Remote Measurement, Monitoring and Control System

Software Architecture Report

Contents

1 Client-Server View

UML: component diagram Shows the elements from this view and their interrelations.

1.1 Context Diagram

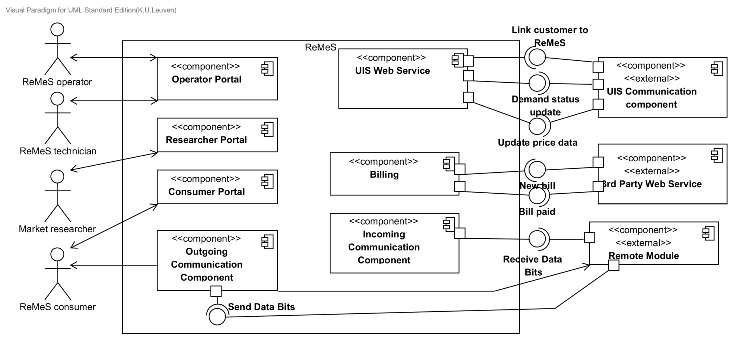

Visual Paradigm for UML Standard Edition(K.U.Leuven)
ReMeS operator
ReMeS technician
Market researcher
ReMeS consumer
ReMeS
<<component>> Operator Portal
<<component>> Researcher Portal
<<component>> Consumer Portal
<<component>> Outgoing Communication Component
<<component>> UIS Web Service
<<component>> Billing
<<component>> Incoming Communication Component
Link customer to ReMeS
Demand status update
Update price data
New bill
Bill paid
Receive Data Bits
Send Data Bits
<<component>> <<external>> UIS Communication component
<<component>> <<external>> 3rd Party Web Service
<<component>> <<external>> Remote Module

Figure 1: ReMeS Context Diagram

1.2 Full Diagram

Figure 2 illustrates the full client-server view. In the subsequent section we describe each component.

1.3 Element Catalog

1.3.1 Incoming Communication Component

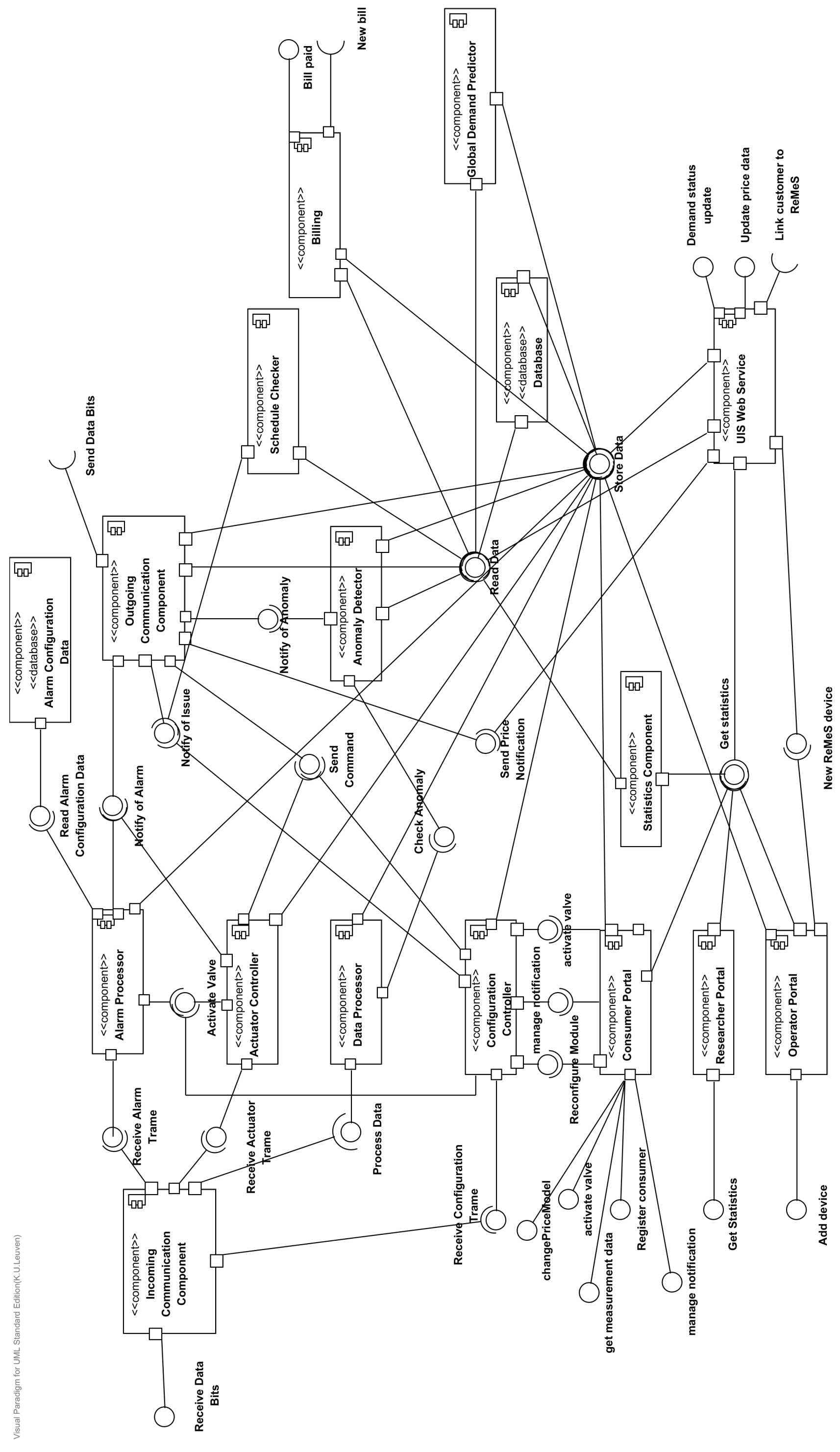
This component is connected to a modem and network. We abstract away from low-level implementation specifics, such as, how the precise communication with the modem is realized. We assume that this component simply receives bits of data that represent, e.g., a data trame, an alarm trame or a configuration trame. The received bits are converted from their bit representation to a data trame, an alarm trame or a configuration trame (otherwise the set of bits are discarded). The trame unpacking is a bit-level operation and is extremely fast. Then the trames are sorted according to their priorities. Alarms and configuration trames relevant to the control valves are treated with higher priority than data and other kinds of configuration trames. Alarm trames are passed to the Alarm Processor component, configuration trames related to actuators are passed to the Actuator Controller, other kinds of configuration trames are passed to the Configuration Controller and data trames are passed to the Data Processor component (a load balancer could be added that based on the current load of each instance of the Data Processing Node passes to the least loaded Data Processor).

1.3.2 Data Processor

Data Processor component is in charge of storing the data trame in the measurements database and passing this trame to the Anomaly Detector component for further processing.

1.3.3 Anomaly Detector

Anomaly Detector component is triggered by the Data Processor to analyze the incoming data for any anomalies. This component features various algorithms to create and update a consumption profile and check whether the current consumption fits this profile. Upon the detection of a possible anomaly this component stores that information and contacts the Outgoing Communication Component to handle the anomaly notification.



<<component>> Global Demand Predictor
<<component>> Billing
Bill paid
New bill
<<component>> Schedule Checker
<<database>> Database
<<component>> UIS Web Service
Demand status update
Update price data
Link customer to ReMeS
Send Data Bits
<<component>> <<database>> Alarm Configuration Data
<<component>> Outgoing Communication Component
<<component>> Anomaly Detector
Notify of Anomaly
Read Data
Store Data
Notify of Issue
Send Command
Send Price Notification
<<component>> Statistics Component
Get statistics
New ReMeS device
Read Alarm Configuration Data
Notify of Alarm
Check Anomaly
<<component>> Alarm Processor
Activate Valve
<<component>> Actuator Controller
<<component>> Data Processor
<<component>> Configuration Controller
activate valve
manage notification
Reconfigure Module
<<component>> Consumer Portal
<<component>> Researcher Portal
<<component>> Operator Portal
Receive Alarm Trame
Receive Actuator Trame
Process Data
Receive Configuration Trame
changePriceModel
activate valve
Register consumer
get measurement data
manage notification
Get Statistics
Add device
<<component>> Incoming Communication Component
Receive Data Bits
Visual Paradigm for UML Standard Edition(K.U.Leuven)

Figure 2: Client-Server view

1.3.4 Outgoing Communication Component

This components handles all outgoing communication not only towards remote monitoring modules, control valves, but also towards consumers (in the form of an email or an SMS). Again we assume that the low level network and modem communication is readily available. This component contains a prioritizer as well in order to handle the outgoing data according to their priorities (e.g., actuator commands have priority over device reconfiguration commands, all commands have priority over notifications, etc.). For notifications this component consults the database to obtain the relevant user information (e.g., email to which the notification should be sent).

1.3.5 Alarm Processor

Alarm Processor is a component that handles the alarms that are sent by the remote monitoring modules. This component determines whether there is a control valve installed for a given module from which an alarm is received and initiates a valve shut down via the Actuator Controller component. Alarm Processor also stores the alarm information in the measurements database.

1.3.6 Actuator Controller

Actuator Controller is activated by the Alarm Processor in case a control valve must be shut down due to an alarm. This component is in charge of sending the control command via the Sender component and monitoring the acknowledgement of this command (received and passed from the Prioritizer component).

1.3.7 Database

The database component is a database that stores all the data.

1.3.8 Alarm Configuration Data

Alarm Configuration Data is a database view on top of the Database component that provides a rapid access to the configuration information relevant to the actions in case of an alarm (e.g., triggering a control trame that will shut off the control valve). The precise setup of this view is the task of the designer who leverages on our architecture.

1.3.9 Consumer Portal

Internet Portal component is available through the WWW and provides a set of functionalities to the end consumers. It allows new consumers to register online for ReMeS and provides to the authenticated consumers information on their consumption, alarms, predictions, etc.

1.3.10 Researcher Portal

Researcher Portal is a component available through the WWW as well and it provides mainly statistics functions to market researchers and other authorized users interested in obtaining statistics from the ReMeS system. All users of the Researcher Portal must be authenticated using a username and a password. The provision of this authentication data is outside the scope of this assignment.

1.3.11 Operator Portal

Operator Portal component is similar to the Consumer Portal, but it is available only for ReMeS operators and ReMeS technicians. ReMeS technicians mainly use it to add new monitoring modules to the system. ReMeS operators have the complete functionality to manage the consumers, their monitoring modules and produces measurements and alarms, etc.

1.3.12 Configuration Controller

Configuration Controller is in charge of handling the device reconfiguration via the Consumer Portal and Operator Portal components. The Configuration Controller component stores all reconfiguration requests and passes them to the Outgoing Communication Component as a command. Configuration Controller component also monitors the acknowledgements of the reconfiguration commands from the remote modules.

1.3.13 Billing

The Billing component is in charge of creating the bills in a timely fashion and pushing the created bills towards the web service provided by a third party in charge of the billing. The Billing component features a process running in batch mode that triggers the bill creation. The third party must follow up on the bills and calls back on this component to notify that a bill has been paid.

1.3.14 Global Demand Predictor

Global Demand Predictor is a background component that runs various algorithms to predict current and forecast future consumption demand. The predicted values are stored in the database.

1.3.15 Price Rate Notifier

In an advanced smart grid setting the Price Rate Notifier component is in charge of sending regularly the price rate notifications to all relevant remote modules (typically only for electricity monitoring modules). This component fetches the market pricing information from the database and combines that generic price with the user specific price formula also stored in the database. The pricing notification is sent via the Outgoing Communication Component as trames towards each remote monitoring module. *Note that at this point we assume that a direct communication channel to any remote monitoring module is available (e.g., fixed IPs). Moreover, it is the responsibility of the remote monitoring module to locally notify all "smart" electricity devices regarding the price rates.*

1.3.16 Schedule Checker

Schedule Checker component is a batch component that scans for modules for which no measurement data has been received. If data is missing, the Outgoing Communication Component is contacted for creating and sending a notification to the consumer.

1.3.17 Statistics Component

Statistics Component is used by various other components of the system to fetch various statistics for the interested stakeholders (e.g., UIS, consumer, market researchers). This component consults information in the database and creates the required statistics. The performance-related aspects of this component (i.e., making sure that heavy requests do not overpower the component) are left out of scope.

1.3.18 UIS Web Service

Each instance of the UIS Web Service component serves the respective UIS for providing status update containing information relevant for the UIS (e.g., current and future consumption demand, number of alarming incidents, etc.). This component also features an interface to get live price information potentially used by the Billing component.

1.4 Interface specifications

1.4.1 Incoming Communication Component

1.4.1.1 receiveData

Interface identity Receiver - receiveData.

Resources provided This interface provides the remote monitoring and control modules the necessary mechanism to send trames to ReMeS. The implementation of this interface is out of scope.

1.4.2 Outgoing Communication Component

1.4.2.1 notifyAlarm

Interface identity OutgoingCommunicationComponent - notifyAlarm.

Resources provided Creates a new alarm notification for the provided module and sends a notification to the consumer linked to this module.

- Syntax: void notifyAlarm(Module m, Details d).
- Semantics: A new notification is created based on the provided module and additional details (if any). The notification is persisted. The component fetches the consumer linked to this module along with his preferences and notifies him as specified by the consumer previously.

1.4.2.2 notifyAnomaly

Interface identity OutgoingCommunicationComponent - notifyAnomaly.

Resources provided Creates a new anomaly notification for the provided module and sends the notification to the consumer linked to this module.

- Syntax: void notifyAlarm(Module m, Details d).
- Semantics: A new notification is created based for the provided module and additional details (if any). The notification is stored in the database. The component fetches the consumer linked to this module along with his preferences and notifies him as specified by the consumer previously.

1.4.2.3 notifyPriceChange

Interface identity OutgoingCommunicationComponent - notifyPriceChange.

Resources provided Creates a new price change notification for the provided module.

- Syntax: void notifyPriceChange(Module m, Details d).
- Semantics: A new price notification is created based for the provided module and additional details (if any). The price notification is stored in the database.

1.4.2.4 sendCommand

Interface identity OutgoingCommunicationComponent - sendCommand.

Resources provided Sends a command to a remote monitoring or a control module.

- Syntax: void sendCommand(Module m, Command c).
- Semantics: The command is stored in the database. The command is translated into a low level representation and sent to the specified module. The component requesting the command to be sent is in charge of accepting any acknowledgement message that arrives asynchronously on the Incoming Communication Component.

1.4.3 Data Processor

1.4.3.1 processData

Interface identity DataProcessor - processData.

Resources provided This interfaces stores the provided data trame and forwards it for further processing.

- Syntax: void processData(NativeDataTrame t)
- Semantics: The provided trame is stored in the database. The trame is further passed to Anomaly Detector component for further processing.

1.4.4 Anomaly Detector

1.4.4.1 checkAnomaly

Interface identity AnomalyDetector - checkAnomaly.

Resources provided This interface checks the provided trame for any potential anomaly.

- Syntax: void checkAnomaly(NativeDataTrame t)
- Semantics: The provided data trame is checked for any anomalies. Potentially history of measurements is also fetched from the database (the precise algorithm details are out of scope). When an anomaly is detected the Outgoing Communication Component is notified of anomaly and it handles the further processing of it.

1.4.5 Alarm Processor

1.4.5.1 receiveAlarmTrame

Interface identity AlarmProcessor - receiveAlarmTrame.

Resources provided This interface receives alarm trames and processes them further.

- Syntax: void receiveAlarmTrame(NativeAlarmTrame t)
- Semantics: First of all the Alarm Configuration Data is checked if there is a remote control module linked to the module to which the provided alarm trame (in native format) is linked. If one is found the activateValve interface of the Actuator Controller is invoked for processing a shut down request for the remote control module. Afterwards an alarm notification is requested via the Outgoing Communication Component. Finally, the alarm details are stored in the Measurement Data database.

1.4.6 Actuator Controller

1.4.6.1 activateValve

Interface identity ActuatorController - activateValve.

Resources provided This interface processes the request to remotely shut off the valve of a remote control module.

- Syntax: void activateValve(RemoteControlModule r)
- Semantics: A new command for activating the valve of the provided remote control module is sent via the sendCommand interface of the Outgoing Communication Component. An internal data structure is kept to keep track of all commands that are in processing mode (i.e., they are sent to the Outgoing Communication Component and an acknowledgement is expected). If the acknowledgement does not arrive after a certain period of time this component resends the control command. If no acknowledgement is received after several attempts this component gives up and creates a new alarm notification via the notifyOfAlarm interface of the Outgoing Communication Component that notifies the consumer of this chain of events.

1.4.6.2 receiveActuatorTrame

Interface identity ActuatorController - receiveActuatorTrame.

Resources provided This interface receives a trame either acknowledging a successful execution of a command or an error message.

- Syntax: void receiveActuatorTrame(NativeActuatorTrame t)
- Semantics: If the received trame acknowledges a successful execution of a previous command the internal data structure that holds the commands that are in processing mode is updated (see activateValve interface). If the received trame reports of an error message a new alarm notification is created via the notifyOfAlarm interface of the Outgoing Communication Component that notifies the consumer of this error.

1.4.7 Database

1.4.7.1 readData

Interface identity Database - readData.

Resources provided This interface provides access to any data stored in the database.

- Syntax: Data readData(Query q)
- Semantics: The provided query is executed and the result of the query is returned.

1.4.7.2 storeData

Interface identity Database - storeData.

Resources provided This interface allows provides access to store data in the database.

- Syntax: void storeData(Query q)
- Semantics: The provided update or insert query is executed.

1.4.8 Consumer Portal

1.4.8.1 registerConsumer

Interface identity ConsumerPortal - registerConsumer.

Resources provided This interface allows consumers that are new to ReMeS to register themselves in the system.

- Syntax: void registerConsumer(ConsumerData d)
- Semantics: A new consumer is registered in the system. All relevant information is added to the database. From this point on ReMeS modules can be linked to this consumer profile. The consumer receives a letter at a home address he has specified containing his login information.

1.4.8.2 login

Interface identity ConsumerPortal - login.

Resources provided This interface authenticates a consumer..

- Syntax: void login(Credentials c)
- Semantics: This interface identifies a given consumer (based on the provided credentials) in the system. From this point on the consumer may use all other interfaces provided by this component.

1.4.8.3 getMeasurementData

Interface identity ConsumerPortal - getMeasurementData.

Resources provided This interface returns the measurement data.

- Syntax: MeasurementData getMeasurementData(Module m, Specification s)
- Semantics: The measurement data is fetched from the database for the provided module and according to the provided specification. The specification is typically a period for which the measurement data should be fetched. The raw measurement data is returned. Data visualization for the consumer is out of the scope at the architectural level.

1.4.8.4 getStatistics

Interface identity ConsumerPortal - getStatistics.

Resources provided This interface returns statistical information.

- Syntax: Blob getStatistics(OptionsList o)
- Semantics: This interface accepts as a parameter a list of options specifying the statistical information to be retrieved. The interface creates a low-level SQL query and leverages on the StatisticsComponent - getStatistics interface to obtain the necessary information. The result is packaged in a blob that represents a textual or graphical representation of the requested information (the precise blob representation is a design/implementation level concern). *Note that the getMeasurementData interface deals mostly with raw measurements, while a more aggregated views are handled by the getStatistics interface.*

1.4.8.5 getAlarmData

Interface identity ConsumerPortal - getAlarmData.

Resources provided Retrieves information and alerts related to alarms and alarm notifications.

- Syntax: AlarmData getAlarmData(Module m, Specification s)
- Semantics: This interface fetches alarm data from the database for the given module and according to the provided specification. The specification could be related to the time period for which information should be retrieved and the alarm type. Raw alarm data is returned.

1.4.8.6 getNotifications

Interface identity ConsumerPortal - getNotifications.

Resources provided Retrieves all notifications.

- Syntax: Notifications getNotifications(Module m, Specification s)
- Semantics: This interface fetches all notifications from the database for a given module and according to the provided specification. The specification could relate to the time period for which notifications should be retrieved and the notification type (e.g., alarm notification, reconfiguration notification, etc). A list of notifications is returned.

1.4.8.7 manageNotification

Interface identity ConsumerPortal - manageNotification.

Resources provided Provides an interface to manage (add, modify or delete) the circumstances for which a notification should be created and the notification recipient.

- Syntax: void manageNotification(Module m, Type t, Condition c, Recipient r, Operation o)
- Semantics: Based on the specified operation a notification specification is created, modified or deleted. The notification contains the relevant module, the notification type (e.g., alarm notification, anomaly notification, etc.), the condition when it should be created and a recipient information (e.g., email address, SMS recipient specification, etc.).

1.4.8.8 changePriceModel

Interface identity ConsumerPortal - changePriceModel.

Resources provided Provides an interface to change the price model of a module.

- Syntax: void changePriceModel(Module m, PriceModel m)
- Semantics: This interface leverages on the reconfigureModule of the Configuration Controller component (see further) that handles the complete reconfiguration workflow. The specified price model is translated into a new configuration of the module and passed to the Configuration Controller. This interface is available only for electricity modules.

1.4.9 Configuration Controller

1.4.9.1 reconfigureModule

Interface identity ConfigurationController - reconfigureModule.

Resources provided This interface sends a reconfiguration command to the remote module.

- Syntax: void reconfigureModule(Module m, Configuration c)
- Semantics: The provided configuration is packaged into a command and sent via the sendCommand interface of the Outgoing Communication Component. The command is added to an internal data structure that keeps track of all commands that are in processing mode. If the command is not acknowledged (see receiveConfigurationTrame interface) the command is resent. After several retries the consumer is notified of the problem via the notifyOfIssue interface of the Outgoing Communication Component.

1.4.9.2 receiveConfigurationTrame

Interface identity ConfigurationController - receiveConfigurationTrame.

Resources provided .

- Syntax: void receiveConfigurationTrame(NativeTrame t)
- Semantics: If the received trame acknowledges a successful execution of a reconfiguration command the internal data structure that holds the commands that are in processing mode is updated (see reconfigureDevice interface). If the received trame reports of an error message a notification is created via the notifyOfIssue interface of the Outgoing Communication Component that notifies the consumer of this error.

1.4.10 Operator Portal

The intranet portal contains all interfaces provided by the Internet Portal. The main difference is that when using these interfaces via the Intranet Portal the selected consumer is not fixed. I.e., for the Consumer Portal an authorization mechanism makes sure that consumers cannot access any information that is not related to their own modules. The following are the interface in addition to ones specified in Internet Portal component.

Interface identity OperatorPortal - addRemesModule.

Resources provided A new ReMeS module is added to the system and linked to a specified consumer.

- Syntax: void addRemesModule(Module m, Consumer c)
- Semantics: A new module linked to a given consumer is stored in the database. The module is marked as inactive. The module is preprogrammed in a way that it sends an initial data trame to ReMeS as soon as it is physically installed. From that point on the module is marked as active.

1.4.11 Researcher Portal

1.4.11.1 login

Interface identity ResearcherPortal - login.

Resources provided This interface authenticates a researcher.

- Syntax: void login(Credentials c)
- Semantics: This interface identifies a given researcher (based on the provided credentials) in the system. From this point on the researcher may use all other interfaces provided by this component. *Note that we leave out of scope the workflow of how the researchers are added to ReMeS.*

1.4.11.2 getStatistics

Interface identity ResearcherPortal - getStatistics.

Resources provided This interface returns statistical information.

- Syntax: Blob getStatistics(OptionsList o)
- Semantics: This interface accepts as a parameter a list of options specifying the statistical information to be retrieved. The interface creates a low-level SQL query and leverages on the StatisticsComponent - getStatistics interface to obtain the necessary information. The result is packaged in a blob that represents a textual or graphical representation of the requested information.

1.4.12 Statistics Component

1.4.12.1 getStatistics

Interface identity StatisticsComponent - getStatistics.

Resources provided Retrieves statistical information.

- Syntax: Data[] getStatistics(Query q)
- Semantics: Statistical information based on the provided query is retrieved from the database and returned in its raw form (as an array of data points)

1.4.13 Billing

1.4.13.1 billPaid

Interface identity Billing - billPaid.

Resources provided This interface marks a bill as paid.

- Syntax: void billPaid(BillInformationMessage m)
- Semantics: The corresponding bill information is retrieved from the provided message. This bill is marked as paid in the database.

1.4.14 UIS Web Service

1.4.14.1 updatePriceData

Interface identity UISWebService - updatePriceData.

Resources provided Updates the current market price information of a utility.

- Syntax: void updatePriceData(PriceData pd)

- Semantics: A new price data specification is added to the database based on the received information. The UISWebService component also checks the database for all modules that are affected by this price modification. For each such module a price change notification is created and passed towards the Outgoing Communication Component that notifies the remote module of the current price rate. The price data is used further by the Billing component to calculate the total cost for the given period and create a bill.

1.4.14.2 demandStatusUpdate

Interface identity UISWebService - demandStatusUpdate.

Resources provided Provides an update of the current demand for a specified utility company.

- Syntax: Blob demandStatusUpdate(UtilityCompanyId u)
- Semantics: The global demand predictions for the provided utility company are retrieved from the database and returned to the requesting component.

2 Deployment View

UML: deployment diagram In this view the components from the Client-Server view are allocated to physical nodes (see above for the structure of the documentation).

2.1 Context Diagram

The context diagram (see fig. 3) illustrates the nodes that are in charge of the external communications.

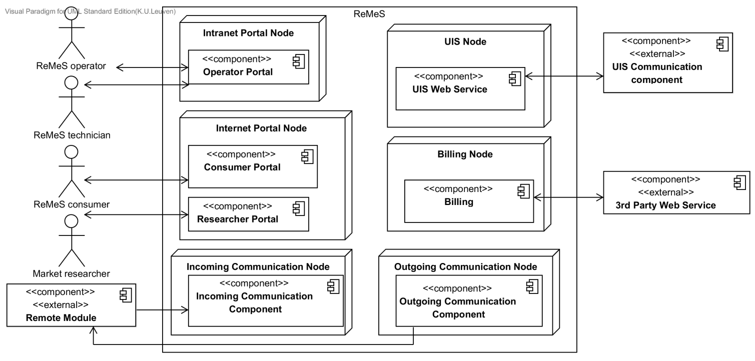


Figure 3: ReMeS Deployment View: Context Diagram

2.2 Full Diagram

Figure 4 illustrates the deployment view of the system architecture. In section 2.3, we briefly describe each node of the deployment view.

2.3 Element Catalog

2.3.1 Incoming Communication Node

This node is in charge of receiving all incoming data from remote monitoring and control modules. It is expected that this node will handle a fairly large amount of data that should be prioritized and balanced between multiple instances of the Data Processing Node. Having a separate node for all the incoming communication is mainly motivated by the performance and availability quality requirements.

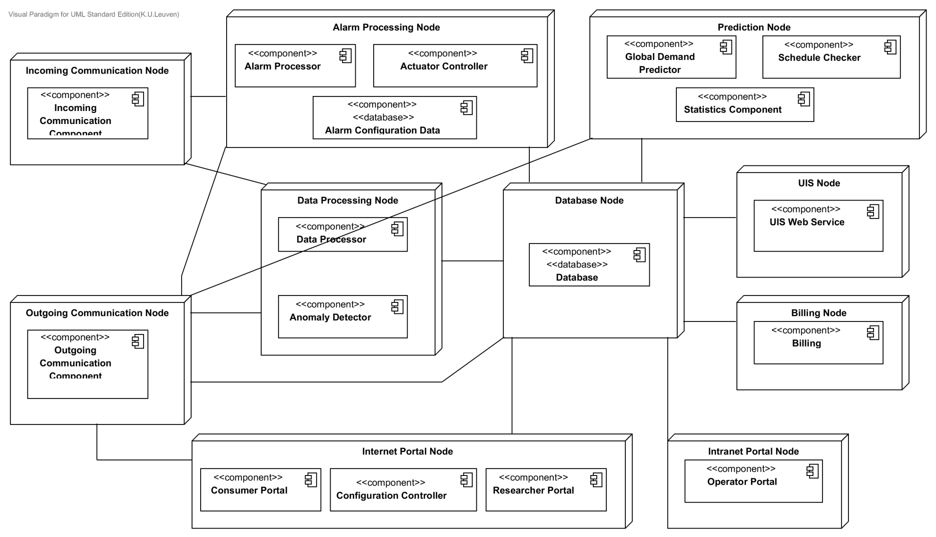

Visual Paradigm for UML Standard Edition(K.U.Leuven)
Alarm Processing Node
<<component>> Alarm Processor
<<component>> Actuator Controller
<<component>> <<database>> Alarm Configuration Data
Prediction Node
<<component>> Global Demand Predictor
<<component>> Schedule Checker
<<component>> Statistics Component
Incoming Communication Node
<<component>> Incoming Communication Component
Data Processing Node
<<component>> Data Processor
<<component>> Anomaly Detector
Database Node
<<component>> <<database>> Database
UIS Node
<<component>> UIS Web Service
Billing Node
<<component>> Billing
Outgoing Communication Node
<<component>> Outgoing Communication Component
Internet Portal Node
<<component>> Consumer Portal
<<component>> Configuration Controller
<<component>> Researcher Portal
Intranet Portal Node
<<component>> Operator Portal

Figure 4: ReMeS Deployment View: Main Diagram

2.3.2 Outgoing Communication Node

The Outgoing Communication Node is in charge of sending commands to the remote monitoring and control modules. The reason we have created a separate outgoing communication node is that certain outgoing communication must be handled very quickly (e.g., sending a shut down valve command in case of an alarm).

2.3.3 Data Processing Node

This node processes all incoming data by storing it in the database and running an anomaly detection process. Storing data in the measurements database (i.e., insert queries) is extremely fast. However, the anomaly detection process may required certain processing capacity. This is why the node can be easily replicated if the processing capacity is insufficient.

2.3.4 Alarm Processing Node

Although alarm processing does not involve any computationally costly processing, the alarms must be processed with maximum priority. This is why we have split the alarm processing from the normal data processing and deployed it on a separate machine.

2.3.5 Database Node

This node is central in ReMeS as the system will be paralyzed by any potential problems to this node. An evident solution to the problem from the architectural point of view is active replication with a load balancer (not shown on figure 4).

2.3.6 Internet Portal Node

The Internet Portal Node serves to ReMeS consumers via the world-wide web. Provided that the node is susceptible to DoS attacks its downtime may not affect the overall operation of ReMeS. This is why this node deploys only non-critical system components.

2.3.7 Intranet Portal Node

As opposed to the open Internet Portal Node, this node is more critical as it serves ReMeS employees and partners in the daily ReMeS operational activities. Downtimes should be minimized and access should be limited to users only within the organization (either physically or by the means of a virtual private network).

2.3.8 Prediction Node

Prediction Node hosts components that are not critical and run in batch mode and create statistics and forecasts of consumption based on the recent measurements.

2.3.9 UIS Node

The UIS Node hosts components that communicate with each UIS.

2.3.10 Billing Node

Finally, the Billing Node realizes the communication with a third party billing system.

2.4 Database decomposition

We have decided to decompose the database into two different databases. Figure 5 illustrates this decomposition in detail. The User Data database contains only information that identify the consumers. Measurement and Information Data database contains all the rest of the information.

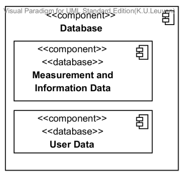

Visual Paradigm for UML Standard Edition(K.U.Leuven)
<<component>>
Database
<<component>>
<<database>>
Measurement and Information Data
<<component>>
<<database>>
User Data

Figure 5: ReMeS Database Decomposition

3 Behavioral views

UML: Sequence diagram

This section will illustrate the most important scenarios.

3.1 Trame processing

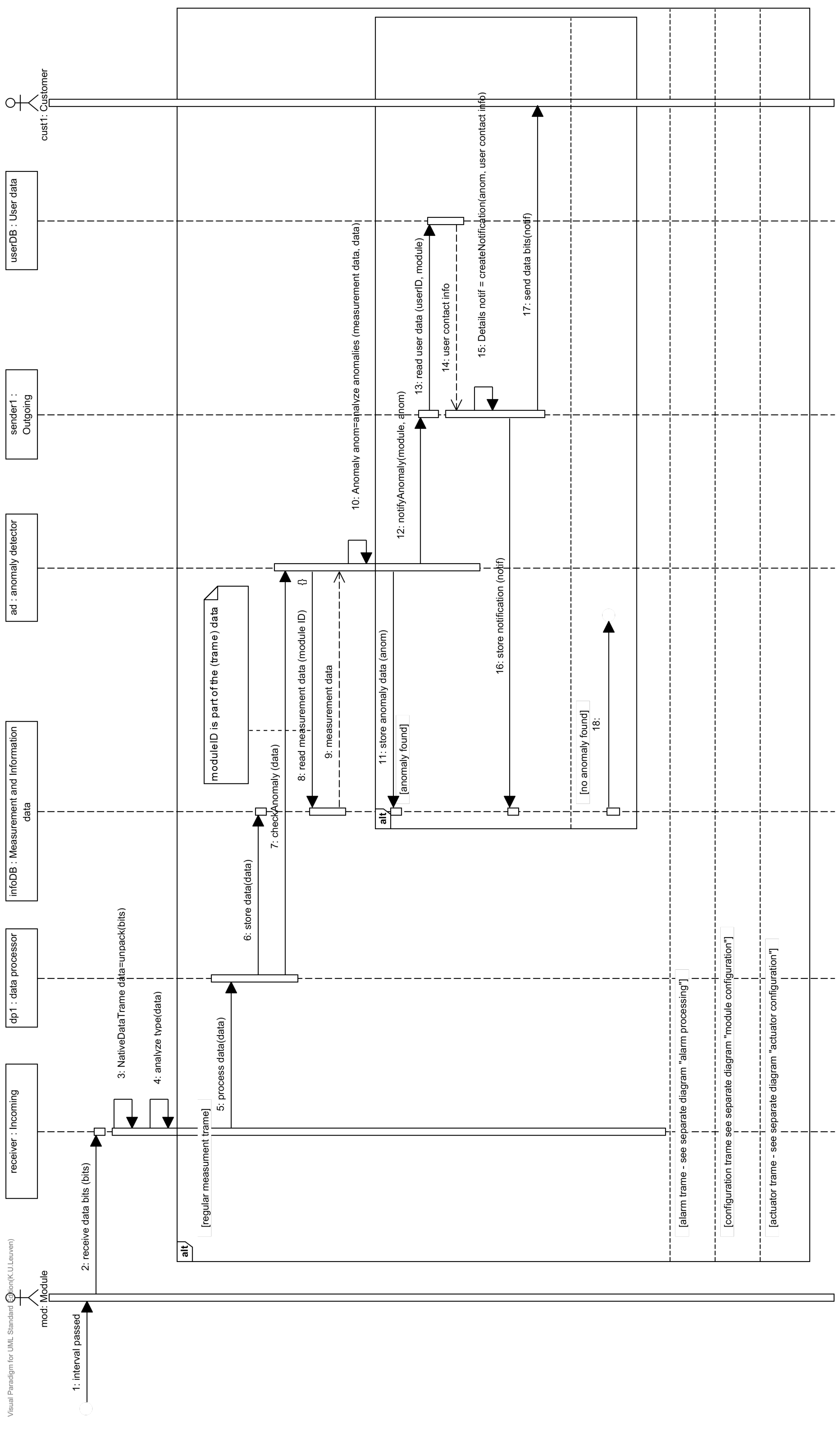
The general flow for processing incoming data trames is represented in Figure 6. This scenario shows that 4 types of trames can be received: 1) regular measurement data, which is processed and check for anomalies; 2) alarm trames, which, as shown in Figure 7, can trigger valve actuation and will notify the consumer; 3) configuration trames that confirm the (re)configuration of measurement trames, as illustrated in Figure 9; and 4) actuator trames, that confirm the actuation of the valve, as shown in Figure 8.

3.2 Consumer registration and portal use

Each new consumer can register himself using the consumer portal, as illustrated in Figure 10. When registered, he can contact a ReMeS operator which will add ReMeS measurement and actuator devices to the consumer's account. Once the consumer is registered, he can log in to the consumer portal using his registration data, as shown in Figure 11. A similar scenario exists for ReMes operators who log in to the operator portal, depicted in Figure 12. Once logged in, the consumer can request his measurement data, manage the contact information and conditions for the notifications sent to him (Figure 14), compare his own consumption and pricing with green alternatives, etc., however, each action is first authorized as illustrated in Figure 13. Although not depicted in the diagrams, the same authorization applies to the operator portal and research portal. An example scenario of the researcher portal is the researcher who requests a set of data, as shown in Figure 15 (although this diagram does not show authentication or authorization, it is implied). Note that both the authentication enforcer and authorization enforcer are not depicted in the client-server view, as they are considered part of each portal.

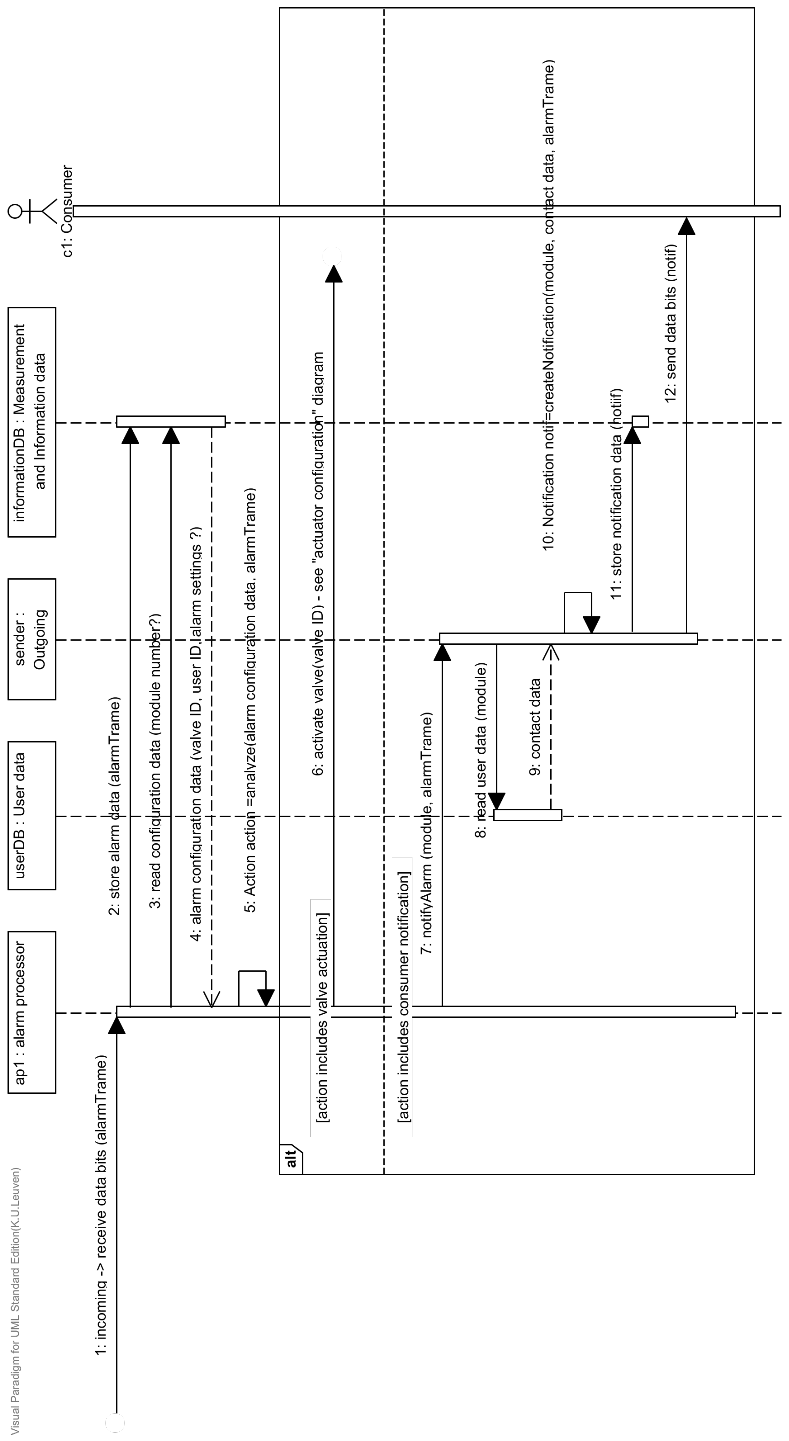
3.3 Utility Company interactions

The utility company can demand status updates at any time, as shown in Figure 16. Also, whenever the utility company decides to use a new price, this is immediately pushed to the ReMeS system as it is important for the price calculations. Also, some customers may have registered for price change notifications (e.g. washing machine should be turned on when the prices are low, fridge can be shut down for half an hour when prices are high). These notifications are sent out whenever a severe price change is observed by ReMeS as illustrated in Figure 17.



Visual Paradigm for UML Standard Edition(K.U.Leuven)
mod: Module
receiver : Incoming
dp1 : data processor
infoDB : Measurement and Information data
ad : anomaly detector
sender1 : Outgoing
userDB : User data
cust1: Customer
1: interval passed
2: receive data bits (bits)
3: NativeDataTrame data=unpack(bits)
4: analyze type(data)
5: process data(data)
6: store data(data)
7: checkAnomaly (data)
8: read measurement data (module ID)
9: measurement data
moduleID is part of the (trame) data
{}
10: Anomaly anom=analyze anomalies (measurement data, data)
11: store anomaly data (anom)
12: notifyAnomaly(module, anom)
13: read user data (userID, module)
14: user contact info
15: Details notif = createNotification(anom, user contact info)
16: store notification (notif)
17: send data bits(notif)
18:
alt
[regular measument trame]
[anomaly found]
[no anomaly found]
[alarm trame - see separate diagram "alarm processing"]
[configuration trame see separate diagram "module configuration"]
[actuator trame - see separate diagram "actuator configuration"]

Figure 6: ReMeS Processing Data scenario



Visual Paradigm for UML Standard Edition(K.U.Leuven)
ap1 : alarm processor
userDB : User data
sender : Outgoing
informationDB : Measurement and Information data
c1: Consumer
1: incoming -> receive data bits (alarmTrame)
2: store alarm data (alarmTrame)
3: read configuration data (module number?)
4: alarm configuration data (valve ID, user ID,alarm settings ?)
5: Action action =analyze(alarm configuration data, alarmTrame)
alt
[action includes valve actuation]
6: activate valve(valve ID) - see "actuator configuration" diagram
[action includes consumer notification]
7: notifyAlarm (module, alarmTrame)
8: read user data (module)
9: contact data
10: Notification notif=createNotification(module, contact data, alarmTrame)
11: store notification data (notif)
12: send data bits (notif)

Figure 7: ReMeS Processing Alarm scenario

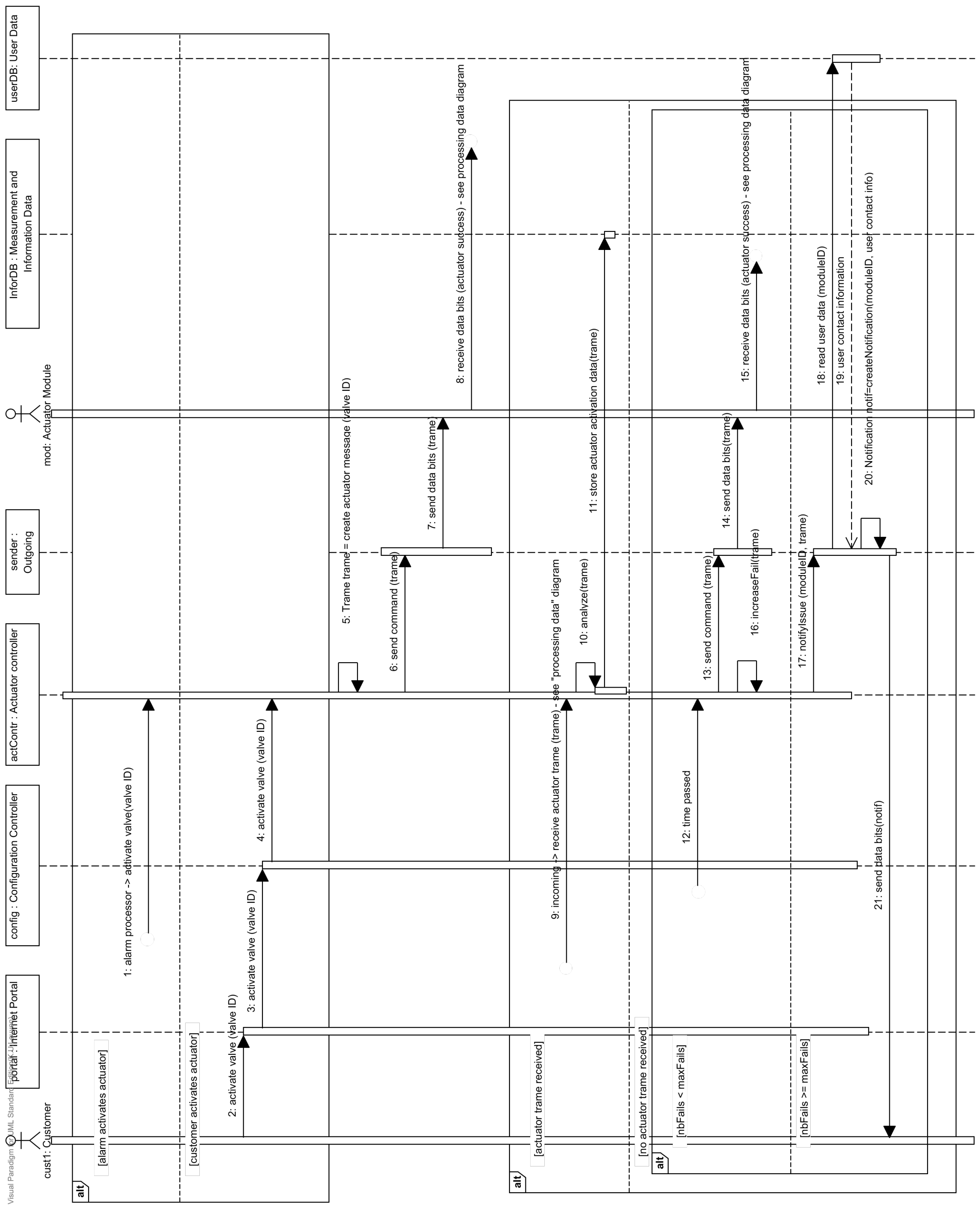

cust1: Customer
portal : Internet Portal
config : Configuration Controller
actContr : Actuator controller
sender : Outgoing
mod: Actuator Module
InforDB : Measurement and Information Data
userDB: User Data
alt
[alarm activates actuator]
[customer activates actuator]
1: alarm processor -> activate valve(valve ID)
2: activate valve (valve ID)
3: activate valve (valve ID)
4: activate valve (valve ID)
5: Trame trame = create actuator message (valve ID)
6: send command (trame)
7: send data bits (trame)
8: receive data bits (actuator success) - see processing data diagram
alt
[actuator trame received]
[no actuator trame received]
9: incoming -> receive actuator trame (trame) - see "processing data" diagram
10: analyze(trame)
11: store actuator activation data(trame)
12: time passed
13: send command (trame)
14: send data bits(trame)
15: receive data bits (actuator success) - see processing data diagram
16: increaseFail(trame)
alt
[nbFails < maxFails]
[nbFails >= maxFails]
17: notifyIssue (moduleID, trame)
18: read user data (moduleID)
19: user contact information
20: Notification notif=createNotification(moduleID, user contact info)
21: send data bits(notif)
Visual Paradigm for UML Standard Edition

Figure 8: ReMeS actuator control scenario

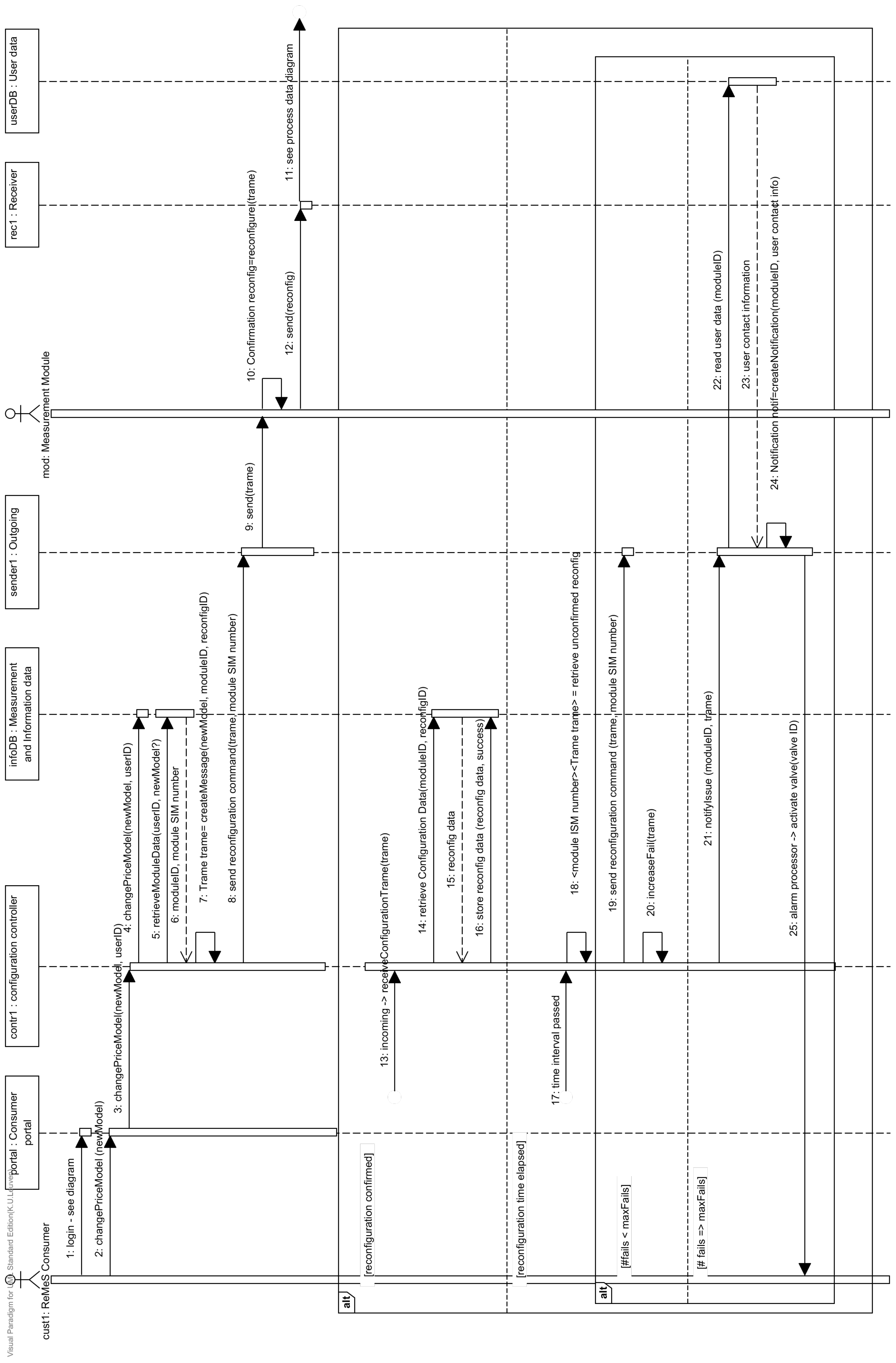

cust1: ReMeS Consumer
portal : Consumer portal
contr1 : configuration controller
infoDB : Measurement and Information data
sender1 : Outgoing
mod: Measurement Module
rec1 : Receiver
userDB : User data
1: login - see diagram
2: changePriceModel (newModel)
3: changePriceModel(newModel, userID)
4: changePriceModel(newModel, userID)
5: retrieveModuleData(userID, newModel?)
6: moduleID, module SIM number
7: Trame trame= createMessage(newModel, moduleID, reconfigID)
8: send reconfiguration command(trame, module SIM number)
9: send(trame)
10: Confirmation reconfig=reconfigure(trame)
11: see process data diagram
12: send(reconfig)
13: incoming -> receiveConfigurationTrame(trame)
14: retrieve Configuration Data(moduleID, reconfigID)
15: reconfig data
16: store reconfig data (reconfig data, success)
17: time interval passed
18: <module ISM number><Trame trame> = retrieve unconfirmed reconfig
19: send reconfiguration command (trame, module SIM number)
20: increaseFail(trame)
21: notifyIssue (moduleID, trame)
22: read user data (moduleID)
23: user contact information
24: Notification notif=createNotification(moduleID, user contact info)
25: alarm processor -> activate valve(valve ID)
alt
[reconfiguration confirmed]
[reconfiguration time elapsed]
alt
[#fails < maxFails]
[# fails => maxFails]
Visual Paradigm for UML Standard Edition(K.U.Leuven)

Figure 9: ReMeS module configuration scenario

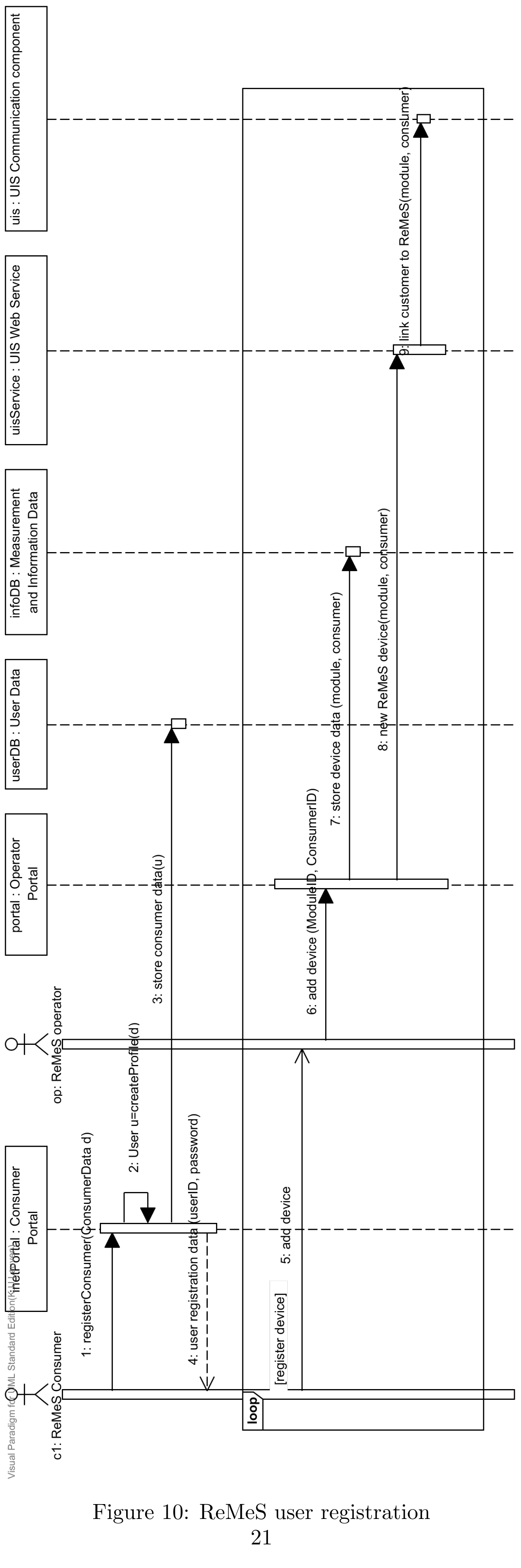

c1: ReMeS Consumer
mePortal : Consumer Portal
op: ReMeS operator
portal : Operator Portal
userDB : User Data
infoDB : Measurement and Information Data
uisService : UIS Web Service
uis : UIS Communication component
1: registerConsumer(ConsumerData d)
2: User u=createProfile(d)
3: store consumer data(u)
4: user registration data (userID, password)
loop
[register device]
5: add device
6: add device (ModuleID, ConsumerID)
7: store device data (module, consumer)
8: new ReMeS device(module, consumer)
9: link customer to ReMeS(module, consumer)
Visual Paradigm for UML Standard Edition(K.U.Leuven)

Figure 10: ReMeS user registration

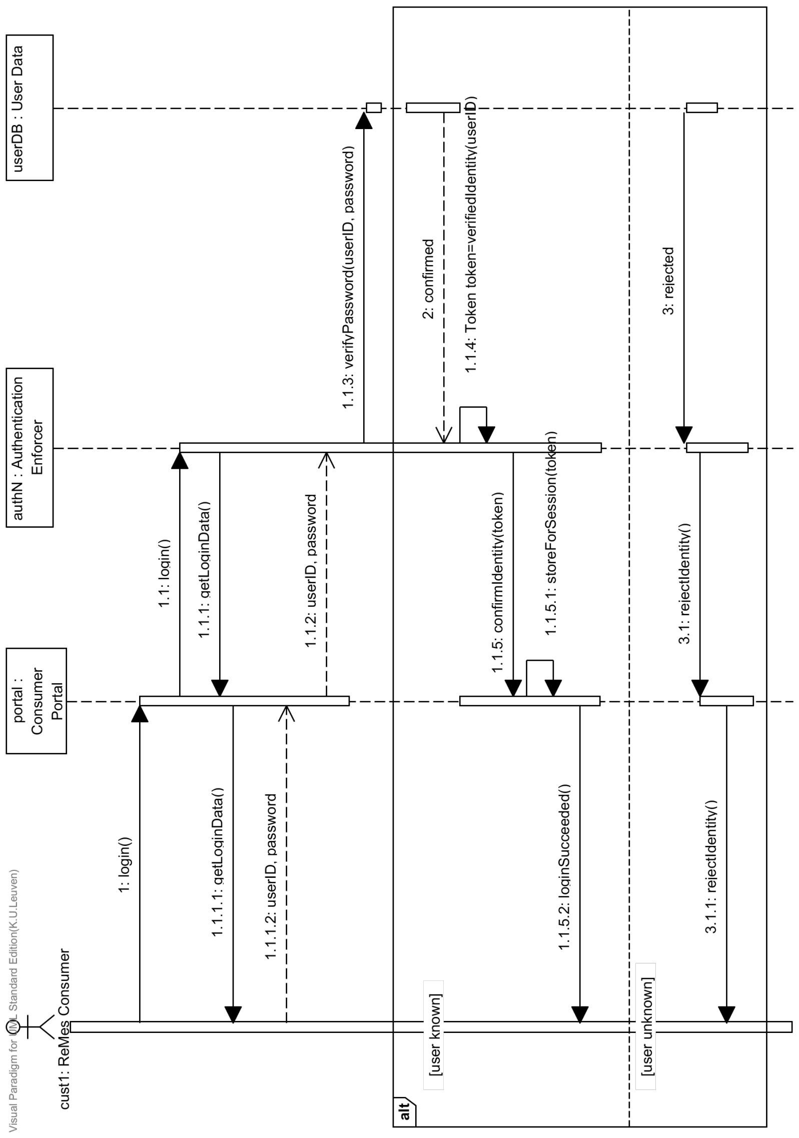

Visual Paradigm for UML Standard Edition(K.U.Leuven)
cust1: ReMeS Consumer
portal : Consumer Portal
authN : Authentication Enforcer
userDB : User Data
1: login()
1.1: login()
1.1.1: getLoginData()
1.1.1.1: getLoginData()
1.1.1.2: userID, password
1.1.2: userID, password
1.1.3: verifyPassword(userID, password)
2: confirmed
1.1.4: Token token=verifiedIdentity(userID)
1.1.5: confirmIdentity(token)
1.1.5.1: storeForSession(token)
1.1.5.2: loginSucceeded()
3: rejected
3.1: rejectIdentity()
3.1.1: rejectIdentity()
alt
[user known]
[user unknown]

Figure 11: ReMeS consumer login - authentication

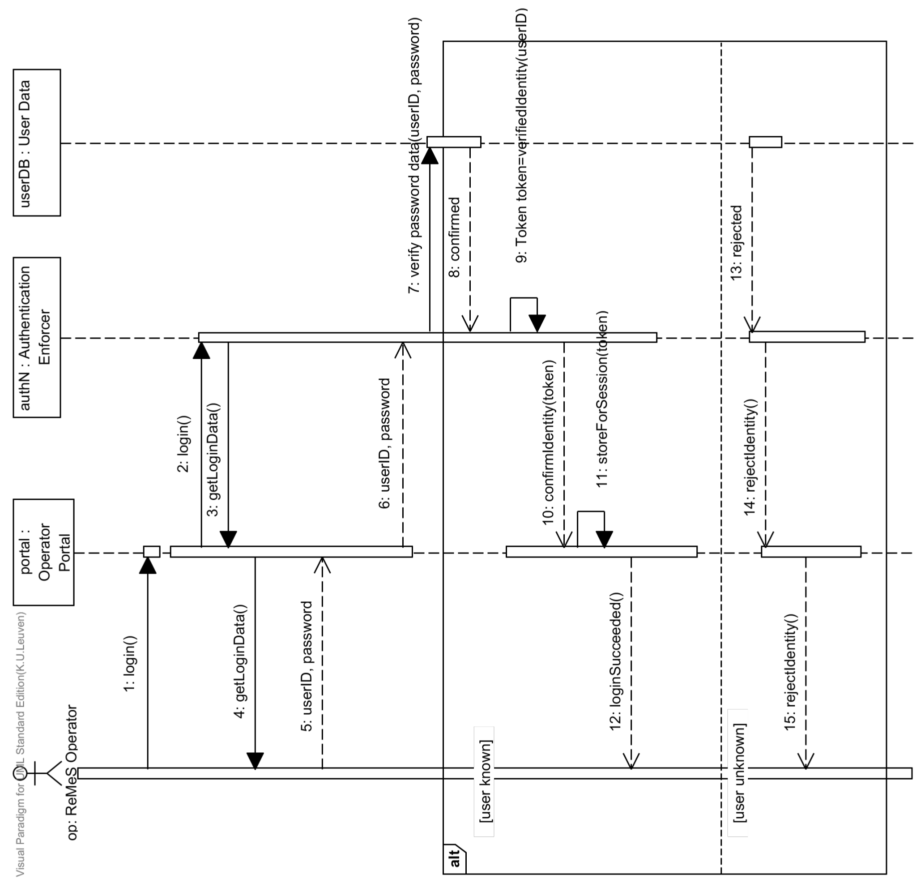


Figure 12: ReMeS employee login - authentication

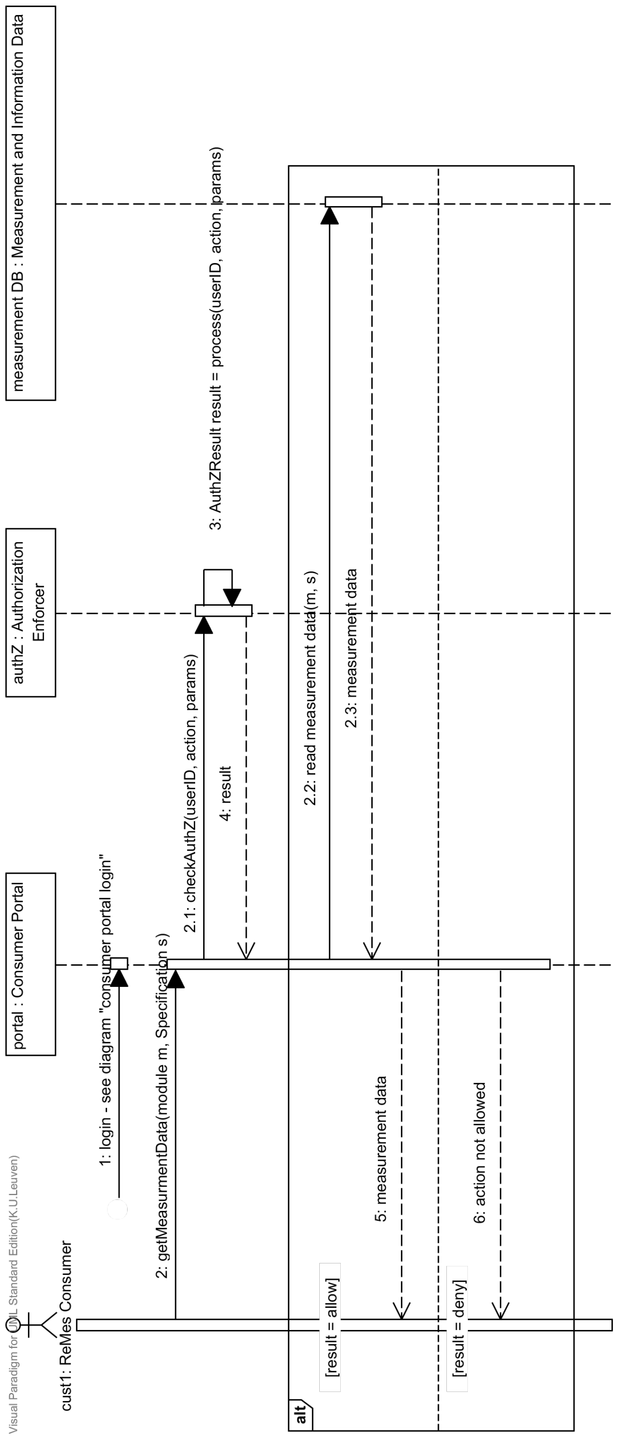

Visual Paradigm for UML Standard Edition(K.U.Leuven)
cust1: ReMes Consumer
portal : Consumer Portal
authZ : Authorization Enforcer
measurement DB : Measurement and Information Data
1: login - see diagram "consumer portal login"
2: getMeasurmentData(module m, Specification s)
2.1: checkAuthZ(userID, action, params)
3: AuthZResult result = process(userID, action, params)
4: result
2.2: read measurement data(m, s)
2.3: measurement data
5: measurement data
6: action not allowed
alt
[result = allow]
[result = deny]

Figure 13: ReMeS portal use - authorization

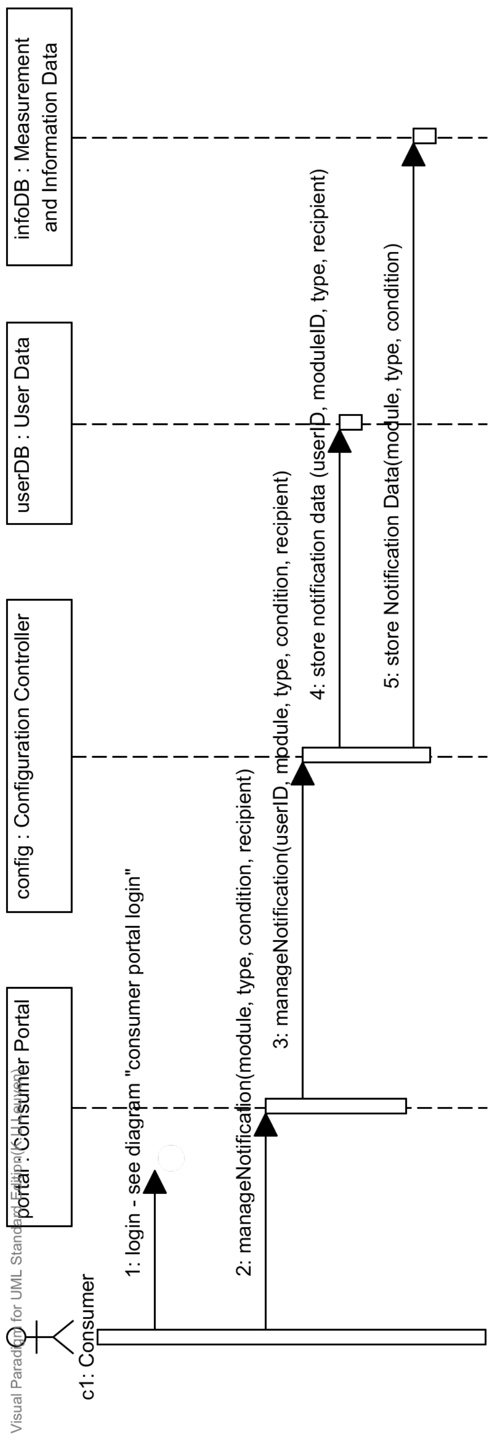

c1: Consumer
portal : Consumer Portal
config : Configuration Controller
userDB : User Data
infoDB : Measurement and Information Data
1: login - see diagram "consumer portal login"
2: manageNotification(module, type, condition, recipient)
3: manageNotification(userID, module, type, condition, recipient)
4: store notification data (userID, moduleID, type, recipient)
5: store Notification Data(module, type, condition)
Visual Paradigm for UML Standard Edition(K.U.Leuven)

Figure 14: ReMeS notification management scenario

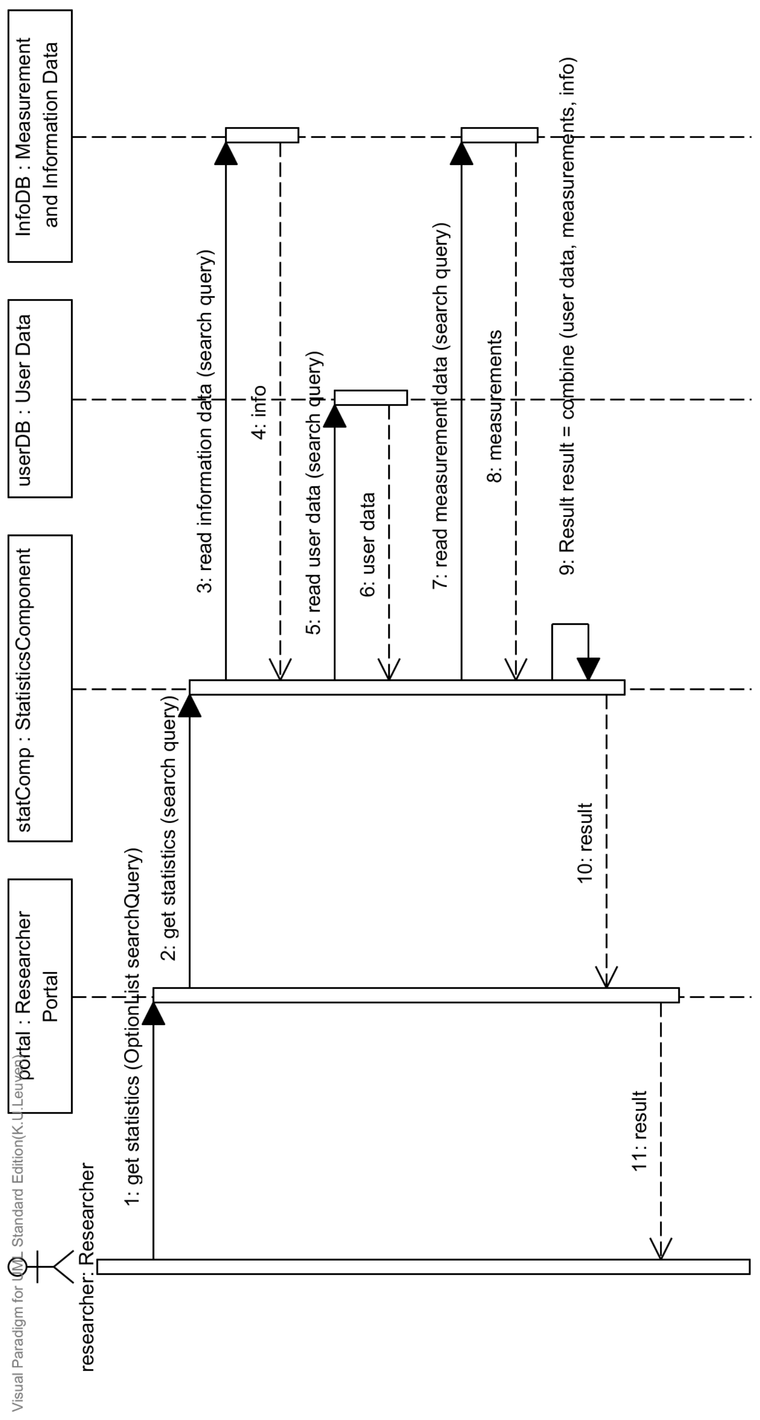

researcher: Researcher
portal : Researcher Portal
statComp : StatisticsComponent
userDB : User Data
InfoDB : Measurement and Information Data
1: get statistics (OptionList searchQuery)
2: get statistics (search query)
3: read information data (search query)
4: info
5: read user data (search query)
6: user data
7: read measurement data (search query)
8: measurements
9: Result result = combine (user data, measurements, info)
10: result
11: result
Visual Paradigm for UML Standard Edition(K.U.Leuven)

Figure 15: ReMeS statistics retrieval scenario

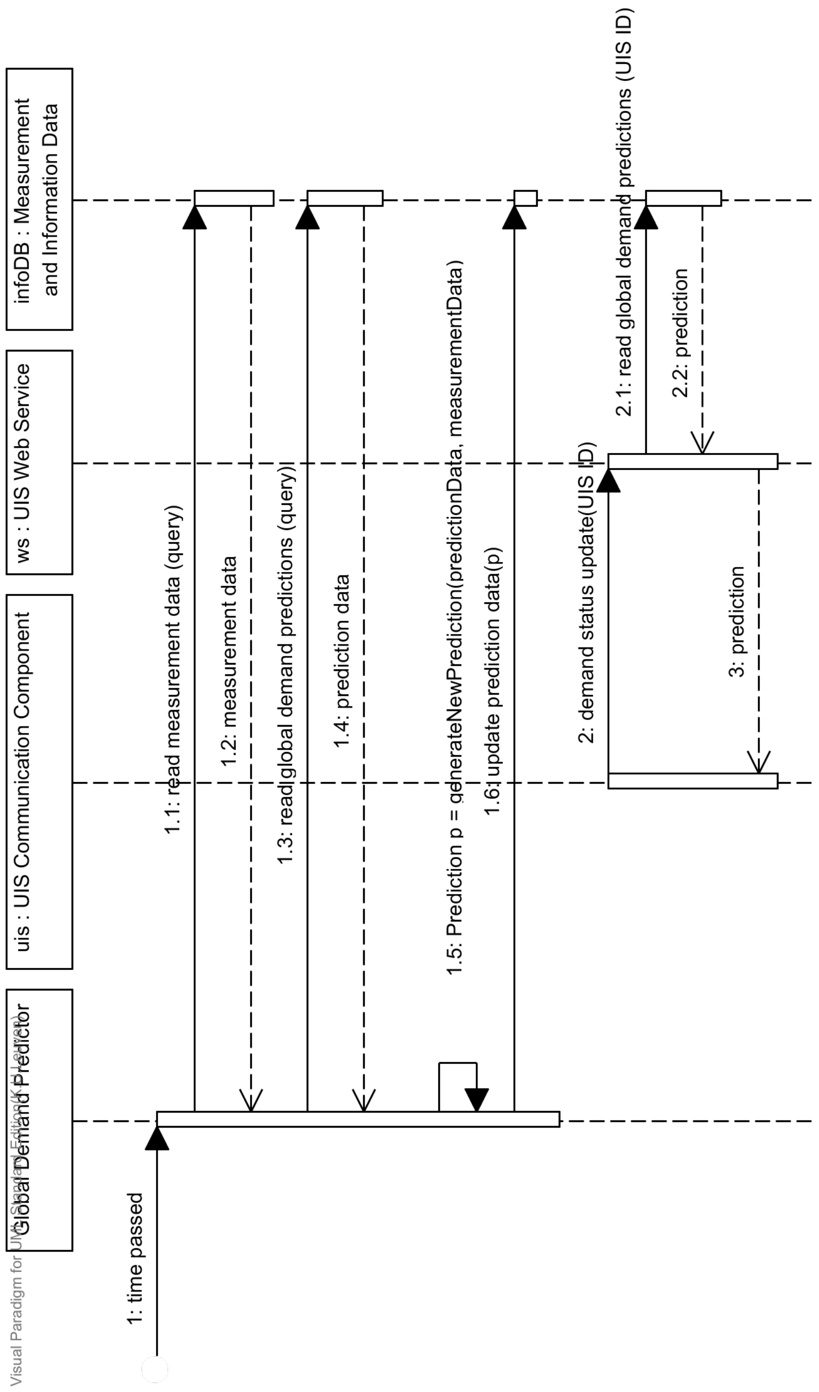

Global Demand Predictor
uis : UIS Communication Component
ws : UIS Web Service
infoDB : Measurement and Information Data
1: time passed
1.1: read measurement data (query)
1.2: measurement data
1.3: read global demand predictions (query)
1.4: prediction data
1.5: Prediction p = generateNewPrediction(predictionData, measurementData)
1.6: update prediction data(p)
2: demand status update(UIS ID)
2.1: read global demand predictions (UIS ID)
2.2: prediction
3: prediction
Visual Paradigm for UML Standard Edition(K.U.Leuven)

Figure 16: ReMeS utility company demands status update

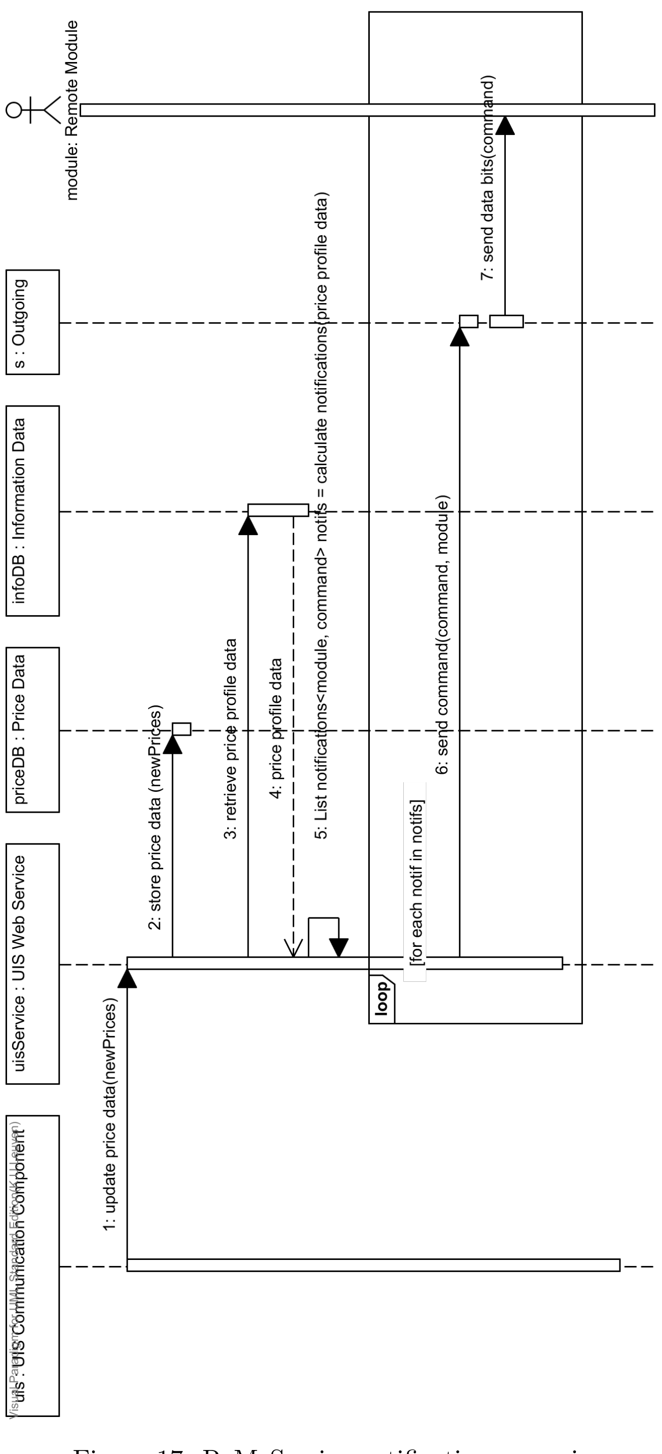

uis : UIS Communication Component
uisService : UIS Web Service
priceDB : Price Data
infoDB : Information Data
s : Outgoing
module: Remote Module
1: update price data(newPrices)
2: store price data (newPrices)
3: retrieve price profile data
4: price profile data
5: List notifications<module, command> notifs = calculate notifications(price profile data)
loop
[for each notif in notifs]
6: send command(command, module)
7: send data bits(command)

Figure 17: ReMeS price notification scenario