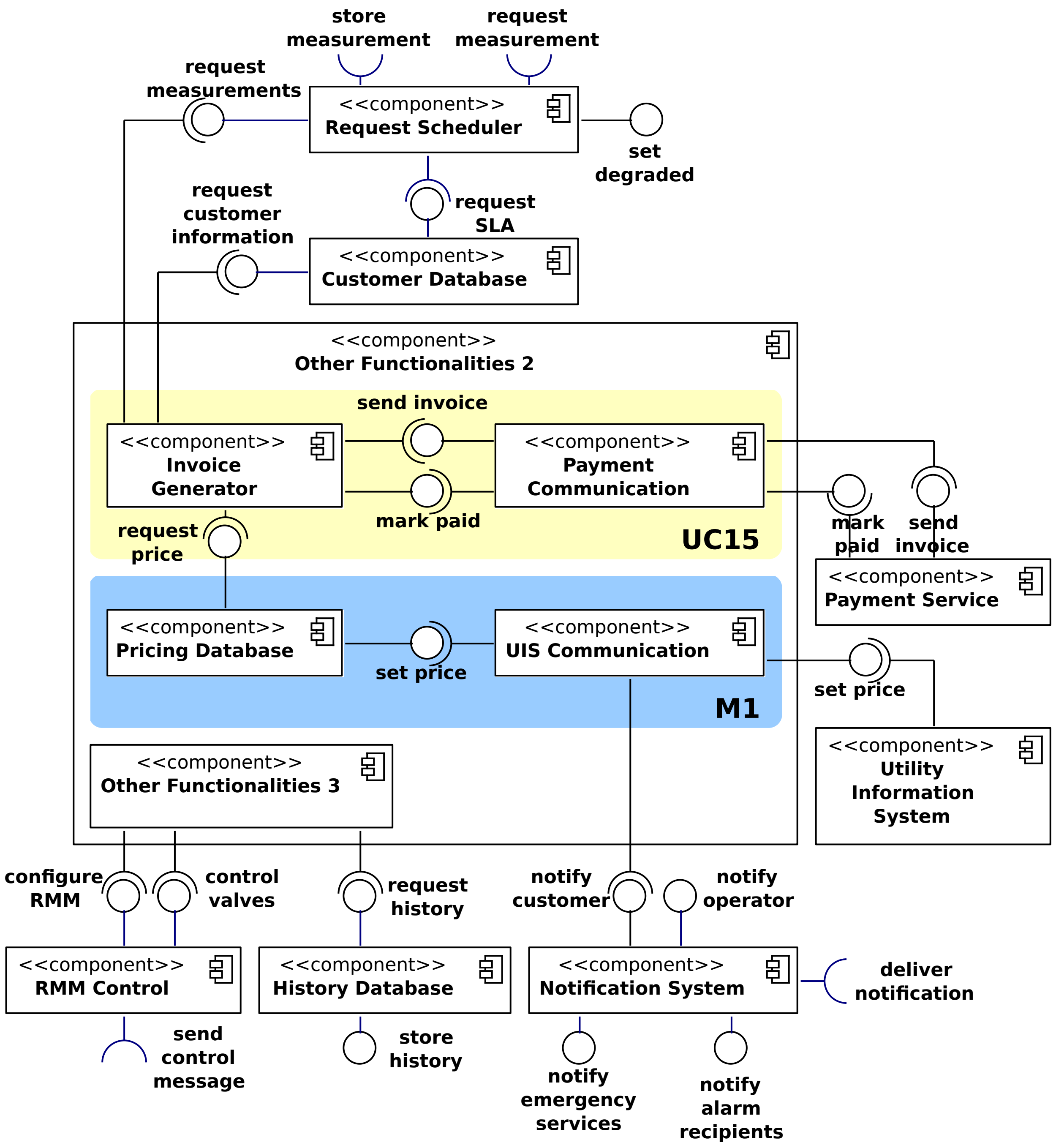
store measurement
request measurement
request measurements
<<component>> Request Scheduler
set degraded
request SLA
request customer information
<<component>> Customer Database
<<component>> Other Functionalities 2
send invoice
<<component>> Invoice Generator
<<component>> Payment Communication
mark paid
request price
UC15
mark paid
send invoice
<<component>> Payment Service
<<component>> Pricing Database
set price
<<component>> UIS Communication
M1
set price
<<component>> Utility Information System
<<component>> Other Functionalities 3
configure RMM
control valves
request history
notify customer
notify operator
<<component>> RMM Control
<<component>> History Database
<<component>> Notification System
deliver notification
send control message
store history
notify emergency services
notify alarm recipients

Figure 7: Component-and-connector diagram of iteration 3.

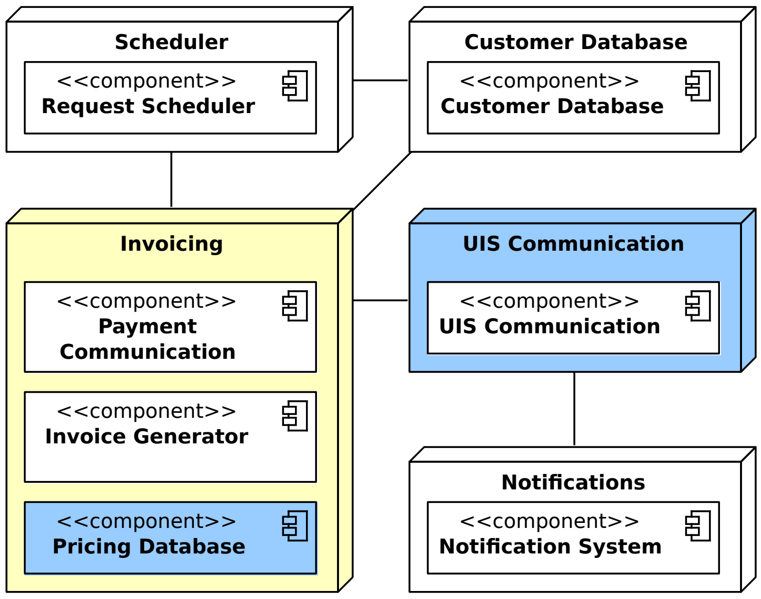

Figure 8: Deployment diagram of iteration 3.

Deployment

2.3.5 Interfaces for child modules

Payment Communication

- Mark paid
  - `void markPaid(Invoice invoice)`
    - Effect: The completion of an invoice is passed along to the Invoice Generator.
    - Exceptions: None
- Send Invoice
  - `void sendInvoice(Invoice invoice)`
    - Effect: The invoice is sent to the appropriate Payment Service.
    - Exceptions: None

Invoice Generator

- Mark paid
  - `void markPaid(Invoice invoice)`
    - Effect: The completion of an invoice is passed along to the Invoice Generator.
    - Exceptions: None

Pricing Database

- Request price
  - `Price requestPrice(PaymentGroup customer, TimePeriod period, Utility utility) throws NoSuchCustomerException`
    - Effect: The price valid during the period for the customer for the utility is returned.
    - Exceptions:
      - NoSuchCustomerException: thrown when the given customer is not found.
- Set Price

  - `void setPrice(Price price, Utility utility, PaymentGroup customer)`
    * Effect: The new price is stored in the database.
    * Exceptions: None

UIS Communication

- Mark paid
  - `void markPaid(Invoice invoice)`
    * Effect: The completion of an invoice is passed along to the Invoice Generator.
    * Exceptions: None
- Send Invoice
  - `void sendInvoice(Invoice invoice)`
    * Effect: The invoice is sent to the appropriate Payment Service.
    * Exceptions: None

Notification System

- Notify Customer
  - `void notifyCustomer(Price update, PaymentGroup customer)`
    * Effect: The customer is informed of a price update.
    * Exception: None
- Notify alarm recipients (same as iteration 2)
- Notify emergency services (same as iteration 2)
- Notify operator (same as iteration 1)

2.3.6 Data type definitions

A short description of the occurrences.

Invoice This data element represents an invoice.

PaymentGroup Group of customers.

Price Value in the appropriate currency.

TimePeriod Period of time.

2.3.7 Verify and refine

This section describes per component which (parts of) the remaining requirements it is responsible for.

Payment Communication

- *Av3*: Third party billing service failure.
- *UC14.1,3*: Request consumption predicion.

Invoice Generator

- *M1e*: Billing is extended to dynamic pricing.
- *UC15*: Generate invoice.
- *UC16*: Mark invoice paid.

Pricing Database

- *M1*: Dynamic pricing.

UIS Communication

- *M1a*: Communication is extended to retrieve prices.

Other Functionalities 3

- *Av1a*: The availability of the database does not affect the availability of other types of persistent data.
- *M2*: Fine-grained metering for enterprises.
- *M3*: Decentralized electricty generation.
- *UC1*: Log in.
- *UC2*: Log off.
- *UC3*: Register customer.
- *UC4*: Unregister customer.
- *UC5*: Associate device to customer.
- *UC6*: Customize custmer profile.
- *UC7.4*: The RMM receives and processes the trame.
- *UC7.5*: The RMM sends a confirmation trame.
- *UC8.1*: RMM sends measurement.
- *UC11.1-5,7*: Operate actuator remotely.
- *UC12.1-4,6*: Set alarm recipients.
- *UC13.1*: The RMM sends an alarm trame to *ReMeS*.
- *UC17*: Perform research.

3 Resulting partial architecture

This section provides an over of the architecture constructed through ADD.

3.1 Deployment view

A short discussion of the allocation of components to physical nodes based on a context diagram and a deployment diagram.

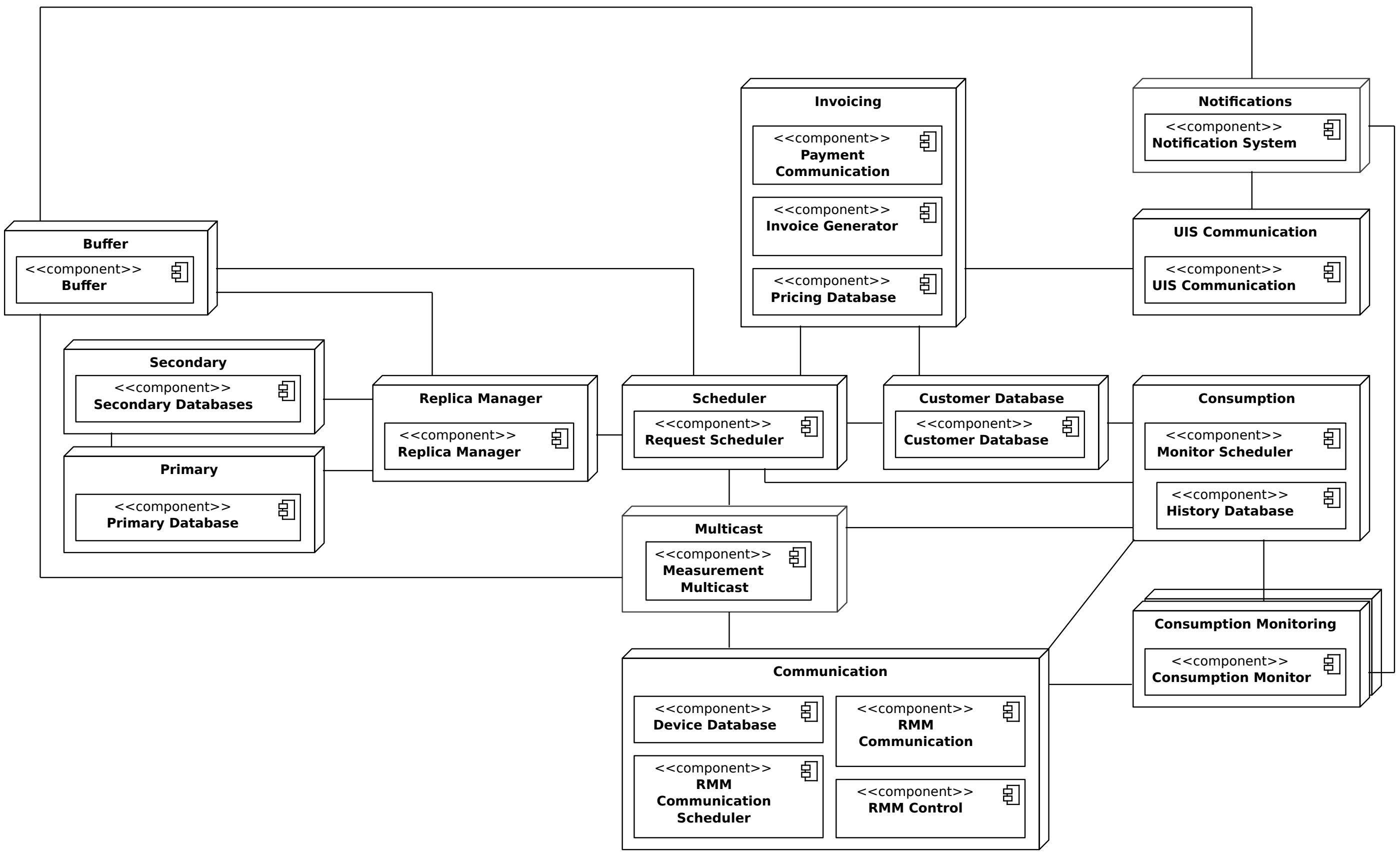

Figure 9: Primary diagram for the allocation view.