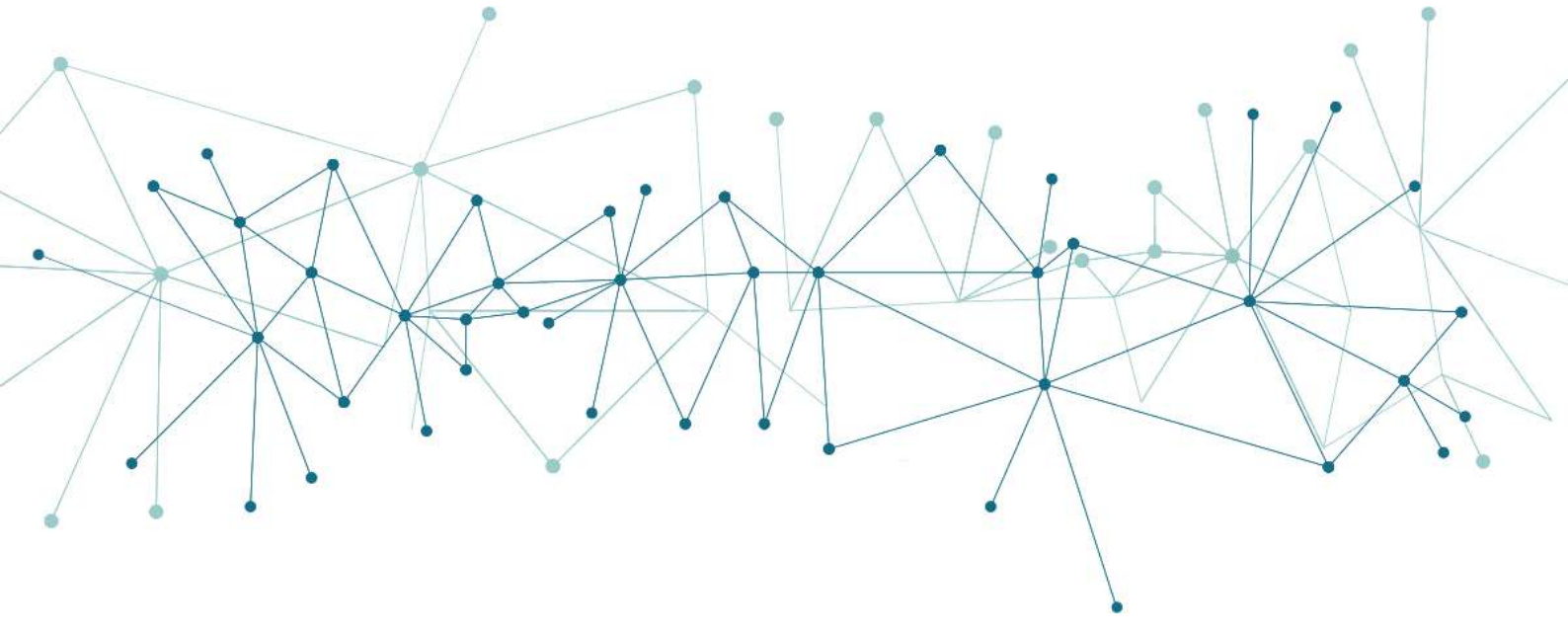
The eDREAM project is co-founded by the EU's Horizon 2020 innovation programme under grant agreement No 774478

eDREAM

enabling new Demand REsponse Advanced, Market oriented and secure technologies, solutions and business models

DELIVERABLE: 5.5 Self-enforcing smart contracts for DR tracking and control V2

Authors: Tudor Cioara, Claudia Pop, Ioan Salomie



Imprint

D5.2. Self-enforcing smart contracts for DR tracking and control V2 (Month 29)

Contractual Date of Delivery to the EC:	31.05.2020
Actual Date of Delivery to the EC:	31.05.2020
Author(s):	Claudia Pop (TUC), Marcel Antal (TUC), Tudor Cioara (TUC), Ionut Anghel (TUC), Viorica Chifu (TUC), Cristina Pop (TUC), Ioan Salomie (TUC), Andreea Valeria Vesa (TUC)
Participant(s):	Lead Partner: Technical University of Cluj-Napoca (TUC) Contributors: ENG, ASM, EMOT
Project:	enabling new Demand Response Advanced, Market oriented and secure technologies, solutions and business models (eDREAM)
Work package:	WP5 – Blockchain-enabled decentralized network control optimization and DR verification
Task:	T5.2 - Blockchain-driven self-enforcing smart contracts for DR energy transactions modelling, tracking and decentralized control
Confidentiality:	public
Version:	0.99

Legal Disclaimer

The project enabling new Demand Response Advanced, Market-oriented and secure technologies, solutions and business models (eDREAM) has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 774478. The sole responsibility for the content of this publication lies with the authors. It does not necessarily reflect the opinion of the Innovation and Networks Executive Agency (INEA) or the European Commission (EC). INEA or the EC are not responsible for any use that may be made of the information contained therein.

Copyright

Table of Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

API	Application Programming Interface
DEP	Distributed Energy Prosumer
DR	Demand Response
DSO	Distributed System Operator
eDREAM	enabling new Demand Response Advanced, Market oriented and secure technologies, solutions and business models
MINLP	Mathematical Program of class Mixed Integer Nonlinear Program
NP	Nondeterministic Polynomial time
P2P	Peer-to-peer
TSO	Transmission System Operator
WP	Work Package

Executive Summary

In this deliverable we present the development work and enhancements brought in addition to the eDREAM blockchain-based platform for micro-grid energy management the initial version of which was realised in deliverable “D5.2 - Self-enforcing smart contracts for DR tracking and control V1”.

For each type of decentralized management solution exposed by the eDREAM blockchain platform, in this deliverable, we present the defined and implemented algorithms matching the demand and the offer. First, we present the matching algorithms for the blockchain-driven peer-to-peer energy marketplace implementation which are aiming to determine at the end of the energy market session both the market clearing price and the pair of energy bids and offers that will form the direct peer-to-peer transactions. Two types of algorithm have been implemented: (i) a greedy approach that sorts the lists of orders and bids in descending order and tries to match them from the largest one to the smaller ones and (ii) an augmented bipartite graph-based approach where the vertices are formed by the sets of bids and offers while the edges are representing the energy flow between prosumers and transmission capacity as constraints.

Second, we have refined and improved the algorithm that is matching the flexibility request of an aggregator to a set of flexibility orders signals to be followed by individual prosumers. The algorithm was implemented and integrated with the blockchain-driven DR and flexibility services management and control and it starts by optimally decomposing the flexibility request and then determines a subset of prosumers from the aggregator portfolio able to deliver a specific amount of flexibility. Third, we have implemented an algorithm for optimal matching of flexibility bids and flexibility offers, considering the associated price as an additional vector to the actual amount of flexibility committed. We addressed this optimization problem using a greedy heuristic that is derived from the best-fit approximation algorithm used for bin-packing. The algorithm has been implemented and integrated with the blockchain-based price-driven flexibility marketplace.

Besides the smart contracts-based implementation of the peer-to-peer energy marketplace and of the decentralized provisioning and control of flexibility services which have been detailed in D5.2, in this deliverable we present the smart contract-based implementation of a price-driven flexibility marketplace allowing the trading of energy flexibility between sellers (i.e. prosumers or aggregators) and buyers (DSO, aggregators or even the TSO). Non-fungible energy-adapted tokens are defined and used to transform the energy into a transactable digital asset. The flexibility sellers and flexibility buyers participate in the flexibility market sessions by leveraging on self-enforcing smart contracts to submit energy flexibility bids/offers with the associated prices. The flexibility market session smart contracts keep a record of all the registered flexibility energy bids and offers, their matching being managed outside of the chain via Oracles-based integration of defined algorithms, while the market manager contracts keep track of all the opened sessions and make the necessary validations regarding the participants' activity. The decision on the actual share of energy flexibility which has been effectively delivered by each flexibility seller and associated financial settlement is conducted using smart contracts that are considering the actual monitored energy information.

Finally, we have provided an overview of the functionality implemented by the eDREAM blockchain-based micro-grid for each type of energy stakeholder and exposed via relevant web interfaces. The web interfaces are presented for each type of management solution provided (i.e. peer-to-peer energy trading, decentralized flexibility services management, and price-driven flexibility trading) using interaction steps for the success use-case scenario.

1 Introduction

1.1 Purpose

This report provides an overview of the work carried out in Task 5.2 between months 18 and 29 for developing and improving the blockchain-based platform for the distributed control and management of the micro-grid initially presented in D5.2.

The work has been concentrated in three main directions:

- The definition, implementation and integration of demand-to-offer-matching algorithms for all the functionalities of the blockchain platform: P2P energy trading, decentralized control and delivery of energy flexibility, price-driven energy flexibility marketplace;
- Implementation of a price-driven flexibility marketplace which is leveraging on self-enforcing smart contracts for allowing the buyers and sellers to trade energy flexibility;
- Refinement and improvement of the blockchain platform features released in D5.2 and implementation of web-based interfaces to facilitate the interaction of the main actors.

1.2 Relation to other activities

WP5 and T5.2 use the outputs of WP2 in terms of requirements and use-cases as well as the outputs of WP3 in terms of energy demand/generation forecasting, prosumers' baseline and flexibility assessment. These are used to set up and to drive the execution of the self-enforcing smart contracts to deliver the functionalities expected with the three micro-grid management alternatives considered in eDREAM.

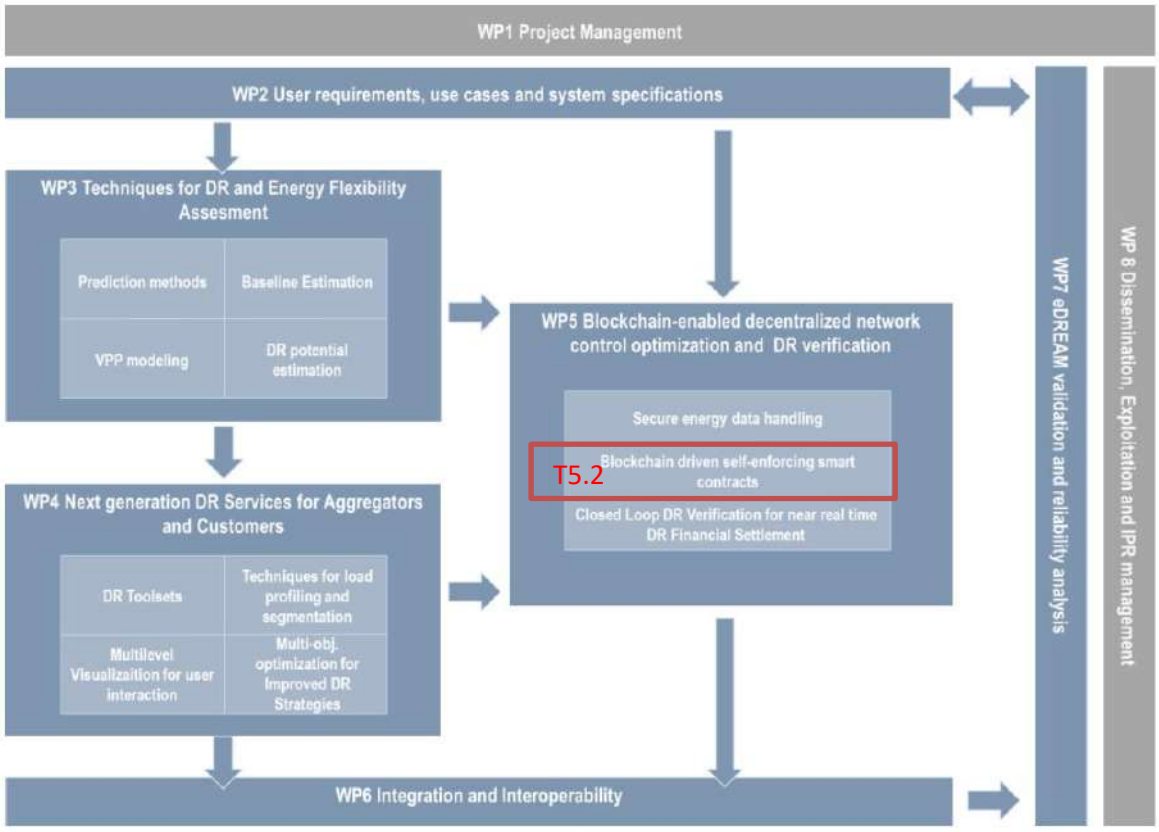


Figure 1. eDREAM PERT chart showing WP5 and T5.2 in relation to other work packages

1.3 Structure of the document

The remainder of the report is organized as follows.

- Section 2 presents the mechanism defined and implemented for matching in a decentralized fashion the demand and the offer. The section reports the algorithms for the decentralized matching of energy bids and offers, flexibility request and flexibility orders and finally the price-driven flexibility bids and flexibility orders;
- Section 3 describes a blockchain implementation of the price-driven energy flexibility marketplace which empowers the prosumers to sell their flexibility while other energy players such as the DSO and aggregators will compete on buying flexibility;
- Section 4 presents the blockchain platform for micro-grid management focusing on to the main web interfaces and functionalities implemented for actors' interaction;
- Section 5 concludes the deliverable.

2 Mechanism for Demand - Offer Matching

In this section we detail the mechanisms we have defined and implemented for matching the demand and offer in three blockchain-driven micro-grid energy management alternatives supported by the eDREAM platform: (1) P2P energy marketplace, (2) DR and flexibility decentralized management and control and (3) price-driven energy flexibility marketplace.

For (1) and (2) the blockchain design and associated smart contracts implementation have been presented in detail in D5.2. while for (3) the smart contracts are presented in Section 3.

2.1 Energy Bids-Offers Matching

The mechanisms presented in this section are related to the blockchain driven peer-to-peer energy marketplace implementation. Their role is to determine at the end of the energy market session both the market clearing price and the pair of energy bids and offers that will form the direct peer-to-peer transactions.

Firstly, the set of energy bids and offers are extracted from the market session, and the session clearing algorithm from is executed to compute the clearing price and determine the bids and offers that could be potentially matched (see Figure 2). The algorithm has a polynomial complexity.

All valid energy offers are put in ascending order (i.e. considering their associated price) on an aggregated energy supply curve and all valid energy bids are put in descending order (i.e. considering their associated price) on an aggregated energy demand curve (see lines 1-2). Then, a loop is used to iterate through price – ordered energy bids and offers and to determine the index of the energy bid and the energy offer where the ascending offer price and the descending bid price intersect (see lines 7-17). The intersection of the two curves determines the clearing price, at which all accepted bids and offers are remunerated (see line 20), the overall quantity of energy traded in the session (see lines 18-19) and the selected energy bids/offers. A method that considers several corner cases of intersecting two polylines is called to compute the exact intersection price. Finally, the clearing price and the list of energy bids and offers selected to be matched are returned (see line 21).

Input: list of energy offers and energy bids from the market session $Offers[P], Bids[Q]$
Output: market session clearing price $Price_{clearing}$
selected energy bids and offers $Offers_{selected}[N], Bids_{selected}[M]$
Begin

1. $Sort_{ascending}(Offers)$
2. $Sort_{descending}\ (Bids)$
3. $Index_O = 1$
4. $Index_B = 1$
5. $Displacement_{bid} = 1$
6. $Displacement_{offer} = 1$
7. **while** $(offer[index_o].price <= bid[index_b].price)$ **do**
8. $currentBidDisplacement = Displacement_{bid} + bid[index_b].amount$
9. $currentAskDisplacement = Displacement_{offer} + offer[index_o].amount$
10. **if** $(currentBidDisplacement > currentAskDisplacement)$ **then**
11. $Index_O++$
12. $Displacement_{offer} = currentAskDisplacement$
13. **else**
14. $Index_B++$
15. $Displacement_{bid} = currentBidDisplacement$

16. **end if**
17. **end while**
18. $Offers_{selected} = offer[0\ to\ index_o]$
19. $Bids_{selected} = bids[0\ to\ index_b]$
20. $Price_{clearing} = Compute_{clearing_P}(Displacement_{bid}, Displacement_{offer}, Index_O, Index_B)$
21. **return** $clearing_{price}, Offers_{selected}, Bids_{selected}$

End

Figure 2. Session energy clearing price calculation

Secondly, an algorithm is defined and implemented to determine the minimum number of matchings between the selected energy bids and energy offers (i.e. returned by the previous one) that will form the peer-to-peer energy transactions. Because the problem at hand is an NP problem, a heuristics algorithm is defined and implemented to solve it.

After running the algorithm from Figure 2, we know that the total amount of the M bids is equal with the total amount of the N offers:

$$\sum_{i=1}^{N} offer_i = \sum_{j=1}^{M} bid_j$$

We aim to determine a mapping between the set of offers and the set of bids, with the least number of mappings.

We construct a matrix $P \in R^{M*N}$ of size $M * N$, where each element $P(i,j)$ defines the percentage of bid_i that is considered to address the $offer_j$. Thus, each row of the matrix decomposes bid_i to each of the offers placed as headers of the columns:

$$\sum_{j=1}^{N} P(i,j) = 1, \forall i \in \{1..M\}$$

The offer is composed by the weighted sum of the bids from each column:

$$offer(j) = \sum_{i=1}^{M} P(i,j) * bid(i), \forall i \in \{1..N\}$$

The goal is to minimize the number of elements of the matrix that are not zero, thus the objective can be defined as:

$$MIN(\sum_{i=1}^{N}\sum_{j=1}^{M} f(P(i,j)))$$

where the function f is defined as a step function:

$$f: [0,1] \to \{0,1\}, f(x) = \begin{cases} 0, if\ x = 0 \\ 1, if\ x > 0 \end{cases}$$

The above defined optimization problem is classified as Nonlinear Programming, because the unknown variable in the optimization function is the matrix P containing real values in the interval $[0,1]$ and the objective function is not linear (i.e. the step function f).

The optimization problem is a NP-complete similar to the partition problem in computer science which is aiming to partition a set of positive integers S into two subsets S_1 and S_2 such that the sum of the numbers in S_1 equals the sum of the numbers in S_2 [1]. This can be easily shown by considering the particular case of our matching problem with N offers and only 2 bids. In this case, the set S is given by the N offers with a total amount of energy offer of $\sum_{i=1}^{N} offer(i)$ that has to be split in two sub-sets that are matching the energy value of each individual bid. A solution of the partition problem will generate N mappings between the set of energy offers and the set of energy bids that is the minimum number of mappings possible. If one could solve the partition problem in polynomial time, it could also solve the energy bid and offers matching problem, implying that this problem is also NP.

The first solution defined to solve the energy bids/offers matching problem is based on a *greedy approach* that sorts descending the lists of orders and bids and tries to match them from the largest one to the smaller ones (Figure 3). This solution has a run time complexity of $O(Nlog(N) + Mlog(M))$ given by the sort operation complexity, while the actual matching computation has the complexity of $O(M + N)$.

Input: Energy bids & offers selected by the energy clearing price calculation algorithm: $Offers_{selected}[N]$, $Bids_{selected}[M]$ where M is the total number of selected energy bids and N the total number of selected energy offers

Outputs: pair of matched energy bids and offers: $P[index_{offer}][index_{bid}]$ with the value 1

Begin

1. $Sort_{descending}(Offers_{selected})$
2. $Sort_{descending}(Bids_{selected})$
3. $index_{offer} = 0$
4. $index_{bid} = 0$
5. **while** $(index_{offer} < N \; and \; index_{bid} < M)$ **do**
6. **If** $(Offers_{selected}[index_{offer}] > Bids_{selected}[index_{bid}])$ **then**
7. $Offers_{selected}[index_{offer}] = Offers_{selected}[index_{offer}] - Bids_{selected}[index_{bid}]$
8. $P[index_{bid}][index_{offer}] = 1$
9. $index_{bid}++$
10. **else**
11. $Bids_{selected}(index_{bid}) = Bids_{selected}(index_{bid}) - Offers_{selected}(index_{offer})$
12. $index_{offer}++$
13. $P[index_{offer}][index_{bid}] = 1$
14. **end if**
15. **end while**

End

Figure 3. Energy bids / offers matching using a Greedy approach

To generate the associated energy trades, a parser id is defined and used to iterate the P matrix and, in case an element $P[index_{bid}][index_{offer}] = 1$, meaning that there is a match mapping between the bid with id $ID(index_{bid})$ and the offer with id $ID(index_{bid})$, it creates an energy trade with the structure presented in Figure 4. By parsing entire matrix P, a list of such trades is generated and published to the blockchain by the Oracle as a single transaction.

```
struct Trade{
        bytes32 id;
        bytes32 buyOrderId;
        bytes32 sellOrderId;
        address prosumerBuyingAddress;
        address payable prosumerSellingAddress;
        uint timestamp;
        uint tokenId;
        uint quantity;
        uint price;
}
```

Figure 4. Energy trade structure implementation in Solidity

The second solution defined to solve the energy bids/offers matching problem is based on a *Graph-based approach*. The problem is modelled as a bipartite graph $G = \{V_1, V_2, E\}$, where the vertices are formed by the sets of bids and offers:

$$V_1 = \{offer_i | i \in \{1..N\}\}, V_2 = \{bid_i | i \in \{1..M\}\}$$

An edge $e_{ij} \in E$ represents a transaction from $offer_i$ to bid_j. We define the function $f: \{V_1 \cup V_2\} \rightarrow R$ as an energy flow function between the vertices, with the following constraint:

$$f_{ij} = f(e_{ij}) \leq offer_i$$

We use concepts from the Minimum-cost flow problem to compute an approximate solution of the problem. We add a source connected to all the nodes from V_1 and a sink to all the nodes from V_2. We connect all the nodes from V_1 to all the nodes from V_2, by creating a complete bipartite graph (see Figure 5).

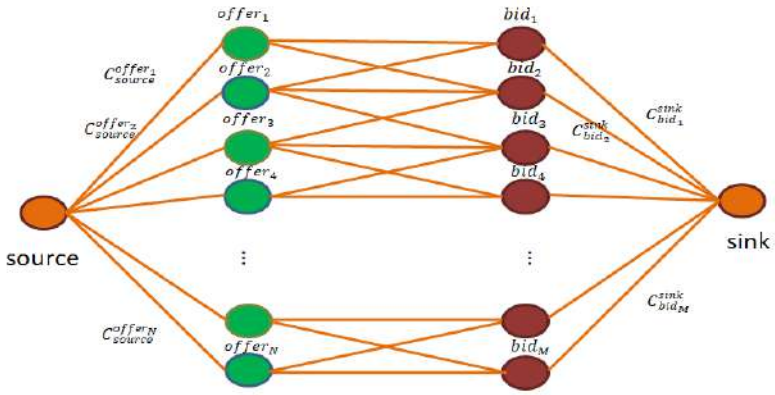


Figure 5. Augmented graph for modelling the energy flow among offers and bids

We augment the graph by defining the energy capacity of the edges as follows:

- All edges connecting the source and the nodes from V_1 have the capacities $C_{source}^{offer_i}$ defined as

$$C_{source}^{offer_i} = offer_i.value$$

- All edges connecting the nodes from V_2 with the sink have capacities $C_{bid_i}^{sink}$ defined as

$$C_{bid_i}^{sink} = bid_i.value$$

- All edges connecting nodes from V_1 to all the nodes from V_2 have the capacity C_{ij} defined as:

$$C_{ij} = offer_i.value$$

- All the edges connecting nodes from V_1 to V_2 have a positive costs a_{ij}, considered as weights, while, nodes from V_2 to V_1 have negative weights $-a_{ij}$

$$a_{ij} = |bid_j - offer_i|$$

- The cost of sending a flow along the edge e_{ij} is computed as

$$cost_{ij} = f_{ij} * a_{ij}$$

In this case the bids and offers matching problem is represented in the graph as an amount of energy s to be sent from source to sink:

$$s = \sum_{i=1}^{N} offer_i = \sum_{j=1}^{M} bid_j$$

The aim is to minimize the cost of the flow over the edges:

$$MIN(\sum_{i=1}^{N}\sum_{j=1}^{M} cost_{ij})$$

With the following constraints:

- Capacity constraints: $f_{ij} \leq c_{ij}$
- Symmetry: $f_{ij} = -f_{ji}$
- Flow conservation: $\sum_{w \in V_1 \cup V_2} f_{uw} = 0, \forall u \in V_1 \cup V_2$
- Required flow: $\sum_{w \in V_1 \cup V_2} f_{source-w} = s = \sum_{w \in V_1 \cup V_2} f_{w-sink}$

We use Figure 6 algorithm to determine the maximum flow from the source to the sink.

Input: The augmented bipartite graph constructed for energy bids and offers submitted in a market session
Output: Energy flows on the edges of the graph
Begin

1. Apply Bellman-Ford algorithm to increase the costs of the edges and make them positive
2. **While** ($\exists$ $path\ from\ source\ to\ sink$)
3. Path (source, sink) = Dijkstra (with edge e_{ij} and weights as a a_{ij})
4. flow = $Min_{capacity}$ (path (sink, source))
5. **For** $e_{ij} \in path(sink, source)$ **do**
6. $e_{ij}.capacity = e_{ij}.capacity - flow$
7. **End for**
8. **End while**

End

Figure 6. Energy bids / offers matching using an augmented graph-based solution

The graph-based solution starts by applying the Bellman Ford algorithm that has a complexity of $O(|N + M| * |E|)$ to increase the edge weights to positive values, so that the Dijkstra algorithm can be applied. According to the graph construction, the number of edges is $E = N + N * M + M$ and the number of vertices is $V = N + M + 2$. Dijkstra is applied in the main loop of the algorithm due to its lower complexity of $O(|E| +$

$|V|\log(|V|))$ when using a Fibonacci heap. It is computing, the shortest path from the source to the sink (see lines 2-8), with the weights represented by the augmented costs a_{ij}, while the flow is computed as the minimum of the capacities (see lines 5-7). The graph-based algorithm ends when no paths are found from the source to the sink having an overall complexity of $O(N * M^2 * log\ N)$.

The matchings between the bids and the offers are computed by the number of energy flows between the sources and the sinks the solution being contained in the energy flow values of the edges connecting nodes of the bipartite graph. To generate the list of transactions with the structure from Figure 4, the graph data structure are iterated and in case flow $f_{uw} > 0$ a trade is generated to mark a transaction between offer with identifier u and the bid with the identifier w. Then, a list of such trades is generated and published to the blockchain by the Oracle as a single transaction.

Since blockchain smart contracts have limited resources for on-chain logic computation, and each instruction is paid by the caller, we have implemented a solution that can be run off-chain while assuring the secure integration with the blockchain implementation of the peer-to-peer energy market. We have used the concept of Oracles for supporting the call of the complex logic or other APIs from outside the blockchain and to provide in response the results necessary in the smart contracts execution. Such an Oracles-based solution is presented in Figure 7.

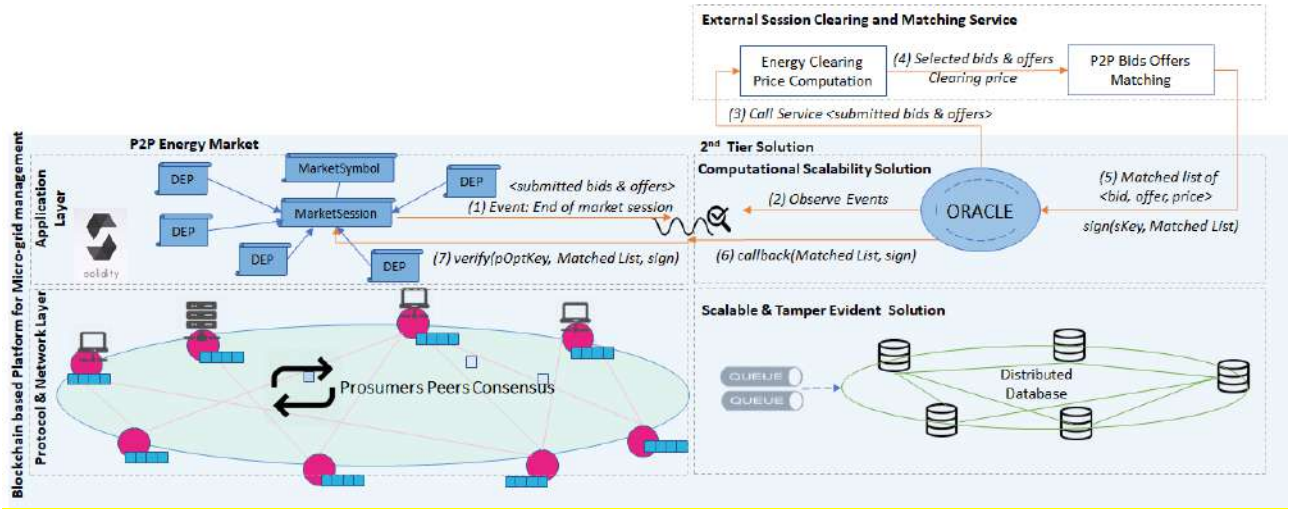


Figure 7. Matching energy bids and offers matching using Oracles

The Oracle is a module that integrates the blockchain with the off-chain world. For our implementation, we consider the oracle module to continuously listen for events that notify the necessity of results from the off-chain algorithm (see step 2). Such an event will be always intercepted whenever the symbol market will throw an end of a session event in the peer-to-peer energy marketplace. Once the Oracle intercepts such an event, it will forward the request to the Session Clearing and Matching Service together with the list of energy bids and offers submitted in the session with their price associated (see step 3). The clearing and matching service will have assigned a pair of public-private keys, where the public key can be stored on-chain. Once the results of the algorithm are obtained in terms of paired energy bids and offers, the service will sign them using the private key and return them to the oracle (see step 5). The signature is necessary, since the oracle or any other entity that intercepts the request can be a malicious entity aiming to tamper the matching results.

Furthermore, the Session Clearing and Matching Service must prove itself as the entity authorized to make this computation through this signature. The results, together with the signature, are injected on chain using a call-back function (see step 6). Once reaching the smart contract of the MarketSymbol and MarketSession

the contract will validate that the results are received from the authorized matching module (see step 7), and that the data have not been tampered with, thus trusting the received results.

2.2 Flexibility Request – Order-matching

The mechanism presented in this section is a refined version of the one described in D5.2 and is related to the blockchain driven DR and flexibility management and control. Its role is to determine a subset of prosumers from the aggregator portfolio able to deliver a specific amount of flexibility and to calculate their individual flexibility order curves to optimally match the aggregated request.

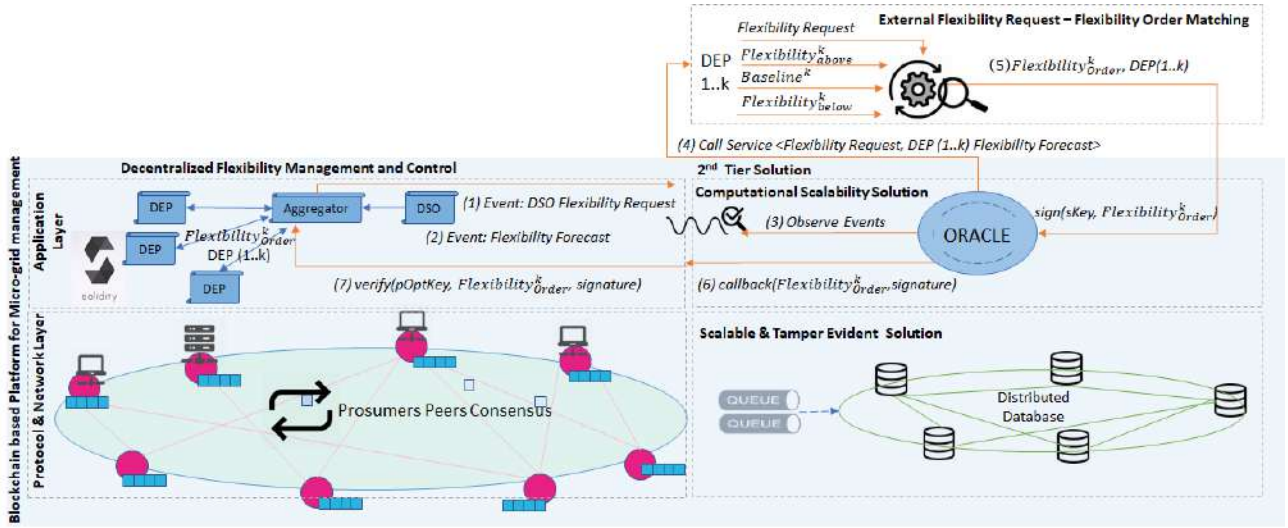


Figure 8. Oracle-based matching of a flexibility request to a set of DEPs flexibility orders

In this case, the optimization problem is to find and select the optimal subset of DEPs from the aggregator portfolio and issue individual flexibility order signals so that their aggregated energy profiles will match the flexibility request signal received by the aggregator from the DSO.

Due to the high overhead of the blockchain-based computations, the algorithm is implemented in an Oracle that is activated when the aggregator receives a flexibility request from the DSO and collects the forecast of energy flexibility form its enrolled distributed energy prosumers (DEPs) to determine at what extend it may satisfy the request.

For each individual prosumer from its portfolio the aggregator needs the individual baseline energy demand and flexibility availability in terms of above and below the baseline maximal values for the time interval of the flexibility request $[0..T]$:

$$DEP_k = \{Baseline^k, Flexibility^k_{Below}, Flexibility^k_{Above}\}, k \in \{1,..N_{DEP}\}$$

The events regarding the information collected for all prosumers (step 1) and the flexibility request (step 2) are intercepted by the Oracle (step 3), which will invoke the external service Flexibility Request – Flexibility Order Matching. The service will select the subset of prosumers and the associated flexibility order of each individual prosumer such that will optimally match as aggregated the flexibility request.

For each prosumer k that is selected, the service will compute a *flexibility order* for the prosumer, with the constraint of being bounded within the availability limits:

$$Flexibility^k_{Below}(t) \le Flexibility^k_{Order}(t) \le Flexibility^k_{Above}(t), t \in \{1..T\}$$

The Flexibility Optimization Service will have assigned a pair of public-private keys, where the public key can be stored on-chain, the results are signed with the private key and return the results and the signature back to the Oracle. The results together with the signature are injected on-chain using a call-back function. Once reaching in the aggregator associated smart contract a validation will be conducted to determine if the results are received from the authorized optimization service, and that the data has not been tampered with.

We model the action of selecting a prosumer k to deliver flexibility for matching the request by a variable $s(k) \in \{0,1\}$. The aggregated flexibility of the determined subset is computed as the sum of flexibility requested to be delivered by the selected prosumers for each time step within the time window T:

$$Flexibility_{Provided}^{Aggregator}(t) = \sum_{k=1}^{N_{DEP}} s(k) * Flexibility_{Order}^{k}(t), s(k) \in \{0,1\}, k \in \{1..N_{DEP}\}$$

Finally, the service aims to minimize the distance between the flexibility request and the aggregated flexibility of the selected prosumers. We use the Manhattan distance as the metric:

$$d\left(Flexibility_{Request}^{DSO}, Flexibility_{Provided}^{Aggregator}\right) = \sum_{t=1}^{T} |Flexibility_{Request}^{DSO}(t) - Flexibility_{Provided}^{Aggregator}(t)|$$

Due to the continuous variables $Flexibility_{Order}^{DEP} \in R^{N_{DEP}*T}$ and integer variables $s \in Z^{N_{DEP}}$, as well as the non-linear objective function, the optimization problem is classified as mixed-Integer nonlinear problem.

2.3 Price-driven Flexibility Bids-Offers Matching

In Section 3 we present in detail the implementation of a price-driven flexibility marketplace over the eDREAM blockchain platform in which various stakeholders can compete on buying and selling energy flexibility. To make it operational beside the smart contracts, a fundamental component is the *matching of flexibility bids and flexibility offers* considering the associated price as an additional vector to the actual amount of flexibility committed. The matching algorithm implementation and integration with the blockchain platform is detailed in the next paragraphs.

In the Price-driven Flexibility Marketplace facilitates the interaction of two types of actors: flexibility buyers and flexibility sellers. The flexibility buyers are entities such that the Aggregators, DSO (Distribution System Operator) or even the TSO (Transmission System Operator), while flexibility sellers are energy prosumers that can adjust their energy demand to deliver flexibility. Furthermore, Aggregators can act either as flexibility buyers (i.e. buy and aggregate flexibility from individual prosumers) or flexibility sellers (i.e. sell aggregated flexibility on their enrolled to interested players such as the DSO) in a specific market session.

To be able to submit a flexibility offer on the Price-driven Flexibility Marketplace, the flexibility seller will leverage its forecast energy data over the time interval of the offer $[0..T]$, as shown in Table 1.

Table 1. Energy information of flexibility seller used for flexibility offer construction

Data structure	Measure Unit	Description
$\boldsymbol{Flex_{seller}^{k}.baseline[T]}$	kWh	Calculated baseline energy demand of the $Flex_{seller}^{k}$ expressed as an array of T energy values over the time interval $[0..T]$
$\boldsymbol{Flex_{seller}^{k}.flex_{above}[T]}$	kWh	Flexibility availability above the baseline of the $Flex_{seller}^{k}$ expressed as an array of T values over the time interval $[0..T]$. These values

		represent the estimated maximum increase of the energy demand of the $Flex^k_{seller}$ over its regular energy baseline.
$\boldsymbol{Flex^k_{seller}.flex_{below}[T]}$	kWh	Flexibility availability below the baseline of the $Flex^k_{seller}$ expressed as an array of T values over the time interval $[0..T]$. These values represent the estimated maximum decrease of the energy demand of the $Flex^k_{seller}$ below its regular energy baseline.

The $Flex^k_{seller}$ sells flexibility, as all basic energy flexibility sellers that participate on Flexibility Market are considered energy consumers, thus they place a flexibility offer showing how much they are willing to alter their baseline, by increasing or decreasing their energy demand. $Flex^k_{seller}.flex_{offer}$ at timestamp t is positive if the $Flex^k_{seller}$ states that it will increase its energy demand, otherwise it is negative showing that the $Flex^k_{seller}$ will decrease its energy demand.

$$Flex^k_{seller}.flex_{offer}[t] = \begin{cases} Flex^k_{seller}[t].flex_{above} - Flex^k_{seller}[t].baseline > 0, if\ Flex^k_{seller}\ increases\ its\ demand \\ Flex^k_{seller}[t].flex_{below} - Flex^k_{seller}[t].baseline < 0, if\ Flex^k_{seller}\ decreases\ its\ demand \end{cases}$$

To submit the offer, the flexibility seller will associate a price to its total energy flexibility $Flex^k_{seller}.price[T]$ to be delivered, measured in $€/kWh$ or $Gwei\ /\ kWh$

Thus, the flexibility offer of a $Flex^k_{seller}$ consists of a tuple indicating how much it is willing to modify its baseline by shifting the flexible energy at the price specified by the $Flex^k_{seller}.price[T]$:

$$Flex^k_{seller}.offer[T] = \{flex_{offer}[T], price[T]\}$$

The flexibility buyer flexibility bids are either positive or negative. If the request is positive, the matched flexibility sellers will have to increase their energy demand by shifting flexible energy in the time interval of the request. If the flexibility request is negative, the matched flexibility seller will have to decrease its energy demand by shifting energy flexibility away from the flexibility request time interval.

$$Flex^k_{buyer}.flex_{request}(t) = \begin{cases} > 0, the\ flexibility\ sellers\ need\ to\ increase\ demand \\ < 0, the\ flexibility\ sellers\ need\ to\ decrease\ demand \end{cases}$$

The bid submitted by a flexibility buyer $Flex^k_{buyer}$, will consist of a tuple from the request and the price associated with the request the buyer is willing to pay.

$$Flex^k_{buyer}.bid[T] = \{flex_{request}[T], flex_{Price}[T]\}$$

The Price-driven Flexibility Marketplace during an open market session collects bids and offers from N flexibility sellers and M flexibility buyers. An Oracle is intercepting the market session end event and will forward a request to an external matching service to determine which of the N offers matches best the M bids (see Figure 9).

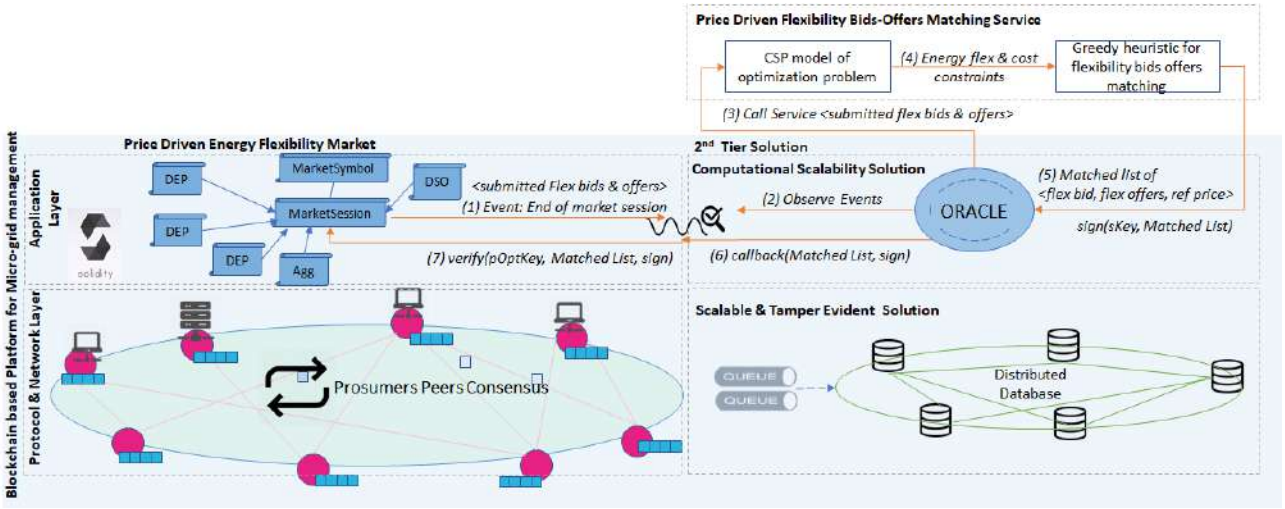


Figure 9. Oracle-based matching of flexibility bids and flexibility offers in the Price-driven Flexibility Marketplace

The Price-driven Flexibility Bids-Offers Matching Service will implement an algorithm that is aiming to find the subset of flexibility sellers that, grouped together, can match each of the M registered flexibility bids over the time window $[0..T]$ at the lowest cost. We define a mapping function f_{match} that maps $Flex^{k}_{seller}.offer$ to the flexibility bid $Flex^{k}_{buyer}.bid$:

$$f_{match}:\{1..N\} \rightarrow \{0..M\}, f_{match}(k) = \begin{cases} p, p \in \{1..M\}, meaning\ that\ Flex^{k}_{seller}.offer\ is\ matched\ to\ Flex^{p}_{buyer}.bid \\ 0, meaning\ that\ Flex^{k}_{seller}.offer\ is\ not\ matched\ to\ any\ flexibility\ buyer\ bid \end{cases}$$

Thus, the input of the matching algorithm is represented by a set of N flexibility offers and M flexibility bids and the output is an instance of the f_{match} mapping function that minimize the optimization objective composed of two components:

- the distance between a $Flex^{p}_{buyer}.bid$ and the total flexibility offered by the set of matched flexibility sellers $Flex^{k}_{seller}.offer$
- the total cost of flexibility computed considering the flexibility prices of the selected prosumers $Cost_{flex}$

These components are weighted by two factors α and β having the sum 1, leading to the optimization objective:

$$MIN(\alpha * \sum_{p=1}^{M}(dist(Flex^{p}_{buyer}, Flex^{p}_{match}) + \beta * \sum_{p=1}^{M}(Cost^{p}_{flex}))$$

The $Flex_{match}(k)$ is computed as the aggregated amount of the flexibility offers of the selected set of flexibility sellers over the time interval $[0..T]$:

$$Flex^{p}_{match}(t) = \sum_{k=1}^{N} selected_{p}(k) * Flex^{k}_{seller}.flex_{offer}[t], \forall\ t \in \{0..T\}$$

where $selected_p(k)$ is a binary array constructed for each $Flex^p_{buyer}$ with the following rule that states that if the match function f_{match} for $Flex^k_{seller}$ is p, meaning that if the sell offer k is matched to the bid p, we mark in the array on position k a value of 1, otherwise we mark 0.

$$selected_p(k) = \begin{cases} 1, if\ f_{match}(k) = p\ for\ flex^k_{seller}.offer \\ 0, otherwise \end{cases}$$

The flexibility cost objective for matching a specific bid p is computed as the sum of the unit prices asked by the selected flexibility offers multiplied with the flexibility amount offered.

$$Cost^p_{flex}(t) = \sum_{k=1}^{N} selected_p(k) * Flex^k_{seller}.flex_{offer}[t] * Flex^k_{seller}.price(t), \forall\ t \in \{0..T\}$$

Finally, the reference price is computed as the average of the prices asked by the selected flexibility sellers offers matched for each of the M flexibility buyer bids for each timestamp from the time interval $[0..T]$:

$$Price_{REF}(T) = \frac{1}{M} * \sum_{p=1}^{M} \frac{1}{\sum_{k=1}^{N} selected_p(k)} * \sum_{k=1}^{N} selected(k) * Flex^k_{seller}.price(t), \forall\ t \in \{0..T\}$$

Figure 10 presents the price-driven flexibility bids and offers matching problem expressed as a constraint satisfaction problem.

Inputs: the set of flexibility offers and flexibility bids from a session $Flex^k_{seller}.offer[T][N]$, $Flex^k_{buyer}.bid[T][M]$
Outputs: the matched flexibility offers to the corresponding bids f_{match}, the flexibility reference price $Price_{REF}[T]$
Objective: Determine the matching function f_{match} **such that**

$$MIN(\alpha * \sum_{p=1}^{M} (dist(Flex^p_{buyer}, Flex^p_{match}) + \beta * \sum_{p=1}^{M} (Cost^p_{flex}))$$

With the following constraints expressed as equations:

C1: $Flex^p_{match}(t) = \sum_{k=1}^{N} selected_p(k) * Flex^k_{seller}.flex_{offer}[t], \forall\ t \in \{0..T\}$

C2: $Cost^p_{flex}(t) = \sum_{k=1}^{N} selected_p(k) * Flex^k_{seller}.flex_{offer}[t] * Flex^k_{seller}.price(t), \forall\ t \in \{0..T\}$

C3: $Price_{REF}(T) = \frac{1}{M} * \sum_{p=1}^{M} \frac{1}{\sum_{k=1}^{N} selected_p(k)} * \sum_{k=1}^{N} selected(k) * Flex^k_{seller}.price(t), \forall\ t \in \{0..T\}$

And the following constraints expressed as inequalities:

C5: $Cost^p_{flex}(t) < Flex^p_{buyer}.Flex_{Price}(t), \forall\ t \in \{0..T\}, k \in \{1..M\}$

Figure 10. Optimization problem of matching flexibility bids and offers

The optimization problem defined in Figure 10 for the flexibility matching algorithm is classified as a mathematical program of class mixed integer nonlinear program (MINLP), having both integer and continuous unknowns. This class of problems is known to be NP-hard [2], thus no polynomial deterministic algorithm can be used to determine a solution. Furthermore, the optimization problem can also be reduced to the well-known bin-packing problem [3], namely the Vector Bin Packing problem [4], that is also NP-hard.

Thus, we propose a solution for solving the optimal flexibility matching problem that uses a greedy heuristic that is derived from the best fit approximation algorithm used for bin-packing. It is derived from the Next Fit Decreasing Height algorithm [3] and used to determine an approximate solution for multidimensional packing problems. It has as inputs the list of N flexibility offers and M flexibility bids that have to be matched, and as output the mapping function f_{match} that optimal matches considering the defined energy and price constraints and optimization objective as well as the reference price of the matching solution computed as

an average of the selected offers prices (see Figure 11Figure 11. Greedy heuristic for flexibility bids offers matching problem).

Inputs: the set of flexibility offers and flexibility bids from a session $Flex^{k}_{seller}.offer[T][N]$, $Flex^{p}_{buyer}.bid[T][M]$
Outputs: the matched flexibility offers to the corresponding bids f_{match}, the flexibility reference price $Price_{REF}[T]$
Begin

1. **Foreach** $Flex^{p}_{buyer}.bid$ in $Flex^{p}_{buyer}.bid[T][M]$ **do**
2. Compute total flexibility bid cost for $Flex^{p}_{buyer}.bid$ as $\sum_{t=1}^{T} flex_{request}[t] * flex_{Price}[t]$
3. **End Foreach**
4. Sort descending by total cost the list of $Flex^{p}_{buyer}.bid[T][M]$
5. **Foreach** $Flex^{k}_{seller}.offer$ in $Flex^{k}_{seller}.offer[T][N]$ **do**
6. Compute total offer price for $Flex^{k}_{seller}.offer$ as $\sum_{t=1}^{T} flex_{offer}[t] * price[t]$
7. **End Foreach**
8. Sort ascending by total price the list of $Flex^{k}_{seller}.offer$
9. $index_{offer} = 0$, Initialize f_{match} with 0
10. **Foreach** $Flex^{k}_{buyer}.bid$ in $Flex^{p}_{buyer}.bid[T][M]$ **do**
11. **While** ! $greaterThan(Flex^{k}_{match}, Flex^{k}_{buyer}.bid)$ **do**
12. $index_{offer}++;$
13. $f_{match}(index_{offer}) = k$
14. **End while**
15. **End Foreach**
16. **Compute** $Price_{REF}$
17. **Return** $f_{match}, Price_{REF}[T]$

End

Figure 11. Greedy heuristic for flexibility bids offers matching problem

The algorithm starts by computing for each flexibility bid the total flexibility cost as the product of the flexibility request and the price associated per each flexibility unit by the flexibility buyer (see lines 1-3). Then, the flexibility bids set submitted in the market session is sorted descending, giving priority to the bids of the flexibility buyers that have associated the largest sum of money (see line 4). The same computation is done for the flexibility offers to determine the associated price and the flexibility sellers are sorted ascending giving priority to the flexibility offers with the smallest price associated (see line 8). The algorithm consists of a for loop that iterates through the flexibility bids set (see lines 11-14), and for each bids of a flexibility buyer p it determine the subset of matching flexibility offers (line 13), while the $Flex^{p}_{match}$ is *not greater than* the $Flex^{p}_{buyer}.bid$. The operator *greater than* is constructed according to the following formula and is true only if all the elements of the first array are greater than the elements of the second array given as argument.

$$greaterThan(Flex^{p}_{match}, Flex^{p}_{buyer}.bid) = \begin{cases} true, if\ \forall t \in \{1..T\}, s.t. Flex^{p}_{match}(t) \geq Flex^{p}_{buyer}.bid(t) \\ false, if\ \exists t \in \{1..T\}, s.t. Flex^{p}_{match}(t) < Flex^{p}_{buyer}.bid(t) \end{cases}$$

The algorithm increments an index that iterates the set of flexibility offers and constructs the mapping f_{match} by assigning the flexibility offer with $index_{offer}$ the flexibility bid with index p. Finally, the algorithm computes the reference price (see line 16).

3 Price-driven Flexibility Marketplace

In this section, we describe the smart contracts-based implementation of the price-driven flexibility marketplace which allows the flexibility trading between flexibility providers (i.e. prosumers) and flexibility requesters (Aggregators or DSO).

Each energy flexibility asset in the micro-grid is enabled to register via smart contracts his flexibility potential as a flexibility offer in a Price-driven Flexibility Market. Similarly, whenever the flexibility buyers have a certain request for flexibility, they will place flexibility bids via smart contracts on the flexibility marketplace. All the bids and offers are registered by the flexibility market contracts, and a flexibility bids/offer matching algorithm (see the previous section for its description) is run via Oracle-based integration to compute the optimal solution for matching the sellers and the buyers of flexibility in the grid. The security and correct integration of the smart contracts with the matching service are ensured by the Oracle entity.

Self-enforcing smart contracts are defined to manage the actual delivery of flexibility (i.e. from the matched flexibility sellers to the flexibility buyers), associating incentive and penalties rates. For each flexibility seller, smart contracts are responsible to evaluate the potential differences between the matched flexibility order and the flexibility actual delivered (i.e. as provided by monitored energy values registered in the distributed ledger). If relevant deviations (i.e. above 10%) are identified, the smart contracts ensure that the flexibility sellers are financially responsible for the imbalance created in the grid by their failure to deliver the promised flexibility.

Five types of smart contracts are involved in Price-driven Flexibility Market operations:

- The market actors' operations are represented and validated on chain by the flexibility buyer contracts (representing the DSO, Aggregators or other actors willing to pay for the delivery of flexibility) and the flexibility seller contract (representing the Prosumers, Aggregator or other actors willing to sell their flexibility);
- The Flexibility Token contract, as an adaptation of ERC-721 [5] modelling the flexibility as a promise of an asset to be delivered.
- The Flexibility Market Manager contract that is managing the actors and their rights to participate in different market sessions, the deposits registered per flexibility bids/offers ensuring financial responsibility for the actors, and the market sessions opening/closing times.
- The Flexibility Market Session contract is responsible to register and store the published flexibility bids and offers corresponding to the active session at the time of registering. Furthermore, the matched flexibility trades are validated, and the correct clearing and settlement are overseen by this contract.

In Figure 12 the interaction flow among the smart contracts involved in the Price-driven Flexibility Marketplace is depicted:

- Step 1: The Flexibility Market Session Smart Contract creates and opens a new energy flexibility trading session for a specific time frame allowing the prosumers to register and publish their flexibility bids/offers.
- Step 2: The flexibility requesters can publish through their associated flexibility buyer contracts their flexibility bid for a specific time period. The buyer needs to specify (2a) a flexibility request profile, specifying the desired quantity of flexibility for each hour interval and the reward offered per unit of flexibility delivered. Furthermore, the buyer needs to prove that he owns the financial means to acquire the specified flexibility requested, thus a payment deposit is made in the Flexibility Market

Manager contract proportional to the requested flexibility. Upon successful validation of the registered flexibility bid (2b), the Flexibility Market Manager will forward (2c) the bid to the corresponding market session, while the manager contract will continue to act as a custodian for the buyer's deposit until the end of the market session.

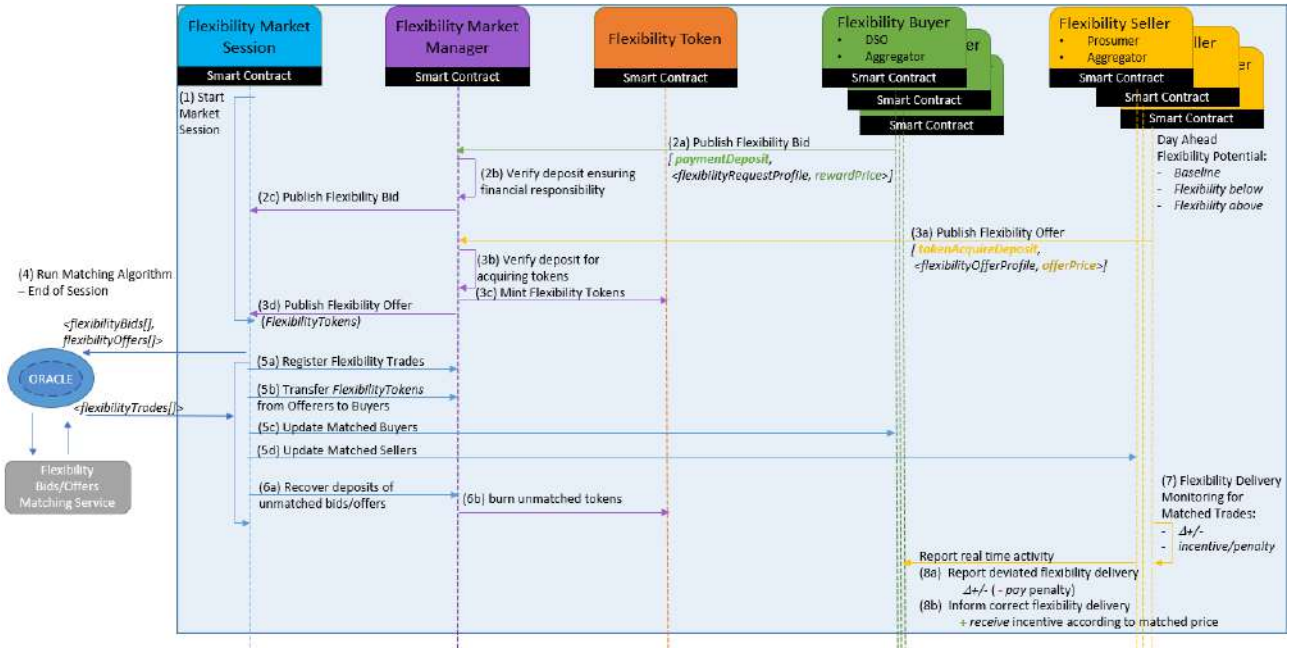


Figure 12. Smart contracts interaction flow in the Price Driver Flexibility Market implementation

- Step 3: The flexibility sellers, can publish their flexibility offer through the flexibility seller contract. The flexibility offer of an actor should be bound by their flexibility potential. The potential of a flexible entity is determined as the capacity to increase -*flexibility above*- or decrease -*flexibility bellow*- their energy consumption baseline at a given time. Since the asset traded in the Flexibility Market is the *FlexibilityToken*, the sellers are required to acquire flexibility tokens proportional to the profile offered (3a). For acquiring the tokens, a deposit is required by the Flexibility Market Manager contract (3b) in order to have an insurance of predictive behaviour from the flexibility seller. The flexibility tokens are generated (3c) and then the sell orders are published in the Flexibility Market Session (3d).
- Step 4: The Market Session Smart Contract will accept flexibility bids and offers until the end of session, at which point the matching algorithm will be triggered off-chain through the Oracle. The matching service will receive the flexibility bids and offers registered during the market session and will return a list of flexibility trades (matched flexibility bids to flexibility orders).
- Step 5: The flexibility trades resulting from matching the bids and offers published, are validated by the Market Session Smart Contract and forwarded to the Flexibility Market Manager Contract (5a). Based on the registered transactions, the transfer of energy token is done from the flexibility sellers to the flexibility buyers (5b). The flexibility buyers and sellers are updated about the matched profiles that are then evaluated during near-real time.
- Step 6: The unmatched orders will be forwarded to the Flexibility Market Manager Smart Contract (6a) who will burn the unused tokens (6b) and will return to each prosumer his initial deposits made upon registration in steps 2 and 3.
- Step 7: In real time, the monitored energy will be forwarded to the flexibility seller smart contract, where the deviation is computed between the flexibility order matched and the flexibility delivered (i.e. computed as the increased/decreased energy monitored towards the baseline value). The

flexibility delivered will be further reported to the flexibility buyer. According to the registered flexibility, if a significant deviation (i.e. above 10%) is registered, the flexibility seller is held accountable and will be required to pay a penalty (8a), otherwise, the seller will be rewarded or his service corresponding to the prices settled by the price-driven flexibility market matching algorithm (8b).

The flexibility buyer Contract is responsible for managing the states and the functionality of the actors that will request flexibility through the Price-driven Flexibility market. Table 2 shows the state variables of the flexibility buyer smart contract.

Table 2. Flexibility buyer smart contract state variables

State Variables		Description
Energy Flexibility Bid		The Flexibility Bid registered by the flexibility buyer in the Price-driven Flexibility Market. An energy demand curve that needs to be followed by one or a group of flexibility sellers due to energy flexibility shifting.
Flexibility service	Incentives	The incentive offered as a reward for making available the flexibility.
	Penalties	The penalties imposed for noncompliance.

In Figure 13, a code snippet from the flexibility buyer contract responsible for the registration and validation of flexibility bids is depicted. The contract requires that the actor signing the published flexibility transaction also deposits enough tokens to be able to ensure the payments in case the flexibility bid is matched. The financial deposit and the bid information are forwarded to the Flexibility Market Manager, that is responsible to process the bid, and redirect it to the corresponding Market Session contract.

```
function publishFlexibilityBuyOrder( int32[] calldata _flexibilityRequest, uint _orderPrice, uint _startHour)  external payable onlyBy(owner) {
    uint deposit = 0;
    for (uint i = 0; i < periodCardinality; i++) {
        deposit += _orderPrice * uint(abs(_flexibilityRequest[i]));
    }
    require (deposit <= msg.value);
    EnergyMarketLibrary.TokenMetadata memory metadata = EnergyMarketLibrary.TokenMetadata(
                                                                    _startHour,
                                                                EnergyMarketLibrary.Type.FLEXIBILITY,
                                                                tx.origin);
    FlexibilityMarketManager manager = FlexibilityMarketManager(flexibilityMarketManager);
    manager.registerBuyOrder.value(msg.value)(_flexibilityRequest, _orderPrice, metadata);
    emit PublishedBuyOrder(_flexibilityRequest, _orderPrice, _startHour);
}
```

Figure 13. Buyer registering a flexibility bid

The flexibility seller contract is responsible to manage the states and the functionality of the actors selling flexibility in the Price-driven Flexibility market. In Table 3 the state variables of the flexibility seller smart contract are depicted.

Table 3 Flexibility seller smart contract state variables

State Variable	Description
Baseline Energy Profile	Regular energy profile of an asset determined based on the average of measured energy values in the past.

Below Flexibility Potential	Estimated capacity to lower energy consumption due to energy flexibility shifting. Computed based on the historical energy consumption.
Above Flexibility Potential	Estimated capacity to increase energy consumption due to energy flexibility shifting. Computed based on the historical energy consumption.
Current Energy Profile	Monitored energy consumption values of the asset acquired by the IoT smart energy metering devices
Flexibility Order Profile	Flexibility profile resulted after the matching algorithm has run. Injected by the market session contract into the *FlexibilitySeller* smart contract and needs to be followed during delivery time.

Figure 14 shows the smart contracts associated with flexibility buyers and sellers where implemented in the price-driven flexibility marketplace.

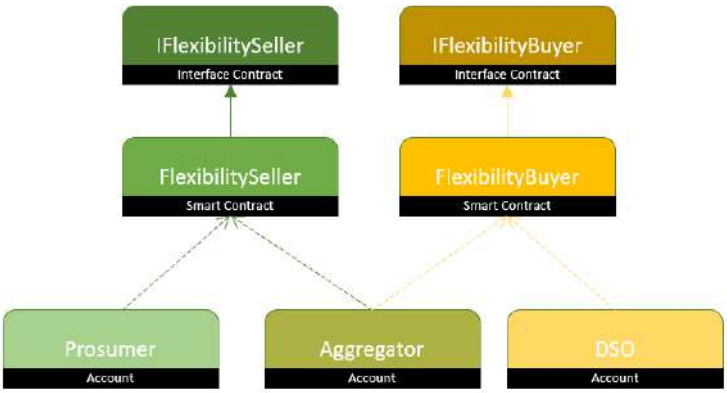


Figure 14. Flexibility buyer and seller smart contract modelled in the flexibility marketplace

The Market Manager and Market Session contracts are implemented in a compatible manner according to the specification of the eDREAM blockchain platform detailed in D5.2.

The Flexibility Market Session contract will register all the flexibility bids and offers as long as the session is opened. By the end of the session, the flexibility bids/offers matching algorithm will be triggered. The matched trades will update the flexibility buyer contract, signalling that the request has been matched, and the activity of the matched sellers should be registered and evaluated based on the monitored values. The financial settlement rules will be applied by the flexibility buyer contract, in near-real-time according to the specification of the eDREAM solution detailed in deliverable "D5.3 - Consensus based techniques for DR validation and financial settlement".

Based on the registered flexibility bids from the Flexibility Market Session a flexibility seller can choose to either decrease or increase his energy consumption, according to his flexibility availability, thus a flexibility sell offer will be published accordingly, as depicted in Figure 15. Since the seller promises the delivery of flexibility as asset in the future, the seller should offer a guarantee of good behaviour. Thus, a token creation deposit is required, proportional to the promised flexibility (see line 7). The guarantee will be returned to the seller as soon as the real time activity is registered.

```
function publishFlexibilitySellOrder( int32[] calldata _estimatedFlexibilityValues,uint _orderPrice, uint _startHour)  external  payable onlyBy(owner) {
    require(_estimatedFlexibilityValues.length == periodCardinality);
    FlexibilityMarketManager mngm = FlexibilityMarketManager(flexibilityMarketManager);
    uint tokenDepositPerUnit = mngm.getTokenDepositPerUnit();
    uint tokenDeposit =0;
    for(uint i =0; i< periodCardinality; i++){
        tokenDeposit += tokenDepositPerUnit * uint(abs(_estimatedFlexibilityValues[i]));
    }
    require (tokenDeposit <= msg.value);
    EnergyMarketLibrary.TokenMetadata memory metadata = EnergyMarketLibrary.TokenMetadata(
                                                        _startHour,
                                                        EnergyMarketLibrary.Type.FLEXIBILITY,
                                                        tx.origin);

    mngm.registerSellOrder.value(tokenDeposit)( _estimatedFlexibilityValues, _orderPrice, metadata);
    emit PublishedSellOrder(_estimatedFlexibilityValues, _orderPrice, _startHour);
}
```

Figure 15. Seller registering a flexibility offer

By the end of the flexibility market session, after the flexibility bids and offers are matched, the *flexibility seller* contract will be updated with information about the matched flexibility profile (i.e. the flexibility order) and the matched *flexibility buyer* who will evaluate the activity in real time will be provided.

We have adopted the ERC-721 standard for representing the traded energy flexibility assets as non-fungible tokens in the blockchain system. The specifics of a token instance are specified by different fields in the TokenMetadata structure. Several adaptations have been made in order to provide a market compatible representation of the flexibility asset to follow the specification of the eDREAM blockchain platform (see Figure 16). The Flexibility Market Session Contract enforces the trading rules by allowing the publisher full management rights over the lifecycle of the bid/offer, being able to update, suspend or re-activate it. The bids/offers and the flexibility trade structures used by the smart contracts are also presented in Figure 16.

```
struct FlexibilityOrder {
    bytes32 id;
    OrderSide orderSide;
    address _address;
    uint timestamp;

    uint tokenId;
    TokenMetadata metadata;

    int32[] quantity;
    uint price;
}
```

```
struct FlexibilityTrade{
    bytes32 id;
    bytes32 buyOrderId;
    bytes32 sellOrderId;
    address buyingAddress;
    address payable sellingAddress;
    uint timestamp;

    uint tokenId;
    int32[] quantity;
    uint price;
}
```

Figure 16 Flexibility Order (i.e. bid or offer) and Flexibility Trade data structure

4 Blockchain Platform for Micro-Grid Energy Management

The eDREAM blockchain based prototype platform for micro-grid energy management offers a peer-to-peer energy marketplace, decentralized management and control of flexibility services and a price-driven flexibility marketplace (see Figure 17).

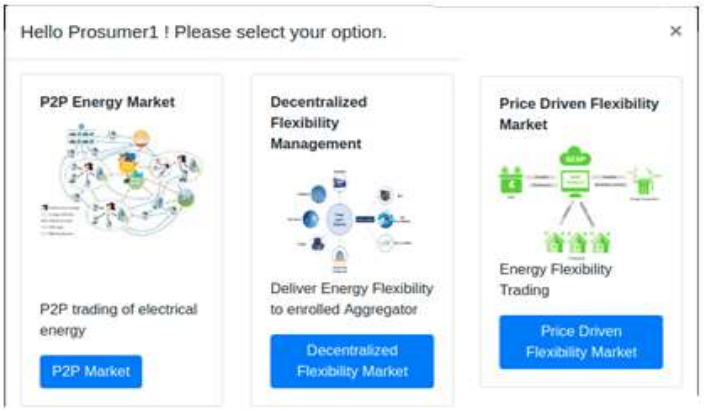


Figure 17. Energy stakeholder options in the eDREAM blockchain platform for micro-grid energy management

4.1 Peer-to-peer Energy Trading

It allows the energy prosumers to trade energy in a peer-to-peer fashion. The main actors involved in this process are the energy buyers or sellers (i.e. usually the prosumers) and the market operator.

After login a prosumer login has access to a page displaying relevant information on its forecast energy production or consumption as well as the reference energy price from the previous market session. All this information will be used for placing and energy bid for buying energy or an energy offer for selling energy in the current market session (see Figure 18).

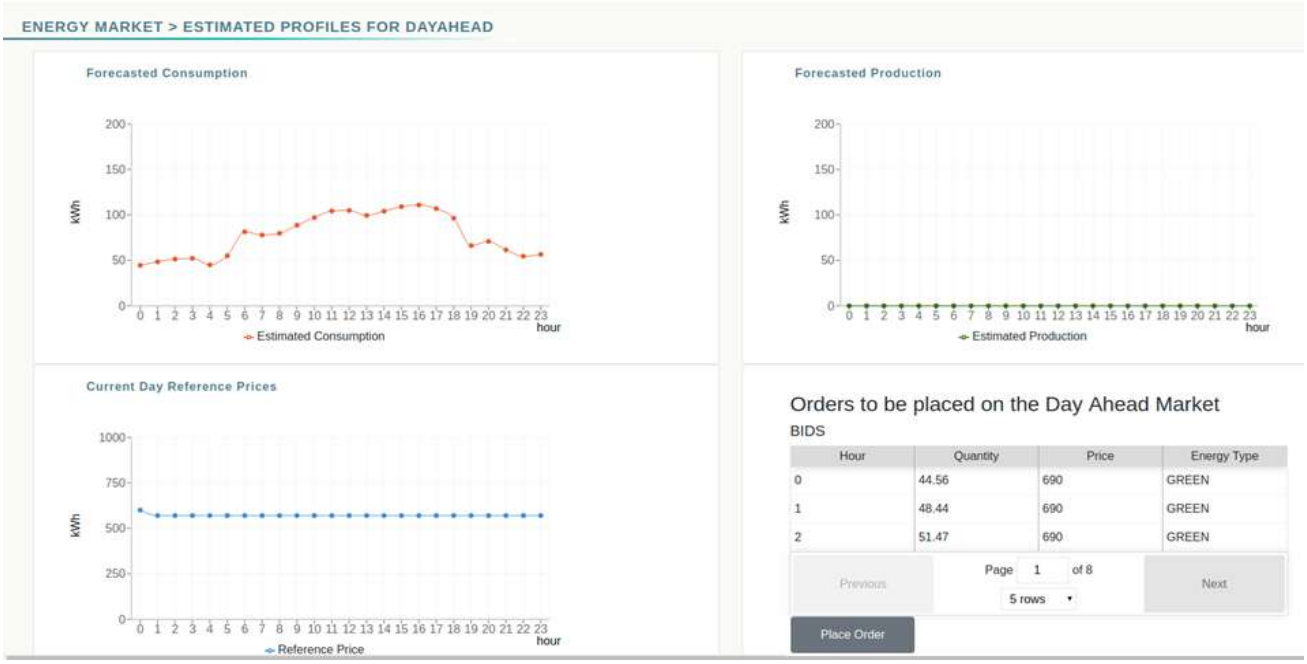


Figure 18. Web page for showing the Prosumer estimated energy profiles and potential bids/offers

After the market session ends, the matching algorithms from Section 2.1 are run and the energy bids and offers as energy transactions are returned (see Figure 19).

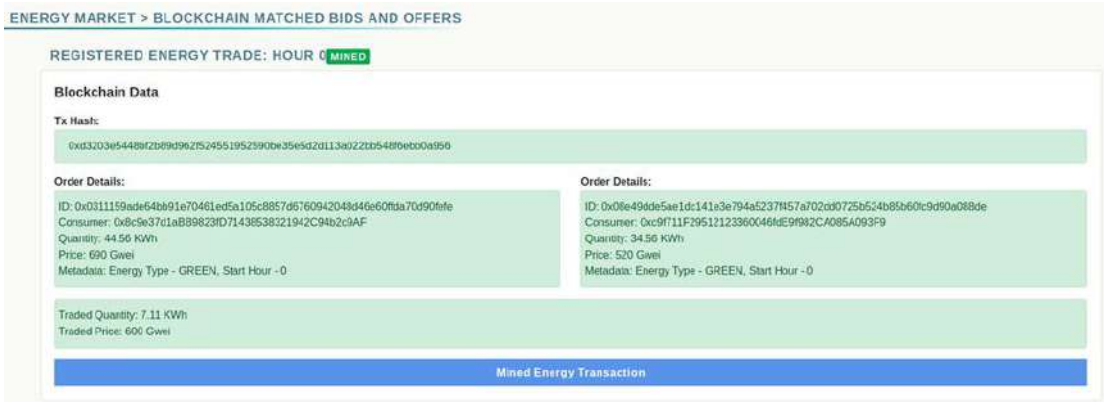


Figure 19. Energy transaction registered on the blockchain platform

The energy transactions between peers are not yet validated until the actual moment of energy delivery. At that point the monitored energy values will be used to validate the transactions and the financial settlement is conducted by delivering the tokens to the wallets of participants in the transaction (see Figure 20).

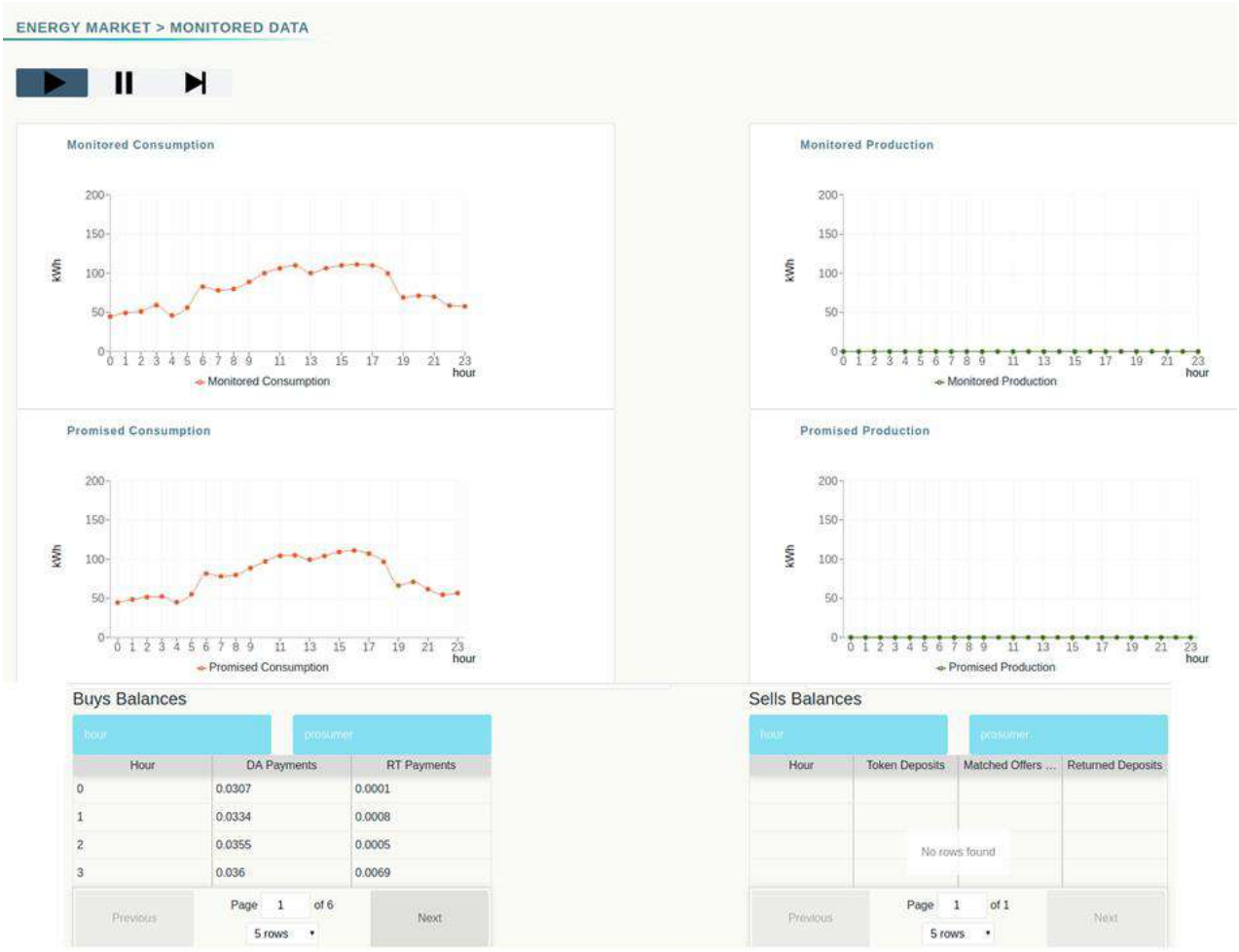


Figure 20. Energy transaction validation using monitored data and participants accounts settlement

The Energy Market Operator, upon its login, will see all the energy bids and offers registered in a market session in terms of promised volumes the be traded and associated prices (see Figure 21).

Figure 21. Energy Market Operator view on energy bids and offers submitted in a market session

In addition, after running the energy bids and offers matching algorithms it can check the matched ones as trades as well as the calculated reference price for the session (see Figure 22).

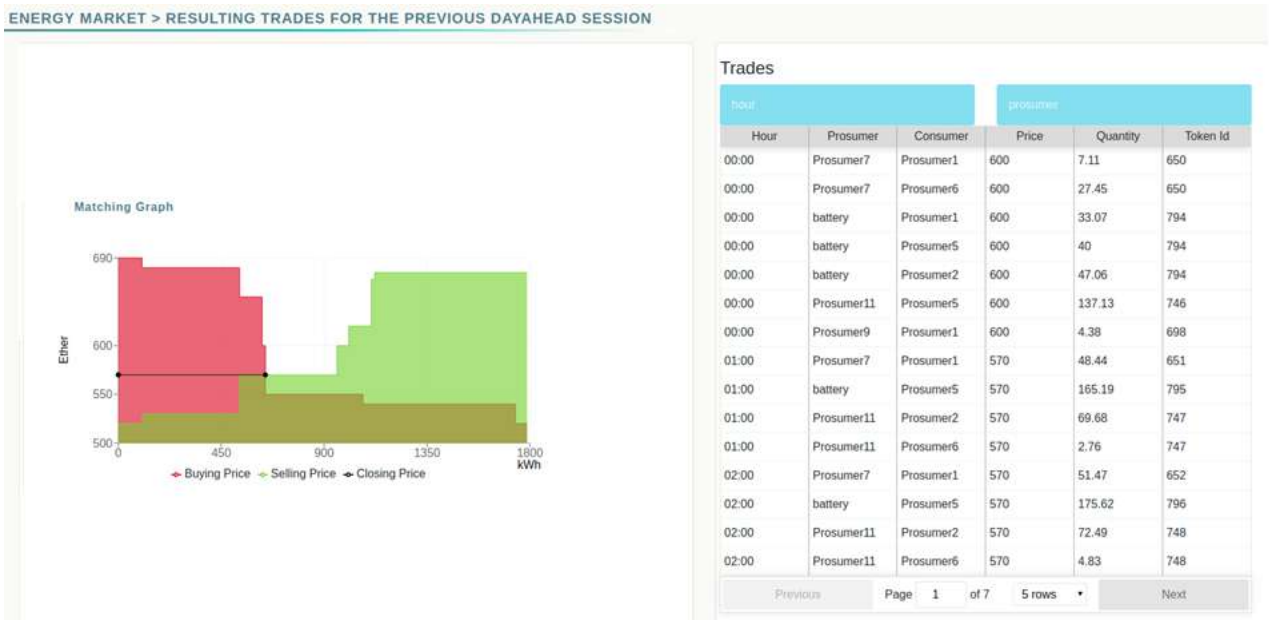


Figure 22. Energy Market Operator view matched bids and offers and reference price calculation for a market session

4.2 Decentralized Flexibility Services Management and Control

In the case of decentralized flexibility services management and control, specific web interfaces have been developed to expose the functionality and facilitate the interaction of the following types of actors such as DSO, flexibility aggregator and prosumers.

The first actor is the DSO, who - after the login to the platform - checks the forecast energy demand and production at the level of specific micro-grid and may create a specific flexibility request in case of potential un-balances are detected. In case the predicted levels on renewable energy generation are above the consumption, the DSO will ask for energy flexibility shifting in the direction of increasing the energy demand and consume as much as possible the generated energy locally. If the predicted generation is below the

consumption, then it may ask for energy flexibility shifting in the direction of lowering the consumption in specific time windows. Accordingly, the flexibility request is being issued (see Figure 23).

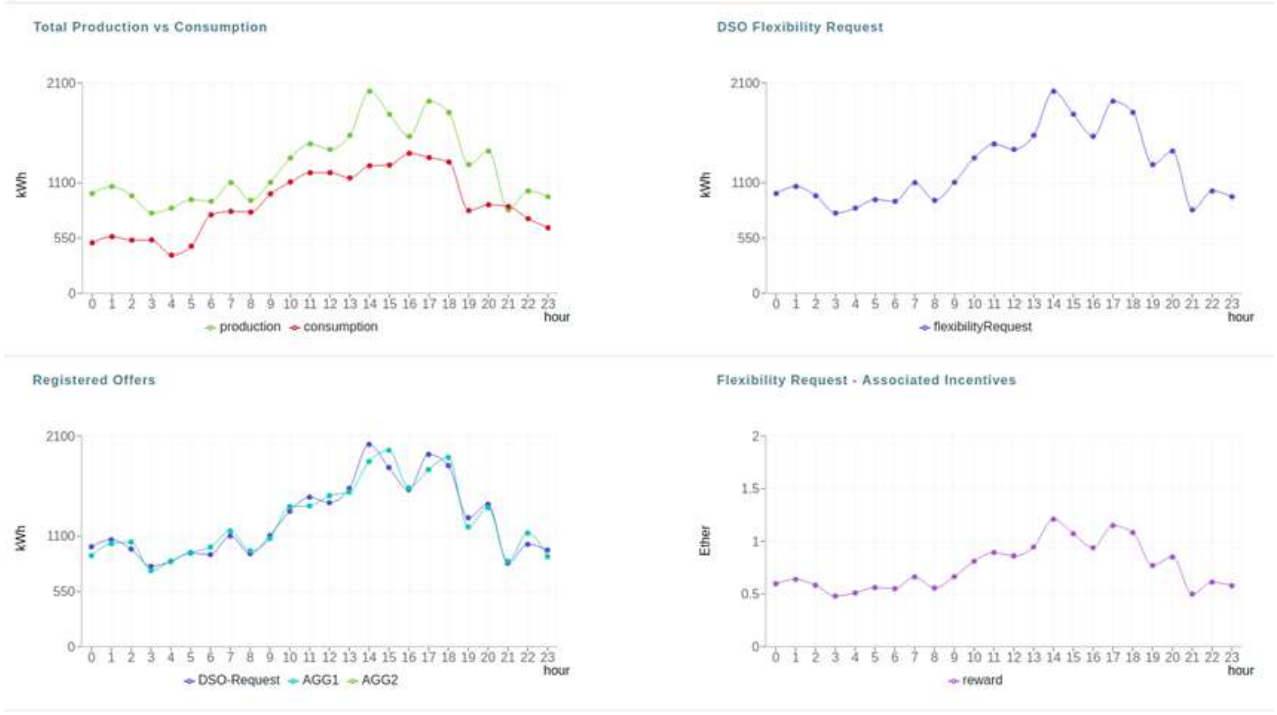
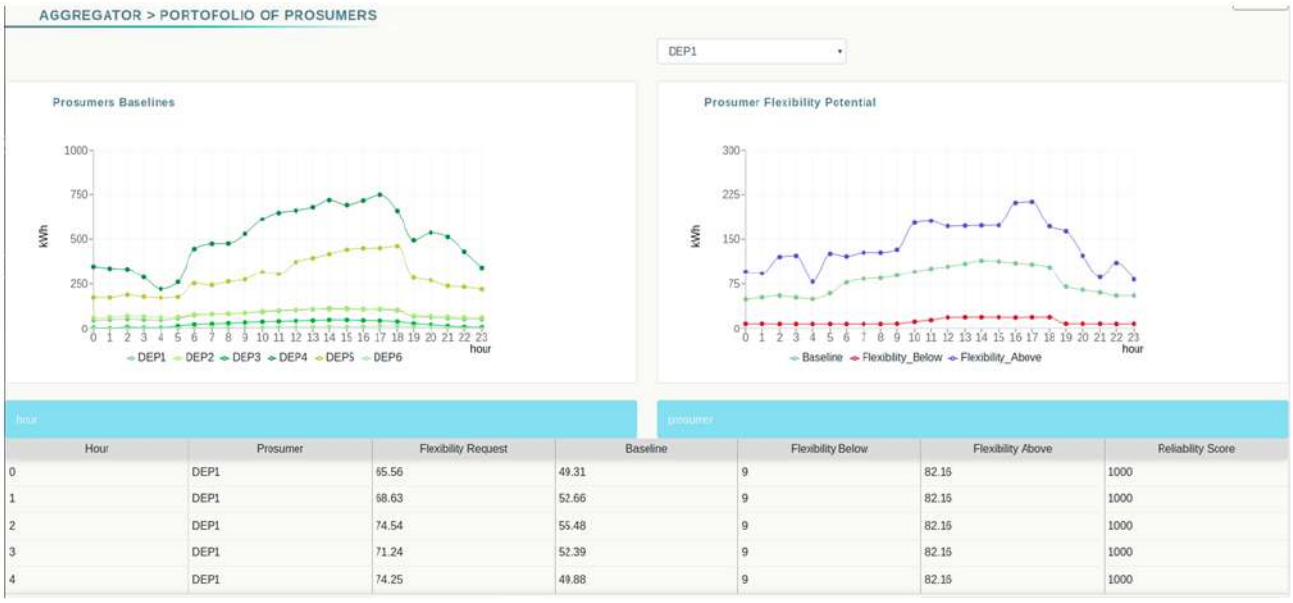


Figure 23. DSO view on micro -grid predicted state and flexibility request generation

After login, the aggregator can see the flexibility request issued by the DSO and will start by investigating how to aggregate the flexibility of its enrolled prosumers to address the flexibility request and deliver the aggregated amount. The aggregator may check for the entire portfolio the forecast flexibility availability (i.e. above and below the regular baseline) as well as spilt for each individual prosumer (see Figure 24).



Hour	Prosumer	Flexibility Request	Baseline	Flexibility Below	Flexibility Above	Reliability Score
0	DEP1	65.56	49.31	9	82.16	1000
1	DEP1	68.63	52.66	9	82.16	1000
2	DEP1	74.54	55.48	9	82.16	1000
3	DEP1	71.24	52.39	9	82.16	1000
4	DEP1	74.25	49.88	9	82.16	1000

Figure 24. Aggregator view on the forecast flexibility availability of enrolled prosumers

After running the flexibility request to flexibility order, the aggregator may check the split of the flexibility request onto flexibility order signals tailored to each individual prosumer flexibility availability (see Figure 25).

Hour	Prosumer	Flexibility Request	Baseline	Flexibility Below	Flexbility Above	Reliability Score
0	DEP1	65.56	49.31	9	82.16	1000
1	DEP1	68.63	52.66	9	82.16	1000
2	DEP1	74.54	55.48	9	82.16	1000
3	DEP1	71.24	52.39	9	82.16	1000
4	DEP1	74.25	49.88	9	82.16	1000

Previous Page 1 of 29 5 rows Next

Figure 25. Flexibility request to prosumer flexibility order split

Moreover, during actual flexibility delivery, the aggregator may check the compliance of the enrolled prosumers to the flexibility order curve based on the actual monitored data assess the potential relevant deviations in terms of delta +/- energy differences and use this information for the financial settlement of the flexibility service (see Figure 26).

0	DEP1	-4.49	6.56	-6.56	-4.49
1	DEP1	9.33	6.86	-6.86	9.33
2	DEP1	9.71	7.45	-7.45	9.71
3	DEP1	-7.2	7.12	-7.12	-7.2
4	DEP1	-10.11	7.43	-7.43	-10.11

Previous Page 1 of 29 5 rows Next

Figure 26. Flexibility delivery and compliance of the prosumers enrolled with the service

Finally, a prosumer who is enrolled with the aggregator may check the forecast flexibility availability levels above and below the baselines as well as the flexibility order signal received from the aggregator (see Figure 27).

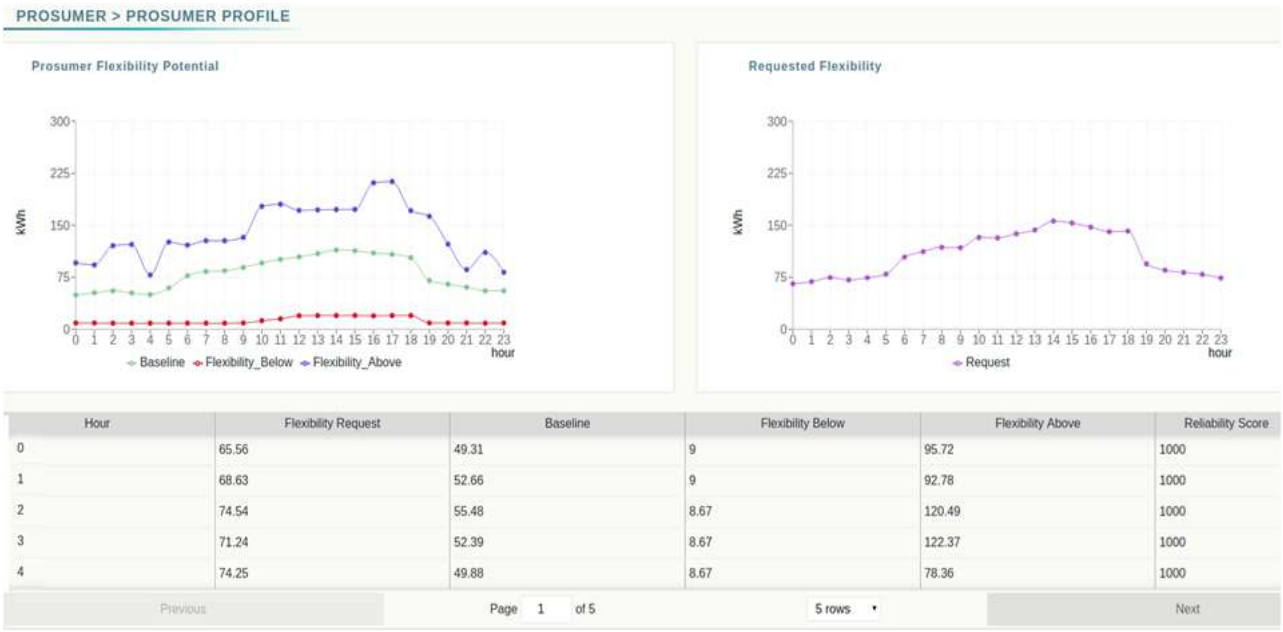


Figure 27. Prosumer view on flexibility availability and aggregator flexibility order signal

During flexibility delivery time the prosumer may check its levels of compliance to the flexibility order signal based on the actual monitored energy data (see Figure 28).

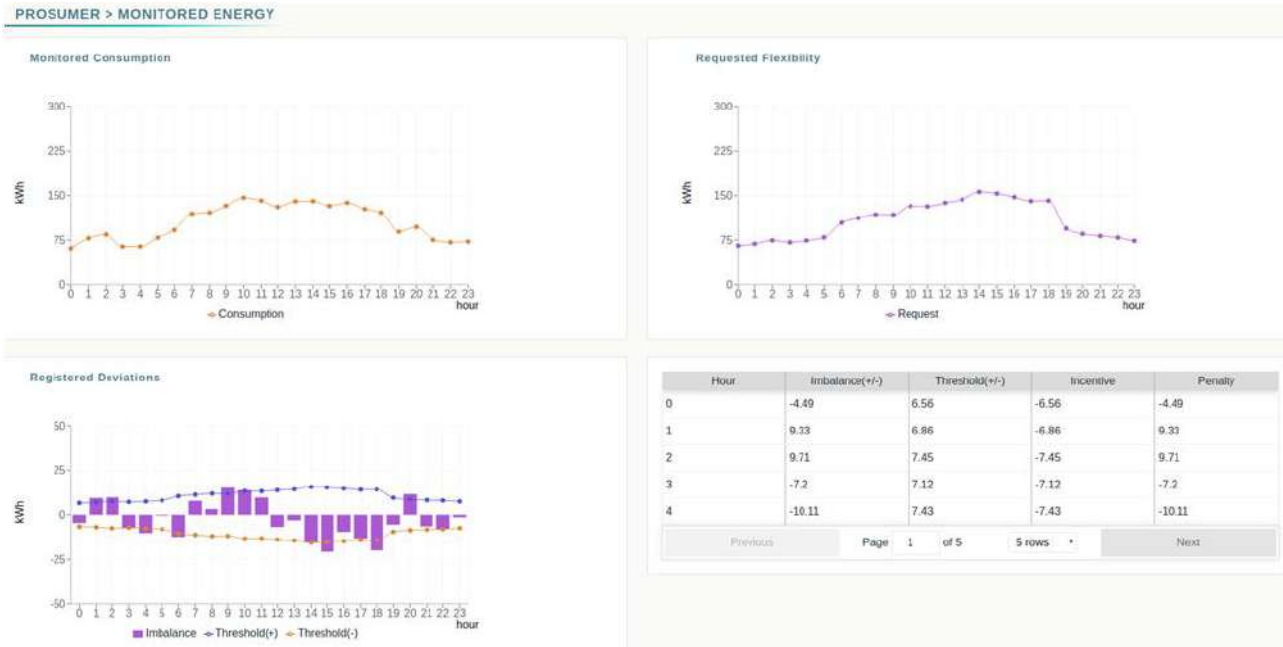


Figure 28. Prosumer actual flexibility delivery information

4.3 Price-driven Flexibility Marketplace

In the price-driven flexibility marketplace, we have three types of roles: flexibility sellers (i.e. prosumers or aggregators), flexibility buyers (DSO, aggregators, etc.) and flexibility market operator.

The flexibility seller may place flexibility offers in the flexibility market session in terms of increasing or decreasing their energy profile in relation to the baseline (i.e. by shifting energy flexibility). After their registration they will be shown information from the previous market session relevant in constructing flexibility sell offers (see Figure 29).

Figure 29. Previous market session flexibility bid / offers volumes and prices

Then the flexibility seller may check their calculated baseline and estimated flexibility availability for the next day either above or below the baseline. This information is then used for placing flexibility sell offer on the flexibility market (see Figure 30).

Figure 30. Forecast flexibility availability and sell flexibility offer

At the end of the session, the flexibility seller may visualize the matched energy flexibility transaction stored in the blockchain platform (see Figure 31).

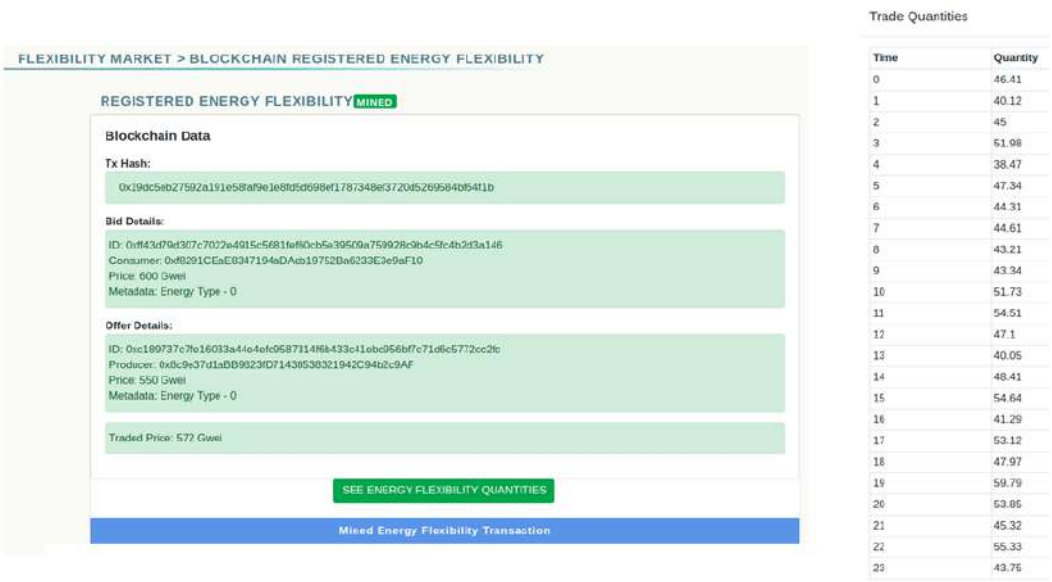


Figure 31. Registered flexibility transaction in the blockchain

At the moment of the delivery, the transaction will be validated using the monitored energy data of the flexibility seller, deviations from the matched flexibility bid curve will be determined as delta +/- energy differences and the wallets of the flexibility transactions participants will be settled accordingly (see Figure 32).

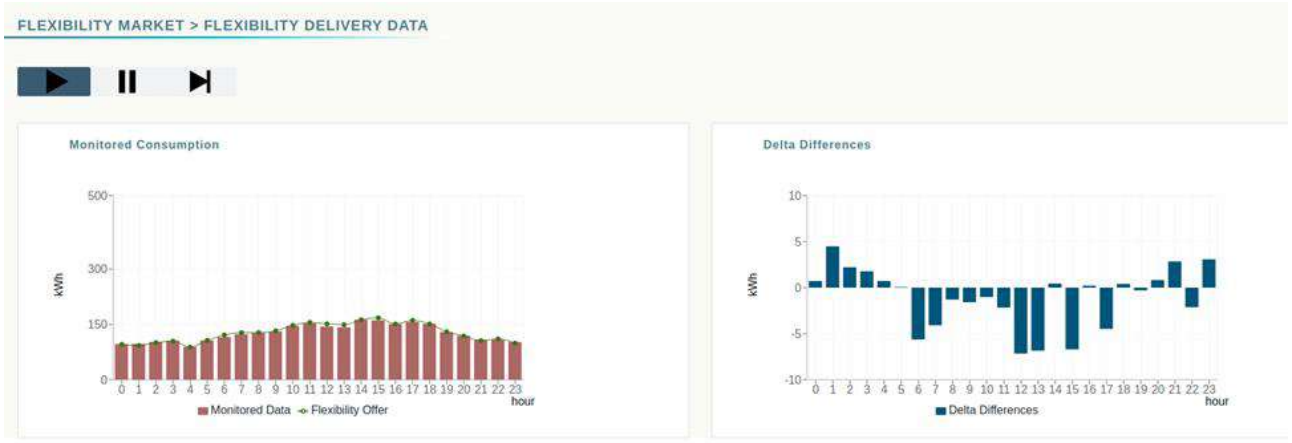


Figure 32. Monitoring the flexibility delivery and flexibility transaction settlement

Flexibility buyers will follow a very similar interaction flow, but in the case of DSO, for example, other relevant information will be displayed for supporting the publish of flexibility buy request on the flexibility market (see Figure 33).

Figure 33. Relevant information for creating flexibility buy bid displayed for the DSO

At the same time, the flexibility buyer may check the actual delivery of flexibility by the group of flexibility sellers (i.e. at the level of individual participant and as a whole) that were matched to its flexibility bid request and use the potential delta +/-energy deviations in the financial statement of the flexibility transaction (see Figure 34).

Figure 34. Flexibility buyer monitoring the flexibility delivery of matched flexibility sellers

Finally, the market operator may check the total flexibility bids and flexibility offers submitted during a market session and upon the session end the ones matched as energy flexibility transactions and the flexibility reference price (see Figure 35).

Figure 35. Flexibility Market Operator view on energy flexibility bids and offers submitted in a market session as well as on the matched ones

5 Conclusion

In this deliverable, we have presented the final version of eDREAM blockchain-based platform for the management of energy micro-grids. The following management solutions have been implemented and can be used by various energy stakeholders (i.e. prosumers, aggregators, DSO, etc.): a P2P energy marketplace, a mechanism for decentralized implementation and control of flexibility services, and a price-driven flexibility marketplace for decentralized trading of assets flexibility.

in addition to the blockchain platform's initial version (reported in D5.2), in this deliverable we have presented the definition, implementation and integration of Demand-Offer matching mechanism for all the management alternatives offered by the eDREAM blockchain platform. At the same time, we have described the implementation of the price-driven flexibility marketplace which is leveraging on self-enforcing smart contracts for allowing the flexibility buyers and the sellers to trade the energy flexibility of various assets. Finally, we have reported the refinement and implementation details of the web-based interfaces to facilitate the interaction of the energy stakeholders with the blockchain platform for exploiting the defined energy management solutions.

The blockchain platform has been tested and validated using the datasets provided by ASM Terni and Kiwi project pilots, showing its effectiveness in addressing various management problems of the micro-grids (see also D5.1, D5.2, D5.3, and D5.4). Furthermore, the developed decentralized management solutions, algorithms and techniques have been validated by the scientific community (i.e. they have been published in relevant Web of Knowledge Quartile 1 and 2 computer science journals).

References

[1] Partition problem, https://en.wikipedia.org/wiki/Partition_problem

[2] P. Belotti, C. Kirches, S. Leyffer, J. Linderoth, J. Luedtke, and A. Mahajan, Mixed-Integer Nonlinear Optimization, Acta Numerica 22 (2013) pp. 1-131, https://doi.org/10.1017/S0962492913000032

[3] Binzhou Xia, Zhiyi Tan, Tighter bounds of the First Fit algorithm for the bin-packing problem, Discrete Applied Mathematics, Volume 158, Issue 15, 2010, Pages 1668-1675, ISSN 0166-218X,

[4] Henrik I. Christensen, Arindam Khan, Sebastian Pokutta, Prasad Tetali, Approximation and online algorithms for multidimensional bin packing: A survey, Computer Science Review, Volume 24, 2017, Pages 63-79, ISSN 1574-0137, https://doi.org/10.1016/j.cosrev.2016.12.001

[5] ERC-721 open standard foe non-fungible tokens on the Ethereum blockchain, http://erc721.org/