Table of contents

1 Abstract 2

2 Introduction 3

2.1 Purpose 3

2.2 Scope 3

2.3 Terms and abbreviations 3

2.4 References 3

3 System overview 4

3.1 System context 4

3.2 Actors 7

3.3 Reference architecture 8

3.4 USEF layered views 9

3.5 USEF Component model 12

4 Diagrams 27

4.1 Common inbound message flow 27

4.2 Hash check flow 28

4.3 Message id check flow 29

4.4 Common inbound message routing flow 30

4.5 Common outbound message flow 31

4.6 Resolve participant flow 31

4.7 File logger flow 32

4.8 Pluggable Business Component invocation flow 33

4.9 Common process flow 34

5 Process coordinator mapping 35

5.1 Triggering 35

5.2 Plan phase 37

5.3 Validate phase 48

5.4 Operate phase 52

5.5 Settlement phase 56

6 Inter actor message flows 62

6.1 Message exchange mechanism 62

6.2 Inter actor messages 62

7 Data store schemas 66

7.1 Message store 66

7.2 Hash store 67

7.3 Common reference 68

7.4 Planboard 80

7.5 AGR portfolio 86

# Abstract

In this document the architectural structure is defined for the software systems that are to be built for the USEF Reference Implementation, based on the USEF specifications [1].

The following is covered in this the document:

* An architectural system overview of USEF showing a common layered approach for each of the individual roles.
* Architectural component models for the major roles in the [1].
* Sequence diagrams demonstrating the most important flows.
* High level modelling of the data stores.

Graphical visualization aspects of the reference implementation are considered to be out of scope and are part of another project.

The steps to deploy and configure the reference implementation are not part of this document. These are covered by the referenced document [2].

Implementation guidelines are part of the referenced document [3].

# Introduction

## Purpose

This document describes the system and software architecture which is applied by the USEF Build team for the reference implementation. This document describes the partitioning of the system and the software and identifies the essential components.

## Scope

This document describes the overall system and software architecture for the different roles with respect to the Use Cases, as specified in [1].

## Terms and abbreviations

Definitions of USEF specific terms and abbreviations can be found the glossary in [1].

Additional terms and abbreviations used in this document:

|  |  |
| --- | --- |
| PBC | Pluggable Business Component |

## References

|  |  |
| --- | --- |
| **Reference** | **Description** |
| [1] | USEF Specifications 2015 |
| [2] | USEF The Framework Implemented – Installation Manual |
| [3] | USEF The Framework Implemented – Implementation Guidelines |
| [4] | USEF The Framework Explained |
| [5] | USEF The Privacy and Security Guideline |

# System overview

## System context

The system context shows a representation of the USEF framework in one view, and identifies the information and control flows that cross via the framework from and to actors (human and system). Its purpose is not to clarify any detail on how the system itself is designed, created, deployed etc.

The actors making use of the USEF framework listed in this chapter, are briefly described. A more elaborated description and an overview of all roles in a smart energy market can be found in chapter 4 of [4]. More information on the common reference can be found in chapter 2.4.1 of [1].



1. System context

Next table lists:

* the interactions between the identified actors via the USEF framework.
* the input towards USEF processes via pluggable business components (marked with (PBC)).

|  |  |
| --- | --- |
| **Actor** | **Interaction** |
| **A\_AGR** |  |
|  | Send AGR Common Reference Update  Query Common Reference  Send D-prognoses (new and updated)  Receive Flex Requests  Send Flex Offers  Receive Flex Orders.  Revoke Flex Offers  Receive Settlement Message  Send Settlement Message Response  Send A-plan (new and updated)  Create Elements (PBC)  Create Profile (PBC)  Create UDI (PBC)  Collect Forecast (PBC)  Non-Udi Collect Forecast (PBC)  Optimize AGR Portfolio (PBC)  Non-UDI Optimize AGR Portfolio (PBC)  New Prognoses Required (PBC)  Create Flex Offers (PBC)  Initialize Non-UDI Clusters (PBC)  Retrieve ADS Goals (PBC)  Determine Net Demands via ADS (PBC)  Identify Change In Forecast (PBC)  Detect Prognoses Deviations (PBC)  Non-UDI Detect Prognoses Deviations (PBC)  Send ADS Messages (PBC)  Send ADS Goals (PBC)  Initiate Settlement Items (PBC)  Validate Settlement Items (PBC) |
| **A\_BRP** |  |
|  | Send BRP Common Reference Update  Query Common Reference  Receive A-plan  Send Flex Requests  Receive Flex Offers  Send Flex Orders  Receive Flex Offer revocation  MDC Query  Send Settlement Message  Receive Settlement Message Response  Received A-plan (PBC)  Prepare Flex Requests (PBC)  Get Not Desirable Flex Offers (PBC)  Create Flex Orders (PBC)  Create Missing A-Plans (PBC)  Initiate Settlement (PBC)  Calculate penalty amount (PBC) |
| **A\_CRO** |  |
|  | Receive DSO updates  Receive DSO query  Receive AGR updates  Receive AGR query  Receive BRP updates  Receive BRP query  Receive MDC query |
| **A\_DSO** |  |
|  | Send DSO Common Reference Update  Query Common Reference  Receive D-prognoses (new and updated)  Send Flex Requests  Receive Flex Offers  Send Flex Orders  Receive Flex Offer revocation  Send Settlement Message  Receive Settlement Message Response  MDC Query  Create Non Aggregator Forecast (PBC)  Create Missing D-Prognoses (PBC)  Perform Grid Safety Analysis (PBC)  Post Coloring (PBC)  Prepare Stepwise Limiting (PBC)  Create Flex Requests (PBC)  Create Flex Orders (PBC)  Monitor Grid (PBC)  Place Operate Flex Order (PBC)  Limit Connections (PBC)  Restore Connections (PBC)  Initiate Settlement (PBC)  Calculate Penalty Amount (PBC)  Generate Connection Meter Events (PBC)  Determine Outage Durations (PBC)  Determine Reduction Periods (PBC)  Calculate Compensations (PBC) |
| A\_MDC |  |
|  | Query Common Reference  Receive MDC Query  Meter Data Query (PBC) |

## Actors

### A\_AGR

The role of Aggregator consists of accumulating flexibility from Prosumers and their Active Demand & Supply, and selling this to the BRP and/or the DSO. It is the goal of the Aggregator to maximize the value of flexibility, taking into account both the customer needs, economical optimization and grid capacity. USEF allows Prosumers to directly access the flexibility market, but in that case they implicitly act as the Aggregator of their own portfolio. USEF is defined in such a way that accumulation of multiple Aggregators into a larger Aggregator is possible.

Two types of Aggregators are identified in USEF.

* **UDI Aggregators**: Aggregators using UDI to communicate with Active Demand & Supply endpoints to realize the portfolio optimization. By doing so, this type of Aggregator can steer consumption and/or production on the connections it represents via the ADS equipment
* **Non-UDI Aggregators**: Aggregators that do not use UDI to realize their portfolio optimization, but that use a fully external demand response solution, a so-called "aggregator-in-a-box" product.

The Aggregator type is configurable for each Aggregator in the environment.

### A\_BRP

A Balance Responsible Party (BRP) is responsible for actively balancing supply and demand for its portfolio of Producers, Aggregators and Prosumers in the most economical way. The BRP forecasts the energy Demand & Supply of its portfolio and seeks the most economical solution for the requested energy to be supplied. The BRP can source the requested energy on behalf of the Supplier in two ways: directly by dispatching power plants with which it has a contractual agreement or indirectly via trading on the various energy markets. Additional value can be created by supporting the TSO in maintaining the system balance, e.g. by trading on the imbalance market. USEF provides an additional possibility to the BRP to optimize its portfolio: activating the flexibility in Active Demand & Supply that Prosumers offer through Aggregators.

### A\_DSO

The role of Distribution Network Operator (DNO) is superseded by the role of the Distribution System Operator (DSO) in the USEF model. The difference between DSO and DNO comes from the possibility of the DSO to perform grid capacity management. Note that in the USEF model, the DSO will not perform frequency control in the distribution grid. The DSO is responsible for the cost-effective distribution of energy in a given area to and from end-users over the distribution grid and the connections to and from the transmission grid. The DSO ensures the long term ability of the distribution system to meet the demands for the distribution of energy. The introduction of the grid capacity management regime enables the DSO to minimize grid capacity costs while simultaneously safeguarding security of supply.

### A\_CRO

A CRO, the common reference, is defined as a system that holds information to relate congestion points to connections and to maintain an administration that lists the AGR and DSO for each connection.

DSOs consult the common reference to determine which AGRs will be sending D-prognoses. DSOs report congestion points and the connections that are impacted by that congestion point to the common reference.

AGRs consult the common reference to determine which of his connections are impacted by a congestion point. AGRs report which connections they manage.

### A\_MDC

The Meter Data Company (MDC) is responsible for acquiring and validating meter data. The MDC acquires smart meter data through the smart meter infrastructure, which is (in general) operated by the DSO.

Each month the MDC determines the monthly energy volumes per Prosumers, based on smart meter data. This is an existing process, with the exception that the frequency may differ from country to country (e.g. once every two months or annually).

For a proper settlement process, completeness of the monthly volumes is required. As the completeness of the smart meter data is not guaranteed, the MDC needs to determine the monthly volume even in the case when no smart meter data is available.

## Reference architecture

When going through the USEF specifications several characteristics can be identified to which a USEF framework implementation must adhere to. Some of them are listed beneath.

* It must be designed with high availability and redundancy in mind (see chapter 1.10 of [5]).
* It must be designed to adhere to the privacy and security principles as described in chapter 1 of [5].
* It must support B2B interactions between the different actors (see chapter 2.3 of [1]).
* It must support event based triggering (see chapter 2.3 of [1]).
* It must support workflow processes (see chapter 2.3 of [1]).
* It must support the composition of USEF simulation scenario’s using different implementations of a Pluggable Business Component (see chapter 4.3 of [3]).

These characteristics (and others like e.g. versioning) correspond very well with an event driven SOA architecture. An event-driven SOA is a form of service oriented architecture (SOA), combining the intelligence and pro-activeness of event-driven architecture with the organizational capabilities found in (sometimes called) SOA 1.0 offerings.

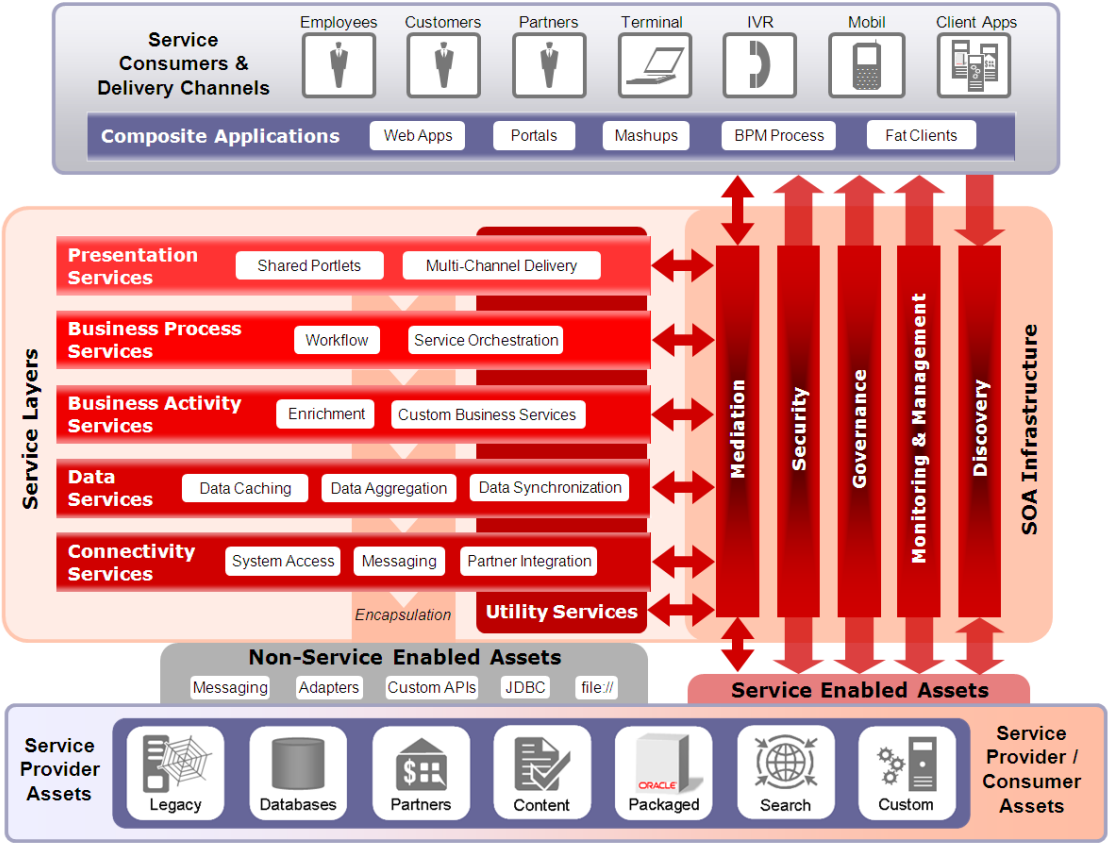
The following architectural diagram shows an abstraction (taken from IBM) of how to construct a Service-oriented Architecture (SOA) as a set of logical layers. In this layered architecture, 3 types can be distinguished:

* Consumer layers (consumer, business process and service layer): These offer services that can be consumed by external parties. B2C and B2B actors can hook in on business processes or invoke business reusable services.
* Provider layers (service, service component, operational systems layer): These support the consumer layers. Note that the service layer is also mentioned here as this will expose the services offered by the service component layer. As the services are loosely coupled with the service components, service components can be replaced with other service components without impacting the overall design of the solution.
* Non-functional layers (integration, quality of service, data architecture and governance layer): These cut across both the consumer layers and the provider layers and will enable the support of the privacy & security principles, integration specifications etc.



1. IBM SOA reference architecture

Next image shows another view on a logical model taken from Oracle. This is added merely to illustrate that the required logical elements for SOA architecture are very similar over different vendors and are widely accepted.



1. Oracle SOA reference architecture

## USEF layered views

### USEF layered view – level 1

The goal is to deliver a reference implementation of USEF 2015 and a working simulation with interchangeable business logic. Besides this, a layered flexibility should be introduced to allow USEF implementer candidates to pick that part of the delivered code fit for purpose. A USEF implementer candidate can choose to:

* Do everything from scratch. In this case no reuse of the delivered code is wanted.
* Use a core ’service’ layer. This layer provides a USEF compliant messaging framework, USEF message validation, encrypted databases and data models, logging functionality, code common to all roles, a set of business and workflow APIs. This layer is role independent.
* Use the core ’service’ layer and a ’workflow’ layer. This ’workflow’ layer provides an implementation of the USEF processes and the business services specified by USEF. Reusing this layer will result in a USEF compliant implementation of the different roles. By not using the delivered ‘Pluggable Business Component’ layer, non USEF specified business components will have to be developed by the USEF implementer. In this the case the implementer will plug in own business logic and can opt to connect to its own backend systems.
* Use the core ’service’ layer, the ’workflow’ layer and the ‘Pluggable Business Component’ layer. The ’Pluggable Business Component’ layer offers a set of implemented non USEF specified business services to create a USEF simulation. As already indicated in the previous bullet point, this layer will also contain the USEF implemented Pluggable Business Components.

As it must be possible for a USEF implementer to choose between the above mentioned scenarios, this layered view will drive the project structuring of the reference implementation.

Following figure illustrates the layered view for a USEF implementer. The workflow layer depends on the service layer and uses the Pluggable Business Component layer.



1. USEF layered view – level 1

### USEF layered view – level 2

This view, goes one level deeper in detail by combining the level 1 view layers with the layers of the reference architecture. The structure of level 1 (bold) and level 2 layers is listed and shown below.



1. USEF layered view – level 2

* Simulator layer: This layer contains components to support the work of the person who prepares and sets up a USEF simulation run.
* **Pluggable Business Component layer**: See level 1 description.
* **Workflow layer**
  + Business process layer: This layer covers the process representation, composition methods, and building blocks for aggregating loosely coupled services as a sequencing process aligned with USEF processes. A process can invoke Pluggable Business Components and components of the business service layer.
  + Business service layer: This layer offers all business components defined by the USEF specifications. These will be invoked by the USEF processes contained in the business process layer.
* **Service layer**
  + Information layer: This layer contains all operational data stores required to realize the application components.
  + Integration layer: This layer enables the integration of services through the introduction of a reliable set of communication capabilities.
  + QoS layer: The Quality of Service layer provides capabilities like: capture (in an operational sense), monitor, log, signal non-compliance.
* Security layer: The security layer will offer capabilities that support the compliancy to the privacy and security principles. This layer is presented as a layer around the workflow and service layer to clearly illustrate that the USEF reference implementation advocates and employs privacy and security by design.

### USEF distributed view

While the USEF layered views illustrate the layering to apply within the USEF reference architecture common to each actor, the distributed view gives another insight (on the same architecture) from a distributed point of view.

As each role will be implemented by an organization fulfilling a role within its’ own organization and service, the architecture must be created in such a way that each set of role specific components can be realized and deployed independently of other role components. An organization taking up the role as e.g. an Aggregator should not be bothered with non-Aggregator components.

* Multiple organizations can take up the same role and/or can take up multiple roles in the smart grid. It must, as a consequence, be possible to identify and contact a USEF participant for a specific role in a unique way.

An organization can take up the same role multiple times for different legal entities. As a consequence, it must also be possible to include several instances of the same role (implemented in different ways) in the smart grid.

This is shown in the next figure.



1. USEF distributed view

## USEF Component model

Next section will describe the logical components (name starting with LC\_) added to the different layers.

The component model describes the structure of a system in terms of its software components with their responsibilities, interfaces, relationships, and the way they collaborate to deliver the required functionality. The component model is the main artefact documenting the functional view of the architecture and serves as an abstraction of the design.

The component model is used to:

* Describe the high-level structure of the system.
* Describe the responsibilities, relationships, and interactions of components.
* Document how application/technical parts of the system are related.
* Specify how existing, acquired, and developed components are related.

These last 2 points are illustrated by a mapping with the physical components (name starting with PC\_) that will realize the logical component. The aim of this mapping is not to go into micro design detail but to link to those physical components to start digging into the micro design. The more detailed interface information will be delivered via Javadoc.

Being a separate component in the logical component model does not implicate by default that the realisation of this component is strictly separated of the realization of all the other components. E.g. when opting for a COTS it is most likely that several component are or will be realized within this COTS.

### Simulator layer

To set up demonstrations and simulations of the USEF framework, someone will have to define and configure scenarios to do so.

While creating these scenarios, this actor will e.g. select what implementation of Pluggable Business Components is to be used and will have to prepare the datasets required to support the complete simulation scenario.

#### Components

LC\_Message injector

A component enabling the injection of USEF messages into the USEF solution. This will allow to send a test message, and receive an automated response indicating that the test message was successfully received.

LC\_Database populator

Before starting a USEF simulation or test, the databases of the different involved actors must be pre-filled with relevant and consistent data to support the simulation or test.

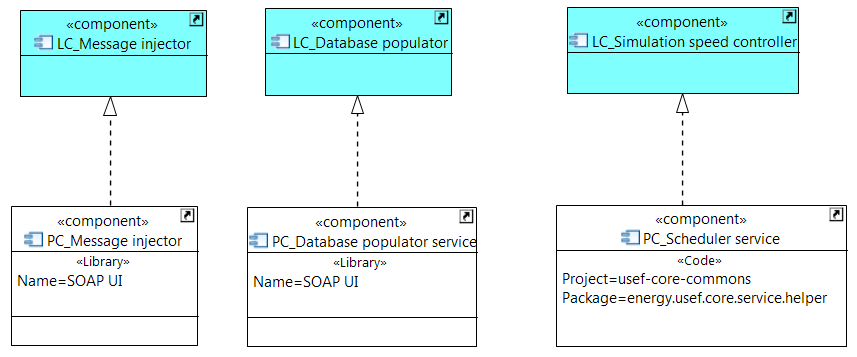
This component enables to inject data in database tables for testing and demonstration purposes. It provides a common mechanism that can be used to populate any table in a database. How to use the database populator can be found in [2].

LC\_Simulation speed controller

In order to run simulations that span a longer time period, a speed up mechanism is required to prevent waiting this complete period to see the results. This component allows to speed up USEF execution by influencing the event triggering timing while still using the normal PTU value in the messages and data.

The speed up factor is based on configuration and can be set before each start of a simulation. More information on this configuration can be found in [2].

#### Component diagram



1. Simulator layer component model

### USEF Business process layer

#### Introduction

The business processes as described in section 2.3 of [1] can be divided in one or more synchronous sub processes.

Each sub process is triggered by one or more of the following types of event:

* A time based event (e.g. intra-day closure).
* A state based event (e.g. all D-prognoses received).
* An event outcome of another sub process (e.g. flexibility orders triggering a re-optimization of an Aggregator portfolio).
* An event that was triggered manually or by an external system (e.g. because of changes in the weather the forecast needs to be re-calculated). This type of events are triggered by calling REST services (endpoints).

A sub process ends when:

* It is specified in the [1] processes.
* The next step in the business process cannot be triggered directly by the last sub process step.

Note that an inbound message (e.g. receive D-prognoses) can also be looked at as an event triggering business logic. These are already managed by the LC\_Message router component of the integration layer that will deliver it to the correct business service. The processing of such a message is a straight forward task within the already known context of the received message: Validate the message business wise, create a message with the validation result and if valid store it on the planboard.

A sequence diagram on the overall logical process flow can be found in the Diagrams chapter of this document.

#### Components

LC\_Process coordinator

This component will coordinate and invoke the proper next step of the sub flow. Pluggable Business Components will be identified based on configuration. This allows for including different Pluggable Business Components realizations without recompiling the solution. A process step can only be invoked via a coordinator.

The naming convention for process coordinators configuration mapping is: Role + Name sub flow + ‘Coordinator’. E.g. DSOGridSafetyAnalysisCoordinator

The chapter Process coordinator mapping gives a view on the sub processes realized in the reference implementation in support of the USEF processes.

LC\_Time based trigger scheduler

This component is a dedicated component to create and schedule all the time based triggers for the sub flow. It uses the LC\_Simulation speed controller to register the triggers. This way trigger logic is separated from business logic so that e.g. different triggers can invoke the same business process.

LC\_Event controller

This components separates control logic from business logic and allows different events to trigger the same sub process. This component will control whenever an event occurred what to do with this event.

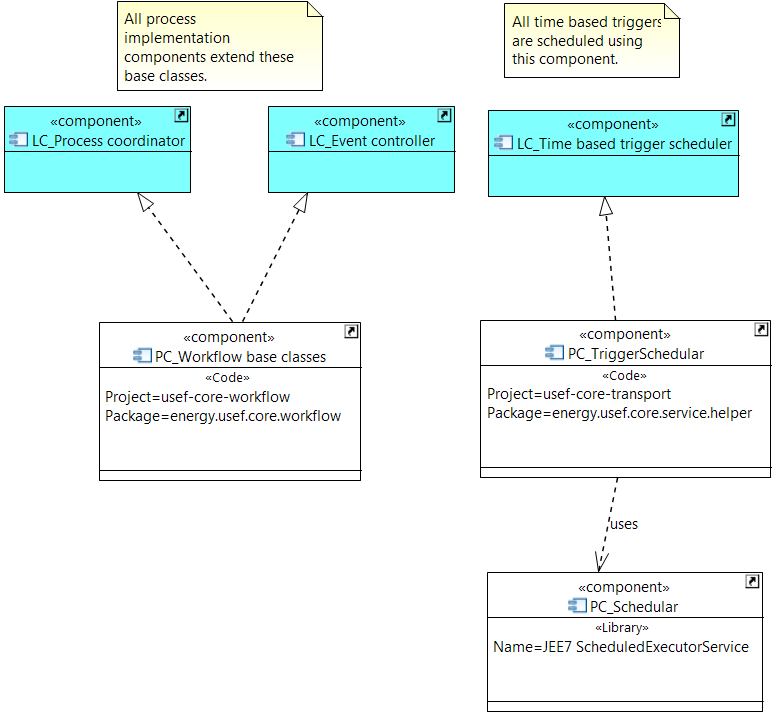
- Decide on the process component to invoke.

- Decide on the operation to invoke of the process component.

- Add a context to the event for the operation invocation.

- Perform state based related logic.

#### Component diagram



1. Business process layer component model

### USEF Business service layer

#### Introduction

A distinction can be made between four kinds of steps in the business processes.

* Steps fully described by the USEF specifications on how to realize them. These will be realized in the reference implementation and are part of this layer, the business service layer.
* Steps partially or not described by the USEF specifications on how to realize them. These will be realized by Pluggable Business Components and are part of the Pluggable Business Components layer.
* Steps realized by an already existing component (blue rectangles in the USEF flows). These will be realized by Pluggable Business Components and are part of the Pluggable Business Components layer.
* A step representing the decision logic to start one of the three types of steps above. These will be realized in the reference implementation and are part of the business process layer.

#### Components

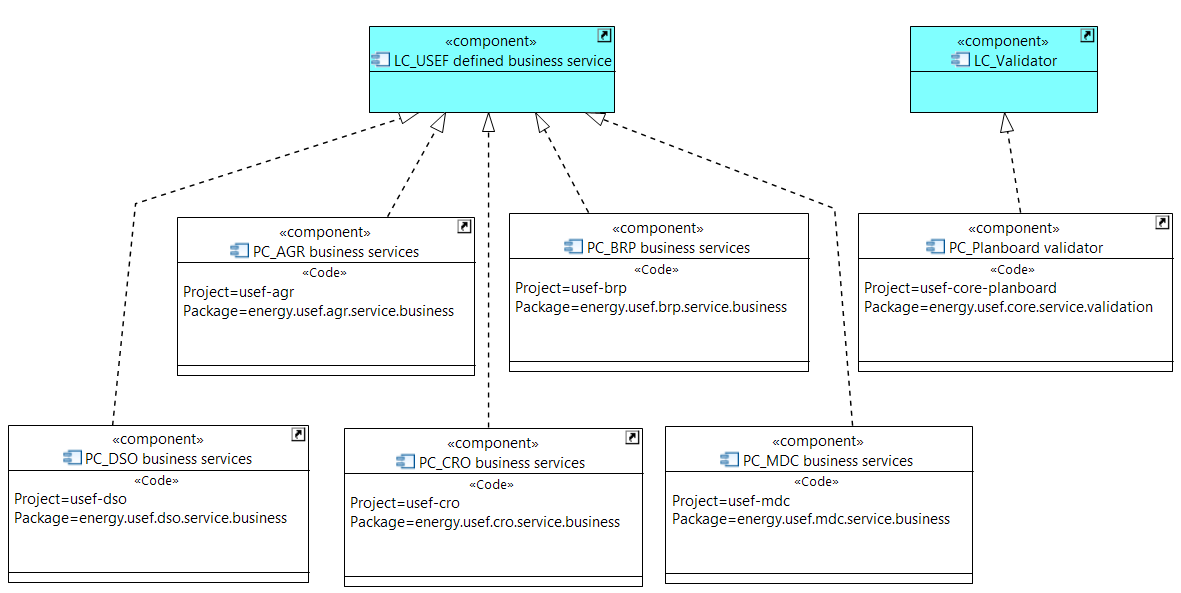
LC\_USEF defined business service

This component is the logical representation of the first kind of step mentioned in the introduction. USEF specifications describe the business logic that this service has to perform.

LC\_Validator

USEF specifies validations to perform during the execution of the processes. This component is the logical representation of such validations. By putting these in a separate component, validations can be reused by several LC\_USEF defined business services over different roles. For example, a validation against the intraday closure is common to all roles.

#### Component diagram



1. Business service layer component model

### Pluggable Business Component layer

#### Components

LC\_Pluggable Business Component

While [1] defines the processes to be supported in the different phases and regimes it does not specify for all process steps what business logic is to be executed. For these steps it is up to the party using the USEF reference framework to provide a meaningful implementation for these steps.

A Pluggable Business Component allows a 3rd party to plug in its own business logic that is realizing a step into the reference implementation.

The reference implementation is accompanied by a set of Pluggable Business Component realizations to demonstrate the USEF framework. These are simple implementations and don’t contain complex business logic.

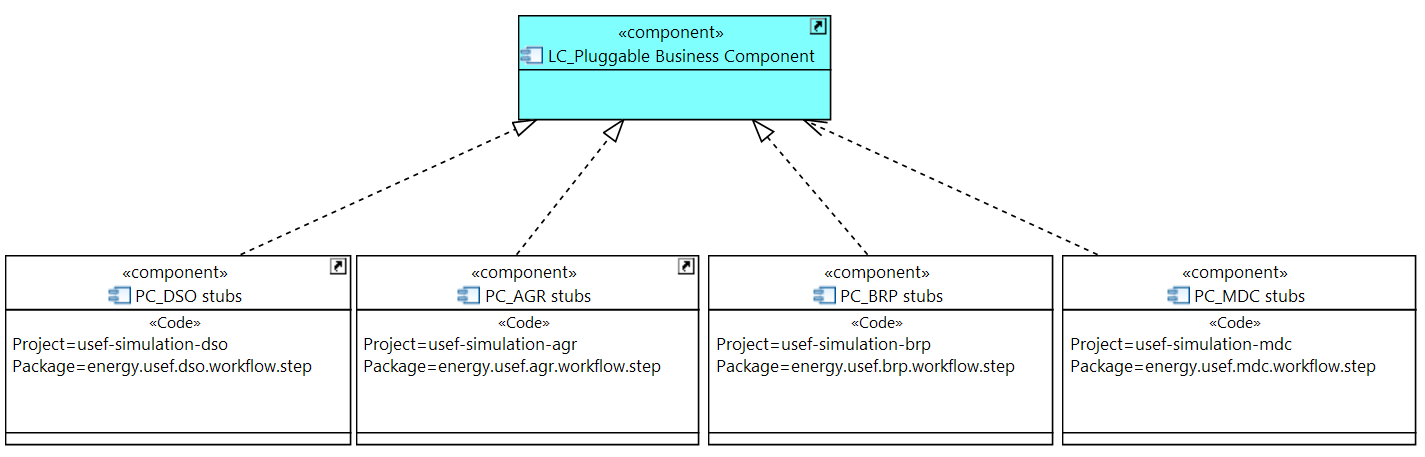
Following givens and assumptions have been made for the implementation of the provided set of Pluggable Business Components. Pluggable Business Components:

* Will implement a common interface.
* Are invoked in a request-reply mode.
* Should not fail on exceptions. On exception while executing, the exception is logged and the related sub process must stop.
* Realizations must be specified by configuration. This allows the use of another Pluggable Business Component realization without recompiling the code.
* Are not loosely coupled in the reference implementation. It is up to the Pluggable Business Component implementing party to use loose coupling or not when a stub is communicating with existing backend systems.
* Can and may not contain any USEF specified logic.
* Must be made available separated from the reference implementation.

A sequence diagram on Pluggable Business Component invocation can be found in the Diagrams chapter of this document.

[3] Contains interface information of each Pluggable Business Component and a list of Pluggable Business Component realizations delivered with the reference implementation.

#### Component diagram



1. Pluggable Business Component layer component model

### USEF Information layer

#### Introduction

This layer stores messages and information exchanged between different participants. The stored data must be protected as specified by the privacy and security principles (chapter 1 of [5]).

To comply with these USEF privacy and security principles all databases must be encrypted. How this is achieved in the reference implementation can be found in [2].

The data store schemas are shown in chapter 6 of this document.

#### Components

LC\_Message exception log

While executing the USEF processes several messages are exchanged between the different participants involved. As a USEF participant, it is important to have a view on messages which have been rejected by the recipient or which have not received a timely reply.

This component will store all messages (transactional and critical) that could not be delivered or for which the reply was not received in a timely manner. It also logs why the message could not be delivered. This log functionality is common over all role types.

A list of possible error message is given in chapter 6.2.5.1 of [1].

LC\_Message log

A USEF participant must be able to consult an audit log showing which participants an actor exchanged which messages with. It also logs if any (security) exceptions occurred during those exchanges. The message log does not contain any sensitive information: any occurrences of personally identifiable information, key material, etc. .. This log functionality is common over all role types.

LC\_Message hash log

As situations can occur that the same message is received multiple times by a USEF participant. The application must be designed in such a way that it can safely receive the same message multiple times.

This component stores the hashes of all (valid and invalid) inbound messages. A validation is done against this component to check if the same idempotent message was already received before. This log functionality is common over all role types.

LC\_Common Reference

As described in chapter 2.4.1 of [1], the common reference contains information on congestion points and the connections related to these congestion points.

This component stores the information with respect to the common reference. Since different roles don’t have the same requirements on congestion points and have different data access rights, the schemas of these stores will differ per role type.

LC\_Planboard

As described in chapter 4 of [4], unleashing the flexibility present in the Active Demand & Supply assets drives the value creation in the Universal Smart Energy Framework. Per PTU flexibility decisions can be taken on how to prevent (plan and validate) and solve (operate) congestion point issues. Afterwards settlement calculations will be done on the actions taken per PTU. For this each USEF participant will need a planboard to support his flex market operations.

A planboard is three-dimensional:

* X-axis: congestion points with associated details (maximum load, total number of connections, active Aggregators and number of connections represented by each);
* Y-axis: days (24-hour periods in the time zone that is customary and fixed for the market). Days do not have any interesting properties by themselves: it is sufficient to simply know which day and congestion point a PTU instance belongs to;
* Z-axis: PTUs (Program Time Units: fixed time slots, of which the exact duration, typically 15 minutes, is customizable but fixed for the market).

The PTU slots on the board should be thought of as stacks, which can have notes of various types pushed onto them. As already mentioned above, the complete collection of notes for a PTU will be ultimately used during the settlement phase, but during the plan, validate and operate phases, the then-current stack will be used to trigger business processes and perform validations.

This component stores the virtual planboard, which is used as the basis for flex market operations.

LC\_Aggregator portfolio

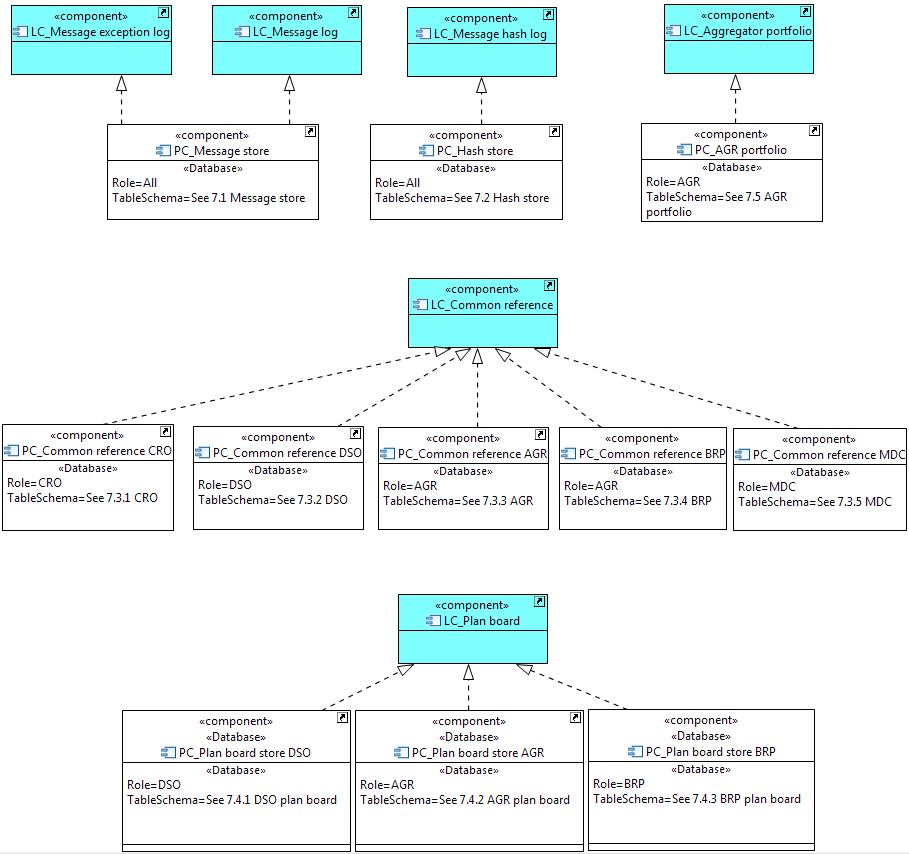
Chapter 4 of [4] goes into detail on the specifications and interactions needed in the Prosumer domain and its devices in order to unleash the flexibility in power from the end-user to the power market.

An Aggregator needs to know in detail what actual and forecast consumption or production is required or delivered by the connections it is managing, as well as potential and allocated flexibility. This information is kept in the Aggregator portfolio store.

A UDI Aggregator communicating with Prosumers via ADS manages this information for all devices of its Prosumers and also knows how to steer the devices of the Prosumers.

A Non-UDI Aggregator needs to manage this information on connection group level and supply this information to the demand response solution that manages all devices on its connections in order to realize the Aggregator’s portfolio.

#### Component diagram



1. Information layer component model

### USEF Integration layer

#### Introduction

The USEF recommended practices for message exchange are described in chapter 6 of [1], and implemented as such by the reference implementation.

Information exchange is required to support the USEF Market-based Control Mechanism (chapter ‎2 of [1]). When selecting a message transfer mechanism, local market conditions should be considered and the USEF Privacy and Security guidelines from Chapter 1 of [5] followed. From a strictly technical point of view, any messaging implementation that supports secure and reliable delivery of XML messages based on the USEF-defined recipient parameters can be used: possible standardized mechanisms include IEC CIM and ISO 15000 (ebXML).

USEF outlines a recommended practice, described in section 6.2 of [1], and provides a reference implementation of this mechanism, which offers a reliable and secure foundation for operation of the USEF role processes. In the reference implementation the USEF best-practice message transfer mechanism is implemented in the integration, QoS and security layer.

The diagrams common inbound and common outbound message flow show the treatment of in- and outbound messages. The diagrams show also the interaction with components of the QoS and security layer.

The exchange mechanism is the same for all message types and roles.

#### Components

LC\_Receiver channel

This component represents the single channel via which an actor will receive messages from another actor. When a message is received, following validations are performed:

* Check on duplicate hash message by the LC\_Idempotent handler.
* Resolve and identify the sender by the LC\_Participant resolver.
* Filter out messages of participants put on a black list by the LC\_Message filter.
* After unsealing the message by the LC\_Encryptor perform a unique message ID check.
* Validate the structure of the received payload.

When all validations are passed with success the message is put in the LC\_Inbound message store.

A response is returned to the sender with the outcome of the validations.

LC\_Sender channel

This component represents the single channel via which an actor can send messages to another actor. It will pick a message from the LC\_Outbound message store and deliver it to the recipient. When sending a message to a recipient following steps are performed.

* Determine the endpoint of the recipient via the LC\_Participant resolver.
* Seal the message by the LC\_Encryptor.
* Deliver the message via the LC\_Exponential back off handler.
* When a successful response is received within a timely matter, the message is stored in the LC\_Message log.
* If not, the message is stored in the LC\_Message exception log. In case of a critical or transactional message (see chapter 6.2.6 of [1]) a notification is sent via the LC\_Failed message notifier.

LC\_Inbound message store

This component will store valid accepted messages received from another actor.

It serves to guarantee the further processing of the messages and to allow the solution to process these messages at 'own pace'.

LC\_Outbound message store

This component will store messages that are to be delivered to another actor.

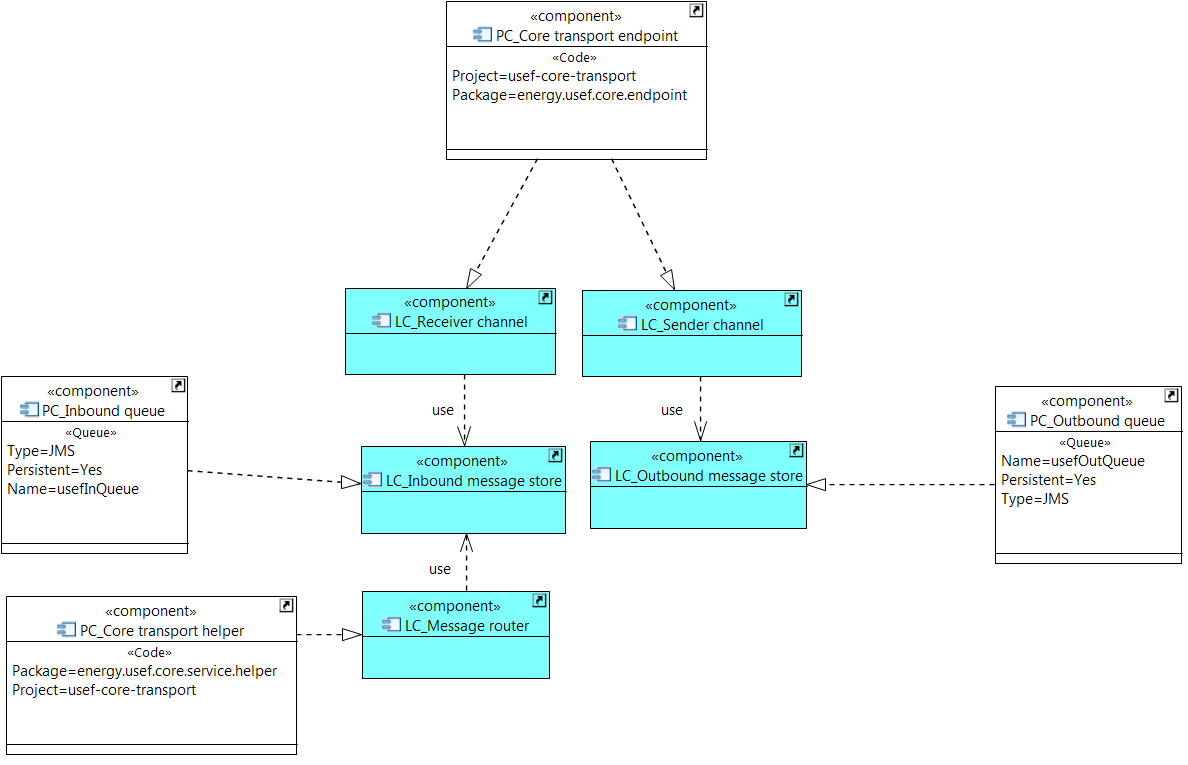
It enables the solution to start working on another process and to decouple the delivery of messages to the recipient.

LC\_Message router

This component will pick up the messages from the LC\_inbound message store, store the message in the LC\_Message log and will route it to the corresponding service that is responsible for the message processing.

A sequence diagram on the common inbound message (and message processing) flow can be found in the Diagrams chapter of this document.

#### Component diagram



1. Integration layer component model

### USEF Quality of Service layer

#### Components

LC\_Exponential back off handler

As communication will often take place over the public Internet, it is not guaranteed that the intended recipient is always available to receive the message. In case of this kind of disruptions, this component provides a mechanism for a configurable number of retries with an exponential back off time interval on sending a message.

LC\_Failed message notifier

Messages with status critical or transactional see chapter 6.2.6 of [1]) that could not be delivered with success must be brought to the attention of the sending USEF participant.

This component offers the notification functionality to the sender of a message. The notification is triggered in case a message is of type critical or transactional and one of the following is true:

* The message could not be delivered.
* The message is considered as not valid by the receiver.
* The response to the message is not received on time.

LC\_Idempotent handler

Chapter 6.2.5 of [1] specifies that a USEF implementations should detect and handle duplicate message identifiers. When receiving a message, 2 specific cases can be identified:

* The sender re-transmitted a message that was previously received and processed successfully.
* Sender re-used a message identifier for a message with different content.

This component will handle both cases. The first case is illustrated in the hash check flow sequence diagram. The second one is illustrated in the message id check sequence diagram.

LC\_File logger

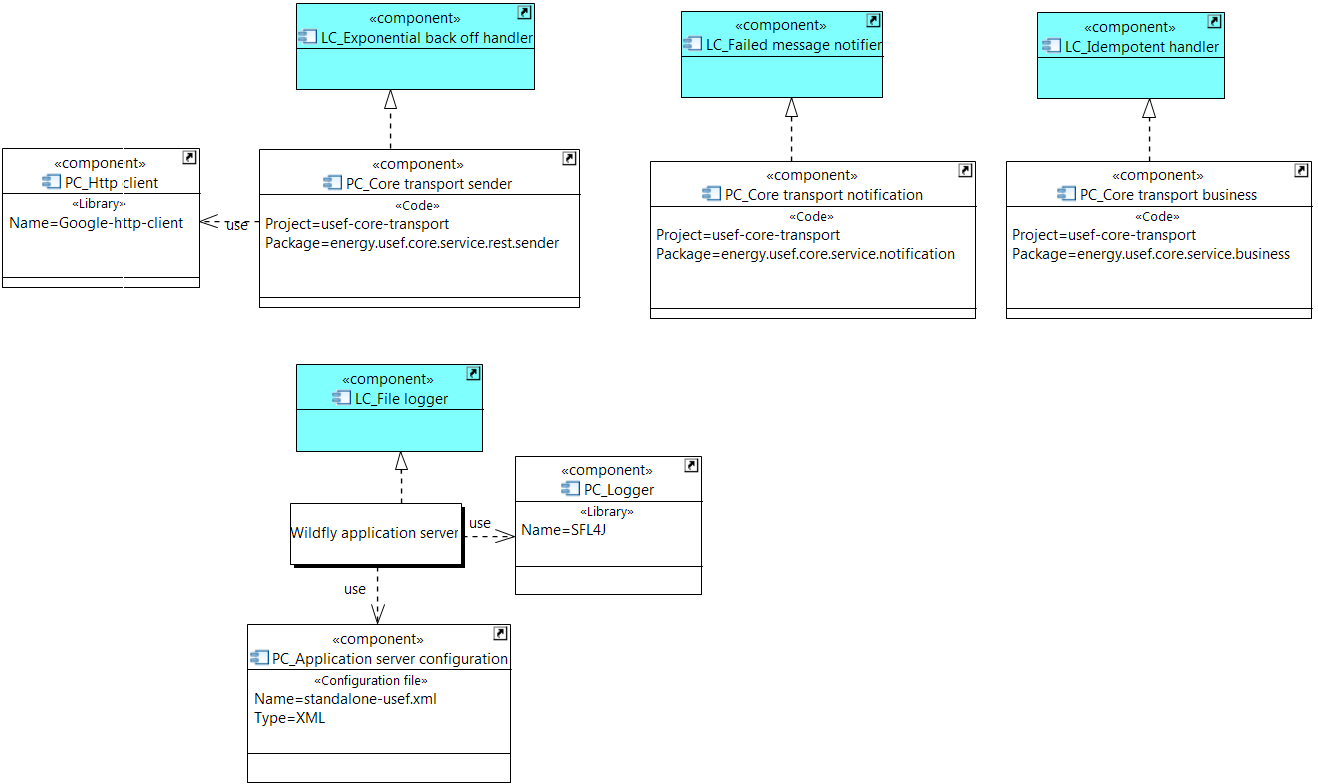
To be able to trace the correct working of and debug a runtime application, log and status messages are written to log files. This is common practice in application development. However, USEF specifications require a specific setup for the logger.

USEF XML messages should never be written to plain-text log files, as their content may very well be sensitive. But it's invaluable for troubleshooting to be able to display message contents on the console. Therefore a mechanism is foreseen to not log sensitive data except explicitly configured to do so.

How to enable confidential logging is shown in [2] and how to use the confidential debugging in the code is shown in [3].

Note that if the configuration is set to log unencrypted messages, the setup is no longer USEF compliant.

#### Component diagram



1. Quality of service layer component model

### The USEF Security layer

#### Introduction

USEF follows the principle of privacy and security by design to mitigate the risk of privacy and security issues (see chapter 1.3 of [5] for real life examples). The USEF privacy and security guidelines (chapter 1 of [5]) require all participants to be able to securely transmit and authenticate messages. A receiver must also be able to protect itself from malicious senders.

The components in the security layer are those needed to adhere to the USEF privacy and security principles.

(Note that in the information layer also has these principles applied by design as the databases are encrypted.)

#### Components

LC\_Message filter

This component will enable the filtering of messages coming from a USEF participant sending unsolicited messages. Filtering is performed in two steps and is based on message filter configuration.

* First, when explicitly configured so, only senders put on this first list are allowed to access the recipient domain.
* Second, if a USEF participant is e.g. sending unsolicited messages or is known not be trustworthy it must be possible to block messages from this participant. Access to the recipient domain by the sender is denied if the sender is put is this second filter list. The second filter step is always performed.

LC\_Message filter configuration

This component stores the criteria on blocking a sending participant and is used by the LC\_Message filter.

LC\_Participant resolver

In order to exchange messages with other USEF participants, a component needs to determine the appropriate entity address, USEF endpoint and encryption/signing keys of each participant.

A USEF participant must as such be able to add USEF-participating contractual partners using a simple procedure involving the trade name of each partner, and store this information. This information can be stored locally in LC\_Local network configuration or in remote location in LC\_Remote DNS configuration.

This component will:

* Decide, based on configuration to use the local or the remote network configuration store.
* Determine the USEF endpoint of the recipient participant in case of sending a message.
* Determine the encryption/signing keys of a sending participant in case of receiving a message. Note that by doing this with success it implicitly also identifies the sender.

Chapter 6.2.4 of [1] goes in more detail on the participant resolving by using DNSSEC.

LC\_Local network configuration

This component stores the local network configuration settings. How to configure the local network configuration can be found in [2].

LC\_Remote DNS configuration

This component stores the remote network configuration settings. See also chapter 6.2.4 of [1].

LC\_Encryptor

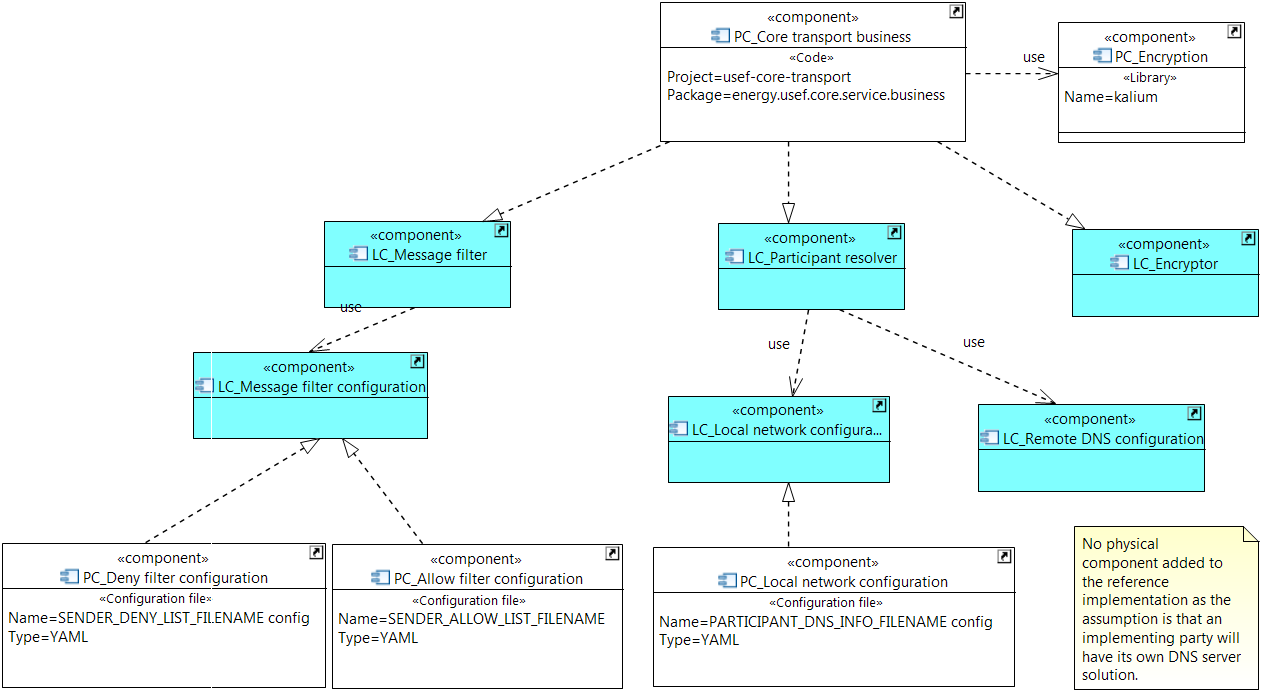
The USEF privacy and security guidelines require all participants to be able to securely transmit and authenticate messages. This is mandatory since only the intended recipient may see and work with the payload of the exchanged messages.

This component is responsible to:

* Sign and seal the payload of an outbound message.
* Unseal the payload of an inbound message.

More info on the applied cryptographic schema can be found in chapter 6.2.2 of [1].

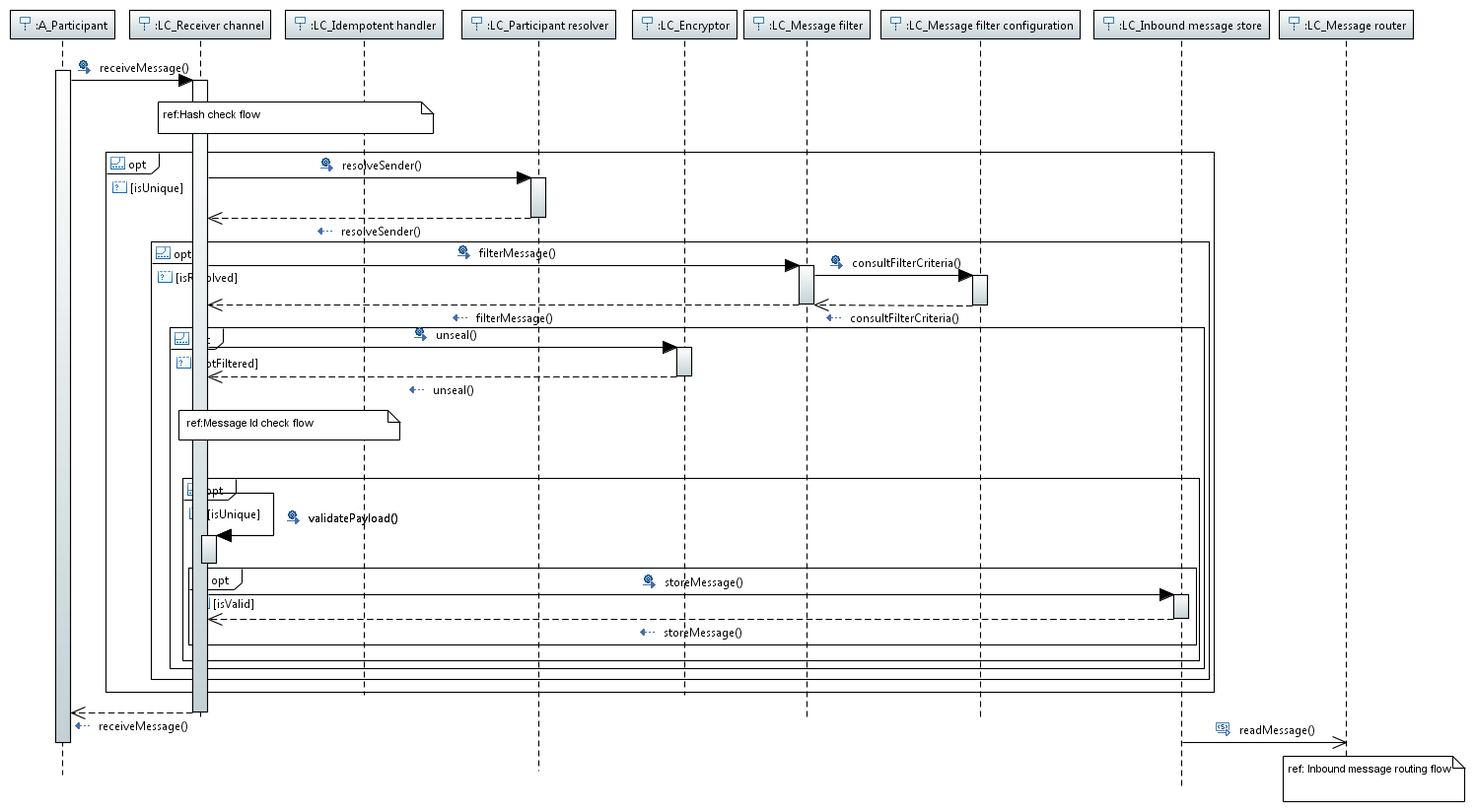
#### Component diagram



1. Security layer component model

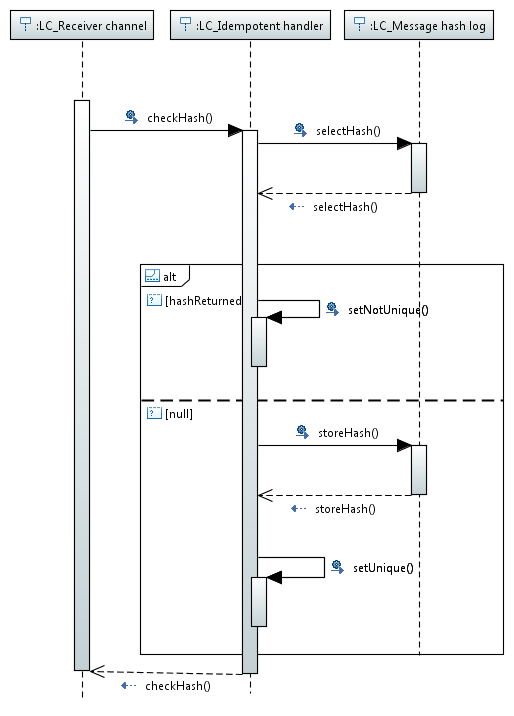
# Diagrams

## Common inbound message flow



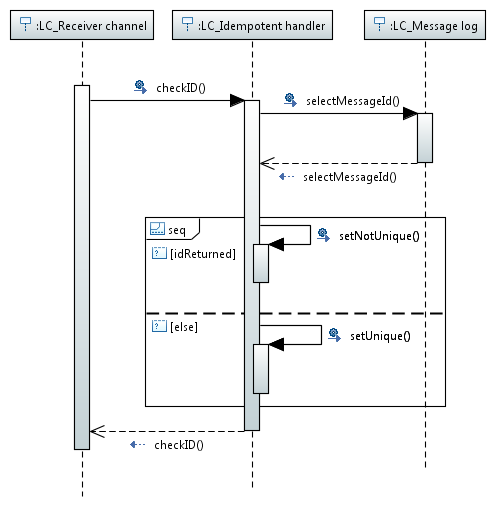
1. Common inbound message flow

## Hash check flow



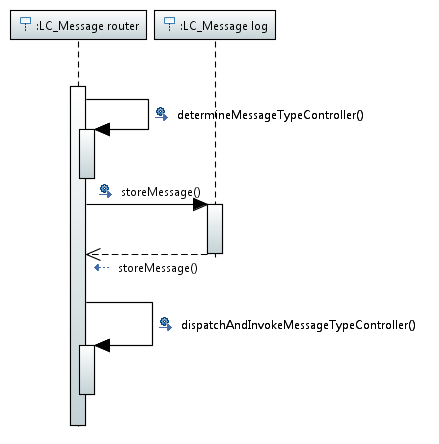
1. Hash check flow

## Message id check flow



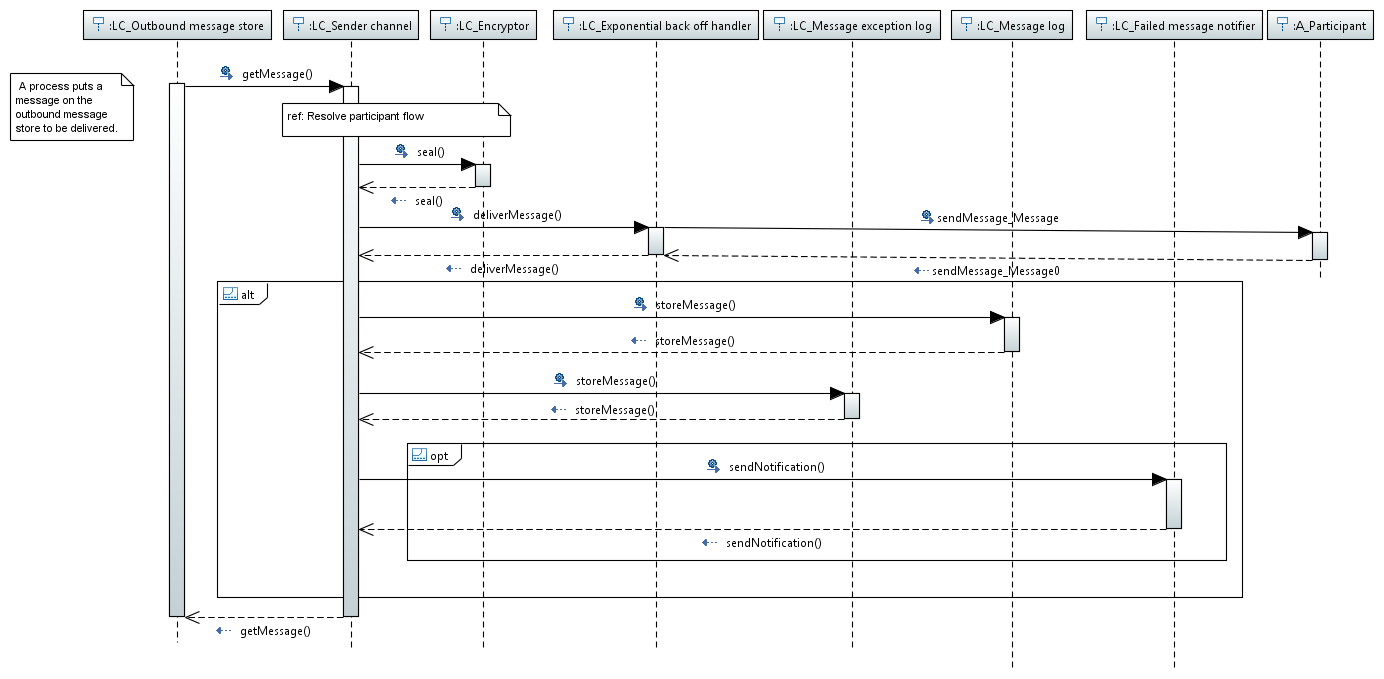
1. Message id check flow

## Common inbound message routing flow



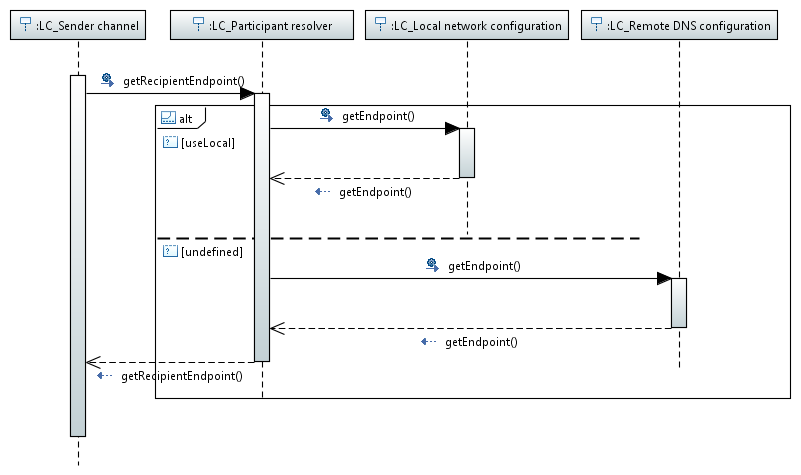
1. Common inbound routing flow

## Common outbound message flow



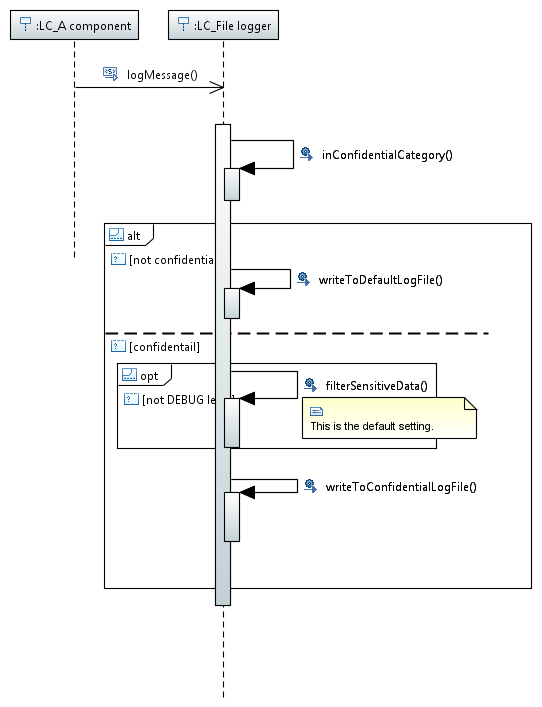
1. Common outbound message flow

## Resolve participant flow



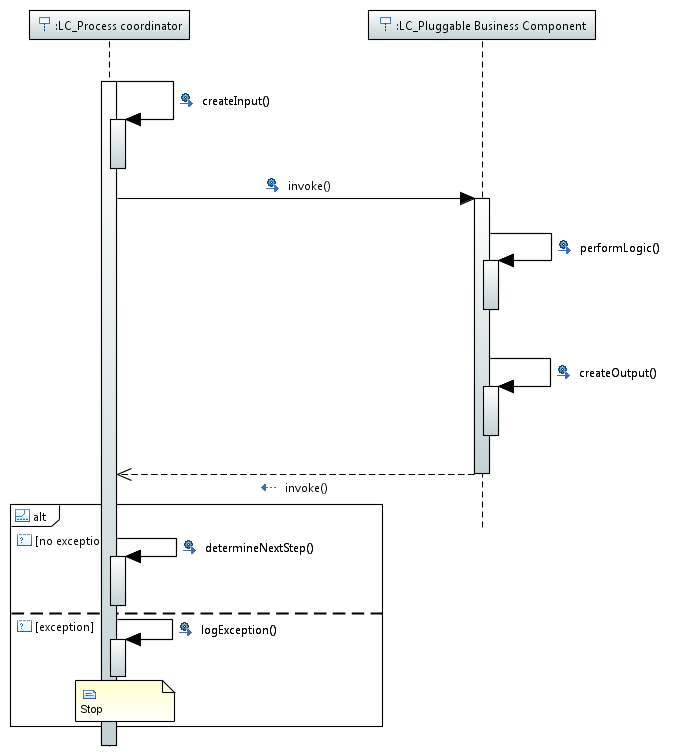
1. Resolve participant flow

## File logger flow



1. File logger flow

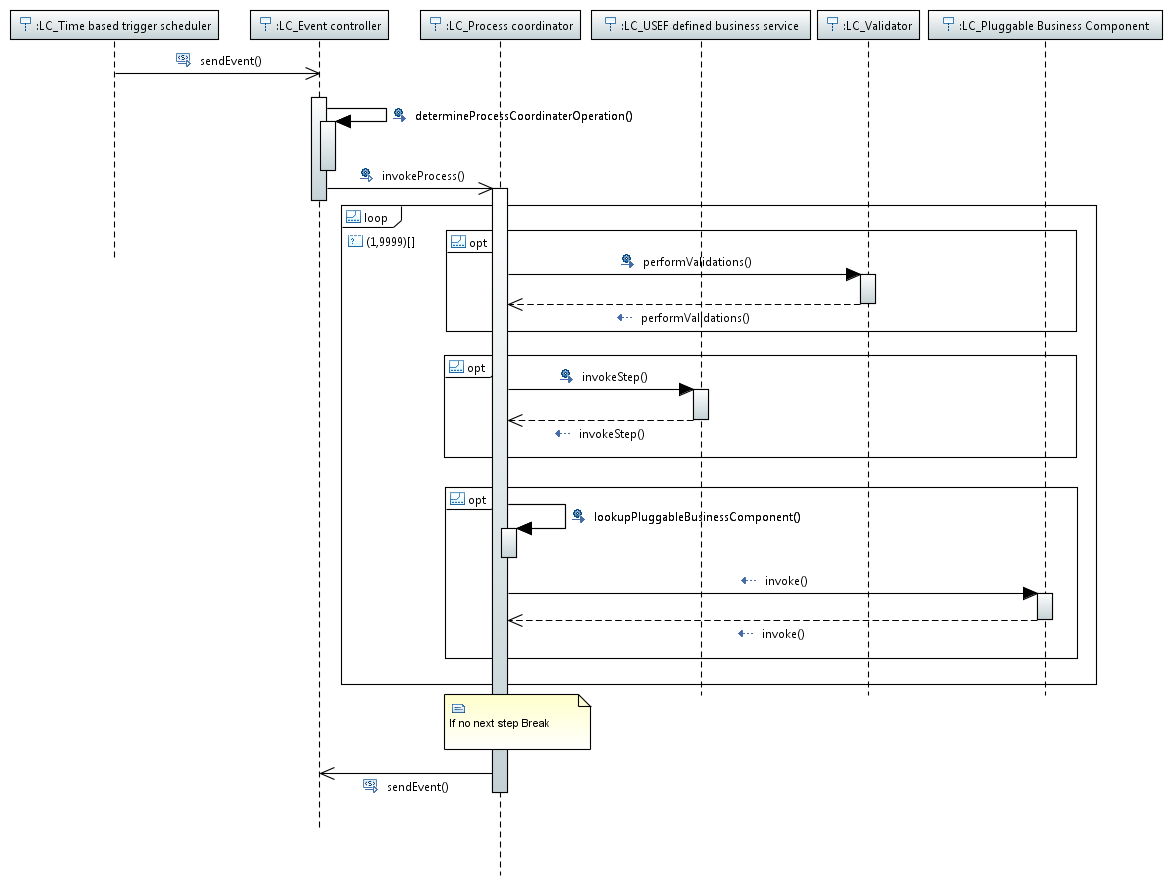
## Pluggable Business Component invocation flow



1. Pluggable Business Component invocation flow

## Common process flow

The diagram below shows a common view on how the identified components interact with each other during USEF sub process execution. As a start the time based trigger is used but the start point could be any of the three types of event.



1. Common process flow

# Process coordinator mapping

This chapter lists the different process coordinators and the Pluggable Business Components required to run the sub flows.

It also shows a mapping of these coordinators on the business process flows as shown in [1] to clearly link the process coordinators to the process flow steps.

For more information on how to start developing Pluggable Business Components see [2].

For more details on the provided Pluggable Business Component implementations, see the appendix Pluggable Business Components catalogue in [3].

## Triggering

As mentioned earlier in the description of the business process layer, process steps can be triggered in different ways. There are for instance time based events and events related to reception of messages, but there are also events that are triggered manually or by an external system. For this type of events endpoints are implemented (REST services) that can be invoked. Some coordinators implement process steps that are triggered by an endpoint.

The following endpoints are implemented:

### Aggregator

FlexOfferRevocationEndpoint:

This endpoint is used to revoke an existing flex offer of an Aggregator.

* URL: /FlexOfferRevocationEndpoint/revokeFlexOffer  
  HTTP method: GET   
  Parameters: /{flexOfferSequence}/{recipientDomainName}/{recipientRole}  
  Example: /FlexOfferRevocationEndpoint/revokeFlexOffer/1234/dso1.usef-example.com/DSO

ReOptimizePortfolio:

This endpoint is used to trigger the re-optimize portfolio process on-demand for an Aggregator for the specified period. Note that this process is also triggered after the collect forecast process is finished, when a deviation in the A-plan or D-prognoses is detected and after flex trading.

* URL: /ReOptimizePortfolio  
  HTTP method: GET  
  Parameters: /{period}  
  Example: ReOptimizePortfolio/2015-09-07

UpdateConnectionForecastEndpoint:

This initial forecast of an Aggregator may change over time because of changes in e.g. weather, in which case the forecast can be re-run for some or all connections.   
This endpoint triggers this process. As input, zero, one or more connection IDs can be passed to the endpoint. If no connections are passed, all the connection portfolios have to get updated. If connection IDs are specified, only the portfolios for the specified connections are updated.

* URL: /connectionportfolio/update  
  HTTP method: GET  
  Parameters: /{connectionEntityAddress}   
  Example: /connectionportfolio/update/ean.100000000003
* URL: /connectionportfolio/update/all   
  HTTP method: GET   
  Parameters: all  
  Example: /connectionportfolio/update/all
* URL: /connectionportfolio/update   
  HTTP method: POST   
  Parameters: a JSON-object containing a list of connection IDs  
  Example JSON-object:  
   {"connectionEntityAddressList": ["ean.100000000003","ean.100000000004"]}

### Balance Responsible Party

PrepareFlexRequests:

This endpoint is used to trigger the flex preparation process of a BRP for the specified period.

* URL: /PrepareFlexRequests   
  HTTP method: GET   
  Parameters: /{period}  
  Example: /PrepareFlexRequests/2015-09-07

FlexOrderEndpoint:

This endpoint is used to trigger the flex order process in operate phase of a BRP.

* URL: /FlexOrderEvent   
  HTTP method: GET   
  Parameters: none  
  Example: /FlexOrderEvent

For each coordinator described in the next paragraphs, the applicable triggers are mentioned.

## Plan phase

This phase is described in chapter 2.3.1, 2.3.2, 4.3.1 and chapter 4.3.2 of [1].

Since the plan and validate phases are iteratively repeated until intraday closure, a number of coordinators are as such applicable in both phases.

### AgrCommonReferenceUpdateCoordinator

#### Process

Goal is to inform the CRO about updates in the common reference information the aggregator is responsible for.

When this process is triggered, the following steps are executed:

* The common reference synchronisation tables are queried to identify all changes in the common reference information of this aggregator.
* After that, a Common Reference Update message is created and sent to the CRO.

The trigger is based on a configuration setting identifying the time at which the common reference update should be triggered each day.

#### Pluggable Business Components

* NA

### AgrCommonReferenceQueryCoordinator

#### Process

Goal is to initialize the planboard of an AGR a number of days in advance and retrieve common reference information.

When initialization of the planboard is triggered, the following steps are executed:

* The common reference is queried to identify all congestion points and connections on which the AGR represents customers. If the common reference is unavailable and remains unavailable after a number of retries, historical data of the previous common reference query is used.   
  If no historic data is available either, all connections are considered not to belong to a congestion point.
* After that, the planboard is initialized with PTU containers in plan phase for the number of days specified in the trigger.
* Finally, the AgrUpdateElementDataStoreCoordinator and AgrCreateConnectionProfileCoordinator are called to update the elements data store and populate the portfolio with profile data.

The trigger is based on configuration settings identifying the time at which the initialization will be triggered and the number of days to be planned ahead. This number also specifies the frequency of triggering. E.g. if the value is 3, the trigger is fired every three days to plan three days ahead starting from the day after the trigger date.

The connections and congestion points retrieved from the common reference are valid and used during this period. When the trigger is fired again after the specified number of days, the plan for the next number of days is created and a new common reference query is done. The retrieved common reference information is stored in the database; new or changed connections and congestion points are added, for unchanged items the validity periods are adjusted, if necessary.

#### Pluggable Business Components

* NA

### AgrUpdateElementDataStoreCoordinator

#### Process

Goal is to update the element data store in order to supply up-to-date data to the subsequent portfolio initialization process.

When triggered, the element data store is filled with the details of the portfolio elements associated with the aggregator’s connections. These details include profile power values, which are generated by an internal process of the AGR that is not specified by USEF.

This process is triggered by the AgrCommonReferenceQueryCoordinator.

#### Pluggable Business Components

* Create Elements

### AgrCreateConnectionProfileCoordinator

#### Process

Goal is to initialize the portfolio with information from the element data store.

When triggered, the portfolio is initialized with profile power values for all connections by using the profile values from the element data store. This is an internal process of the AGR that is not specified by USEF.

For UDI aggregators, after the portfolio initialization, the AgrCreateUdiCoordinator is called to create the appropriate UDIs and associated capabilities.

This process is triggered by the AgrCommonReferenceQueryCoordinator.

#### Pluggable Business Components

* Create Profile

### AgrCreateUdiCoordinator

This process only applies to UDI aggregators.

Goal is to create the UDIs and associated UDI events and capabilities for all managed devices in the element data store. This is an internal process of the AGR that is not specified by USEF.

When triggered, the portfolio is initialized with profile power values for all connections by using the profile values from the element data store.

This process is triggered by the AgrCreateConnectionProfileCoordinator.

#### Pluggable Business Components

* Create UDI

### AgrConnectionForecastPlanBoardCoordinator

#### Process

Goal is to collect and store connection forecast information after the planboard is initialized.

When the forecast collection process is triggered, the following steps are executed:

* For all connections a load forecast is created for each PTU. The generation of these forecasts is an internal process of the AGR and is therefore not specified by USEF.
* The forecast information for connections per PTU is stored in the AGR planboard for the number of days specified in the trigger.

The trigger is based on configuration settings identifying the time at which the forecast collection will be triggered and the number of days to be planned ahead. This number also specifies the frequency of triggering. E.g. if the value is 3, the trigger is fired every three days to plan three days ahead starting from the day after the trigger date.

#### Pluggable Business Components

* Collect Forecast
* Non-UDI Collect Forecast

### AgrUpdateConnectionForecastCoordinator

#### Process

This initial forecast may change over time because of changes in e.g. weather. The goal is to update the planboard by re-running the forecast for some or all connections.

When triggered:

* For updates of the forecast, the CRO is not consulted again. Instead, all (or a subset) of the existing congestion points and connections are used in the forecast calculation.
* For all connections a load forecast is created for each PTU. The generation of these forecasts is an internal process of the AGR and is therefore not specified by USEF.
* The forecast information for connections per PTU is stored in the AGR planboard for the number of days specified in the trigger.

A REST endpoint UpdateConnectionForecastEndpoint is foreseen to trigger the process of updating a forecast. A list of connection IDs can be passed to the endpoint, if no connections are passed ALL the connection portfolios have to get updated.

#### Pluggable Business Components

* Collect Forecast
* Non-UDI Collect Forecast

### AgrReOptimizePortfolioCoordinator

#### Process

Goal is to re-optimize the portfolio in order to maximize the income by influencing consumption or production of the devices (via ADS) behind the connections managed by an Aggregator.

Portfolio optimisation is a process that takes place in both the Plan and the Operate phase.

When triggered:

* The process verifies if it is already running for the requested period. If that is the case, the process will not re-execute immediately.
* If the process is not yet running for the requested day, the portfolio information for the requested period is retrieved from the portfolio store.
  + This information is transferred to a process re-optimizing the portfolio of the Aggregator. The process is an internal process of the AGR and is therefore not specified by USEF.
  + An updated connection portfolio is returned and stored, as well as a list of device messages, specifying the actions to be taken to optimize the portfolio based on the UDI events as stored in the connection portfolio for each connection.
* If, while the process was running, additional events have triggered this process for the same period, the re-optimize process will be re-executed for the requested period.
* Finally, the AgrReCreatePrognosesCoordinator is triggered to determine if new prognoses need to be (re-)created.

This process step is triggered to perform a complete portfolio re-optimization when either of the following occurs:

* The collect forecast process has finished at the start of the Plan phase,
* Changes in the forecast are identified by the AgrIdentifyChangeInForecastCoordinator and the portfolio needs to be re-optimized
* When the aggregator has received flex orders in the Plan, Validate or Operate phase,
* In the Operate phase, a deviation in the A-plan or D-prognoses is detected,
* The REST endpoint ReOptimizePortfolio is called.

#### Pluggable Business Components

* Optimize AGR Portfolio
* Non-UDI Optimize AGR Portfolio

### AgrReCreatePrognosesCoordinator

#### Process

After portfolio optimization, an aggregator needs to decide whether new A-plans and/or D-prognoses are required in order to actually create and send such plans when needed. This process step takes place in both the Plan and the Operate phase.

When triggered:

* The coordinator checks if an A-plan exists. If that is not the case, an event is fired to create a new A-plan.
* If an A-plan already exists, a process is called that makes the decision whether to actually re-create A-plans and/or D-prognoses. This is both a technical and business decision, which is left to an internal process of the AGR and is therefore not specified by USEF.  
  This process returns:
  + - A list of A-plan identifications indicating which A-plans need to be re-created.
    - A list of D-prognosis identifications indicating which D-prognoses need to be re-created.
  + If the list of A-plan identifications is not empty, the “Re-create A-plan” process step is triggered immediately.
  + If the list of D-prognosis identifications is not empty, the “AGR Create and Send D-Prognoses” step is triggered immediately.

This process step is triggered when the portfolio optimization process is completed. The event triggering this process step may contain device messages as a result of portfolio optimization.

#### Pluggable Business Components

* New Prognoses Required

### AgrCreateAPlanPlanboardCoordinator

#### Process

Goal is to create and supply A-plans on a daily basis to the BRP(s) operating connections that the Aggregator represents as well.

When triggered:

* Per BRP an A-plan is created informing each BRP about the forecast aggregate consumption or production of the customer connections linked to the Aggregator in each PTU of a given period. This information is needed by the BRP to perform its portfolio optimization.
* An A-plan can be submitted at any time and shall be approved by the BRP. The AGR stays in the Plan phase until:
  + - For all A-plans that have been sent to BRPs an approval has been received from the BRPs.
    - Less than one hour remains before the day-ahead gate closure time is reached. In this case, an event is triggered to invoke the AgrUpdateAPlanPlanboardCoordinator.

The A-plans are stored on the planboard.

This process step is triggered by:

* The process that creates the connection portfolio in case no A-plan exists yet.
* An event triggered by the AgrReCreatePrognosesCoordinator, which determines if and which A-plans need to be re-created.

#### Pluggable Business Components

* NA

### AgrUpdateAPlanPlanboardCoordinator

#### Process

This process step checks all non-finalized A-plans for the next period (with state SENT or PENDING\_FLEX\_TRADING) and makes them final.

The trigger is a time based trigger that will fire a configurable number of PTUs before the day-ahead gate closure time is reached.

#### Pluggable Business Components

* NA

### AgrFlexOfferCoordinator

#### Process

Goal is to compose and deliver flex offers to participants (DSOs and BRPs) that have sent flex requests.

When triggered:

* The planboard is queried to check if there are flex requests available for which a flex offer is needed and still useful. If the last PTU for which flex is required, is already passed the intraday gate closure time for that PTU, no flex offers should be created.
* The flex requests are analysed by the aggregator to check if it is able to supply any flexibility for the congestion point or connections and the PTU(s) it applies to. If so, a flex offer is composed and sent to the originating requester, detailing the flexibility that can be supplied and the price setting for doing so. The analysis and generation of flex offers, is an internal process of the AGR and is therefore not specified by USEF.
* Note that the result list may be empty, if no flex is available.

The trigger is a time based trigger that will fire with a configurable (short) interval to check if requests are posted.

#### Pluggable Business Components

* Create Flex Offers

### AgrFlexOfferRevocationCoordinator

#### Process

Goal is to enable an Aggregator to revoke a previously made offer if the Aggregator is no longer willing or capable to fulfil the flex offer.

When triggered:

* A message is composed to revoke the identified flex offer. Revocations are only sent to a participant (DSO or BRP) for flex offers which have previously been accepted by that participant.

A REST endpoint FlexOfferRevocationEndpoint is foreseen to trigger the revocation of an offer. Two parameters are to be provided: the domain name of the targeted participant and the flex offer sequence number.

#### Pluggable Business Components

* NA

### BrpCommonReferenceUpdateCoordinator

#### Process

Goal is to inform the CRO about updates in the common reference information the BRP is responsible for.

When this process is triggered, the following steps are executed:

* The common reference synchronisation tables are queried to identify all changes in the common reference information of this BRP.
* After that, a Common Reference Update message is created and sent to the CRO.

The trigger is based on a configuration setting identifying the time at which the common reference update should be triggered each day.

#### Pluggable Business Components

* NA

### BrpCommonReferenceQueryCoordinator

#### Process

Goal is to initialize the planboard of a BRP a number of days in advance.

When triggered:

* The common reference is queried to identify all congestion points and connections on which the AGR represents customers. If the common reference is unavailable and remains unavailable after a number of retries, historical data of the previous common reference query is used.   
  If no historic data is available either, all connections are considered not to belong to a congestion point.
* After that, the planboard is initialized with PTU containers in plan phase for the number of days specified in the trigger.

The trigger is based on configuration settings identifying the time at which the process will be triggered and the number of days to be planned ahead. This number also specifies the frequency of triggering. E.g. if the value is 3, the trigger is fired every three days to plan three days ahead starting from the day after the trigger date.

The connections and congestion points retrieved from the common reference are valid and used during this period. When the trigger is fired again after the specified number of days, the plan for the next number of days is created and a new common reference query is done. The retrieved common reference information is stored in the database; new or changed connections and congestion points are added, for unchanged items the validity periods are adjusted, if necessary.

#### Pluggable Business Components

* NA

### BrpAplanCoordinator

Power vs. Energy

Whereas USEF messages deal with Power (values in Watts), all BRP-internal data stores should work with Energy (values in Watt-hours).

* When emitting USEF messages, all Energy values should be expressed as Power, by multiplying each value by the number of PTUs per hour.
* When consuming USEF messages, all Power values should be expressed as Energy in internal data stores, by dividing each value with the number of PTUs per hour.

#### Process

The BRP wants to optimize its portfolio and attain an economically optimized program. During this optimization, the BRP will negotiate with its Aggregators to exploit the available flexibility in the market and optimize its value.

The BrpAplanCoordinator is the starting point of this optimization. Its goal is to receive and process A-plans from Aggregators and create flexibility requests if necessary. The A-plan only includes the forecasted load of those Prosumers served by the Aggregator and that have a connection related to the BRP.

When triggered:

* After reception of an A-Plan, the messages are checked and stored in the database.
* All A-plans for the same period as the one just received are then processed and validated. This leads to either accepting or rejecting A-plans.
* Validation of the A-plans is an internal process of the BRP and is therefore not specified by USEF.
* For all accepted A-plans an approval message is sent to the corresponding aggregator.
* An event is called to trigger the process that analyses if flex trading is required and possible and creates flexibility requests to the originating Aggregator, indicating the desire to adjust the A-plan by engaging in flex trading.
* Generation of these flex requests is an internal process of the BRP and is therefore not specified by USEF.
* After the list of flex requests has been returned by the Pluggable Business Component, the BrpCreateFlexRequestCoordinator is triggered to sends the flex requests.

The trigger is the reception of an A-Plan message from an Aggregator.

The process that analyses if flex trading is possible and generates flexibility requests can also be triggered by means of a REST endpoint PrepareFlexRequests.

#### Pluggable Business Components

* Received A-Plan
* Prepare flex requests

### BrpCreateFlexRequestCoordinator

#### Process

The goal is to send the flexibility requests generated earlier by the BrpAplanCoordinator.

When triggered, the flex requests provided in the input are completed and sent to the appropriate aggregators.

#### Pluggable Business Components

* NA

### BrpFlexOrderCoordinator

#### Process

The goal is to optimize the BRPs portfolio and order the flexibility offered earlier by Aggregators.

When triggered:

* First a decision making process is fired identifying which offers are desirable and should possibly be turned into flex orders. This is a BRP internal process.
  + If all offers for an A-plan are not desirable, this A-plan is approved and no flex trading will take place now for this A-plan.
* Then a decision making process is fired identifying what desired offers of which Aggregator to use. This is a BRP internal process. This internal process will typically run multiple times and will return a decision with respect to the given time context.
* A decision can be deferred if time is still left to receive better/more offers. Orders can be selected or not. No flex orders should be generated if the last PTU for which flex is required is less than INTRADAY\_GATE\_CLOSURE\_PTUS away from the Operate phase
* Selected offers become orders and are stored on the planboard.
* The orders are transferred to the corresponding Aggregator.

The trigger is a time based trigger that will fire with a configurable (short) interval. This process can also be triggered in operate phase by means of a REST endpoint FlexOrderEndpoint.

#### Pluggable Business Components

* Get Not Desirable Flex Offers
* Create Flex Orders

### BrpFlexOfferRevocationCoordinator

#### Process

Goal is to handle the fact that an Aggregator has revoked a flex offer.

When triggered:

* A check is done if the offer exists and if all PTU’s in the flex offer are before the Operate phase.
* If so the revocation is accepted, otherwise rejected towards the AGR.

The trigger is the receipt of the flex offer revocation message from an Aggregator.

#### Pluggable Business Components

* NA

### BrpFinalizeAPlansCoordinator

#### Process

The goal of a BRP is to end the Plan phase with final A-plans for all aggregators, even if the normal planning and trading processes have somehow failed to result in that state.

When triggered, this process sets the status of any remaining non-approved A-plans to Final.

Note that the finalized A-plans are not sent to the aggregator.

The trigger is a configurable time based trigger that will fire a number of PTUs before day ahead gate closure.

#### Pluggable Business Components

* NA

### BrpCreateMissingAPlansCoordinator

#### Process

The goal of a BRP is to end the Plan phase with final A-plans for all aggregators. If an aggregator fails to deliver its A-plan to the BRP by the day-ahead gate closure time, the BRP is left with an incomplete forecast for that period.

Before the BRP is able to decide whether it wants to engage in flex trading, it needs to supply its own replacement forecasts for the missing aggregator connections.

When triggered, this process creates the missing A-plans. The generation of these missing A-plans is an internal process of the BRP and is therefore not specified by USEF.

The trigger is a configurable time based trigger that will at the day ahead gate closure.

#### Pluggable Business Components

* Create Missing A-Plans

### DsoCommonReferenceUpdateCoordinator

#### Process

Goal is to inform the CRO about updates in the common reference information the DSO is responsible for.

When this process is triggered, the following steps are executed:

* The common reference synchronisation tables are queried to identify all changes in the common reference information of this DSO.
* After that, a Common Reference Update message is created and sent to the CRO.

The trigger is based on a configuration setting identifying the time at which the common reference update should be triggered each day.

#### Pluggable Business Components

* NA

### DsoCommonReferenceQueryCoordinator

#### Process

Goal is to initialize the planboard of a DSO a number of days in advance and retrieve common reference information.

When initialization of the planboard is triggered, the following steps are executed:

* The common reference is queried to identify what Aggregators are active on each congestion point, and how many connections they serve. If the common reference is unavailable and remains unavailable after a number of retries, historical data of the previous common reference query is used. If no historic data is available either, the congestion point is considered as having zero Aggregators.
* After that, the planboard is initialized with PTU containers in plan phase for the number of days specified in the trigger.

The trigger is based on configuration settings identifying the time at which the initialization will be triggered and the number of days to be planned ahead. This number also specifies the frequency of triggering. E.g. if the value is 3, the trigger is fired every three days to plan three days ahead starting from the day after the trigger date.

The connections and congestion points retrieved from the common reference are valid and used during this period. When the trigger is fired again after the specified number of days, the plan for the next number of days is created and a new common reference query is done. The retrieved common reference information is stored in the database; new or changed connections and congestion points are added, for unchanged items the validity periods are adjusted, if necessary.

#### Pluggable Business Components

* NA

### DsoConnectionForecastPlanBoardCoordinator

Goal is to create and store the Non-Aggregator connection forecast information.

When triggered:

* A load forecast for all connections within a congestion point not served by an Aggregator is created. This is mandatory by USEF as these connections do contribute to the use of the available grid capacity. The generation of this forecast however, is an internal process of the DSO and is therefore not specified by USEF.
* The forecast information is stored in the DSO planboard for the number of days specified in the trigger.

The trigger is based on configuration settings identifying the time at which the process will be triggered and the number of days to be planned ahead. This number also specifies the frequency of triggering. E.g. if the value is 3, the trigger is fired every three days to plan three days ahead starting from the day after the trigger date.

#### Pluggable Business Components

* Create Non Aggregator Forecast

## Validate phase

This phase is described in chapter 2.3.1, 2.3.2, 4.3.1 and chapter4.3.2 of [1].

Since the plan and validate phases are iteratively repeated until intraday closure, a number of coordinators are as such applicable in both phases.

### AgrCreateDPrognosisPlanboardCoordinator

#### Process

Goal is to create and supply D-prognoses on a daily basis to the DSO(s) operating congestion points on which the Aggregator represents connections.

When triggered:

* Per DSO, a D-prognosis is created informing each DSO about the forecast aggregate consumption (or production) of the customer connections linked to the Aggregator per congestion point in each PTU of a given period. This information is needed by the DSO to perform its grid safety analysis. A D-prognosis can be submitted at any time, but at least one D-prognosis must arrive prior to the day-ahead gate closure time of the first PTU of any period in which the Aggregator represents connections on a given congestion point. The D-prognosis is created by decomposing an approved A-plan into per-congestion point data. No D-prognoses should be sent anymore if the last PTU of the period is less than INTRADAY\_GATE\_CLOSURE\_PTUS away from the Operate phase.
* The D-prognosis are stored on the planboard.

This process step can be triggered by:

* A time based event fired once a day taking into account the day ahead closure time.
* An event fired after all A-plans have been approved by the BRPs.
* An event fired by the AgrReCreatePrognosesCoordinator, which determines if and which D-prognoses need to be re-created.

#### Pluggable Business Components

* NA

### DsoCreateMissingDPrognosesCoordinator

#### Process

Goal is to create missing D-prognoses before a grid safety analysis is executed.

When triggered:

* A check is done to verify if there are any missing D-prognoses
* If so, these missing D-prognoses are created. The generation of these missing prognoses is an internal process of the DSO and is therefore not specified by USEF.
* Finally, if any missing D-prognoses were created, the gird safety process is triggered.

This process is triggered by:

* A time based trigger fired each day at the moment the day-ahead closure minus the DAY\_AHEAD\_GATE\_CLOSURE\_PTUS is reached. When this trigger is fired it is mandatory to perform create any missing D-prognoses and subsequently perform a grid safety analysis.

#### Pluggable Business Components

* Create Missing D-Prognoses

### DsoGridSafetyAnalysisCoordinator

#### Process

Goal is to analyse the grid operated by the DSO to identify whether congestion is expected during a PTU and whether an attempt can be done to purchase flexibility or not to solve the congestion. In the latter case the connections related to the congestion point will have to be limited (orange regime).

* The analysis is based on D-prognoses received from Aggregators and Non-Aggregator connection forecasts for those connections not related to an Aggregator.
* If not all Aggregators have delivered a valid D-prognosis at the day-ahead gate closure time, the DSO has an incomplete forecast for a congestion point. The missing D-prognoses are created by the DsoCreateMissingDPrognosesCoordinator.
* The grid safety analysis is an internal process of the DSO and is therefore not specified by USEF
* If flexibility request have been created as part of the grid safety analysis, an event is triggered to send these flexibility requests.
* If congestion is expected but no aggregators are available to solve the congestion by means of flex trading, the DsoColoringCoordinator is triggered to mark the appropriate PTUs Orange.

Prior to the Operate phase, this analysis can be performed multiple times as different events can happen that can influence the outcome of a grid safety analysis. (E.g. a new D-prognosis is received or a flex offer is purchased).

Different triggers can start the grid safety analysis and have different validations before continuing.

* If a valid new or updated D-prognosis is received from an Aggregator, a grid safety analysis must be done if each Aggregator related to a congestion point has delivered a valid D-prognosis.
* A time based trigger fired each day at the moment the day-ahead closure minus the DAY\_AHEAD\_GATE\_CLOSURE\_PTUS is reached. When this trigger is fired, first the DsoCreateMissingDPrognosesCoordinator creates any missing D-prognoses, which then triggers the grid safety analysis.

#### Pluggable Business Components

* Perform Grid Safety Analysis

### DsoColoringProcessCoordinator

#### Process

If the grid safety analysis indicates that congestion is expected in a certain PTU for a certain congestion point, but there no flexibility can be procured to resolve the expected congestion, PTU(s) need to be marked as Orange in order to prepare for connection limiting.

When triggered:

* Each PTU for which congestion is expected is marked as Orange for the specific congestion point.
* Next, the process that prepares the stepwise limiting of connections is invoked. This is an internal process of the DSO and is therefore not specified by USEF.
* Once the coloring process is complete, a process is triggered to take further action. This is an internal process of the DSO and is therefore not specified by USEF.

This process it triggered by:

* The grid safety analysis process, when congestion is expected and no aggregators are available.
* The flexibility ordering process, when there are no appropriate flexibility offers.

#### Pluggable Business Components

* Prepare Stepwise Limiting
* Post Coloring Process

### DsoCreateFlexRequestCoordinator

#### Process

Goal is to create and send flex requests to all Aggregators linked to an identified congestion point (PTU). When triggered:

* The same flex request is sent to all Aggregators linked to the congestion point. The USEF flex request message contains a list of all PTUs for the period of 1 day for which flexibility is wanted. Each PTU in the day can have different request information. Some PTUs will request consumption reduction or increase, and others will indicate available headroom.
* The creation of flex requests, is an internal process of the DSO and is therefore not specified by USEF. No flex requests should be generated if the last PTU for which flex is required is less than INTRADAY\_GATE\_CLOSURE\_PTUS away from the Operate phase.
* All flex requests are stored on the planboard.

The trigger to create flex requests is based on the outcome of the grid safety analysis. If PTUs within the day have as disposition status REQUESTED and Aggregators are related to that congestion point, the process is triggered taking into account the offset towards the Operate phase as mentioned above.

#### Pluggable Business Components

* Create Flex Requests

### DsoFlexOrderCoordinator

#### Process

Goal is to order flexibility to avoid congestion during some PTU(s) on a given congestion point.

When triggered:

* A decision making process is fired identifying what offers of which Aggregator to use. This internal process will typically run multiple times and will return a decision with respect to the given time context. This is an internal process of the DSO and is therefore not specified by USEF.
* A decision can be deferred if time is still left to receive better/more offers. Orders can be selected or not. No flex orders should be generated if the last PTU for which flex is required is less than INTRADAY\_GATE\_CLOSURE\_PTUS away from the Operate phase
* Selected offers become orders and are stored on the planboard.
* The orders are transferred to the corresponding Aggregator.
* If no flex orders are placed, because there are no applicable flexibility offers, the DsoColoringCoordinator is triggered to mark the appropriate PTUs Orange.

The trigger is a time based trigger that will fire with a configurable (short) interval.

#### Pluggable Business Components

* Create Flex Orders

### DsoFlexOrderAcknowledgementCoordinator

#### Process

Goal is to get a view on the order status sent to Aggregators at these Aggregators.

When triggered:

* A check is done if a response came from the Aggregator within limited time. If not, the assumption is made that the order did not reach the Aggregator and that new flex orders must be created to avoid possible congestion.
* If a response came but the order was rejected new flex orders must be created to avoid possible congestion. Aggregators rejecting an order for a given congestion point/PTU will not be contacted again for that congestion point/PTU.
* If a response came and the order is accepted by the Aggregator, the status is updated at the DSO and the order becomes binding between the DSO and the Aggregator.

The trigger is the transmission of the flex order towards an Aggregator from which a timely response is expected based on the common message framework. Depending on the outcome, the corresponding action is triggered.

#### Pluggable Business Components

* NA

### DsoFlexOfferRevocationCoordinator

#### Process

Goal is to handle the fact that an Aggregator has revoked a flex offer.

When triggered:

* A check is done if the offer exists and if all PTU’s in the flex offer are before the Operate phase.
* If so the revocation is accepted, otherwise rejected towards the AGR.

The trigger is the receipt of the flex offer revocation message from an Aggregator.

#### Pluggable Business Components

* NA

## Operate phase

This phase is described in chapter 2.3.3 and 4.3.3 of [1].

For the Aggregator the following coordinators are described in the plan phase section of this document, since the portfolio optimization and (re-)creation of prognoses processes take place in both plan and operate phase:

* AgrReOptimizePortfolioCoordinator
* AgrReCreatePrognosesCoordinator

### AgrNonUdiInitializeCoordinator

#### Process

This process only applies to Non-UDI aggregators.

Goal is to share the common reference information for a certain period with the aggregator-in-a-box (demand-response) solution the Non-UDI aggregator uses.

When triggered:

* Common reference information is retrieved for the period specified in the triggering event. All connection groups with connections that are active on the specified day are retrieved and will be sent to the aggregator-in-a-box (demand-response) solution.
* Sending this information to the aggregator-in-a-box (demand-response) solution is an internal process of the AGR and is therefore not specified by USEF.

This process is started by a time based trigger.

#### Pluggable Business Components

* Initialize Non-UDI Clusters

### AgrNonUdiRetrieveAdsGoalRealizationCoordinator

#### Process

This process only applies to Non-UDI aggregators.

Non-UDI aggregators have fairly limited control over the execution of their plans. They will have to rely on their external ADS solution to execute these to the best of its ability. However, Non-UDI aggregators monitor their ADS and feed its statistics about goal realization into the Detect Deviations from A-Plans and/or D-Prognoses process so plans can be adjusted, if necessary.

When triggered:

* For all connection groups (BRPs and congestion points) power values will be retrieved from the external ADS system and stored in the aggregator portfolio as observed power values.
* Retrieving this information from the aggregator-in-a-box (demand-response) solution is an internal process of the AGR and is therefore not specified by USEF.

The trigger for this process is a time based trigger that by default will fire once every PTU.

#### Pluggable Business Components

* Retrieve ADS Goals

### AgrDetermineNetDemandCoordinator

#### Process

Goal is to fetch detailed net monitoring information of the devices behind each connection represented by an Aggregator via the ADS managed by the Aggregator. The information serves to detect deviations from A-plan and/or D-prognoses.

When triggered:

* Net demand information is fetched for each connection managed by the Aggregator via the ADS USEF Device Interface (UDI). The monitoring of the devices via UDI is an internal process of the AGR and is therefore not specified by USEF.
* The retrieved information is stored in the portfolio database of the Aggregator. Information retrieved for each UDI linked to the connection is the actual power consumption, forecast for future DTUs and a list of energy consumption or production events that will occur and may or may not be influenced by the Aggregator. The data is delivered on DTU (device time unit being an integer subdivision of PTU) level.

This process is started by a time based trigger.

#### Pluggable Business Components

* Determine Net Demands via ADS

### AgrIdentifyChangeInForecastCoordinator

#### Process

An aggregator needs to identify relevant changes in the forecast during the Plan phase, which may impact its portfolio and the A-plan(s) and D-prognoses previously sent to other participants.

When triggered:

* A process is called that determines the forecast changes for each period in the forecast and makes the decision if the connection portfolio needs to be re-optimized for that period. This is an internal process of the AGR and is therefore not specified by USEF.
* If the process has decided that the portfolio needs to be re-optimized, the list of returned PTUs, for which the forecast has changed, is checked.
* The result has 2 options
  + If changes in the forecast for PTUs for the current period within the intraday gate closure range are detected an event is created to trigger the “Detect deviations from A-Plan and/or D-Prognoses” process. This always triggers the Re-optimize Portfolio process and, if necessary, the creation of A-plans and D-prognoses for the current period.
  + If changes in the forecast are detected for future periods, the plan phase is entered by triggering the re-optimize portfolio process for these periods.

The trigger for this process is a time based trigger that by default will fire once every PTU.

#### Pluggable Business Components

* Identify Change In Forecast

### AgrDetectDeviationCoordinator

#### Process

Goal is to compare the actual and forecast usage from the portfolio obtained via the connection monitoring process with the current D-prognoses and A-plans. This to make sure that all contractual obligations still can be met. If not, the portfolio re-optimization process is triggered to steer demand in the right direction.

When triggered:

* For all active connection groups for the given day, the prognoses and forecast information are retrieved and transferred to the decision process to detect deviations.
* This decision process is an internal process of the AGR and is therefore not specified by USEF.
* As soon as a deviation is detected in any of the prognoses a re-optimization of the Aggregator portfolio is triggered.

This process is triggered when changes in the forecast for PTUs for the current period within the intraday gate closure range are detected.

#### Pluggable Business Components

* Detect Prognoses Deviations
* Non-UDI Detect Prognoses Deviations

### AgrControlActiveDemandSupplyCoordinator

#### Process

Goal is to realize the optimization of the portfolio by sending the device messages created by the portfolio optimization process to ADS endpoints. By doing so, an Aggregator can steer consumption and/or production on the connections it represents via the ADS equipment.

When triggered:

* The device messages are read out of the aggregator’s portfolio.
* All new device messages are sent to the ADS devices. Since the communication protocols and business logic are aggregator specific, this logic is not added in the USEF reference implementation but is represented by a Pluggable Business Component.
* Note that previously failed device messages are not sent again.

The trigger for this process is a time based trigger that by default will fire once every PTU.

#### Pluggable Business Components

* Send ADS Messages

### AgrNonUdiSetAdsGoalsCoordinator

#### Process

This process only applies to Non-UDI aggregators.

A Non-UDI aggregator communicates detailed ADS goals to his "aggregator in a box" solution. This is done every time a new A-plan or D-prognosis is created.

When triggered:

* The newly created A-plan or D-prognosis is transformed and sent to the "aggregator in a box" solution.
* Since the "aggregator in a box" solutions are AGR specific, these are not added in the USEF reference implementation and are represented by a Pluggable Business Component.

This process is triggered every time an A-plan or D-prognosis is (re-)created.

#### Pluggable Business Components

* Send ADS Goals

### DsoOperateCoordinator

#### Process

Goal is to monitor the grid operated by the DSO in the operate phase and identify and resolve possible congestion on the grid. This is done by purchasing any available flex orders, but it may result in graceful degradation in case insufficient flexibility is available.

When triggered:

* Monitor all congestion points and record the actual and maximum load values per-PTU and an indication if congestion is detected and store these values on the planboard. The grid monitoring process is an internal process of the DSO and is therefore not specified by USEF.
* In case congestion is detected, verify if valid flexibility offers are still available and, if possible, place flex orders in order to resolve the congestion. Change the regime to Yellow. The purchasing of flex orders is an internal process of the DSO and is therefore not specified by USEF.
* In case congestion is not resolved by purchasing flex orders, connections need to be limited until the actual load and purchased flex orders do not exceed the maximum load anymore. The regime is changed to Orange. Limiting connections is an internal process of the DSO and is therefore not specified by USEF.
* If congestion is no longer ongoing while operating under the Orange regime, previously limited connections can be restored to increase grid capacity again. The regime is changed back to Yellow this case. Restoring connections is an internal process of the DSO and is therefore not specified by USEF.

The trigger is a time based trigger that will fire multiple times per PTU (for instance every minute).

#### Pluggable Business Components

* Monitor Grid
* Place Operate Flex Orders
* Limit Connections
* Restore Connections

## Settlement phase

This phase is described in chapter 2.3.4 and 4.3.4 of [1].

### AgrInitiateSettlementMessageCoordinator

#### Process

Goal is to prepare the settlement process before settlement messages are received from the DSO and BRP. When both the preparation is done and the settlement message is received from the DSO or BRP, the settlement is processed.

The preparation of the settlement consists of the following steps:

* Gather the details of all acknowledged flex orders placed during the settlement period, grouped by day and congestion point or BRP.
* Retrieve the load forecast in the latest D-prognosis and the details (amount of the flexibility sold) of the flex orders associated with it.
* For each settlement item, the delivered flex quantity and power deficiency for the PTU is calculated. This is an internal process of the AGR and is therefore not specified by USEF.
* For each settlement item, the price per PTU is calculated. Settlement price is equal to the order price, unless the delivered flex power is less than ordered, in which case the price = delivered power/ordered power \* order price.
* Store this information in the database to be validated later upon receipt of the settlement message sent by the DSO or BRP.
* Store records for expected settlement messages in the database with status SettlementPending.

The trigger for the settlement preparation is a time based trigger that will fire on a configurable day each month.

#### Pluggable Business Components

* Initiate Settlement

### AgrReceiveSettlementMessageCoordinator

#### Process

Goal is to validate and process the settlement messages sent by the DSO and BRP.

These settlement messages contain details of purchased flex and fines for non-delivery of such, and serve as the basis for invoices to the DSOs and BRPs.

When both the preparation is done and the settlement message is received from the DSO or BRP, the settlement is processed.

Processing of the settlement consists of the following steps:

* Check if the preparation of the settlement is done by the aggregator and if the appropriate settlement messages are received from the DSO or BRP. If so:
  + - Compare the settlement message with the preparation performed earlier and perform a validation and compare the amount of flex sold and actually delivered for all settlement items in the messages. If necessary and possible, the actual consumption/production per PTU can be retrieved and used in the validation. This is an internal process of the Aggregator and is therefore not specified by USEF.
    - Based on the outcome of these checks and the content of the settlement message, the settlement is accepted or disputed by adding this information and details on the settlement items to the response messages.
    - Send the settlement response messages to the appropriate DSOs and BRPs.

This process is triggered by the message controller after it has received and processed a settlement message from a DSO or BRP.

#### Pluggable Business Components

* Validate Settlement Items

### BrpInitiateSettlementCoordinator

#### Process

The start of the settlement process consists of the following steps:

* For each day in the settlement period where flex ordering took place, create a list of entity addresses of connections that were active during that day.
* Send Meter Data Query messages for each of these day to the MDCs to retrieve the consumption per PTU for these connections.

The trigger for the settlement preparation is a time based trigger that will fire on a configurable day each month.

The settlement process continues after reception of all responses from the MDC. The following steps are performed:

* Receive and process the Meter Data Query Response from the MDC.
* Create a list of settlement items based on the previous month’s flex orders.
* For each settlement item, the consumption per PTU as received from the MDC is stored.
* For each settlement item, the delivered flex quantity and power deficiency for the PTU is calculated. This is an internal process of the BRP and is therefore not specified by USEF.
* For each settlement item, the price per PTU is calculated. Settlement price is equal to the order price, unless the delivered flex power is less than ordered, in which case the price = delivered power/ordered power \* order price.
* For each aggregator penalties are calculated and stored for all PTUs in the settlement period. This is an internal process of the BRP and is therefore not specified by USEF.
* Finally, when response messages have been received from the MDC for each requested day, the BrpSendSettlementCoordinator is triggered that will send the settlement messages.

There are two triggers for the processing of the settlement.

* One trigger is fired by the message controller that receives a Meter Data Query Response message from the MDC.
* If not all Meter Data Query Response messages are received within a configurable number of hours after the preparation, the settlement process stops. If a Meter Data Query Response message is received after this trigger is fired, this data will not be used.

#### Pluggable Business Components

* Initiate Settlement
* Calculate Penalty Amount

### BrpSendSettlementMessagesCoordinator

#### Process

Goal is to send settlement messages to aggregators, containing information on purchased flex orders and penalties that was prepared earlier.

When triggered:

* Collect all settlement items grouped by aggregator for the settlement period and group them into one settlement message per aggregator.
* Send the settlement messages to the appropriate aggregators.

The trigger is an event that is fired by the BrpInitiateSettlementCoordinator when he is finished gathering all required settlement information.

#### Pluggable Business Components

* NA

### BrpSettlementMessageResponseCoordinator

#### Process

Goal is to process the settlement response messages from aggregators.

When triggered:

* Read the settlement response messages from the aggregators, containing the result of the settlement. If the result is “Accepted”, all settlement items in the response are evaluated and all statuses are updated accordingly. Missing settlement items are considered “Disputed”.
* If the result of the response message is “Rejected” or if no response message is received within a configurable time period (e.g. 5 days), the result of the settlement is set to “Rejected” and all related settlement items are set to “Disputed”.
* Handling disputes and invoices is not in scope of the reference implementation.

The trigger is a message controller that receives a settlement response message from an aggregator.

#### Pluggable Business Components

* NA

### DsoInitiateSettlementCoordinator

#### Process

Goal is to initiate the settlement process, on a configurable day each month, this process generates settlement items based on the previous month’s flex orders.

The start of the settlement process consists of the following steps:

* For each day in the settlement period where flex ordering took place, create a list of entity addresses of connections that were active during that day.
* Send Meter Data Query messages for each of these day to the MDCs to retrieve the consumption per PTU for these connections.

The trigger for the settlement preparation is a time based trigger that will fire on a configurable day each month.

The settlement process continues after reception of all responses from the MDC. The following steps are performed:

* Receive and process the Meter Data Query Response from the MDC.
* Create a list of settlement items based on the previous month’s flex orders
* For each settlement item, the consumption per PTU as received from the MDC is stored
* For each settlement item, the delivered flex quantity and power deficiency for the PTU is calculated. This is an internal process of the DSO and is therefore not specified by USEF.
* For each settlement item, the price per PTU is calculated. Settlement price is equal to the order price, unless the delivered flex power is less than ordered, in which case the price = delivered power/ordered power \* order price.
* For each aggregator penalties are calculated and stored for all congestion points and all PTUs in the settlement period. This is an internal process of the DSO and is therefore not specified by USEF.
* Finally, when response messages have been received from the MDC for each requested day, the DsoSendSettlementCoordinator is triggered that will send the settlement messages.

There are two triggers for the processing of the settlement.

* One trigger is fired by the message controller that receives a Meter Data Query Response message from the MDC.
* If not all Meter Data Query Response messages are received within a configurable number of hours after the preparation, a trigger is fired and the settlements will be created based on data from grid monitoring instead of real meter data. If a Meter Data Query Response message is received after this trigger is fired, this data will not be used.

#### Pluggable Business Components

* Initiate Settlement
* Calculate Penalty Amount

### DsoSendSettlementCoordinator

#### Process

Goal is to send settlement messages to aggregators, containing information on purchased flew orders and fines that was prepared earlier.

When triggered:

* Collect all settlement items grouped by aggregator for the settlement period and group them into one settlement message per aggregator.
* Send the settlement messages to the appropriate aggregators.

The trigger is an event that is fired by the DsoInitiateSettlementCoordinator when he is finished gathering all required settlement information.

#### Pluggable Business Components

* NA

### DsoSettlementMessageResponseCoordinator

#### Process

Goal is to process the settlement response messages from aggregators.

When triggered:

* Read the settlement response messages from the aggregators, containing the result of the settlement. If the result is “Accepted”, all settlement items in the response are evaluated and all statuses are updated accordingly. Missing settlement items are considered “Disputed”.
* If the result of the response message is “Rejected” or if no response message is received within a configurable time period (e.g. 5 days), the result of the settlement is set to “Rejected” and all related settlement items are set to “Disputed”.
* Handling disputes and invoices is not in scope of the reference implementation.

The trigger is a message controller that receives a settlement response message from an aggregator.

#### Pluggable Business Components

* NA

### DsoCollectOrangeRegimeDataCoordinator

#### Process

Goal is to initiate the settlement process between the DSO and its customers, on a configurable day each month. This process determines the extent and duration of each capacity reduction event that occurred during a settlement period, in order to compensate its customers for those events.

The settlement process starts with the following steps:

* Determine the PTUs in the period prior to the current settlement period in which the regime was Orange.
* Determine the list of connections (from historic data) affected by the regime.
* Send Meter Data Query messages to the all MDCs available to this DSO to retrieve the reduction and outage meter events for these connections per day. The messages are sent on one day basis.

The trigger for the settlement preparation is a time based trigger that will fire on a configurable day each month.

The settlement process continues after reception of a response from the MDC. The following steps are performed:

* Receive and process the Meter Data Query Response from the MDC.
* For each connection, the reduction and outage meter events as received from the MDC are stored.
* In case two MDCs sent data for the same connection, the first wins.
* Finally, after receiving the last MDC response and generating missing data, the DsoDetermineCapacityLimitationPeriodsCoordinator is triggered that will process send the settlement messages.

There are two triggers for the processing of the settlement.

* One trigger is fired by the message controller that receives a Meter Data Query Response message from the MDC.
* In case MDCs does not deliver connection event data for some a trigger is fired and the meter events are generated by the DSO. This is an internal process of the DSO and is therefore not specified by USEF. The decision whether to invoke this process is taken after receiving of the last MDC response.

#### Pluggable Business Components

* Generate Connection Meter Events

### DsoDetermineCapacityLimitationPeriodsCoordinator

#### Process

Goal is to process the gathered capacity reduction and outage meter events for the settlement period and calculate compensation due to prosumers that were affected by the orange regime.

When triggered:

* The DSO converts the retrieved capacity reduction and outage meter events into capacity reduction and outage periods and stores the period details for further processing. The process checks the data against internal records, determining if a sequence of events classifies as a capacity reduction or outage, and emits a list of actual capacity reduction and outage periods. Generation of these capacity reduction and outage periods is done by two internal processes of the DSO and is therefore not specified by USEF.
* The generated capacity reduction and outage periods that occurred in a given settlement period are then processed. The DSO calculates compensations due to each prosumer and posts the compensations to an accounting system. This is an internal process of the DSO and is therefore not specified by USEF.

The trigger is an event that is fired by the DsoCollectOrangeRegimeDataCoordinator when that is finished collecting all orange regime data.

#### Pluggable Business Components

* Determine Outage Durations
* Determine Reduction Periods
* Calculate Compensations

### MdcMeterDataQueryCoordinator

#### Process

Goal is to process the meter data query that was received from a participant.

When triggered:

* Process the Meter Data Query message.
* Depending on the type of request, retrieve the actual consumed or produced power value or the meter data events for each connection during the specified data range provided in the query. This is an internal process of the MDC and is therefore not specified by USEF.
* Generate a Meter Data Query Response message with the retrieved power values or meter data events.

The trigger is a message controller that receives a Meter Data Query message from a participant.

#### Pluggable Business Components

* Meter Data Query

# Inter actor message flows

Chapter 6 of [1] goes in detail on the message transport and message description specifications.

This chapter:

* gives a view on the message exchange mechanism in the reference implementation in line with chapter 6.1 of [1].
* lists all message transport flows that follow the common in- and outbound message flow
* shows the relation between the interaction flows listed in the system context chapter (an outbound message flow from a sender will trigger an inbound message flow at a receiver)
* makes a reference to the message descriptions contained in chapter 6.3 of [1].

## Message exchange mechanism

The reference implementation offers a message exchange mechanism in line with the USEF specifications that is illustrated in chapter 6.1 of [1] common for all roles.

For this, the reference implementation:

* uses a point-to-point integration between the different actors exchanging messages between each other. Opting for a hub and spoke model would introduce the dependency of an organization providing the central hub that would also be accountable for the good distribution of messages.
* uses a synchronous message exchange mechanism. It is a good practice to perform a set of message validations on inbound messages before allowing them into the backend and trigger business logic. The performed validations are shown in the common inbound message flow sequence diagram. The result of these validations is returned to the sender.

The synchronous message exchange is realized by using REST over https.

* decouples message exchange from actual business logic execution.

Performing business logic triggered by an inbound message in a synchronous way will lead to all kind of performance and resource issues as the reply to the sender will only be sent after the business logic has finished.

Sending messages at the end of some business execution in a synchronous way will also lead to all kind of performance and resource issues and will among others lead to extra complexity for retries, error handling etc...

The decoupling is realized by using queues as shown in the USEF integration layer component model.

* offers one inbound and one outbound channel per actor instance. These single in- and outbound channels are used for all exchanged messages. As a consequence each actor instance will have one REST endpoint to receive inbound messages, one inbound message queue and one outbound message queue.
* seals the message payload using the NaCl library as described in chapter 6.1.2 of [1].

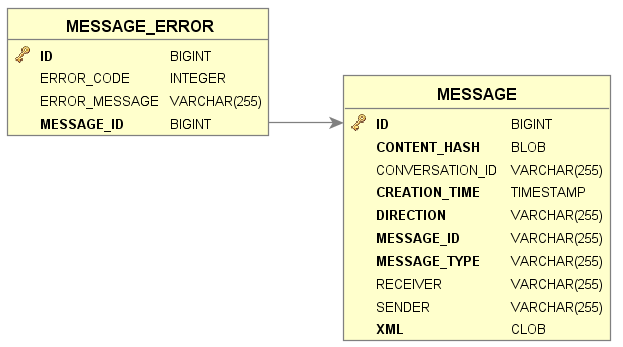
## Inter actor messages

|  |  |  |
| --- | --- | --- |
| **Sender** | **Receiver** | **[1] Schema reference** |
| **A\_DSO**  Send DSO Common Reference update. | **A\_CRO**  Receive DSO updates. | Chapter 6.4.7  CommonReferenceUpdate  Chapter 6.4.8 CommonReferenceUpdateResponse |
| **A\_DSO**  Query Common Reference. | **A\_CRO**  Receive DSO query. | Chapter 6.4.9  CommonReferenceQuery  Chapter 6.4.12 CommonReferenceQueryResponse |
| **A\_DSO**  Send Flex Requests. | **A\_AGR**  Receive Flex Requests. | Chapter 6.4.17  FlexRequest  Chapter 6.4.18 FlexRequestResponse |
| **A\_AGR**  Send AGR Common Reference update. | **A\_CRO**  Receive AGR updates. | Chapter 6.4.7  CommonReferenceUpdate  Chapter 6.4.8 CommonReferenceUpdateResponse |
| **A\_AGR**  Query Common Reference. | **A\_CRO**  Receive AGR query. | Chapter 6.4.9  CommonReferenceQuery  Chapter 6.4.12 CommonReferenceQueryResponse |
| **A\_AGR**  Send D-prognoses. | **A\_DSO**  Receive D-prognoses. | Chapter 6.4.14  Prognosis  Chapter 6.4.16  PrognosisResponse |
| **A\_AGR**  Send Flex Offers. | **A\_DSO**  Receive Flex Offers. | Chapter 6.4.19  FlexOffer  Chapter 6.4.20  FlexOfferResponse |
| **A\_DSO**  Send Flex Orders. | **A\_AGR**  Receive Flex Orders. | Chapter 6.4.23  FlexOrder  Chapter 6.4.24  FlexOrderResponse |
| **A\_AGR**  Revoke Flex Offers. | **A\_DSO**  Receive Flex Offers revocation | Chapter 6.4.21  FlexOfferRevocation  Chapter 6.4.22  FlexOfferRevocationResponse |
| **A\_DSO**  Send Settlement Message | **A\_AGR**  Receive Settlement Message | Chapter 6.4.25  SettlementMessage  Chapter 6.4.28  SettlementMessageResponse |
| **A\_BRP**  Send BRP Common Reference update. | **A\_CRO**  Receive BRP updates. | Chapter 6.4.7  CommonReferenceUpdate  Chapter 6.4.8 CommonReferenceUpdateResponse |
| **A\_ BRP**  Query Common Reference. | **A\_CRO**  Receive BRP query. | Chapter 6.4.9  CommonReferenceQuery  Chapter 6.4.12 CommonReferenceQueryResponse |
| **A\_AGR**  Send A-plan. | **A\_BRP**  Receive A-plan. | Chapter 6.4.14  Prognosis  Chapter 6.4.16  PrognosisResponse |
| **A\_BRP**  Send Flex Requests. | **A\_AGR**  Receive Flex Requests. | Chapter 6.4.17  FlexRequest  Chapter 6.4.18 FlexRequestResponse |
| **A\_AGR**  Send Flex Offers. | **A\_BRP**  Receive Flex Offers. | Chapter 6.4.19  FlexOffer  Chapter 6.4.20  FlexOfferResponse |
| **A\_BRP**  Send Flex Orders. | **A\_AGR**  Receive Flex Orders. | Chapter 6.4.23  FlexOrder  Chapter 6.4.24  FlexOrderResponse |
| **A\_AGR**  Revoke Flex Offers. | **A\_BRP**  Receive Flex Offers revocation | Chapter 6.4.21  FlexOfferRevocation  Chapter 6.4.22  FlexOfferRevocationResponse |
| **A\_BRP**  Send Settlement Message | **A\_AGR**  Receive Settlement Message | Chapter 6.4.25  SettlementMessage  Chapter 6.4.28  SettlementMessageResponse |
| **A\_ MDC**  Query Common Reference. | **A\_CRO**  Receive BRP query. | Chapter 6.4.9  CommonReferenceQuery  Chapter 6.4.12 CommonReferenceQueryResponse |
| **A\_ DSO**  Query Meter Data | **A\_MDC**  Receive Meter Data query | Chapter 6.4.30  MeterDataQuery  Chapter 6.4.36  MeterDataQueryResponse |
| **A\_ BRP**  Query Meter Data | **A\_MDC**  Receive Meter Data query | Chapter 6.4.30  MeterDataQuery  Chapter 6.4.36  MeterDataQueryResponse |

# Data store schemas

## Message store

See the LC\_Message log and LC\_Exception message log components for a description of the need and usage of this schema.



1. Message store

* MESSAGE table: Each in- and outbound message is stored in this table.

CONTENT\_HASH: Hashed content of the message.

CONVERSATION\_ID: Unique identifier used to correlate responses with requests.

CREATION\_TIME: Date time of message creation

DIRECTION: Indicator on in- or outbound message

MESSAGE\_ID: The unique identifier of the message

MESSAGE\_TYPE: Indicates the importance and impact of the message,

RECEIVER: Identifies the receiver of the message.

SENDER: Identifies the sender of the message.

XML: The message.

For a more elaborated description of these fields and how they are used, see chapter 6.4.3 of [1].

* MESSAGE\_ERROR table

ERROR\_CODE: Code identifying the type of error.

ERROR\_MESSAGE: A description describing the error.

The message error table stores the message exceptions as listed in chapter 6.2.5.1 of [1].

## Hash store

See the LC\_Message hash log component for a description of the need and usage of this schema.



1. Hash store

* SIGNED\_MESSAGE\_HASH table

HASHED\_CONTENT: Hashed content of the message.

## Common reference

See the LC\_Common reference component for a description of the need and usage of these schemas.

### CRO common reference

The CRO schema stores the connection and congestion point information as received from the different USEF participants. Each participant has a daily process that sends changes in this information to the CROs it is connected to.

A DSO will pass its congestion point and related connections, while an AGR and BRP will give information on what connections they are active on. Note that not all connections need to be linked to a congestion point.

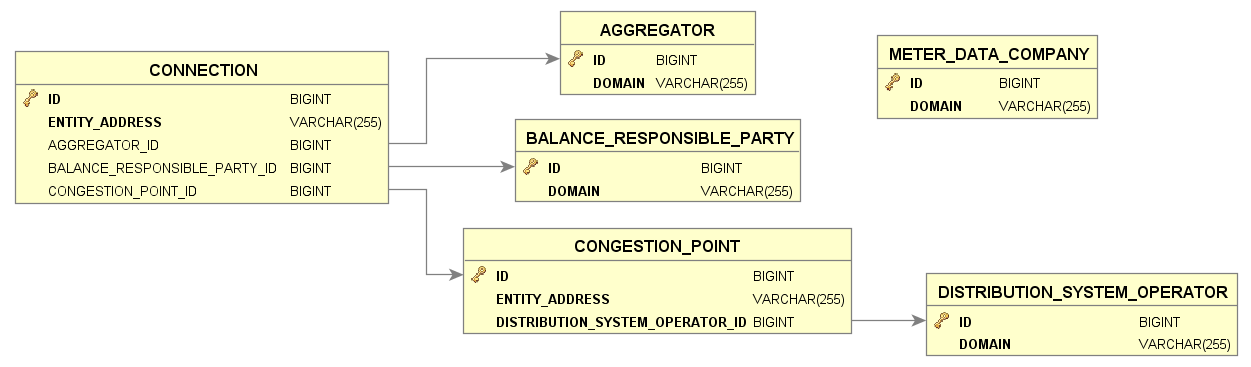
When a DSO indicates a congestion point can be removed, it will be removed from the common reference. Also links from connections to this removed congestion point are removed.

When an AGR or BRP indicates they are no longer active on a certain connection, the link between the connection and that AGR or BRP is removed.

If a connection is neither linked to a congestion point, nor to a BRP, nor to an AGR, it is removed as well.

The CRO stores a reference to the MDC it can receive requests from. Within the common reference there is only one MDC participant.

A DSO, BRP, AGR and MDC can then query the CRO database to retrieve the common reference information that is applicable for them and perform their respective business logic.



1. CRO common reference

* AGGREGATOR table

DOMAIN: Identifies an AGR.

* BALANCE\_RESPONSIBLE\_PARTY table

DOMAIN: Identifies a BRP.

* DISTRIBUTION\_SYSTEM\_OPERATOR table

DOMAIN: Identifies a DSO.

* METER\_DATA\_COMPANY table

DOMAIN: Identifies an MDC. There is only 1 MDC.

* CONNECTION table

ENTITY\_ADDRESS: Identifies the connection associated to a congestion point.

AGGREGATOR\_ID: Identifies the AGR active on the connection.

BALANCE\_RESPONISBLE\_PARTY\_ID: Identifies the BRP active on the connection.

CONGESTION\_POINT\_ID: Identifies the congestion point the connection is associated with.

* CONGESTION\_POINT table

ENTITY\_ADDRESS: Identifies the congestion point.

DISTRIBUTION\_SYSTEM\_OPERATOR\_ID: Identifies the DSO associated with the congestion point.

### DSO common reference

A DSO is mainly interested in all congestion points (for flex trading with the AGR). The DSO will receive and store congestion point and related AGR information from the CRO (which aggregators are active on a congestion point and how many connections do they represent).

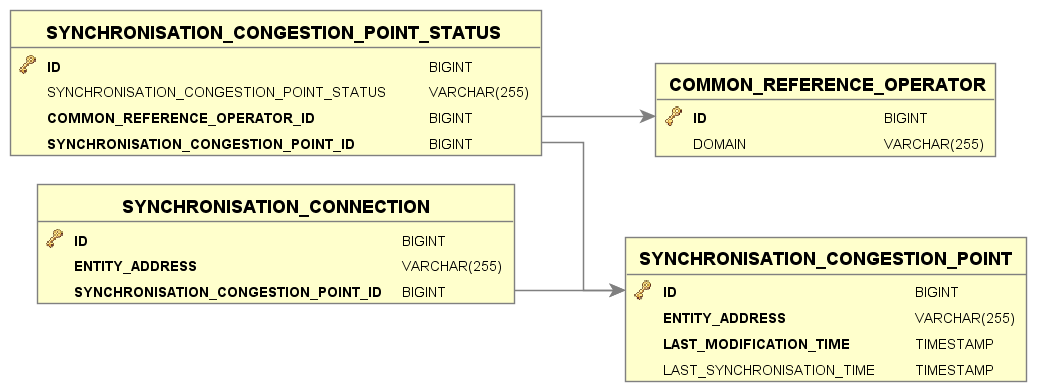
The DSO will also receive connection information from the CRO, but without specification of the representing BRP and AGR. This information is only used to define the relations between connections and congestion points.

The common reference is split in two distinct parts, one for sending updates to the CRO and one for storing information received from the CRO

The first part is for sending updates on the common reference to the CROs and keeping track of the responses from CROs. Each participant has a daily process that sends changes in this information to the CROs it is connected to.

After the updates are sent and confirmed by the CROs, data in this part is deleted.

To delete a congestion point as DSO, the DSO sends a message with status DELETED containing the congestion point without any associated connections.



1. DSO common reference – synchronise updates to CRO

The common reference related to updates for the DSOs consists of the following tables:

* SYNCHRONISATION\_CONNECTION table: used to communicate connection updates to the CROs. When an update is sent to and confirmed by all CROs, the actual connection update and all related synchronisation records are removed from this table.

ENTITY\_ADDRESS: Identifies a connection.

SYNCHRONISATION\_CONGESTION\_POINT\_ID: Identifies the related congestion point.

* SYNCHRONISATION\_CONGESTION\_POINT table: used to communicate congestion point updates to the CROs. When an update is sent to and confirmed by all CROs, the actual congestion point update and all related connections and synchronisation records are removed from this table.

ENTITY\_ADDRESS: Identifies a connection.

LAST\_SYNCHRONISATION\_TIME: A date time field containing the last date time this congestion point record was synchronized with the CRO.

LAST\_MODIFICATION\_TIME: A date time field containing the last modification date time.

* SYNCHRONISATION\_CONGESTION\_STATUS table: Stores status information on synchronisation of updates to the CRO. When an update is sent to and confirmed by the CRO, the actual congestion point update and all related connections and synchronisation records are removed from this table.

COMMON\_REFERENCE\_OPERATOR\_ID: Identifies the associated Common Reference Operator.

SYNCHRONISATION\_CONGESTION\_POINT\_ID: Identifies the associated congestion point.

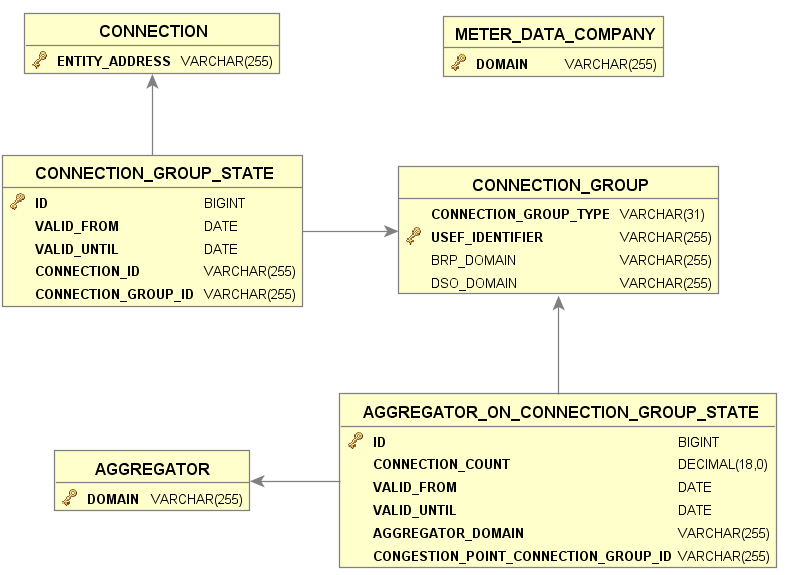
SYNCHRONISATION\_CONGESTION\_POINT\_STATUS: Possible values: MODIFIED, DELETED, SYNCHRONIZED.  
Identifies if a congestion point is MODIFIED or DELETED and registers the confirmation of the update at the CRO.  
In case the update is confirmed by the CRO, the status field is set to SYNCHRONIZED.  
In case the update is not confirmed for whatever reason the updated records will be transferred again the next time to the CRO as the status is not changed (MODIFIED).

* COMMON\_REFERENCE\_OPERATOR table. This table must be filled by the DSO.

DOMAIN: Identifies a CRO; a DSO has 1 CROs.

The second part is for storing the common reference information; connections and congestion points as received from the CROs as a result of a common reference query. This data is used to within the business logic of the AGR.

The DSO stores a reference to the MDC from which it will request meter data. Within the common reference there is only one MDC participant.



1. DSO common reference – process common reference query

* CONNECTION table: Identifies a connection in the grid.

ENTITY\_ADDRESS: Identifies a connection.

* CONNECTION\_GROUP\_STATE table: Links connections to connection groups and keeps historical data.

VALID\_FROM: Start date from which time onwards the connection is part of the connection group.

VALID\_UNTIL: Date when the connection was no longer part of the connection group.

CONNECTION\_ID: Identifies the associated connection.

CONNECTION\_GROUP\_ID: Identifies the associated connection group.

* CONNECTION\_GROUP table: Identifies a group to which a number of connections belong. Connections can be grouped by congestion point, but also by BRP.

USEF\_IDENTIFIER: Unique identifier associated with the connection group. For congestion points it is filled with the entity address associated with the congestion point, for BRPs it is filled with the domain.

CONNECTION\_GROUP\_TYPE: Identifies the type of connection group (“CONGESTION\_POINT”, “BRP”, “AGR”)

BRP\_DOMAIN: Identifies the associated Balance Responsible Party.

DSO\_DOMAIN: Identifies the associated Distributed System Operator.

* AGGREGATOR\_ON\_CONNECTION\_GROUP\_STATE table: Links the connection group to an aggregator and specifies the number of connections the aggregator represents in this connection group.

CONNECTION\_COUNT: Number of connections the aggregator represents in this connection group.

VALID\_FROM: Start date from which time onwards the connection is part of the connection group.

VALID\_UNTIL: Date when the connection was no longer part of the connection group.

AGGREGATOR\_DOMAIN: Identifies the associated aggregator.

CONGESTION\_POINT\_CONNECTION\_GROUP\_ID: Identifies the associated connection group.

* AGGREGATOR table: Identifies the aggregators the DSO does business with.

DOMAIN: Identifies the associated Aggregator.

* METER\_DATA\_COMPANY table: Identifies the meter data company that is used for meter data queries. This table must be filled by the DSO.

DOMAIN: Identifies an MDC. There is only 1 MDC.

### AGR common reference

An AGR is interested in all congestion points (for flex trading with the DSO) and all connections (for flex trading with the BRP).

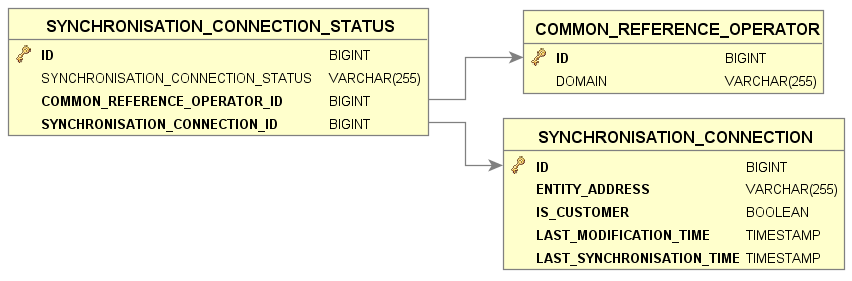
Note that not all connections need to be linked to a congestion point.

The common reference is split in two distinct parts, one for sending updates to the CRO and one for storing information received from the CRO

An AGR is connected to one or more CROs.

The first part is for sending updates on the common reference to the CROs and keeping track of the responses from CROs. Each participant has a daily process that sends changes in this information to the CROs it is connected to.

After the updates are sent and confirmed by the CROs, data in this part is deleted.



1. AGR common reference – synchronise updates to CRO

The common reference related to updates for the AGRs consists of the following tables:

* SYNCHRONISATION\_CONNECTION table: used to communicate connection updates to the CROs. When an update is sent to and confirmed by all CROs, the actual connection update and all related synchronisation records are removed from this table.

ENTITY\_ADDRESS: Identifies a connection.

IS\_CUSTOMER: Flag indicating if a connection is linked to the AGR or not.

LAST\_SYNCHRONISATION\_TIME: A date time field containing the last date time this connection record was synchronized with the CRO.

LAST\_MODIFICATION\_TIME: A date time field containing the last modification date time.

* SYNCHRONISATION\_CONNECTION\_STATUS table: Stores status information on synchronisation of updates to all CROs. When an update is sent to and confirmed by all CROs, the actual connection update and all related synchronisation records are removed from this table.

SYNCHRONISATION\_CONNECTION\_ID: Identifies the associated connection.

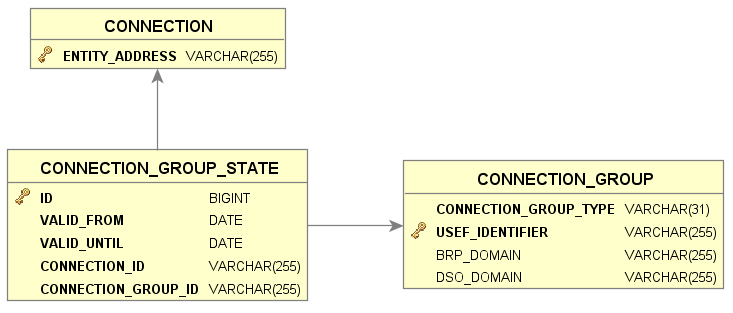
COMMON\_REFERENCE\_OPERATOR\_ID: Identifies the associated Common Reference Operator.

SYNCHRONISATION\_CONNECTION\_STATUS: : Possible values: MODIFIED, SYNCHRONIZED.  
Identifies if a connection point is modified and registers the confirmation of the update at all CROs.  
In case the update is confirmed by the CRO, the status field is set to SYNCHRONIZED.  
In case the update is not confirmed for whatever reason the updated records will be transferred again the next time to the CRO as the status is not changed (MODIFIED).

* COMMON\_REFERENCE\_OPERATOR table. This table must be filled by the AGR.

DOMAIN: Identifies a CRO; an AGR can have 1 to many CROs.

The second part is for storing the common reference information (connections and congestion points) as received from the CROs as a result of a common reference query. This data is used to within the business logic of the AGR.



1. AGR common reference – process common reference query

* CONNECTION table: Identifies a connection in the grid.

ENTITY\_ADDRESS: Identifies a connection.

* CONNECTION\_GROUP table: Identifies a group to which a number of connections belong. Connections can be grouped by congestion point, but also by BRP.

USEF\_IDENTIFIER: Unique identifier associated with the connection group. For congestion points it is filled with the entity address associated with the congestion point, for BRPs it is filled with the domain.

CONNECTION\_GROUP\_TYPE: Identifies the type of connection group (“CONGESTION\_POINT”, “BRP”, “AGR”)

BRP\_DOMAIN: Identifies the associated Balance Responsible Party.

DSO\_DOMAIN: Identifies the associated Distributed System Operator.

* CONNECTION\_GROUP\_STATE table: Links connections to connection groups and keeps historical data.

CONNECTION\_GROUP\_ID: Identifies the associated connection group.

CONNECTION\_ID: Identifies the associated connection.

VALID\_FROM: Start date from which time onwards the connection is part of the connection group.

VALID\_UNTIL: Date when the connection was no longer part of the connection group.

### BRP common reference

A BRP is mainly interested in all connections (for flex trading with the AGR).

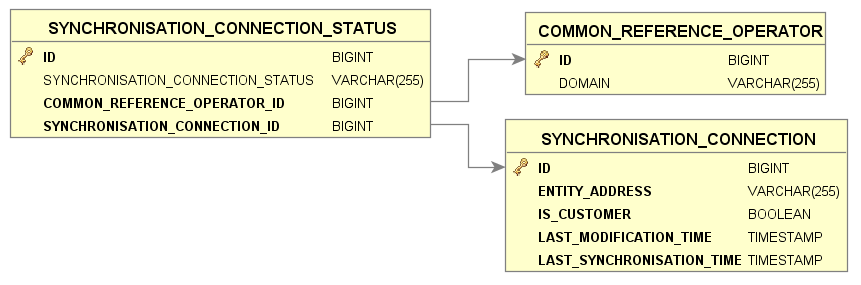
The BRP does not receive congestion point information from the CRO.

The common reference is split in two distinct parts, one for sending updates to the CRO and one for storing information received from the CRO

A BRP is connected to one or more CROs.

The first part is for sending updates on the common reference to the CROs and keeping track of the responses from CROs. Each participant has a daily process that sends changes in this information to the CROs it is connected to.

After the updates are sent and confirmed by the CROs, data in this part is deleted.



1. BRP common reference – synchronise updates to CRO

The common reference related to updates for the BRPs consists of the following tables:

* SYNCHRONISATION\_CONNECTION table: used to communicate connection updates to the CROs. When an update is sent to and confirmed by all CROs, the actual connection update and all related synchronisation records are removed from this table.

ENTITY\_ADDRESS: Identifies a connection.

IS\_CUSTOMER: Flag indicating if a connection is linked to the BRP or not.

LAST\_SYNCHRONISATION\_TIME: A date time field containing the last date time this connection record was synchronized with the CRO.

LAST\_MODIFICATION\_TIME: A date time field containing the last modification date time.

* SYNCHRONISATION\_CONNECTION\_STATUS table: Stores status information on synchronisation of updates to all CROs. When an update is sent to and confirmed by all CROs, the actual connection update and all related synchronisation records are removed from this table.

SYNCHRONISATION\_CONNECTION\_ID: Identifies the associated connection.

COMMON\_REFERENCE\_OPERATOR\_ID: Identifies the associated common reference.

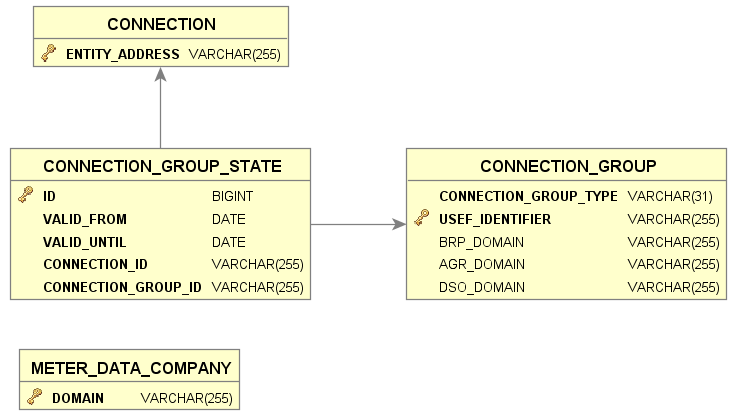
SYNCHRONISATION\_CONNECTION\_STATUS: Possible values: MODIFIED, SYNCHRONIZED.  
Identifies if a connection point is modified and registers the confirmation of the update at all CROs.   
In case the update is confirmed by the CRO, the status field is set to SYNCHRONIZED.   
In case the update is not confirmed for whatever reason the updated records will be transferred again the next time to the CRO as the status is not changed (MODIFIED).

* COMMON\_REFERENCE\_OPERATOR table. This table must be filled by the BRP.

DOMAIN: Identifies a CRO; a BRP can have 1 to many CROs.

The second part is for storing the common reference information (connections and congestion points) as received from the CROs as a result of a common reference query. This data is used to within the business logic of the BRP.

The BRP stores a reference to the MDC from which it will request meter data. Within the common reference there is only one MDC participant.



1. BRP common reference – process common reference query

The common reference related to information received from the CROs and that is used in the BRP business processes, consists of the following tables:

* CONNECTION table: Identifies a connection in the grid.

ENTITY\_ADDRESS: Identifies a connection.

* CONNECTION\_GROUP table: Identifies a group to which a number of connections belong. Connections can be grouped by congestion point, but also by AGR.

USEF\_IDENTIFIER: Unique identifier associated with the connection group. For congestion points it is filled with the entity address associated with the congestion point, for AGRs it is filled with the domain.

CONNECTION\_GROUP\_TYPE: Identifies the type of connection group (“CONGESTION\_POINT”, “BRP”, “AGR”)

BRP\_DOMAIN: Identifies the associated Balance Responsible Party.

DSO\_DOMAIN: Identifies the associated Distributed System Operator.

AGR\_DOMAIN: Identifies the associated Aggregator.

* CONNECTION\_GROUP\_STATE table: Links connections to connection groups and keeps historical data.

CONNECTION\_GROUP\_ID: Identifies the associated connection group.

CONNECTION\_ID: Identifies the associated connection.

VALID\_FROM: Start date from which time onwards the connection is part of the connection group.

VALID\_UNTIL: Date when the connection was no longer part of the connection group.

* METER\_DATA\_COMPANY table: Identifies the meter data company that is used for meter data queries. This table must be filled by the BRP.

DOMAIN: Identifies an MDC.

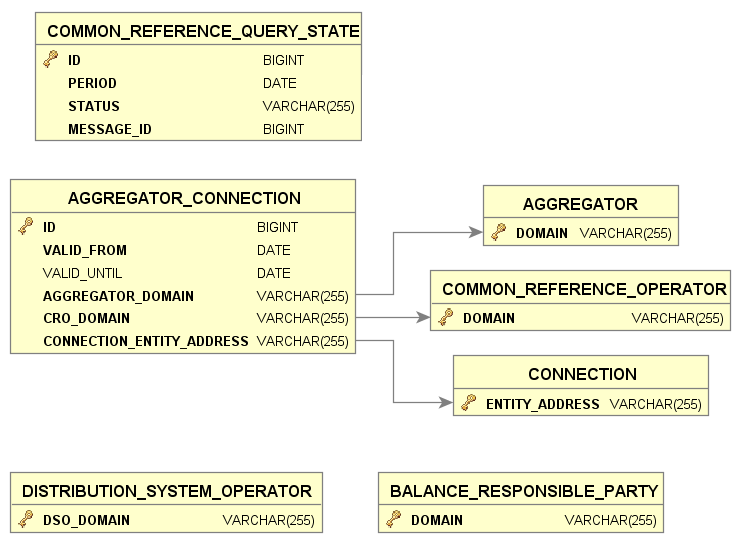
### MDC common reference

The Meter Data Company (MDC) is responsible for acquiring and validating meter data for connections registered at the MDC. The Meter Data Company (MDC) does not send updates to the CRO, it only stores information received from the CRO. An MDC is only interested in the Aggregators that are associated with the connections that are registered at the MDC. The MDC does not receive congestion point information from the CRO.

When the MDC queries the CRO for common reference information, it supplies all connections registered at the MDC. The MDC is responsible to populate the following tables before a query is executed: COMMON\_REFERENCE\_OPERATOR, DISTRIBUTION\_SYSTEM\_OPERATOR, BALANCE\_RESPONSIBLE\_PARTY and CONNECTION.

In response, the CRO will supply the aggregators associated with these connections.

An MDC is connected to one or more CROs. Within the common reference there is only one MDC.



The common reference used in the Meter Data Company consists of the following tables:

* COMMON\_REFERENCE\_QUERY\_STATE table: Stores status information on the common reference query response received from the CROs.

PERIOD: Identifies the associated date.

STATUS: Identifies the status of the query (SUCCES or FAILURE).

MESSAGE\_ID: Identifies the Common Reference Query Response message.

* AGGREGATOR table: Identifies an aggregator associated with connections

DOMAIN: Identifies an AGR.

* COMMON\_REFERENCE\_OPERATOR table: Identifies the CRO from which the connections were received. This table must be filled by the MDC.

DOMAIN: Identifies a CRO.

* DISTRIBUTION\_SYSTEM\_OPERATOR table: Identifies a DSO that is allowed to query the meter data. This table must be filled by the MDC.

DSO\_DOMAIN: Identifies a DSO.

* BALANCE\_RESPONSIBLE\_PARTY table: Identifies a BRP that is allowed to query the meter data. This table must be filled by the MDC.

DOMAIN: Identifies a BRP.

* CONNECTION table: Identifies the connection in the grid that are registered at the MDC. This table must be filled by the MDC.

ENTITY\_ADDRESS: Identifies a connection.

* AGGREGATOR\_CONNECTION table: Identifies the connections in the grid and the associated aggregator (including historic information).

CONNECTION\_ENTITY\_ADDRESS: Identifies the associated connection.

VALID\_FROM: Start date from which time onwards the connection is valid.

VALID\_UNTIL: Date when the connection was no longer valid.

AGGREGATOR\_DOMAIN: Identifies the associated aggregator.

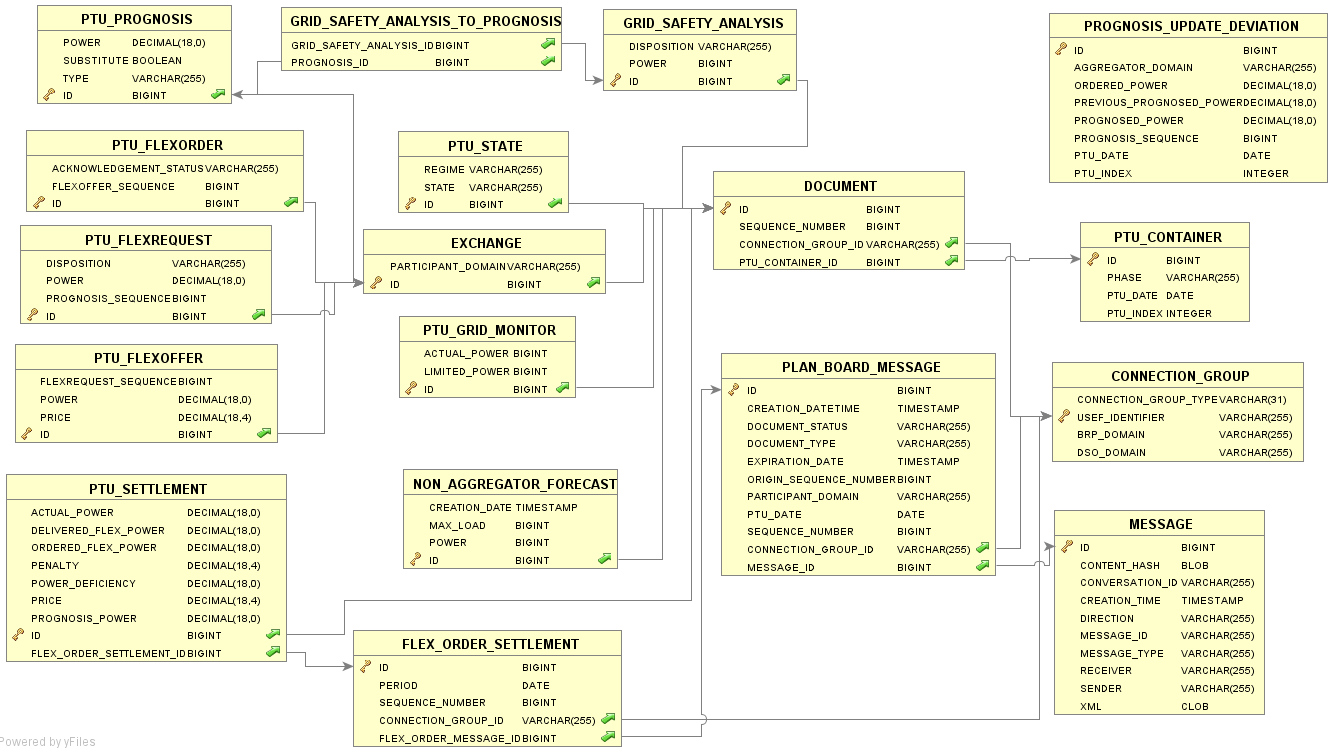
CRO\_DOMAIN: Identifies the associated Common Reference Operator that sent this information.

## Planboard

See the LC\_Planboard component for a description of the need and usage of this schema.

The mechanism and thinking behind the planboard is similar for all roles.

### DSO planboard



1. DSO planboard

See the Message store table description for next tables:

* MESSAGE

See the DSO common reference table description for next tables:

* CONNECTION\_GROUP
* PLAN\_BOARD\_MESSAGE table: The PLAN\_BOARD\_MESSAGE table links all related entities for a certain CONNECTION\_GROUP for all applicable PTUs together.  
  Each in- or outbound message contains 1 type of entity (e.g. flex requests or flex orders, they are never combined) for multiple PTUs. For each in- and outbound message all contained entities are stored individually (per PTU) on the planboard (e.g. in the PTU\_FLEXREQUEST or PTU\_FLEXORDER table) and linked to a record in the PLAN\_BOARD\_MESSAGE table by means of the sequence number.  
  The PLAN\_BOARD\_MESSAGE record is also linked to a CONNECTION\_GROUP (a congestion point or BRP) to which the entities in the message apply. One in- or outbound message can result in multiple records in the PLAN\_BOARD\_MESSAGE table, one for each CONNECTION\_GROUP. The PLAN\_BOARD\_MESSAGE table also keeps track of the status of all enclosed entities.

CREATION\_DATETIME: Date time when this planboard message was created.

DOCUMENT\_STATUS: Identifies the status of the document (NEW, SUBMITTED, ACCEPTED, REJECTED, EXPIRED, REVOKED, PROCESSED, PENDING\_FLEX\_TRADING, ARCHIVED, FINAL, TO\_BE\_RECREATED, DISPUTED). The status of a document is determined by the rules applicable in the USEF processes on a message.

DOCUMENT\_TYPE: Identified the document type corresponding to the type of message (FLEX\_OFFER, FLEX\_ORDER, FLEX\_REQUEST, PROGNOSIS).

EXPIRATION\_DATE: Optional date and time specifying when the message expires.

ORIGIN\_SEQUENCE\_NUMBER: Sequence number of the origin message the current message is related with. A flex request is related to a prognosis, a flex offer is related to a flex request.

PARTICIPANT\_DOMAIN: Identifies the sender in case of an inbound message or the receiver in case of an outbound message.

PTU\_DATE: Date the PTUs referenced in the message belong to.

SEQUENCE\_NUMBER: Sequence number of this message, which is incremented each time a new revision of a message is sent

CONNECTION\_GROUP\_ID: Identifies the associated connection group (in this case a congestion point).

MESSAGE\_ID: Identifies the associated message.

* PTU\_CONTAINER table: The container to put all connection group/PTU specific information in.

PHASE: Identifies the phase the PTU is in (Plan, Validate, Operate, Settlement).

PTU\_DATE: Identifies the date the PTU belongs to.

PTU\_INDEX: Identifies the PTU within the given day.

* DOCUMENT table: An abstract table representing any type of document that is put in a container for a connection group /PTU.

SEQUENCE\_NUMBER: Identifies the version of the document. Relates to the sequence number in the PLAN\_BOARD\_MESSAGE table that contains the in- or outbound message related to this document.

PTU\_CONTAINER\_ID: Identifies the associated PTU container.

CONNECTION\_GROUP\_ID: Identifies which connection group (congestion point, BRP or AGR this document relates to.

* PTU\_STATE table: Contains state and regime information for a certain connection group in a certain PTU.

REGIME: Identifies the phase the PTU is in (GREEN, YELLOW, ORANGE, RED).

STATE: Identifies the state the PTU is in (PlanValidate, DayAheadClosedValidate, IntraDayClosedValidate, Operate, PendingSettlement, Settled).

* NON\_AGGREGATOR\_FORECAST table: Contains the result of a non aggregator forecast.

CREATION\_DATE: Date the forecast was created.

MAX\_LOAD: The maximum load for the congestion point for that day.

POWER: The power forecast for the PTU for Non-Aggregator connections.

* GRID\_SAFETY\_ANALYSIS table: Contains the result of grid safety analysis.

DISPOSITION: Identifies the disposition type of the PTU (AVAILABLE, REQUESTED).

POWER: The power value associated with the disposition type.

* PTU\_PROGNOSIS table: Contains PTU prognosis information.

POWER: Specifies the power in Watt for the associated PTU.

TYPE: Specifies the type of prognosis (A\_PLAN, E\_PLAN, D\_PROGNOSIS).

SUBSTITUTE: When a prognosis message is created to fill a gap in the flow, like a missing D-prognosis, this flag is set to true (default value is false)*.*

* GRID\_SAFETY\_ANALYSIS\_TO\_PROGNOSIS table: Contains the relation between a grid safety analysis and the prognoses used.
* PROGNOSIS\_UPDATE\_DEVIATION table: Contains an updated prognosis.

AGGREGATOR\_DOMAIN: Domain name of the related aggregator.

ORDERED\_POWER: Ordered power (sum of the ACCEPTED orders related to the previous PTU).

PREVIOUS\_PROGNOSED\_POWER: The amount of power previously prognosed for this PTU.

PROGNOSED\_POWER: Amount of power prognosed in the updated prognosis.

PROGNOSIS\_SEQUENCE: Sequence number of the updated prognosis.

PTU\_DATE: Identifies the date of the prognosis.

PTU\_INDEX: Identifies the PTU within the given day.

* PTU\_GRID\_MONITOR table: Contains power information for a certain congestion point in a certain PTU.

ACTUAL\_POWER: Actual power in Watt for this congestion point in this PTU.

LIMITED\_POWER: Amount of power in Watt this congestion point was limited to in this PTU.

* EXCHANGE table: Contains information on the party data has been exchanged.

PARTICIPANT\_DOMAIN: Identifies the participant to who this information was sent (outbound) or from whom the information came (inbound).

* PTU\_FLEXREQUEST table: Contains PTU flex request information.

DISPOSITION: Indicates whether the Power specified for this PTU represents available capacity or a request for reduction/increase (AVAILABLE, REQUESTED).

POWER: Specifies the power in Watt for the associated PTU.

PROGNOSIS\_SEQUENCE: Identification of the prognosis sequence used to create the flex request.

* PTU\_FLEXOFFER table: Contains PTU flex offer information.

POWER: Specifies the power in Watt for the associated PTU.

PRICE: The price offered or accepted for supplying the indicated amount of flexibility in this PTU.

FLEXREQUEST\_SEQUENCE: Identification of the flex request sequence related to the flex offer.

* PTU\_FLEXORDER table: Contains PTU flex order information.

FLEXOFFER\_SEQUENCE: Identification of the flex offer sequence related to the flex order.

ACKNOWLEDGEMENT\_STATUS: Identifies the status the flex order is in (SENT, ACCEPTED, NO\_RESPONSE, REJECTED).

* PTU\_SETTLEMENT table: Contains settlement information related to a flex order and PTU.

ACTUAL\_POWER: The actual power in Watt for the associated PTU.

DELIVERED\_FLEX\_POWER: The amount of power in Watt delivered as part of the flex order.

ORDERED\_FLEX\_POWER: The amount of power in Watt order in the flex order.

PENALTY: The penalty calculated for this settlement.

POWER\_DEFICIENCY: The difference between ordered and delivered flex amount of power.

PRICE: The price of the associated flex order.

PROGNOSIS\_POWER: The amount of power prognosed in the associated PTU in Watt.

FLEX\_ORDER\_SETTLEMENT\_ID: Identifies the associated flex order settlement item.

* FLEX\_ORDER\_SETTLEMENT table: Contains settlement information related to a flex order.

PERIOD: Period the flex order applies to.

CONNECTION\_GROUP\_ID: Identifies the connection group the flex order is associated with.

FLEX\_ORDER\_MESSAGE\_ID: Identifies the planboard message associated with the flex order.

* CONNECTION\_METER\_EVENT table: Contains.

CAPACITY: The capacity of the connection in Watt.

DATETIME: Identifies the date of the connection meter event.

EVENT\_TYPE: Type of event, possible values: CapacityManagement, ConnectionInterruption, ConnectionResumption.

CONNECTION: Specifies the connection of the meter event.

* CONNECTION\_CAPACITY\_LIMITATION\_PERIOD table: Contains information on limited connections.

CAPACITY\_REDUCTION: The capacity reduction in Watt.

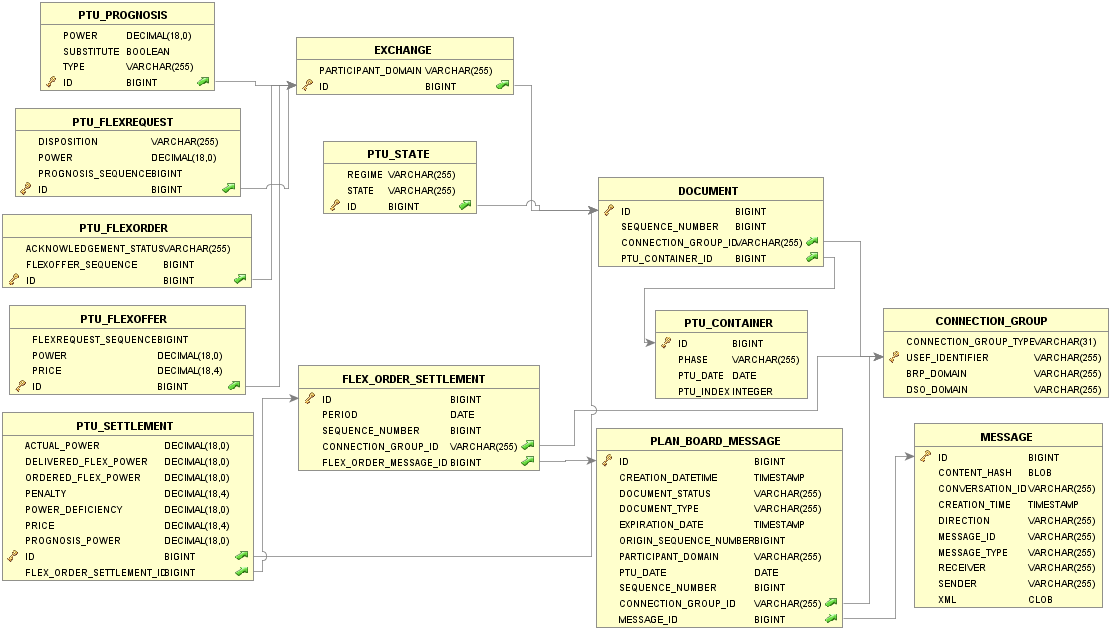
START\_DATETIME: Start of the limitation of the connection.

END\_DATETIME: End of the limitation of the connection.

TOTAL\_OUTAGE: specifies if the limitation was a total outage.

CONNECTION: Specifies the connection.

### AGR planboard



1. AGR planboard

See the Message store table description for next tables:

* MESSAGE

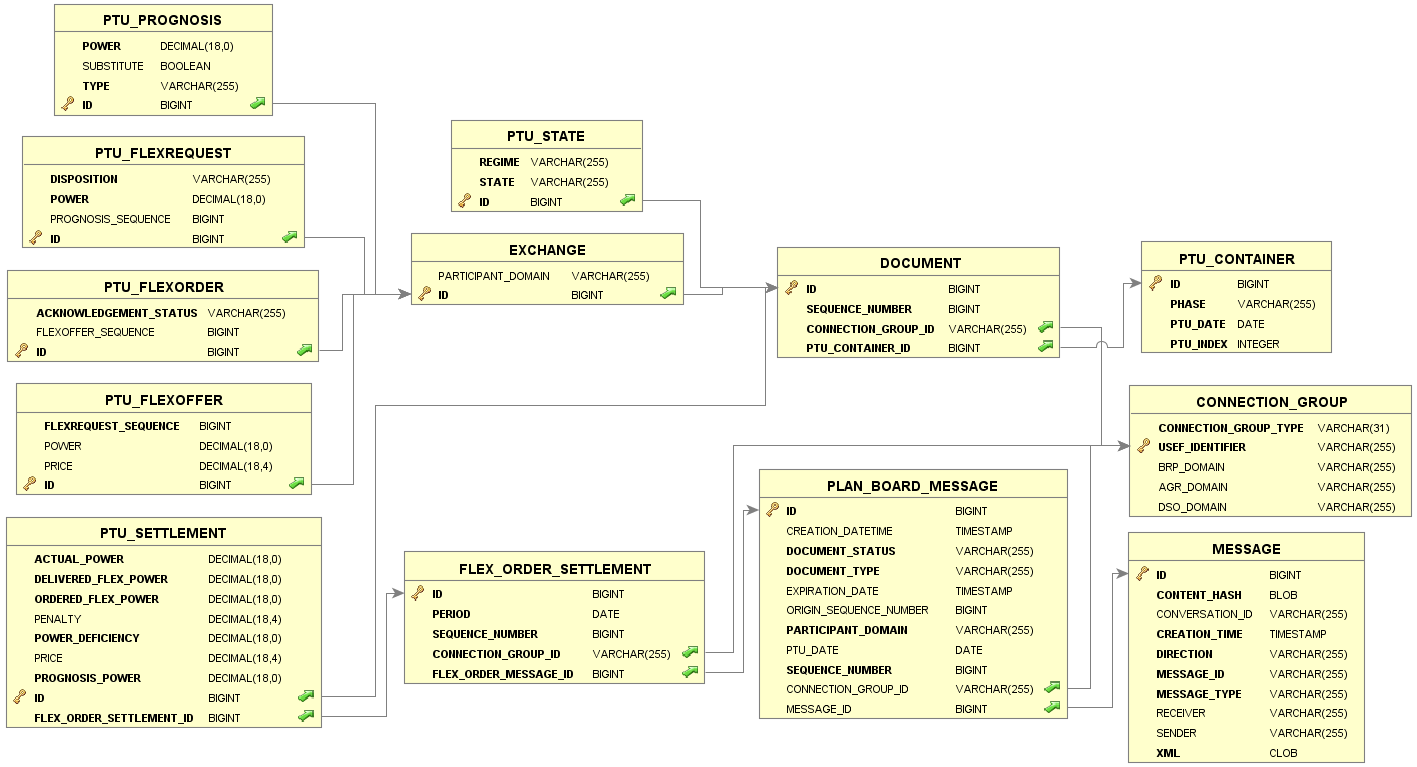
See the AGR common reference table description for next tables:

* CONNECTION\_GROUP

See the DSO planboard table description for next tables:

* PLAN\_BOARD\_MESSAGE
* PTU\_CONTAINER
* DOCUMENT
* PTU\_STATE
* EXCHANGE
* PTU\_PROGNOSIS
* PTU\_FLEXREQUEST
* PTU\_FLEXOFFER
* PTU\_FLEXORDER
* PTU\_SETTLEMENT
* FLEX\_ORDER\_SETTLEMENT

### BRP planboard



1. BRP planboard

See the Message store table description for next tables:

* MESSAGE

See the BRP common reference table description for next tables:

* CONNECTION\_GROUP

See the DSO planboard table description for next tables:

* PLAN\_BOARD\_MESSAGE
* PTU\_CONTAINER
* DOCUMENT
* PTU\_STATE
* EXCHANGE
* PTU\_PROGNOSIS
* PTU\_FLEXREQUEST
* PTU\_FLEXOFFER
* PTU\_FLEXORDER
* PTU\_SETTLEMENT
* FLEX\_ORDER\_SETTLEMENT

## AGR portfolio

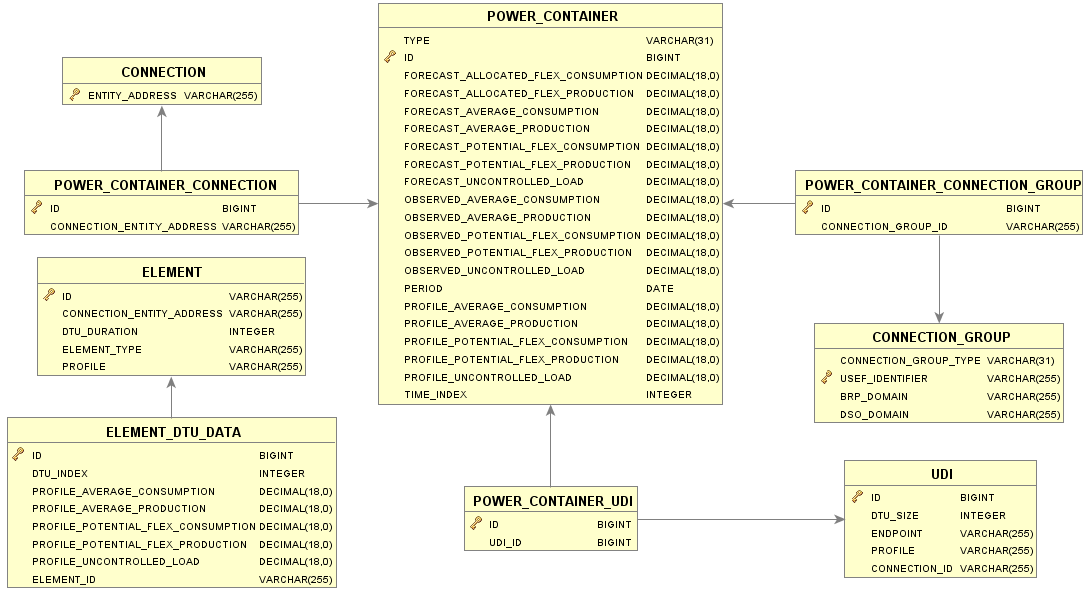
See the LC\_Aggregator portfolio component for a description of the need and usage of this schema.

The aggregator portfolio has four categories:

|  |  |  |
| --- | --- | --- |
| **Category** | **Description** | **Populated by** |
| Profile | Used for very-long-term forecasts, this is the behaviour generally expected to be exhibited by the connection depending on the macro-level class it belongs to (regular household, PV household, EV household, etc.) | N-Day Ahead Initialization |
| Forecast | Based on profile and/or previously observed behaviour, corrected for expected medium-term factors, such as weather | Collect Forecasts |
| Observed | Reported behaviour, which may be based on partial information or even entirely absent | Check ADS Goal Realization |

The common idea of the different power container categories is that observed values have priority over forecast values, which in turn have priority over profile values.

Power containers are used on 3 levels; connection group, connection and UDI. Per category, each specific type of power value (e.g. forecast\_average\_consumption) is stored on only one of these levels and can be aggregated on higher levels. These aggregated values are calculated at runtime and are not stored and should not be present in the database. If a specific power value (e.g. forecast\_average\_consumption) is requested on connection group level, the specific power values of all underlying connections and UDIs are summed. If power values would be present on multiple levels for the same category (e.g. forecast\_average\_consumption on connection and underlying UDI level) the aggregation could result in double counting.



1. AGR portfolio – power containers and elements

See the AGR common reference for the description of the tables CONNECTION and CONNECTION\_GROUP.

* ELEMENT table: Acts as a template for the Portfolio for both UDI and non-UDI processes. The table contains the details of the portfolio elements associated with the aggregator’s connections, in order to consistently initialize the portfolio based on those elements.

ID: Unique identification, used as the device selector.

CONNECTION\_ENTITY\_ADDRESS: The connection this element belongs to.

DTU\_DURATION: The Device Time Unit used by the ELEMENT

ELEMENT\_TYPE: Type of element. Possible values: MANAGED\_DEVICE, SYNTHETIC\_DATA

PROFILE: An AGR-specific string which classifies the element.

* ELEMENT\_DTU\_DATA table: Identifies the template power data per DTU for a specific element. During portfolio initialization the power values in this table are copied to the applicable entries in the POWER\_CONTAINER table in the portfolio.

DTU\_INDEX: Identifies the DTU within a day

PROFILE\_UNCONTROLLED\_LOAD: Profile value for the uncontrolled load.

PROFILE\_AVERAGE\_CONSUMPTION: Profile value for the average consumption.

PROFILE\_AVERAGE\_PRODUCTION: Profile value for the average production.

PROFILE\_POTENTIAL\_FLEX\_CONSUMPTION: Profile value for the potential flex consumption.

PROFILE\_POTENTIAL\_FLEX\_PRODUCTION: Profile value for the potential flex production.

ELEMENT\_ID: Identifies the associated element.

* POWER\_CONTAINER table: Table that holds all possible power values for a UDI, connection or connection group per time unit. The time unit used is either DTU (specified for each UDI), PTU (for connections) or Non-UDI PTU (for connection groups). The last two are configured in the application configuration.

TYPE: Entity type associated with this record. Possible values: CONNECTION\_GROUP, CONNECTION, UDI.

PERIOD: Identifies the associated date.

TIME\_INDEX: Sequential index number identifying the time unit within the given period.

CONNECTION\_ENTITY\_ADDRESS: Refers to the associated connection in case the power values are on connection level.

UDI\_ID: Refers to the associated UDI in case the power values are on UDI level.

CONNECTION\_GROUP\_ID: Refers to the associated connection group in case the power values are on connection group level.

PROFILE\_UNCONTROLLED\_LOAD: The uncontrolled load on profile level, used to determine the amount of production/consumption not 'seen' by the ADS control system.

PROFILE\_AVERAGE\_CONSUMPTION: The average consumption on profile level.

PROFILE\_AVERAGE\_PRODUCTION: The average production on profile level.

PROFILE\_POTENTIAL\_FLEX\_CONSUMPTION: The potential flex consumption on profile level.

PROFILE\_POTENTIAL\_FLEX\_PRODUCTION: The potential flex production on profile level.

FORECAST\_UNCONTROLLED\_LOAD: The uncontrolled load on forecast level, used to determine the amount of production/consumption not 'seen' by the ADS control system.

FORECAST\_AVERAGE\_CONSUMPTION: The average consumption on forecast level.

FORECAST\_AVERAGE\_PRODUCTION: The average production on forecast level.

FORECAST\_POTENTIAL\_FLEX\_CONSUMPTION: The potential flex consumption on forecast level.

FORECAST\_POTENTIAL\_FLEX\_PRODUCTION: The potential flex production on forecast level.

FORECAST\_ALLOCATED\_POTENTIAL\_FLEX\_CONSUMPTION: The potential flex consumption on forecast level already allocated to devices by means of device messages.

FORECAST\_ALLOCATED\_POTENTIAL\_FLEX\_PRODUCTION: The potential flex production on forecast level already allocated to devices by means of device messages.

OBSERVED\_UNCONTROLLED\_LOAD: The uncontrolled load on observed level, used to determine the amount of production/consumption not 'seen' by the ADS control system.

OBSERVED\_AVERAGE\_CONSUMPTION: The average consumption on observed level.

OBSERVED\_AVERAGE\_PRODUCTION: The average production on observed level.

OBSERVED\_POTENTIAL\_FLEX\_CONSUMPTION: The potential flex consumption on observed level.

OBSERVED\_POTENTIAL\_FLEX\_PRODUCTION: The potential flex production on observed level.

* POWER\_CONTAINER\_CONNECTION table: Table that holds the link between a connection and a power container.
* POWER\_CONTAINER\_CONNECTION\_GROUP table: Table that holds the link between a connection group and a power container.
* POWER\_CONTAINER\_UDI table: Table that holds the link between a UDI and a power container.
* UDI table: Identifies the USEF Device Interface endpoints present behind the connection.

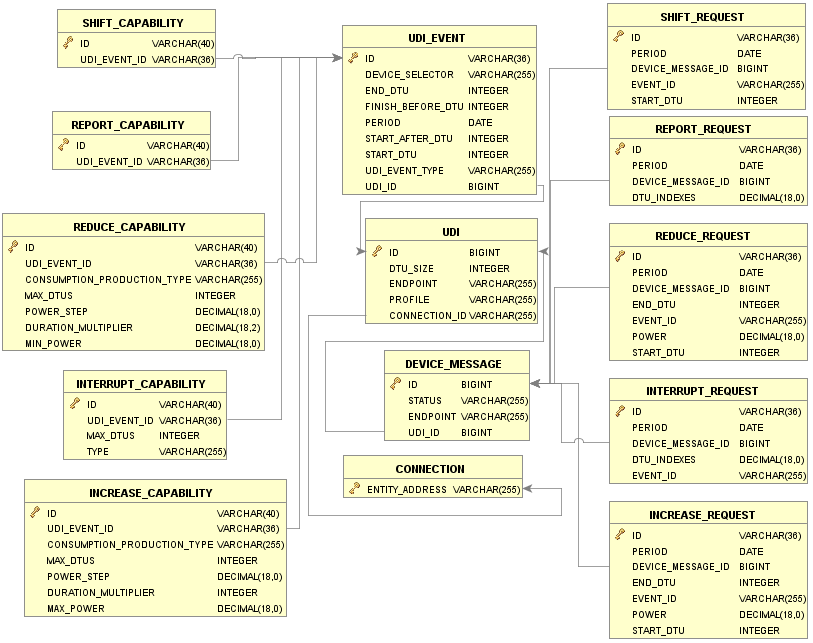
DTU\_SIZE: The Device Time Unit used by the UDI.

ENDPOINT: The network-specific address of the UDI endpoint.

CONNECTION\_ID: Identifies the associated connection.

PROFILE: Type of device (related to the ELEMENT.PROFILE field).

The Element table contains all ADS devices managed by the aggregator. For UDI aggregators, the capabilities of those devices are described in the UDI\_EVENT table and the associated CAPABILITY tables. The UDI control messages, containing one or more device requests, are described in the DEVICE\_MESSAGE table and the associated REQUEST tables.



1. AGR portfolio – UDI device capabilities and device messages

* UDI\_EVENT table: The Element table contains all ADS devices managed by the aggregator. The capabilities of those devices are described in the UDI\_EVENT table and the associated CAPABILITY tables.

ID: Event identifier: an endpoint-specific string uniquely identifying the event within the period. It may include a device identifier.

DEVICE\_SELECTOR: Indicates which specific device controlled by the UDI endpoint this event relates to.

START\_DTU: Currently planned, possible or actual start time of the event, expressed as a DTU.

END\_DTU: Currently planned, possible or actual end time of the event, expressed as a DTU

START\_AFTER\_DTU: DTU specifying the earliest time the process can be started

FINISH\_BEFORE\_DTU: DTU by which time the process should be finished

PERIOD: Period when the event is applicable.

UDI\_EVENT\_TYPE: The type of event. Possible values: Consumption, Production, OnDemandConsumption, OnDemandProduction.

UDI\_ID: Identifies the associated UDI

* SHIFT\_CAPABILITY table: Identifies the shift capability associated with a UDI event.

UDI\_EVENT\_ID: Specifies the associated UDI event

* REPORT\_CAPABILITY table: Identifies the report capability associated with a UDI event.

UDI\_EVENT\_ID: Specifies the associated UDI event

* REDUCE\_CAPABILITY table: Identifies the reduce capability associated with a UDI event.

UDI\_EVENT\_ID: Specifies the associated UDI event

MAX\_DTUS: The maximum number of Device Time Units the process can reduce its power consumption or production.

POWER\_STEP: Amount (in Watts) indicating the granularity with which consumption or production can be decreased.

DURATION\_MULTIPLIER: Decimal value, indicating the amount by which the total duration of the process will increase for each power step by which consumption or production is decreased.

MIN\_POWER: Amount (in Watts) indicating the power floor to which consumption or production can be reduced.

CONSUMPTION\_PRODUCTION\_TYPE: Indicates if the capability relates to power CONSUMPTION or PRODUCTION.

* INTERRUPT\_CAPABILITY table: Identifies the interrupt capability associated with a UDI event.

UDI\_EVENT\_ID: Specifies the associated UDI event

MAX\_DTUS: The maximum number of Device Time Units the process can interrupt its power consumption or production.

TYPE: The extent to which the process described by this event can be interrupted. Possible values: Full, None, Per-DTU.

* INCREASE\_CAPABILITY table: Identifies the increase capability associated with a UDI event.

UDI\_EVENT\_ID: Specifies the associated UDI event

MAX\_DTUS: The maximum number of Device Time Units the process can increase its power consumption or production.

POWER\_STEP: Amount (in Watts) indicating the granularity with which consumption or production can be increased.

MAX\_POWER: Amount (in Watts) indicating the power ceiling to which consumption or production can be increased.

CONSUMPTION\_PRODUCTION\_TYPE: Indicates if the capability relates to power CONSUMPTION or PRODUCTION.

* DEVICE\_MESSAGE table: Identifies the UDI control messages, containing one or more device requests to be sent to an endpoint.

ID: Unique identification, used as the device message

STATUS: Status of the message. Possible values: NEW, IN\_PROCESS, SENT, FAILURE.

ENDPOINT: Network-specific address of the UDI endpoint, as included in the aggregator portfolio.

UDI\_ID: Specifies the associated UDI

* SHIFT\_REQUEST table: Identifies a shift device request associated with a device message.

DEVICE\_MESSAGE\_ID: Specifies the associated device message

PERIOD: Date the request applies to.

EVENT\_ID: Specifies the associated UDI\_EVENT.

START\_DTU: DTU at which the event should start.

* REPORT\_REQUEST table: Identifies a report device request associated with a device message.

DEVICE\_MESSAGE\_ID: Specifies the associated device message

PERIOD: Date the request applies to.

DTU\_INDEXES: DTU(s) for which the device should report its production or consumption values, if available.

* REDUCE\_REQUEST table: Identifies a reduce device request associated with a device message.

DEVICE\_MESSAGE\_ID: Specifies the associated device message

PERIOD: Date the request applies to.

START\_DTU: DTU in which the device should start decreasing its consumption or production.

END\_DTU: DTU in which the device should stop decreasing its consumption or production.

EVENT\_ID: Specifies the associated UDI\_EVENT.

POWER: Amount of power in Watts (positive integer) by which the device should attempt to decrease its consumption or production.

* INTERRUPT\_REQUEST table: Identifies an interrupt device request associated with a device message.

DEVICE\_MESSAGE\_ID: Specifies the associated device message

PERIOD: Date the request applies to.

EVENT\_ID: Specifies the associated UDI\_EVENT.

DTU\_INDEXES: DTU(s) for which the device should report its production or consumption values, if available.

* INCREASE\_REQUEST table: Identifies an increase device request associated with a device message.

DEVICE\_MESSAGE\_ID: Specifies the associated device message

PERIOD: Date the request applies to.

START\_DTU: DTU in which the device should start increasing its consumption or production.

END\_DTU: DTU in which the device should stop increasing its consumption or production.

EVENT\_ID: Specifies the associated UDI\_EVENT.

POWER: Amount of power in Watts (positive integer) by which the device should attempt to increase its consumption or production.