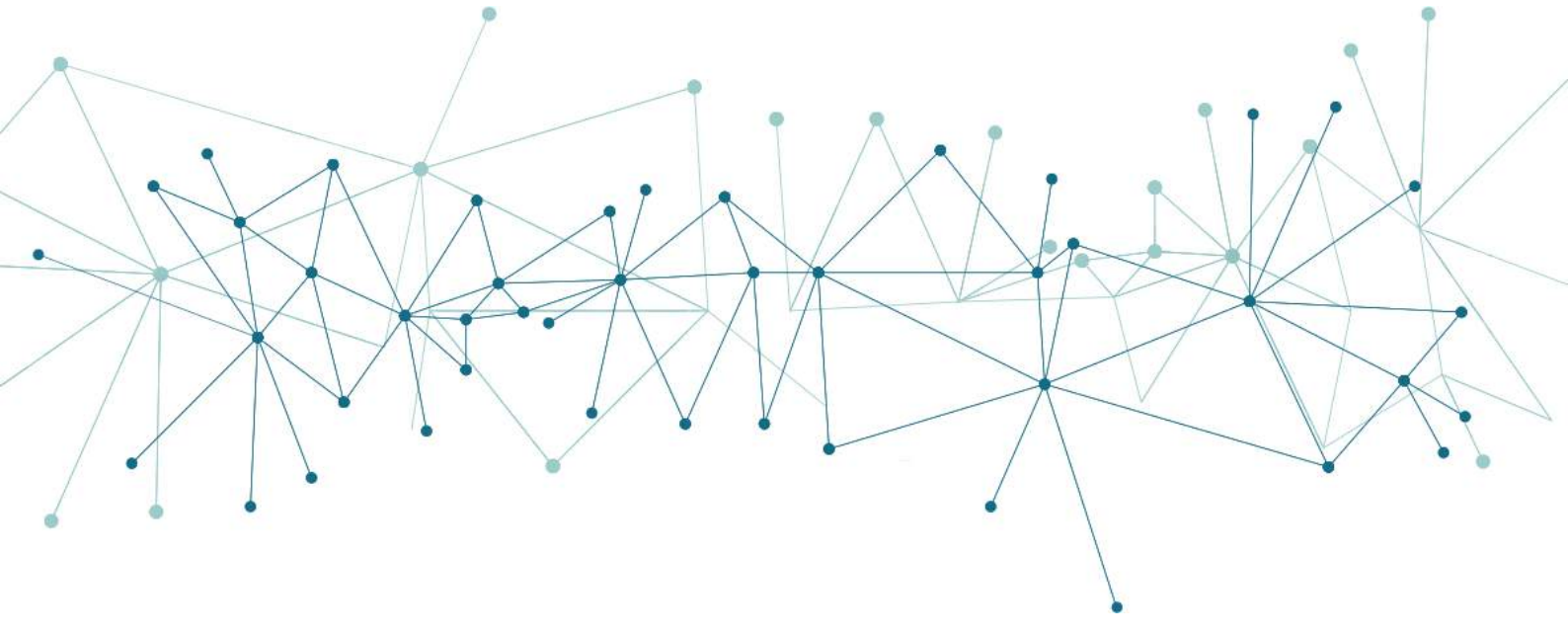
The eDREAM project is co-founded by the EU's Horizon 2020 innovation programme under grant agreement No 774478

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

DELIVERABLE: 5.2 Self-enforcing smart contracts for DR tracking and control V1

Authors: Claudia Pop (TUC), Marcel Antal (TUC), Tudor Cioara (TUC)



Imprint

D5.2. Blockchain-driven self-enforcing smart contracts for DR energy transactions modelling, tracking and decentralized control (Month 17)

Contractual Date of Delivery to the EC:	31.05.2019
Actual Date of Delivery to the EC:	31.05.2019
Author(s):	Tudor Cioara (TUC), Marcel Antal (TUC), Claudia Pop (TUC)
Participant(s):	Contributors: Ionut Anghel (TUC), Viorica Chifu (TUC), Cristina Pop (TUC), Ioan Salomie (TUC), Giuseppe Raveduto (ENG), Francesco Bellesini (EMOT), Alessio Cavadenti (ASM)
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:	1.0

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
JSON	JavaScript Object Notation
P2P	Peer to Peer
REST	Representational State Transfer
URI	Uniform Resource Identifier
WP	Work Package

Executive Summary

In this deliverable we present the development of a blockchain-based platform for micro-grid energy management which provides two types of decentralized solutions: (1) a price-driven peer-to-peer energy marketplace enacting the local trading of energy and consumption of energy production and (2) a flexibility marketplace for distributed provisioning and control of energy flexibility services, from DSO to aggregators and further to their enrolled prosumers. The deliverable presents in detail the self-enforcing smart contracts which are implementing the blockchain-based platform decentralized management and control features.

First, the peer-to-peer energy marketplace is implemented at the local micro-grid level and allows any prosumer regardless of size to directly participate in the market trading sessions. The market acts as an energy management mechanism by rewarding the consumption of energy when it is locally available and making sure that the potential energy imbalances at the micro-grid level are locally addressed. Non-fungible energy-adapted tokens are defined and generated at a rate proportional with energy production, thereby transforming the energy into a transactable digital asset. The producers and consumers use tokens to participate in the electricity market sessions and will leverage on self-enforcing smart contracts to submit energy bids/offers and transact energy in a peer-to-peer fashion. We have used a public permission blockchain deployment and we have managed the new prosumers registration and permission control using smart contracts. The market sessions management contracts keep an order book of all the registered energy bids and offers, while the market manager contracts keep track of all the opened sessions and make the necessary validations regarding the prosumer activity. The energy transactions are registered and replicated in future blockchain blocks across all the nodes in the network. The decision on the actual share of energy which has been effectively delivered/consumed by each peer and associated financial settlement will be unanimously agreed upon through consensus by all the other network peers, also considering the actual monitored energy information. Second, the flexibility services provisioning and management is achieved through self-enforcing smart contracts enabling decentralized coordinated control; this empowers the energy stakeholders such as the DSO or aggregators to assess and trace the share of flexibility, which is activated in near real-time at the micro-grid level. Each prosumer in the micro-grid has been enabled to offer, via smart contracts, his flexibility to specific aggregators to which s/he is enrolled. When the DSO identifies congestion point, the DSO sends a flexibility request via smart contracts to the flexibility marketplace and the aggregators are providing their flexibility offers. The self-enforcing smart contracts are defined to manage the levels of energy demand flexibility (i.e. from aggregators and their enrolled prosumers on one side and from aggregators to the DSO on the other side), with incentive and penalty rates. Smart contracts evaluate the difference between the requested energy flexibility (i.e. a curve signal) and the flexibility actual delivered (as relived by monitored energy values registered in the distributed ledger). If deviations are identified, specific actions will be taken to compensate for the lack of flexibility or rebalance the energy demand with energy production, thus, the smart contracts act as a decentralized control mechanism. At the same time, we have provided an algorithm to select the optimal subset of prosumers from an aggregator's portfolio and their flexibility energy profiles so that the aggregator can match best the DSO flexibility request.

The initial simulated results are showing the potential of the blockchain platform and smart contracts for the distributed management of flexibility at the micro-grid level. The flexibility requests signals are followed with high accuracy, while the deviations in the share of flexibility actual delivered are identified and addressed in a near real-time fashion. Energy results show the potential of matching locally in a peer-to-peer fashion the energy demand and energy production, thereby avoiding escalating potential imbalances. The blockchain-based marketplace allows tracking energy transactions and the settlement of prosumers' wallets based on actual energy monitored data in real time.

1 Introduction

1.1 Purpose

This report provides an overview of the work carried out in Task 5.2 in the direction creating a blockchain-based platform for the distributed control and management of the micro-grid. The platform is leveraging on the implementation of:

- a price-driven peer-to-peer energy marketplace allowing the local trading and consumption of energy produced at the micro-grid level and
- a flexibility marketplace for flexibility services distributed provisioning and management from DSO to aggregators and further to their enrolled prosumers based on blockchain-distributed control, allowing to assess and track in near real time the share of flexible energy actually delivered.

Self-enforcing smart contracts are exploited for the implementation of both micro-grid management approaches.

It addresses the following WP5 objective: "To define and implement self-enforcing smart contracts for tracking and controlling the energy transactions and DR flexibility services in smart energy grids in a fully decentralized manner".

1.2 Relation to other activities

WP5 and in particular T5.2 uses 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 assessment and distributed optimization problem solving to implement micro-grid management mechanisms, leveraging on blockchain and self-enforcing smart contracts.

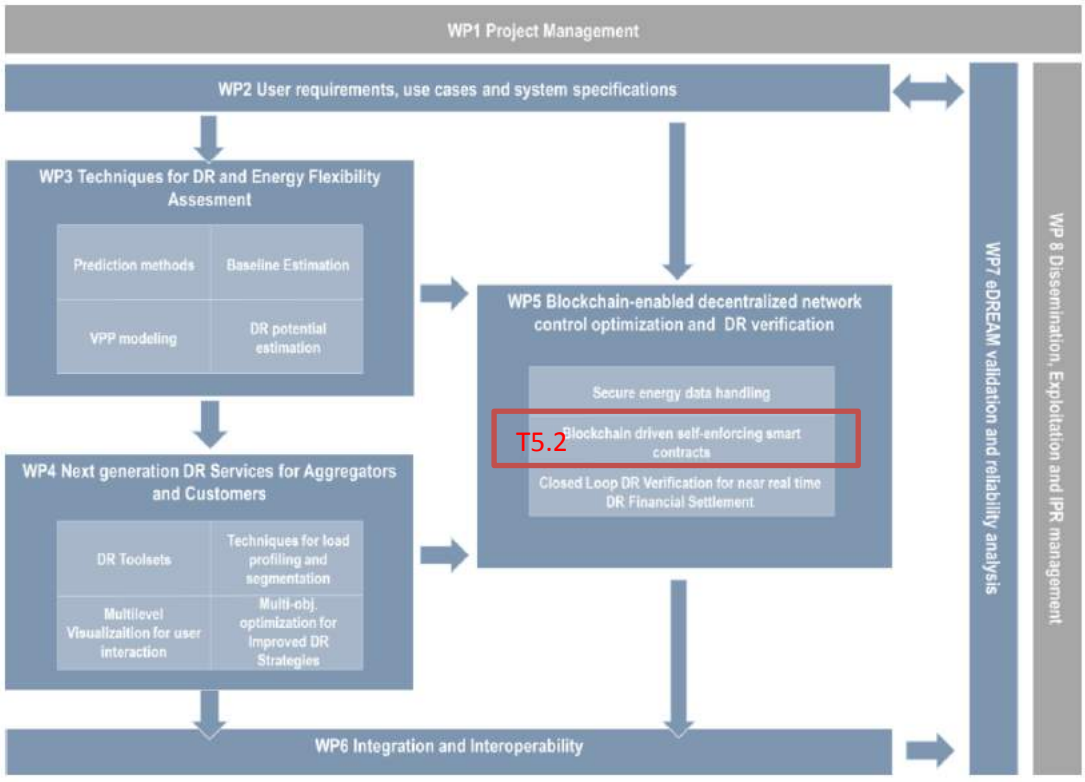


Figure 1. eDREAM PERT chart showing WP5 in relation to other work packages

1.3 Structure of the document

The remainder of the report is organized as follows:

- Section 2 presents a blockchain-based specification of a peer-to-peer energy marketplace which may be implemented at micro-grid level assuring that the generated energy is locally traded and consumed;
- Section 3 describes a blockchain-driven flexibility services provisioning and management from DSO through aggregators and their enrolled prosumers;
- Section 4 presents the architectures and the public APIs of the eDREAM component implementing the blockchain platform for micro-grid energy management;
- Section 5 presents some initial results focused on peer-to-peer energy-trading and decentralized flexibility services provisioning as micro-grid management solutions;
- Section 6 concludes the deliverable and presents future work to be addressed in following iterations.

2 Secured blockchain-driven energy marketplace

We define an energy marketplace that will be operated at the local micro-grid level, allowing any prosumer to directly participate in the market trading sessions. This is of great importance in the context of integrating many small-scale DEPs to provide opportunities for competitive or cooperative procurement models. The market will match consumers with energy producers and will rely on energy tokens for representing the energy as a digital asset in the energy market. For creating energy tokens, we use the ERC721 open standard [1] which allows to represent each energy asset as a non-fungible (i.e. a property that describes an asset as being indistinguishable from another asset) token in the blockchain system. The tokens will be generated at a rate proportional with the forecast energy production at the micro-grid level, transforming the energy into a transactable digital asset. The producers and consumers will then use tokens to participate in the electricity market sessions, leveraging on self-enforcing smart contracts. The market will leverage on self-enforcing smart contracts to manage, in a programmatic manner, the P2P energy-trading between DEPs (see Figure 2).

Self-enforcing smart contracts are distributed at the level of each peer DEP who is voluntarily enrolled with the marketplace and will stipulate the expected energy production/demand levels, energy price in tokens or the associated token-based incentives for rewarding the DEPs consuming the renewable energy when available. During a market session, each prosumer will submit bids and offers (i.e. from their contracts) representing the amount of energy they are willing to buy or sell. The use of smart contracts will allow the participants to automatically submit bids and offers; validity checks ensure that market session rules are not violated; the evaluation is performed by the smart contracts.

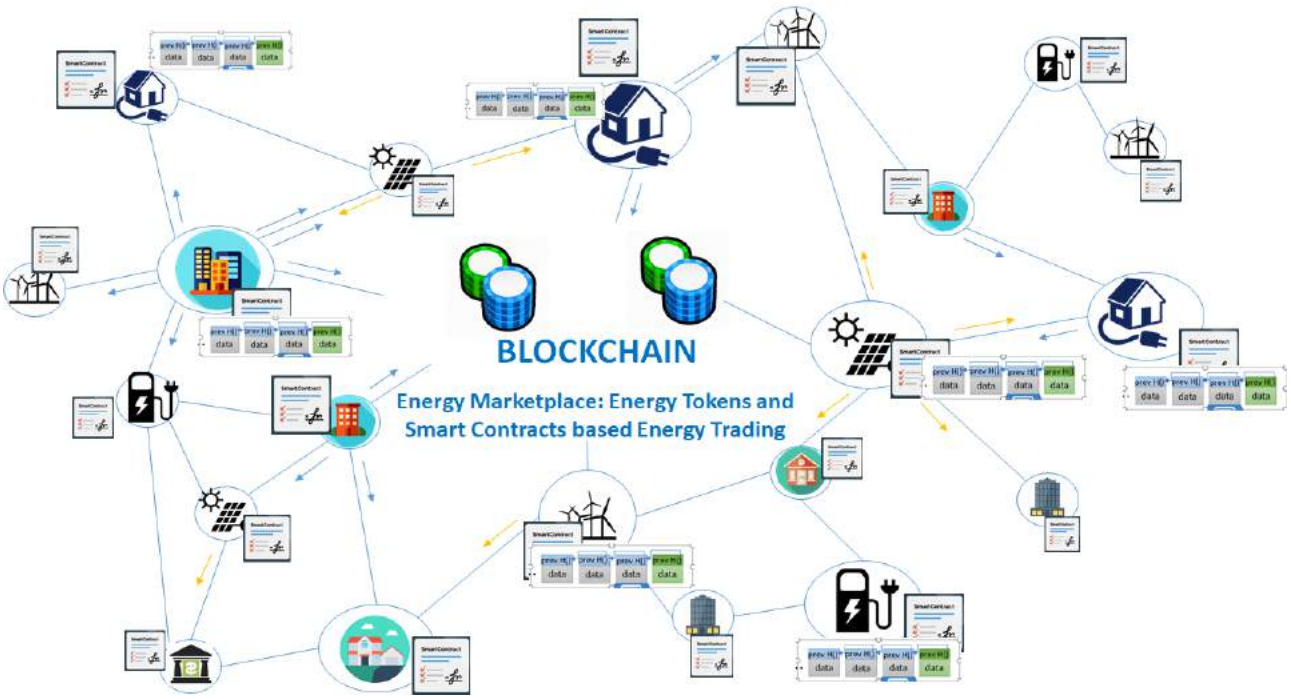


Figure 2. Blockchain-based P2P energy marketplace

A market level smart contract will programmatically deal with market operation processes such as matching bids and offers and calculating per session the energy clearing price. For clearing price calculation, the energy supply offers are sorted in ascending order and the energy demand bids in descending order. The intersection point between the two curves gives the market-clearing price. The offers (supply) having the price lower than the clearing price will be matched with the bids (demand) having a price higher than the clearing price. DEPs may accept or reject the matched offers/bids. The acceptance of an offer/bid implies the market participant's

commitment to inject/withdraw the quantity of energy specified in the offer/bid, or, in case of partial acceptance of the offer/bid, the corresponding share, in a prefixed time frame. As a result, energy transactions are generated, replicated in all the nodes and validated but they are not fully confirmed until a new block containing them will added to the blockchain.

The market's financial settlement is achieved by consensus-based validation. Once issued, the energy transactions are registered and replicated in future blockchain blocks across all the nodes in the network. The consensus mechanisms implemented in the blockchain system keeps track of all these changes and validates the corresponding state updates. Thus, instead of having one authority for keeping all centralized energy transactions, the responsibility is equally shared among every peer node of the network. Each transaction is tracked and validated locally by each peer node before it is unanimously accepted in the blockchain history. Each node is responsible for validating the integrity of the registered market actions: tokens issued, bids and offers, market clearing price computation, monitored values, settled price, energy consumer rewards and penalties. The results of each prosumer node computation will determine whether the actions contained in the block are valid and whether the block will be added as a valid one in the chain history. As a result, the decision on the actual share of energy which has been effectively delivered/consumed by each peer and associated financial incentives will be unanimously agreed upon through consensus by all the other network peers.

2.1 Prosumer registration and permission control

Two types of blockchain infrastructure deployments can be considered for the micro-grid decentralized energy marketplace implementation: public blockchain or private blockchain. The main difference among them lies on the access right and control of new DEP joining a micro-grid and on the reading/writing rights they may be allowed (see Table 1).

Table 1. Public vs private blockchain for micro-grid energy management

	Public		Private	
	Permission-less	**Permissioned**	**Consortium**	**Enterprise**
Access	Any DEP	Any DEP based on Prior Validation	Based on Owners Group Validation	Based on Administrator Validation
Transactions	Any DEP	Owners and Validated DEPs	Owners and Validated DEPs	
Commit to Chain	Any DEP	Owners and Subset of Validated DEPs	Owners and Subset of Validated DEPs	

The DSO's aim is to keep some degree of control on the registration of new DEPs to the grid and this limits the potential blockchain deployments suitable for micro-grid energy marketplace management.

The first alternative we considered is to use private blockchain deployments as a solution to manage the access rights of prosumers and to restrict some of them to a group of owners (e.g. group of energy aggregators or even the DSO in specific cases). In this case, the energy marketplace can be managed by a group of big prosumers, retailers or aggregators having an important stake in the market operation or by a single entity. It is important that the prosumers are known and vetted in advance this decision has a great impact upon the micro-grid operation, both in terms of security and consensus. Since the prosumers are known and could be held accountable for their actions, the use of high energy-consuming consensus algorithms for transactions validation and financial settlement such as Proof-of-Work are not necessarily

justified. In such a private ecosystem, the validators are legally accountable; thus, a certain level of trust between the nodes can be assumed. Thus, more energy efficiency consensus algorithms can be suitable for private blockchain ecosystems (i.e. Proof of Authority [2] or Practical Byzantine Fault Tolerance [3]). However, a private blockchain requires to have trusted entities at least for validating new prosumers and issuing new blocks on the chain.

The second alternative we considered is to use a public permissioned deployment and to manage new prosumers' registration validation of new prosumers using smart contracts. Prior to prosumers registration validation they will be able to read data from the chain, but they will not be able to write new energy transactions and to mine/validate the chain blocks. The management of permissions is achieved in the public blockchain system, by either validating the prosumers before their registration to the micro-grid blockchain-based energy management or by establishing some permissions rules at the level of each decentralized application, by keeping a registry of all the validated prosumers in the smart contracts. As a result, energy management processes will be decentralized, and governed by the decisions and consensus achieved through the collaboration of all the prosumers registered. In this regard the proposed solution considers a public blockchain network, whose robustness and security is intrinsic without requiring a trusted entity to ensure the well-functioning of the system. However, due to its openness features the public chain maybe considered as unsuitable for many institutions and enterprises, raising governance and the privacy concerns.

In both cases, the DSO should have all required data regarding the involved prosumers: spatial position, place and voltage level connection into the grid, power signature, power quality impact (i.e. any voltage fluctuation that they can generate, through the local energy injection into the power grid).

Such smart contracts for prosumers' registration and access control should:

- enforce that every new prosumer who accesses the micro-grid has its blockchain platform account validated by several reputable accounts already registered before being tracked by the account registry function of the smart contract;
- keep track of all the prosumers' accounts from the micro-grid and register information about their actions and accounts reputation;
- decrease a prosumer's account reputation in case it is not respecting the bids or offers it makes during actual delivery;
- consider reputation for granting the prosumers' rights and for enforcing each new action initiated.

Thus, an assurance mechanism is implemented in the peer-to-peer energy marketplace, based on financial deposits to ensure that each registered prosumer will behave honestly. When registering, the prosumer shall provide the estimated energy amount to be sold/bought on the market and a financial deposit proportional with the estimated energy to be traded.

In Figure 3 the code snippets representing the registration mechanism are depicted. When registering in the energy marketplace each prosumer acting as a producer must also make a deposit proportional with the energy to be traded (line 12), deposit that will be recovered once the producer can prove that the promised quantity has been delivered. Similarly, for consumption the prosumer must prove financial capability by registering together with the buy order the actual financial deposit proportionally to the price and bid quantity (line 8). If the prosumer does not provide enough funds to make as deposits the transaction will be rolled back, thus cancelling the prosumer registration in the peer-to-peer market.

```
function enrollP2PMarket(int32[] memory _estimatedEnergyValues, uint[] memory  _tradingPrices,
                         uint _startTime,  EnergyMarketLibrary.EnergyType _energyType,
                         EnergyMarketLibrary.MarketSessionType _marketSessionType) public payable{
    require(msg.sender == ownerProsumer);
    ...
    if(_estimatedEnergyValues[hour] > 0){
            uint deposit =depositPerUnit * uint(abs(_estimatedEnergyValues[hour]));
            marketManager.registerBuyOrder.value(deposit)( _estimatedEnergyValues[hour],  _tradingPrices[hour],
                                                           hour, metadata,  _marketSessionType);
        }else{
            uint deposit = _tradingPrices[hour] * uint(abs(_estimatedEnergyValues[hour]));
            marketManager.registerSellOrder.value(deposit)( _estimatedEnergyValues[hour],  _tradingPrices[hour],
                                                           hour, metadata,  _marketSessionType);
        }
    ...
    participantOfP2PMarket = true;
}
```

Figure 3. Prosumer registration in a P2P energy marketplace

Once registered in the peer-to-peer market, the prosumer will be added in the market registry together the associated contract. Whenever the prosumer will attempt to access a functionality of the market, its identity will be verified.

2.2 Smart contracts for P2P Energy-trading

We have defined several smart contracts for making transactions on energy assets in the peer-to-peer energy marketplace (Figure 4).

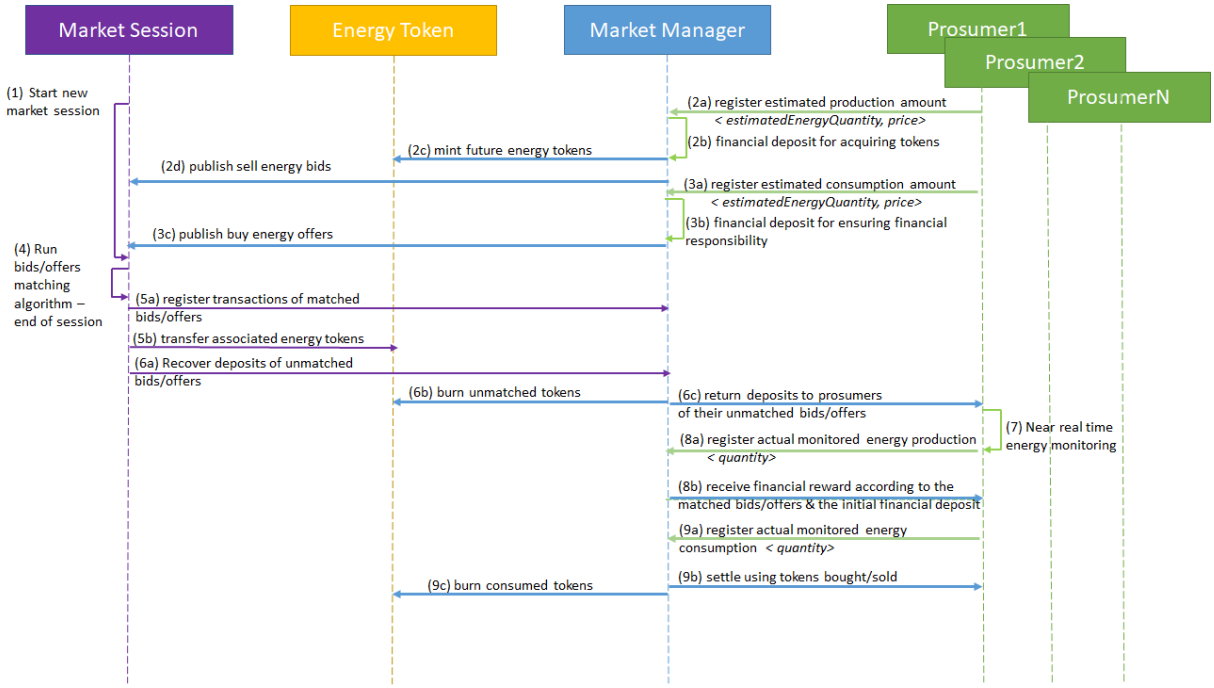


Figure 4. Energy marketplace smart contracts and their interaction

Each prosumer in the micro-grid has associated an account in the chain that can be used to sign transactions and pay different services or receive incentives as a participant in such services. Associated with the prosumer account, we define a *Prosumer Smart Contract* that keeps track of the services the prosumer registers for (peer-to-peer energy-trading or delivery of flexibility services), and of the associated responsibilities. The prosumer's activity with respect with the fulfilment of responsibilities will be continuously evaluated. The energy asset is modelled using an *Energy Token Smart Contract* as energy-adapted ERC721 token and is used as a trading symbol in different market sessions. The market sessions are managed by the *Market Session Smart Contract* that keeps the order book of all the registered energy bids and offers. The *Market Manager*

Smart Contract keeps track of all the opened sessions and can conduct the necessary validations regarding the activity of the underlying prosumers.

In Figure 4 the interaction of the defined energy marketplace smart contracts is depicted.

- *Step 1*: the Market Session Smart Contract creates and opens a new energy-trading session for a specific time frame allowing the prosumers to register and publish their bids/offers.
- *Step 2-3*: using their associated Prosumer Smart Contract, each prosumer can register the estimated (i.e. forecast) production (2a) or consumption (3a), based on which the corresponding selling offers (2d) or buying bids are registered (3c). The financial deposit scheme (i.e. prosumers' stake) is enforced. Before acquiring the energy tokens corresponding to the estimated production (2c), the prosumer must make a deposit established by the Market Manager Smart Contract as registration fee (2b). Similarly, for registering a buy energy bid, the prosumer must pay the sum proving that he owns the financial means to acquire the requested energy (3b).
- *Step 4*: the Market Session Smart Contract will accept energy bids and offers until the end of session, at which point the matching algorithm will be run.
- *Step 5*: the energy transactions resulted from matching the bids and offers published are validated by the Market Session Smart Contract and forwarded to the Market Manager Contract (5a) for future prosumer accountability validations. Based on the registered transactions, the transfer of energy token is done from the energy producers to the consumers (5b), such that in real time each consumer can make proof of the energy tokens bought. The money paid by the consumers for those tokens, however, will be sent to the producer only when proof of the actual energy monitored is registered on chain (i.e. the transaction is actually validated).
- *Step 6*: the unmatched orders will be forwarded to the Market Manager Smart Contract (6a), who will burn the unused tokens (6b) and will return to each prosumer his initial deposits (6c) made upon registration in steps 2 and 3.
- *Step 7*: in real time, the monitored energy will be forwarded to the Prosumer Smart Contract associated to each prosumer in the local micro-grid.
- *Step 8*: when a new energy production value is monitored, the Market Manager Smart Contract (8a) will validate the production value against the energy sold on the previous sessions, and the prosumer will receive together with the initial deposit the financial sum (8b) received from the matched buying offers.
- *Step 9*: when a consumption value is monitored, the Market Manager Smart Contract (9a) will validate its actual value against the energy tokens bought on the previous sessions (9b). If the consumption is according to the quantity bought, the tokens will be consumed (9c) and no additional cost penalty will be put on the consumer.

2.2.1 Prosumer associated contract

Through their associated smart contract, each DEP can register its estimated energy production or demand, and corresponding energy bids and offers in the peer-to-peer energy marketplace (see Figure 5).

Table 2. Prosumer Smart Contract state variables for P2P energy-trading

State Variable		Description
Energy bids/offers	Energy Amount $E_{Estimated}$	The estimated amount of energy a DEP is willing to sell or buy in the energy marketplace.
	Trading Prices E_{price}	The price that the DEP is willing to pay in case of energy bids or the price the DEP desires in case of a sell energy offer.
Energy Transactions	Energy Amount E_{Traded}	Energy transactions of a DEP specifying the energy amounts corresponding to the bids/offers matched during market sessions and associated clearing price calculated. The matched bids/offers act as a promise of energy to be delivered being actuality validated against the actual energy monitored values.
	Clearing Price	
Actual Energy Profile E_{Actual}		Monitored energy consumption values acquired by the IoT smart energy metering devices.

In Table 2 the state variables necessary for the prosumer's participation in a P2P energy marketplace are depicted. When enrolling in the market, the prosumer must provide the estimated energy amount willing to buy or sell, timeslot and trading prices; the information serves as base for registering the corresponding energy bids/offers in the market sessions. Once the market session ends, if the prosumers' bids or offers are matched with the ones of other prosumers, energy transactions are generated containing the traded energy amount (i.e. energy tokens bought or sold) and the calculated market session clearing price. During energy delivery, the prosumer's energy meter registered values (see Figure 5 line 2), are evaluated against the energy bought/sold on the previous session market (line 9) by the Market Manager Smart Contract.

```
1   //====================   DEP CONTRACT - Step (7a)  ======================
2    function monitorValue(uint time, int32 value, bytes32  hash ) public {
3        require(msg.sender == ownerProsumer);
4        ...
5        if(participantOfP2PMarket){
6            evaluateTradedEnergyProgress(hour, value);
7        }
8    }
9   function evaluateTradedEnergyProgress(uint _hour, uint time,  int32 _value) internal {
10       EnergyMarketsManagement mngm = EnergyMarketsManagement(p2pMarketManagement);
11       ...
12       if(_value > 0){
13           mgm.evaluateRealTimeConsumption(_timeIndex, time, _value);// Step (9a)
14       }else{
15           mgm.evaluateRealTimeProduction(_timeIndex, _value);// Step (8a)
16       }
17       ...
18    }
19    //================   MARKET MANAGEMENT CONTRACT - Step (8-9) ====================
20   function evaluateRealTimeProduction(uint _timeIndex,  int32 _value) public {
21       require(prosumerToDEPContract[tx.origin] == msg.sender);
22       uint sumValuePromised = producersPromisedQuantity[tx.origin][_timeIndex];
23       ...
24   }
25
26   function evaluateRealTimeConsumption(uint _timeIndex, uint time,  int32 _value) public {
27       require(prosumerToDEPContract[tx.origin] == msg.sender);
28       uint tokensBought = consumersBoughtTokens[tx.origin][_timeIndex].length;
29       ...
30   }
```

Figure 5. Tracking of prosumers' energy transactions

When energy production values are monitored (line 20), the market's managing contract will evaluate the produced quantity against the quantity promised through the established energy transactions. For violation of the initial transactions promised values, the prosumer will be penalized according to the deviation amount and energy price. As depicted in Figure 5, when the prosumer's sensor registers new monitored values, the prosumer's activity needs to be validated against the energy traded on the previous market sessions (line 20, line 26). However, the manager contract will verify the identity of the transaction signer (line 21, line 27) and if the prosumer is not found in the market participants' registry the transaction will be reverted.

2.2.2 Energy Token Contract

Ethereum has defined several standards regarding the representation of real-life assets using blockchain and specific rules for issuance, tracking, transfer and destruction. However, only one of these standards addresses non-fungible assets. Fungibility is the property that describes an asset as being indistinguishable from another asset. That is, any two assets are identical and interchangeable without any detectable differences. This, however, is a property that cannot be applied to many real-life assets. In this sense, we define the energy tokens as being a non-tangible asset, since for a given quantity of energy there are several aspects that make it distinguishable from another. For example, in case of an energy asset, the source of energy (e.g. solar, wind, fuel) is an aspect that clearly differentiates future tokens. Furthermore, the timeframe or region regarding the generation of a certain energy amount can also be used to differentiate between tokens.

Thus, we have adopted the ERC721 [4] standard for representing the traded energy assets as non-fungible tokens in the blockchain system. The specifics of a token instance can be specified by different fields in the metadata structure. Several adaptations have been made in order to provide a market-compatible representation of each energy asset.

The first adaptation is in regards to the ERC721 metadata [5] is described using the *tokenURI* field, following the *ERC721 Metadata JSON schema.* It provides information about the asset such as name, description, and image. To provide a secure way for hosting the associated metadata, the IPFS [6] system is proposed as a storing mechanism for the images, or other related information.

```
1   //ENERGY ADAPTED ERC721
2   mapping(uint256 => EnergyMarketLibrary.EnergyTokenMetadata) internal _tokensDetails;
3   function tokenDetails(uint256 tokenId) public view returns(EnergyMarketLibrary.EnergyTokenMetadata memory){
4       require(_exists(tokenId))
5       return _tokensDetails[tokenId];
6   }
7
8   //ERC721Metadata
9   mapping(uint256 => string) private _tokenURIs;
10  function tokenURI(uint256 tokenId) external view returns (string memory) {
11      require(_exists(tokenId));
12      return _tokenURIs[tokenId];
13  }
```

Figure 6. ERC721 metadata adaptation for energy assets

We propose an adaptation of the ERC721 metadata specification, by replacing the mapping of the *_tokenURIs* (Figure 6 line 9) with the *_tokenDetails* mapping (Figure 6 line 2) having instead of a metadata URIs, a data structure *EnergyTokenMetadata*, implemented in Solidity language. The motivation behind this adaptation is, that, once registered, specific information about a token as part of the energy sell offer, such as production time and energy type, could be relevant to the energy marketplace management processes. These details will be further verified at runtime by the market contract, to validate whether the sold energy token matches the information requested by the buying side. Furthermore, the metadata information will be verified in real time by the Market Manager Smart Contract, to validate that the actual energy consumed is consistent with the energy tokens owned by the prosumer.

```
1   enum EnergyType {GREEN, BROWN}
2   struct EnergyTokenMetadata {
3       uint startTimeToken;
4       uint endTimeToken;
5       EnergyType energyType;
6       string measureUnit;
7       address producer;
8   }
```

Figure 7. ERC721 metadata adapted for energy tokens

The second adaptation is in regards to the asset quantification of owned tokens in smart contracts. The standard ERC20 uses a mapping of addresses and quantity of tokens owned (Figure 8, line 2), without distinguishing among individual tokens (fungible assets). The ERC721 standard, however, keeps track of the

individual tokens owned by each address (line 5) and the total number of tokens that the owner holds (line 6). We propose a modified version of the ERC721 where a combination of the previously described approaches is used. As a result, we identify the metadata of the energy token, that differentiates it from other individual tokens; however, we consider prosumer to be able to hold more than one such a token. In this sense we defined the mapping structure (line 25) in which, for each address, the token-specific quantity is stored.

The motivation behind this adaptation is that in case of the energy generated by the producer, while the asset must be distinguishable from other assets (green or brown energy), to model a large amount of energy, we also need to map a quantity of energy generated to such energy descriptions. Once registered in the market as a sell energy offer, this total amount of energy can be split by the bids/offers matching algorithm to match one or more energy bids. As a result, the energy token marked by the same metadata, once sold, could have its quantity split among more than one buying prosumers.

```
1   //ERC20
2   mapping (address => uint256) private _balances;
3
4   //ERC721MetadataMintable
5   mapping (uint256 => address) private _tokenOwner;
6   mapping (address => Counters.Counter) private _ownedTokensCount;
7
8   function mintWithTokenURI(address to, uint256 tokenId, string memory tokenURI) public onlyMinter returns (bool) {
9       _mint(to, tokenId);
10      _setTokenURI(tokenId, tokenURI);
11      return true;
12  }
13  function _mint(address to, uint256 tokenId) internal {
14      require(to != address(0));
15      require(!_exists(tokenId));
16
17      _tokenOwner[tokenId] = to;
18      _ownedTokensCount[to].increment();
19
20      emit Transfer(address(0), to, tokenId);
21  }
22
23  //ENERGY ADAPTED ERC721
24  mapping (uint256 => address) private _tokenOwner;
25  mapping(address => mapping(uint256 => uint256)) internal _ownedTokensBalances ;
26
27  function mintWithTokenMetadata(address to, uint256 tokenId,
28                                 EnergyMarketLibrary.EnergyTokenMetadata memory tokenMetadata, uint quantity)
29                                 public onlyMinter returns (uint) {
30      _mint(to, tokenId, quantity);
31      _setTokenMetadata( tokenId, tokenMetadata);
32      return true;
33  }
34  function _mint(address to, uint256 tokenId, uint quantity) internal {
35      require(to != address(0));
36      require(!_exists(tokenId));
37
38      _tokenOwner[tokenId] = to;
39      _ownedTokensBalances[to][tokenId] = quantity;
40
41      emit Transfer(address(0), 0,  to, tokenId, quantity);
42  }
```

Figure 8. The quantification energy tokens in smart contracts

Being a mintable token (can increase its supply based on the promise of energy generation), the adaptation made in terms of token quantity representation also affects the minting functionality. Thus, based on a prosumer's production estimation, a new energy token is minted by the market manager having the producer-related information specified as metadata and the quantity of the token equal to the energy production amount (Figure 8, lines 27 and 39), as opposed to the standard ERC721 where only one token is generated during minting (line 8 and 18).

2.2.3 Market Session Contract

The Market Session Contract is responsible for creating a trading market session with specific timeframes (intra-day or day-ahead) and to keep track of published energy bids and energy offers. Table 3 details the main state variables of the Market Session Contract.

Table 3 Energy Market Session state variables

State Variable	Description
Market Session Configuration	The session configuration stores information about the session type (day-ahead or intraday) and the start time and end time of the market session.
Energy Bids	A key-value pair (<id, Order>) data structure holding the active bids registered in the current session.
Energy Offers	A key-value pair (<id, Order>) data structure holding the active buy requests registered in the current session.
Energy Transactions	A structure holding the results of the matching algorithm run at the end of the session, pairing sell offers and buy bids in transactions to be carried out between the corresponding producers and consumers.
Clearing Price	The clearing price obtained for the market session.

Furthermore, the 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.

For the matching of bids/offers we have implemented a second-tier solution, that fetches all the active bids/offers from the market session, runs the matching algorithm and publishes a list of energy transactions specifying the matched bids/offers and the price and quantity to be traded. The description of bids/offers as orders and the energy transactions are presented in Figure 9.

```
enum OrderSide {BUY, SELL}
struct Order {
    bytes32 id;
    OrderSide orderSide;
    address prosumerAddress;
    uint timestamp;

    uint tokenId;
    EnergyTokenMetadata metadata;

    uint quantity;
    uint price;
}
```

```
struct Trade{
    bytes32 id;
    bytes32 buyOrderId;
    bytes32 sellOrderId;
    address prosumerBuyingAddress;
    address payable prosumerSellingAddress;
    uint timestamp;

    uint tokenId;
    uint quantity;
    uint price;
}
```

Figure 9. Order (bid or offer) and energy transaction data structure

2.2.4 Market Manager Contract

The market manager contract is defined to ensure the coordination between the market sessions and evaluate and track the behaviour of the prosumers registered in the market. Table 4 presents the main state variables of the Market Manager Contract, storing the information necessary to ensure the correct functionality of the sessions and evaluation of the prosumers.

Table 4. Market Manager Contract state variables

State Variable	Description
Opened Session Markets	The manager keeps track of all the active market sessions and their configuration.
Registry of the tokens bought by the consumers	A key-value pair (<hour, tokens>) data structure keeping track of all the tokens bought by each prosumer.
Registry of the producers' promised energy	A key-value pair (<hour, amount>) data structure keeping track of the energy amount to be sold by each prosumer.
Financial deposits of the prosumers	A structure holding information about all the financial deposits made by the prosumers when publishing their bids/offers.

The market manager contract keeps track of all the opened session. As a result, whenever a prosumer wants to publish new bid or offer (see Figure 10), this contract is responsible for validating that the targeted session is indeed open (line 5, line 17), verifying and storing the deposits as a financial assurance of honest behaviour (line 7-8, line 20-21), and forwarding the bid/offer to the corresponding session (line 11, line 22).

```
1   //================   EnergyMarketManager Contract - Step (2a) ====================
2   function registerSellOrder(int32 _tradedValue, uint _tradingPrice, uint _hour, EnergyMarketLibrary.EnergyTokenMetadata memory metadata,
3                                                   EnergyMarketLibrary.MarketSessionType _marketSessionType ) public payable {
4       EnergyMarketLibrary.MarketSessionConfiguration memory session = marketSessions[uint(_marketSessionType)];
5       require(_openedSession(session));
6       uint hourDeposit = depositPerUnit * uint(abs(_tradedValue));
7       require (hourDeposit <= msg.value);
8       producerTokenDeposits[tx.origin][_hour] = hourDeposit;// Step (2b)
9       ...
10      erc.mintWithTokenMetadata (tx.origin, tokenId, metadata, uint(abs(_tradedValue)));// Step (2c)
11      _publishSellOrder(tokenId, metadata, _tradingPrice, uint(abs(_tradedValue)), session.marketSessionAddress ); // Step (2d)
12  }
13  //================   EnergyMarketManager Contract - Step (3a) ====================
14  function registerBuyOrder(int32 _tradedValue, uint _tradingPrice, uint _hour, EnergyMarketLibrary.EnergyTokenMetadata memory metadata,
15                                                  EnergyMarketLibrary.MarketSessionType _marketSessionType ) public payable {
16      EnergyMarketLibrary.MarketSessionConfiguration memory session = marketSessions[uint(_marketSessionType)];
17      require(_openedSession(session));
18
19      uint deposit = _tradingPrice * uint(abs(_tradedValue));
20      require (deposit <= msg.value);
21      consumerBuysDeposits[tx.origin][_hour] = deposit; // Step (3b)
22      _publishBuyOrder(metadata, _tradingPrice, uint(abs(_tradedValue)), session.marketSessionAddress ); // Step (3c)
23  }
```

Figure 10. Energy Market Manager registering orders

Similarly, when at the end of the session the bids/offers are matched, this contract is responsible for updating the registries with the sold/ bought promised energy values corresponding to the DEPs that were matched. This enables the contract to validate and settle their accounts in real time.

3 Blockchain-driven DR and Flexibility Management

DEPs are able to offer and trade their flexibility in terms of loads modulation, indirectly via enabling aggregators or directly with the DSO via direct DR and control of DEP's energy assets (see Figure 11).

Through self-enforcing smart contracts enabling both demand-offer matching and decentralized coordinated control, the energy stakeholders such as the DSO or aggregators will be able to assess and trace the share of contracted flexibility services, actually activated in real time by the enrolled prosumers. This will impact the energy grid management by

- maintaining the balance of supply and demand in a decentralized environment;
- achieving the final goal of reducing the overloading and
- reaching power network stability by means of the flexibility provided by active micro-grids.

A micro-grid can work in either grid-connected mode or island mode and can be registered to DR programs being able to sign/accept smart contracts, each of their DEPs making available their energy flexibility.

The balance between energy demand and energy production is managed by the corresponding DSO who can analyse the actual state of the distribution grid and forecast the needs for energy flexibility to deal with potential distribution grid level congestion problems targeting to identify grid issues and actors that can solve them by providing the required flexibility. The DSO aims to access micro-grid DEPs, taking advantages of micro-grid as to guarantee smart grid stability by making available *flexibility-as-a-service* through smart contracts.

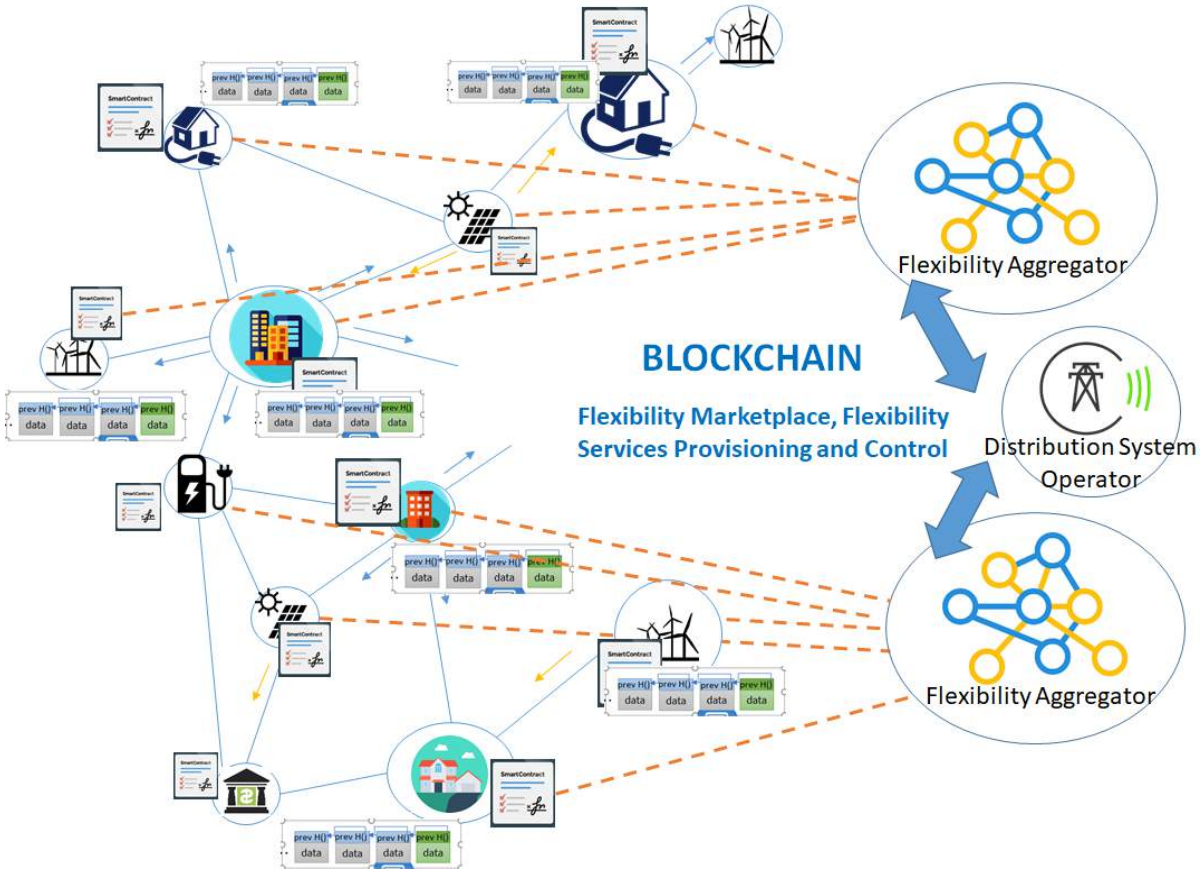


Figure 11. Blockchain-based flexibility trading and control

Each DEP of the micro-grid has been enabled to offer via smart contracts his flexibility to specific aggregators to which it is enrolled. Using our prosumer level forecasting techniques [7] the Aggregator creates and sends

a forecast of the aggregated energy demand of all its registered DEPs to the DSO, who uses these forecasts to estimate the total load and detect potential congestion points [8]. When the DSO identifies a congestion point, the associated grid connections and it has to declare it in the common reference operator registry and the aggregators in that micro-grid are activated and contract prosumers connected to this congestion point offer flexibility. More specifically, the DSO sends a flexibility request and aggregators respond to this flexibility request with a flexibility offer. The DSO can accept only one flexibility offer and then sends a flexibility order to the selected aggregator who will adjust the load of their registered DEPs as to fulfil the flexibility need.

Self-enforcing smart contracts will be defined to manage the levels of energy demand flexibility (i.e. from aggregators and their enrolled prosumers on one side and from aggregators to the DSO on the other side), with associated incentive- and penalty rates. The corresponding smart contracts evaluate the difference between the requested energy flexibility (i.e. a curve signal) and the flexibility actual delivered (as shown by monitored energy values registered in the distributed ledger). If significant deviations are identified, specific actions will be taken to rebalance the energy demand with the energy production, thus, the smart contracts act as a decentralized control mechanism.

The smart contract defines each DEP's energy baseline profile and expected adjustments in terms of the amount of energy flexibility to be shifted during DR event time intervals. The power baseline profile is the normal energy profile of a DEP determined as the average of past measured energy values [14]; it reflects how much energy the prosumers may have consumed in the absence of DR event.

The aggregators will inject individual control signals (i.e. upon the DSO request) in the smart contracts regulating individual DEP's flexibility, requesting them to adapt their energy profile by shifting flexible energy. Aggregators will evaluate the difference between the total amount of flexible energy activated, normalised to the baseline energy consumption of each DEP.

The flexibility requests received by a DEP are recorded in the ledger. Thus, the smart contract verifies in near real time the monitored energy consumption data against the associated DR signal to detect any significant deviations and notifies the aggregator. In case of notable positive or negative deviations, the smart contract calculates the associated penalties for the prosumer. Otherwise, the prosumer is rewarded considering the DR revenue rates and how much of the prosumer energy demand profile has been adapted during a DR event. The total reward for a DEP for its adaptation during a DR event is calculated by the aggregator based on the flat rate provided by the DSO for the aggregated flexibility (i.e. a daily revenue rate for each kW of energy shifted or a discount rate on the regular electricity bill).

3.1 Registration and permission control

Permission mechanisms are ensured at two different levels via smart contracts: at aggregator level and at the individual-prosumer level.

First, the aggregators must be entities authenticated and validated by the DSO, to be able to register flexibility offers as a result of the DSO flexibility request. Thus, the registration process of an aggregator in the blockchain-based flexibility market can only be achieved through the smart contract managing the DSO actions as depicted in Figure 12**Error! Reference source not found.** .

```
1   mapping(address => address) aggregatorRegistry;
2
3   function deployAggregatorContract(uint depositValue) public returns (address){
4       require(aggregatorRegistry[msg.sender] == address(0x0));
5       Aggregator agg = new Aggregator(deviationThreshold, periodCardinality,depositValue);
6       address aggContractAddress = address(agg);
7       aggregatorRegistry[msg.sender] = aggContractAddress;
8       emit AggregatorRegistered(msg.sender, aggContractAddress, depositValue);
9       return aggContractAddress;
10  }
```

Figure 12. Aggregator registration via smart contracts

The aggregator will generate and sign a transaction requesting to be registered as an actor which can participate in the flexibility marketplace (Figure 12, line 3). If no other similar request has been previously made from the same aggregator account, the DSO-associated smart contract will proceed by deploying an aggregator associated smart contract which will manage the aggregators actions in the flexibility marketplace (line 5). The address of the contract will be registered (line 7) in the DSO's *aggregatorRegistry* (line 1), ensuring that at any point in the future, the DSO will be able to check the integrity of any request made by any aggregator and to check that the whitelisted accounts, stored in the registry, have permissions to access contract functionality. For example, whenever an aggregator contract publishes a flexibility offer in the flexibility marketplace the prerequisite is for the associated transaction to be originally signed by a known aggregator account, the method invocation to be made through the associated aggregator contract and both to be already stored in the DSO's registry.

Second, each DEP in the micro-grid willing to offer its flexibility should be registered (i.e. enrolled) with an aggregator via the aggregator-associated smart contract (see Figure 13, line 4).**Error! Reference source not found.** The aggregator smart contract will firstly verify (line 1) that the DEP signing the request transaction has not already registered another DEP-associated smart contract (line 5). To ensure that the DEP has a stake in flexibility offering services, they will be requested upon registration to make a deposit (line 6) that will be forwarded (lines 8 and 17) to the DEP account/smart contract to be able cover eventual penalties that may occur as a result not delivering the agreed flexibility profile. Upon a successful validation the DEP associated contract will be updated as a flexibility services participant via its proxy aggregator (line 19). However, this update can be made only by the aggregator (line 18), so no other third parties can act on behalf of the aggregator.

```
1   // ==============          AGGREGATOR CONTRACT       ==================
2   mapping(address => address) prosumerToDEPContract;
3
4   function registerProsumerContract(address  depContract) public payable returns (address){
5       require(prosumerToDEPContract[msg.sender] == address(0x0));
6       require(registrationSum <= msg.value);
7       DEP dep = DEP(depContract);
8       dep.registeredForFlexibilitMarket.value(msg.value)();
9       prosumerToDEPContract[msg.sender] = depContract;
10      emit DepContractRegisteredForFlexibility(msg.sender,depContract);
11      return msg.sender;
12    }
13
14  // ===============          DEP CONTRACT         ==================
15    bool private participantOfFlexibilityMarket;
16
17    function registeredForFlexibilitMarket() public payable {
18        require(msg.sender == aggregator);
19        participantOfFlexibilityMarket = true;
20        emit LogDepositReceived(msg.sender);
21    }
```

Figure 13. Registration of the DEP with aggregator as an energy flexibility provider

Once the DEP is registered in a flexibility service, its flexibility delivery must be evaluated against the flexibility order profiles defined by the aggregator. For this, each time a new energy value is monitored the signer of the associated transaction is verified, such that no other player of the network can act on behalf of the DEP and register monitored values in the DEP contract. The identity of the DEP signing the original transaction and of the DEP contract forwarding the flexibility delivery reports are verified to avoid any misleading impersonation of other actors in the network.

3.2 Smart Contracts for Flexibility Tracking

We defined and used self-enforcing smart contracts associated with individual DEPs, aggregators and DSO for management of flexibility services at micro-grid level [8].

DEP smart contracts are defined in a distributed fashion at the level of each DEP part of the micro-grid, enrolled with an aggregator and participating in DR-based flexibility services. A DEP's contract will specify (see Figure 14) the baseline energy consumption profile as an energy curve as well as the actual energy values captured by the associated smart meter and stored as transactions in the blockchain.

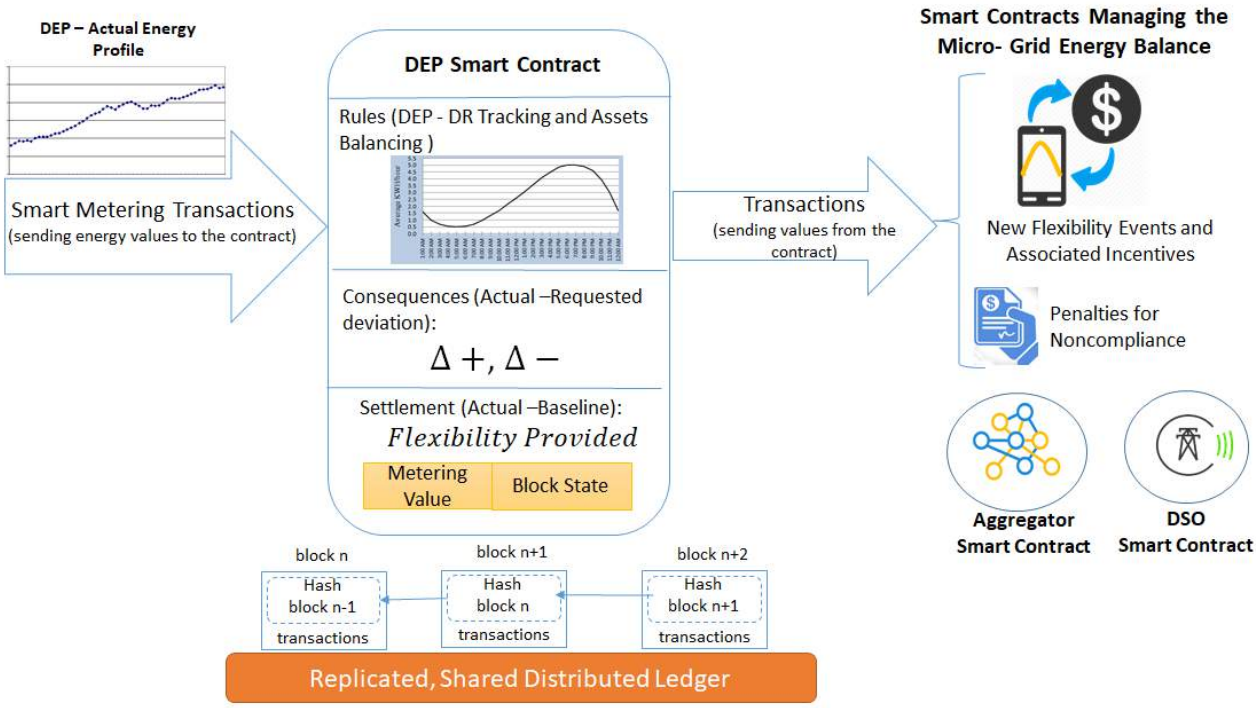


Figure 14. Self-enforcing smart contracts for flexibility services management and decentralized control [8]

Via aggregators smart contract, the aggregator will inject flexibility orders as a DR signal in the smart contract of specific DEPs. Afterwards the contract will automatically evaluate for each energy value, sealed in blocks and provided by the IoT smart metering device associated with the DEP, the difference between the requested energy demand curve and actual sampled energy values. If significant deviations are found, actions are taken to rebalance and match the energy demand with the energy supply thus the smart contract will act as a decentralized control mechanism and energy assets balancing towards the DSO. Based on the registered deviations in the micro-grid new flexibility orders are defined in a decentralized manner as well as the financial incentives for penalizing the prosumer that violates the smart contract and for rewarding the ones that will address the new flexibility service defined.

To determine the flexibility levels actually provided by each DEP, the DEP's smart contract will also evaluate the difference between the baseline energy demand profile of a DEP and the energy consumption values registered by the smart meter. The calculated value will be used to determine the incentives for rewarding the DEP for its provided flexibility.

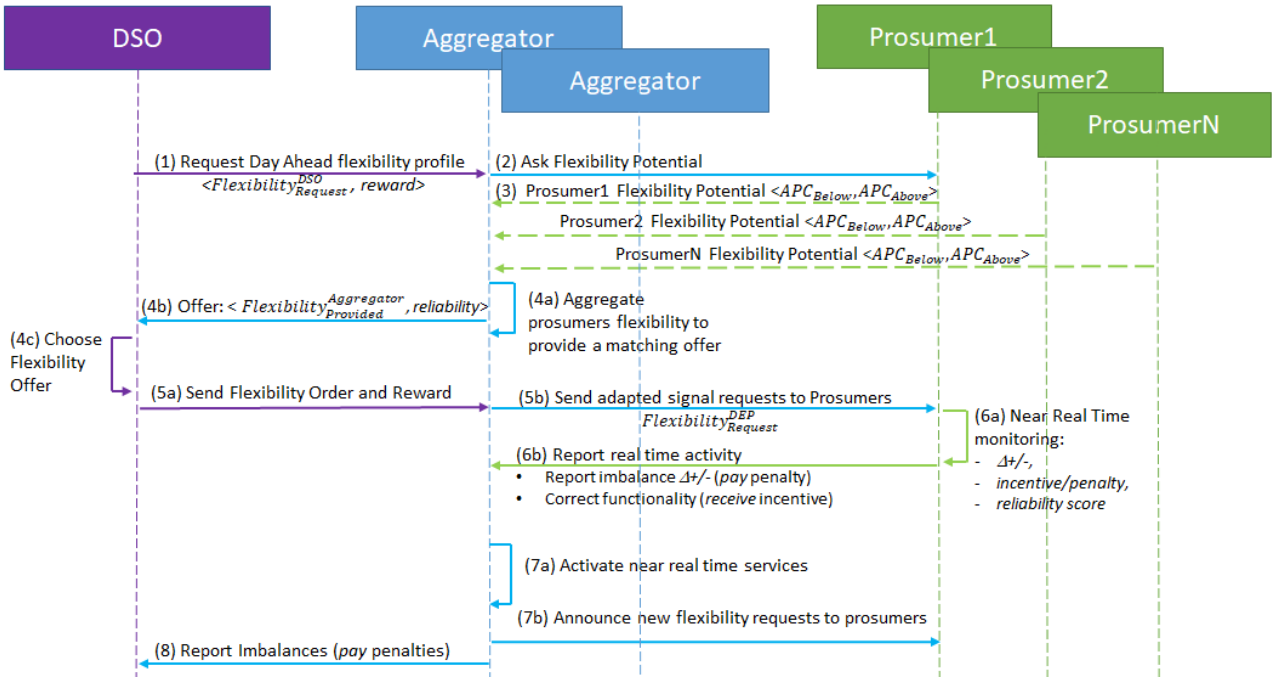


Figure 15. Interaction for flexibility services delivery using smart contracts

In Figure 15 the interaction between smart contracts associated with DEPs, aggregators and DSO:

- *Step 1*: the DSO registers a flexibility request as a bid in the flexibility marketplace specifying an energy profile that must be delivered the next day. Together with the request, the DSO establishes the reward that will be offered to the aggregator able to deliver the requested flexibility. Once the bid is registered, the market session is opened for any aggregator in the area to publish their flexibility offer.
- *Step 2:* based on the flexibility request registered in the flexibility market, the aggregator will try to estimate in its DEP portfolio the flexibility availability for the next day and trying to match the request published by the DSO. For this, the aggregator will gather information from its enrolled DEPs regarding the flexibility potential.
- *Step 3:* each DEP will send its flexibility availability in terms of how much are they able to change their normal behaviour and adapt to new requests both by decreasing and by increasing their normal profile (i.e. information about flexibility profile below and above baseline).
- *Step 4a:* the aggregator will request the off-chain optimization module to provide an optimal subset of its DEP and their flexibility potential such that the DSO flexibility request can be met.
- *Step 4b:* the aggregated profile of the optimal subset of prosumers computed in step 4a is provided to the DSO as an offer, together with a reputation score associated with the DEPs selected in the program.
- *Step 4c:* the DSO will keep the session opened for a period, time in which all the offers from the aggregators are registered. When session will close, the DSO will select the offer of a single aggregator.
- *Step 5a:* the aggregator will withdraw the reward money from the DSO contract and register the flexibility curve as a target for the next day.

- *Step 5b:* the aggregator will notify each DEP from the selected subset at step 4a, of the energy flexibility it must deliver in the next day.
- *Step 6a:* for each monitored time step of the day, each DEP contract will receive the monitored energy value and check this value against the flexibility order received at Step 5b. The deviation from this curve will be computed, and the reputation score of the prosumer will be updated accordingly.
- *Step 6b:* if an imbalance is created by the DEP, the smart contract will report this imbalance to the aggregator together with the money representing the penalty for not following the request. Similarly, if the DEP delivers the quantity of energy requested by the flexibility curve, the aggregator will be notified, and the DEP will receive the incentive corresponding with the flexibility delivered.
- *Step 7a:* in case of imbalances detected by the physical grid, near real-time ancillary services programs are activated in real time in order to provide additional flexibility needed. Different assets with rapid reaction time like batteries and generators can be used in these scenarios to compensate the problems but the costs are very high for activating such mechanisms.
- *Step 7b:* in addition, new flexibility requests can be sent to enrol new DEPs in the program and provide supplementary flexibility.
- *Step 8:* if the flexibility levels are not met, the aggregator will report the imbalances to the DSO.

3.2.1 DEP Smart Contract

The DEP smart contract is a piece of code that defines the rules for DEP participation in flexibility services which need to be verified and agreed upon all interested actors (i.e. prosumer, aggregator, DSO). The rules may describe the behaviour of DEPs during the flexibility delivery events or may even address various constraints for maintain the grid stability and reliability. These contracts are triggered by new transactions signed by the owning DEP (i.e. registering new energy data from the IoT smart meters), which will determine each blockchain node to update its state based on the results obtained after running the smart contract.

Table 5 presents the main state variables defined in a smart contract defining a DEP participation in flexibility services via aggregators.

Table 5. DEP-associated smart contract state variables

State Variable	Description
Baseline Energy Profile $\mathrm{E}_{\mathrm{Baseline}}$	Regular energy profile of a DEP determined based on average of measured energy values in the past.
Below Flexibility Potential $\mathrm{APC}_{\mathrm{Below}}$	Profile contains the values of the actual energy measured that are smaller than the baseline
Above Flexibility Potential $\mathrm{APC}_{\mathrm{Above}}$	Profile contains the values of the actual energy measured that are larger than the baseline
Current Energy Profile $\mathrm{E}_{\mathrm{Actual}}$	Monitored energy consumption values acquired by the IoT smart energy metering devices consumption;
Flexibility Request Profile $\mathrm{Flexibility}_{\mathrm{Request}}^{\mathrm{DEP}}$	Energy profile requested through the flexibility service. Injected by the aggregator into the DEP smart contract
Reputation score	A score computed by the DEP contract, measuring degree in which the prosumer has been able to deliver the ordered flexibility to the aggregator

The data acquired by each energy metering device associated with a DEP triggers the DEP smart contract execution. Monitored energy values are added to the energy curve describing the Current Energy Profile of the DEP considering a time interval T:

$$E^{DEP}_{Actual} = \{P^{DEP}_{Monitored}(t) \mid t \in T\}$$

Being enrolled with the aggregator to provide flexibility the DEP is provided with a flexibility request signal $Flexibility_{Request}$ and financial incentives to adjust their energy demand during DR events. The provisioning of a flexibility request signal by a DEP will be registered in the ledger, thus the DEP smart contract will check in near real time the monitored energy consumption data against the requested curve signal to detect any significant deviations and notifies the DSO accordingly. The deviations are determined as:

$$\Delta_{E^{DEP}_{Actual} - Flexibility^{DEP}_{Request}} = \sum_{t_{start}}^{t_{end}} (P^{DEP}_{Actual}(t) - Flexibility^{DEP}_{Request}(t))$$

where t_{start}, t_{end} represents the interval of the flexibility service. A positive value ($\Delta +$) signals that the DEP has not reduced its energy demand as requested while a negative value ($\Delta -$) signals that the DEP has decreased too much its energy demand.

In case of significant positive or negative deviations (over 10% of the flexibility request signal) the DEP smart contract will calculate the associated penalties for DEP. Otherwise the DEP will be rewarded by the aggregator considering the flexibility service incentives established by the DSO and how much of the DEP energy demand profile has been adapted during the DR event.

To determine how much energy a DEP has shifted the difference between the actual energy demand values and the established baseline profile is calculated for the flexibility service time interval:

$$Flexibility^{DEP}_{Provided} = \sum_{t_{start}}^{t_{end}} |P^{DEP}_{Actual}(t) - P^{DEP}_{Baseline}(t)|$$

Figure 16 shows the DEP smart contract evaluation of the DEP's flexibility delivery (i.e. monitored energy profile values against the flexibility requested profile).

```
1   //====================   DEP CONTRACT - Step (6a)  ========================
2   function monitorValue(uint time, int32 value, bytes32  hash ) public {
3       require(msg.sender == ownerProsumer);
4       ...
5       if(participantOfFlexibilityMarket){
6           evaluateFlexibilityProgress(hour, value);
7       }
8   }
9   function evaluateFlexibilityProgress(uint _hour, int32 _value) internal {
10      Aggregator c = Aggregator(aggregator);
11      ...
12      c.registerProgress.value(penalty)(_value, _promisedValue, baseline[_hour], _hour);
13      ...
14   }
15  //=================   AGGREGATOR CONTRACT - Step (6b) ======================
16  function registerProgress(int32 _value, int32 _promisedValue, int32 _baseline, uint _hour)  public payable{
17      require(prosumerToDEPContract[tx.origin] == msg.sender);
18      ...
19      if( abs(_imbalance) > deviationThreshold){
20          imbalancesProfile[_hour] += _imbalance;
21          require(_penalty <= msg.value );//Step (6c)
22      }else{
23          DEP dep = DEP (msg.sender);
24          dep.flexibilityReward.value(_incentive)();//Step (6d)
25      }
26  }
```

Figure 16. DEP smart contract – flexibility delivery evaluation

The entire process of evaluation from registering the monitored value, the progress, the imbalance, up to the incentive/penalty evaluation is done in an atomic process by the DEP smart contract. Since the entire code, all the evaluation rules and the prosumer's profiles (flexibility and baseline) are already stored by *the previous blockchain blocks in an immutable structure, once the transaction associated with the acquisition of a new monitored the energy value is registered (line 2), the process will automatically be enforced according to the code, without the prosumer or aggregator having the possibility to change it. Specifically, if the current DEP contract is previously registered in a flexibility program (Figure 16, line 5), the monitored value will be automatically checked against the flexibility profiles requested by the aggregator (line 9). The progress will be forwarded to the managing aggregator (lines 12, 16). The aggregator will evaluate the prosumer's progress (line 19), and a penalty (line 21) or a reward (line 24) will be applied. Any attempt to impersonate another prosumer is avoided by evaluating the DEP contract (line 3, line 17), and rolling back the transaction registration process.

3.2.2 Aggregator Smart Contract

The aggregator associated smart contract defines the rules for aggregating the DEPs individual flexibility and offering it to the DSO. After the DSO registers a flexibility bid in the flexibility market the aggregator must choose a subset of its DEPS (based on their potential flexibility) to be able to submit a flexibility offer in the flexibility marketplace. More about this computation is provide in Section 3.3. If the aggregator's offer is accepted by the DSO, the flexibility request has to be decomposed into flexibility orders, tailored to the profile of each selected DEP and to monitor its actual delivery. Thus, smart contract associated with the energy aggregator defines the rules for making available and aggregating the flexibility provided by individual DEPs. The smart contract tracks and aggregates the flexibility $\Delta \pm$ (+ flexibility above the baseline – flexibility below the baseline) registered per DEP, with the overall goal of matching and balancing the monitored energy profiles with the flexibility request made by the DSO. If imbalances are detected between a DEP's monitored values and their corresponding requests, the aggregator smart contract will register and report them to the DSO and apply the corresponding penalties to the misbehaving DEP. Table 6 presents the state variables controlled by the aggregator level smart contract.

Table 6. Aggregator smart contract state variables

State Variables	Description
Aggregated Energy Flexibility $Flexibility_{Provided}^{Aggregator}$	Aggregated energy flexibility computed, based on the potentials of the enrolled DEPs during the flexibility market session, and published as a counter-offer to the DSO request.
Micro-grid Flexibility Delivery State $\Delta_{MicroGrid}$	The energy flexibility state at the level of aggregator computed real time based on the aggregated DEPs behaviour
Flexibility request profiles for each DEP $Flexibility_{Request}^{DEPi}$	Flexibility request signals assigned to each DEP for bringing the smart grid in balanced energy state
Flexibility service incentives and penalties	The incentives offered as a reward for making available the flexibility. The penalties imposed for noncompliance.

The micro-grid energy flexibility delivered state is calculated the difference between the total energy flexibility actual delivered by DEPs and the expected aggregated energy flexibility demand by the DSO. It is calculated by aggregating the $\Delta \pm$ imbalances registered at the level of each DEP enrolled with the aggregator and participating to the flexibility service delivery:

$$\Delta_{MicroGrid} = \sum_{i=1}^{N} \Delta_{Flexibility_{Request}^{DEP^i} - E_{Actual}^{DEP^i}}$$

where N represents the number of DEP enrolled with the aggregator.

The micro-grid energy flexibility state will aggregate both positive and negative values. The positive value represents a deficit of energy flexibility in the micro-grid (i.e. the DEP consumes more than instructed through the flexibility request signal), and the negative value represents a surplus of energy flexibility in the micro-grid (i.e. the DEP consumes less than instructed through the flexibility signal). If such imbalances are determined the aggregator smart contract will construct new flexibility requests signal allowing other DEPs in its portfolio to address them and as result re-balance the energy state of the grid.

The total energy flexibility made available by an aggregator to the DSO using its DEPs portfolio is calculated as:

$$Flexibility_{Provided}^{Aggregator} = \sum_{i=1}^{N} Flexibility_{Provided}^{DEPi}$$

In Figure 17 , the relevant part of the smart contracts managing the Aggregator offer registration is depicted. Once the optimal subset of prosumers matching the requested aggregated curve is determined the subset is registered on chain together with the assigned flexibility delivery curves to be provided by each DEP in the subset. Since the problem of finding the optimal subset of prosumers is a NP-hard problem, it is not feasible to search the solution on chain; however, the validation of the solution can be done on chain.

```
1   //=============== AGGREGATOR CONTRACT - Step  (4a) ==============
2   function registerAggregatedProsumers(ProsumerDataLibrary.ProsumerRequest[] memory _bestProsumersMatch) public {
3       require(msg.sender == ownerAggregator);
4       int32[] memory _flexibilityAggregated = new int32[](periodCardinality);
5       int32 evaluationScore = 0;
6       int32 riskFactor = 0;
7
8       for(uint p=0; p< _bestProsumersMatch.length; p++){
9           ...
10              require(_depFlexibilityOrdered[pj] <= prosumerPotential.flexibilityAbove[pj]);
11              require(_depFlexibilityOrdered[pj] >= prosumerPotential.flexibilityBelow[pj]);
12              _flexibilityAggregated[pj]+= _depFlexibilityOrdered[pj];
13          ...
14          evaluationScore+= prosumerPotential.reliabilityScore;
15          prosumersRequests[prosumerAddress] = _bestProsumersMatch[p];
16      }
17      DSO dso = DSO(dsoAddress); // Step  (4b)
18      dso.registerOffer( _flexibilityAggregated, evaluationScore, riskFactor);
19      emit PlaceOfferForDSO( _flexibilityAggregated, evaluationScore, riskFactor);
20  }
21   //================   DSO CONTRACT - Step  (4b) =========
22  function registerOffer( int32[] memory _flexibilityValues , int32 _evaluationScore, int32 _riskFactor) public {
23      require(aggregatorRegistry[tx.origin] == msg.sender);
24      require((now <= currentRequest.marketEndOfSession) && (currentRequest.marketStartOfSession <= now) );
25      require(_flexibilityValues.length == periodCardinality);
26      int32 _distance = 0;
27      for(uint i=0; i<periodCardinality; i++) {
28          _distance += abs(currentRequest.flexibilityRequest[i] - _flexibilityValues[i]);
29      }
30      if(_distance < bestOffer._distance){// Step  (4c)
31          bestOffer = FlexibilitOffer(tx.origin, _flexibilityValues, _evaluationScore,
32                                      _riskFactor, _distance,  new int32[](periodCardinality));
33          emit FlexibilityOfferRegistered(bestOffer);
34      }
35  }
```

Figure 17. Aggregator smart contract generating a new flexibility offer in the marketplace

The best match provided as input to the aggregator contract (Figure 17, line 3) will be validated against to DEP's restrictions. Thus, the flexibility scheduled to be delivered by each prosumer, will be validated against the flexibility potential bounds set for the corresponding prosumer (lines 11, 12). If any of the flexibility potential (below and above the baseline) are violated, then the transaction is reverted deeming the solution incorrect. However, if the solution is correct, the aggregated flexibility is computed (line 13), together with the aggregated reputation score of the involving prosumers (line 15) and published as a flexibility offer in the flexibility marketplace (line 19). The DSO contract will validate that the offer is published by a registered aggregator and forwarded by the aggregator's contract (line 25). The distance between the proposed offer

and the requested profile is computed (line 30). If the proposed offer provides a smaller distance than the best offer received so far (line 32), the current offer is stored (line 33).

3.2.3 DSO Smart Contract

In a similar fashion we defined the *DSO smart contract* for regulating DSO behaviour towards assuring the right balance of the energy state at the micro-grid and enforcing its stability. The DSO smart contract evaluates at micro-grid level the balance between the energy production and consumption and if imbalances are determined (i.e. load congestion point or surplus of renewable) it opens a flexibility market session, making a request for flexibility to compensate the imbalance by shifting a certain amount of flexible energy from their portfolio in return for a specified reward.

$$Flexibility_{Request}^{DSO} = \Delta_{Grid} = E_{Production} - E_{Consumption}$$

For an aggregator to be rewarded and receive the financial incentives its total energy flexibility should match the amount requested by the DSO:

$$Flexibility_{Request}^{DSO} = Flexibility_{Provided}^{Aggregator}$$

Through this contract, the DSO is able to publish flexibility requests in the flexibility marketplace for the aggregators in the area. Each aggregator is responsible for evaluating the potential of the prosumers managed and respond to the DSO by providing an offer for the flexibility, together with the prosumer reputation factor computed based on the DEPs provided data. The state variables registered by the DSO smart contract are depicted in Table 7.

Table 7. DSO smart contract state variables

State Variables	Description	
Energy Grid State, Δ_{Grid}	Grid level balance between production and consumption. It is used to create the request flexibility from aggregators being injected by the DSO into the aggregators' smart contracts.	$\Delta_{Grid} > 0$ Decrease energy demand by shifting baseline energy flexibility to avoid peak load
		$\Delta_{Grid} < 0$ Increase energy demand by scheduling energy flexibility to match an energy generation peak
Flexibility Request and Reward $< Flexibility_{Request}^{DSO}, Rew >$	The profile of flexibility request to be delivered by the Aggregator, and the reward provided in return for its service.	
Selected Flexibility Offer $Flexibility_{Provided}^{Aggregator}$	DSO selected flexibility offer published by an aggregator during the flexibility market session.	

The market is implemented like an auction-based system, where the best flexibility offers wins. While in the classical auction, the best offer is represented by the highest paid offer, in this situation we consider the best offer as the offer that provides the closest profile to the requested flexibility.

```
1   //================  DSO CONTRACT - Step (1)    ================
2   function publishRequest(uint _marketStartOfSession, uint _marketEndOfSession, uint[] memory _reward , int32[] memory _flexibilityRequest) public payable {
3       require(msg.sender == ownerDSO);
4       require(_reward.length == periodCardinality);
5       require(_flexibilityRequest.length == periodCardinality);
6
7       uint _flexibilityRequestrewardSum =0;
8       for(uint i=0; i<periodCardinality; i++){
9           _rewardSum += _reward[i] * abs( _flexibilityRequest[i]);
10      }
11      require(_rewardSum <= msg.value);
12      currentRequest = FlexibilityRequest(_marketStartOfSession, _marketEndOfSession, _reward, _rewardSum,  _flexibilityRequest);
13      emit FlexibilityRequestCreated(currentRequest);
14  }
```

Figure 18. DSO publishing a flexibility request as a bid in the flexibility marketplace

Figure 18 depicts the code responsible for validation and registration of the flexibility bid request which can only be registered through a transaction signed by the DSO (line 3). Another important aspect of the request registration is the reward associated together with the flexibility bid which is validated by the DSO smart contract when signing the transaction (lines 8-11).

```
1    //=================    DSO CONTRACT Step (7a)    ==================
2    function withdraw() public {
3        require(now > request.marketEndOfSession);
4        require(msg.sender == bestOffer.aggregator);
5        Aggregator agg = Aggregator(aggregatorRegistry[msg.sender] );
6        agg.setDSORequestedValues.value(request.rewardTotal)(request.reward, request.flexibilityRequest);
7        emit Withdraw(msg.sender, request.rewardTotal);
8    }
9    //=================   AGGREGATOR CONTRACT Step (7b) =================
10   function setDSORequestedValues(uint[] memory _reward, int32[] memory _flexibilityRequest )  public payable {
11       require(msg.sender == dsoAddress);
12       ...
13       require(expectedTotalRewardSum <= msg.value);
14       for(uint i=0; i<DEPs.length; i++){...
15           DEP dep = DEP (depAddress); // Step (8)
16           dep.setFlexibilityRequest(prosumersRequests[prosumerAddress].flexibilityOrder );
17       }
18   }
```

Figure 19. Aggregator Claiming Flexibility Request

At the end of the session (Figure 19 line 3), the aggregator that registered the winning flexibility offer (line 4) will take responsibility of the flexibility delivery in the next period (line 5-6) by enforcing the setup of the flexibility profiles on both the aggregator (line 10) and on all the prosumers from the chosen subset (line 14).

3.3 Flexibility request disaggregation using Oracles

We developed an algorithm to solve the optimization problem, that aims to select the optimal subset of DEPs from aggregator's portfolio and their flexibility energy profiles so that the aggregator can match best the DSO flexibility request.

Due to the fact that the optimization problem is an NP-hard problem, a heuristic algorithm is developed. Since smart contracts have limited resources for computing logic on chain, and each instruction is paid by the caller, we must consider a solution that can be run off-chain and a secure integration between the blockchain and such an algorithm. The concept of Oracles has been introduced for such scenarios, where complex logic or other APIs need to be called from outside the chain to provide results necessary in the contract execution. Such an oracle-based solution is presented in Figure 20.

The Oracle is a module that integrates the chain 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. Once the Oracle intercepts such an event, it will forward the request to the flexibility optimization service together with the necessary information. The Oracle is activated when the DSO publishes a flexibility request that may be addressed by an aggregator and forwards it to a Flexibility Optimization Service together with information on the aggregator's portfolio of DEPs. The Flexibility Optimization Service will have assigned a pair of public-private keys; the public key being stored on chain. The computation results are signed with the private key before returning them to the Oracle. The results together with the signature are injected on chain using a call-back function. Once the results are 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 consider that an aggregator has in its portfolio N_{DEP} prosumers, defined by their baseline energy, minimum and maximum flexibility over a discrete time interval $[0..T]$:

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

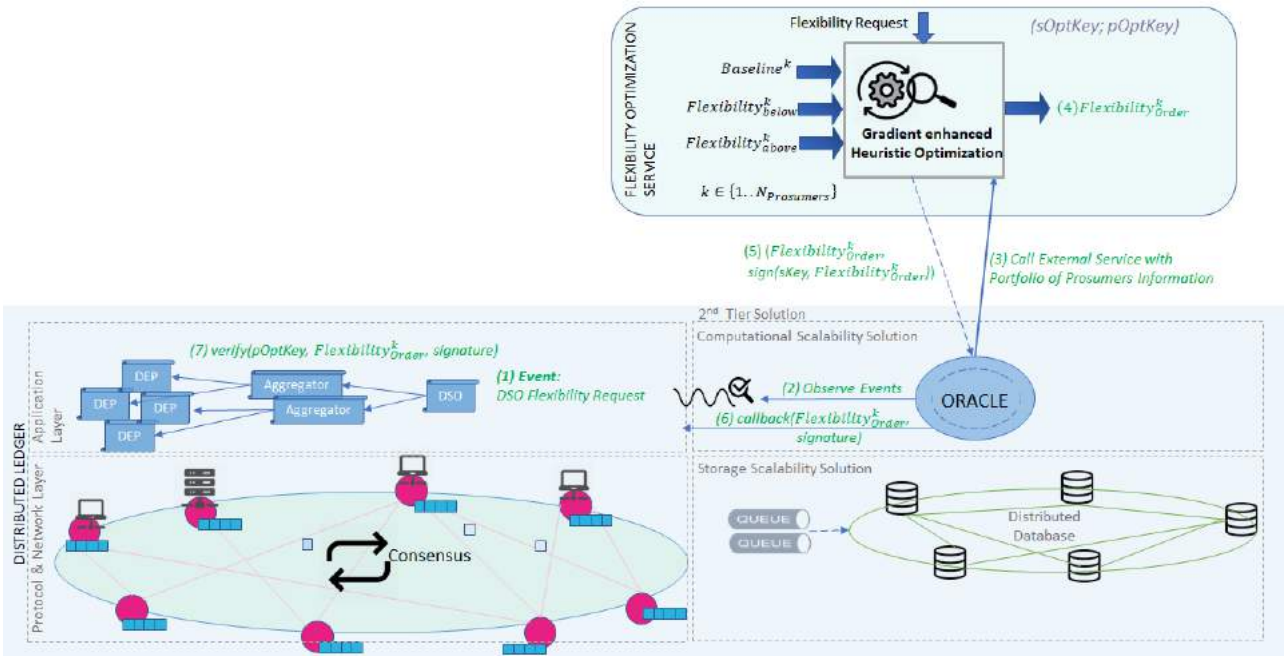


Figure 20. Oracle-based aggregation of DEPs flexibility for matching a DSO request

If the prosumer k is selected, the aggregator will compute a *flexibility order* for the prosumer, with the constraint of being bounded within the availability limits:

$$APC^k_{Below}(t) \leq Flexibility^{DEP^k}_{Request}(t) \leq APC^k_{Above}(t), t \in \{1..T\}$$

We model the action of choosing a prosumer to the aggregation subset by the binary variable $s(k) \in \{0,1\}$, ranging over the number of prosumers in the aggregator's portfolio k.

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:

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

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

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

Due to the continuous variables $Flexibility_{Request}^{DEP} \in R^{N_{DEP}*T}$ 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 program, being NP-hard. For determining an approximate solution, we leverage on a hybrid solver, based on both heuristics and gradient-based search methods. The solver is presented in detail in deliverable D3.3.

4 Blockchain platform for micro-grid energy management

We developed a prototype platform implementing on top of both peer-to-peer energy-trading and the delivery of flexibility services. A Server Application has been developed that connects directly to a node in the network and deals with prosumers' account management. All the information exposed to the prosumers is directly requested and forwarded to the server application that acts like a proxy between them and the blockchain platform. This architecture has been chosen to provide fast prototyping features, allowing the simulation of a large number of prosumers, as opposed to fully decentralized client applications that would require actually deploying a number of client applications proportional with the number of prosumers.

The conceptual architecture of the prototype implemented is presented in Figure 21.

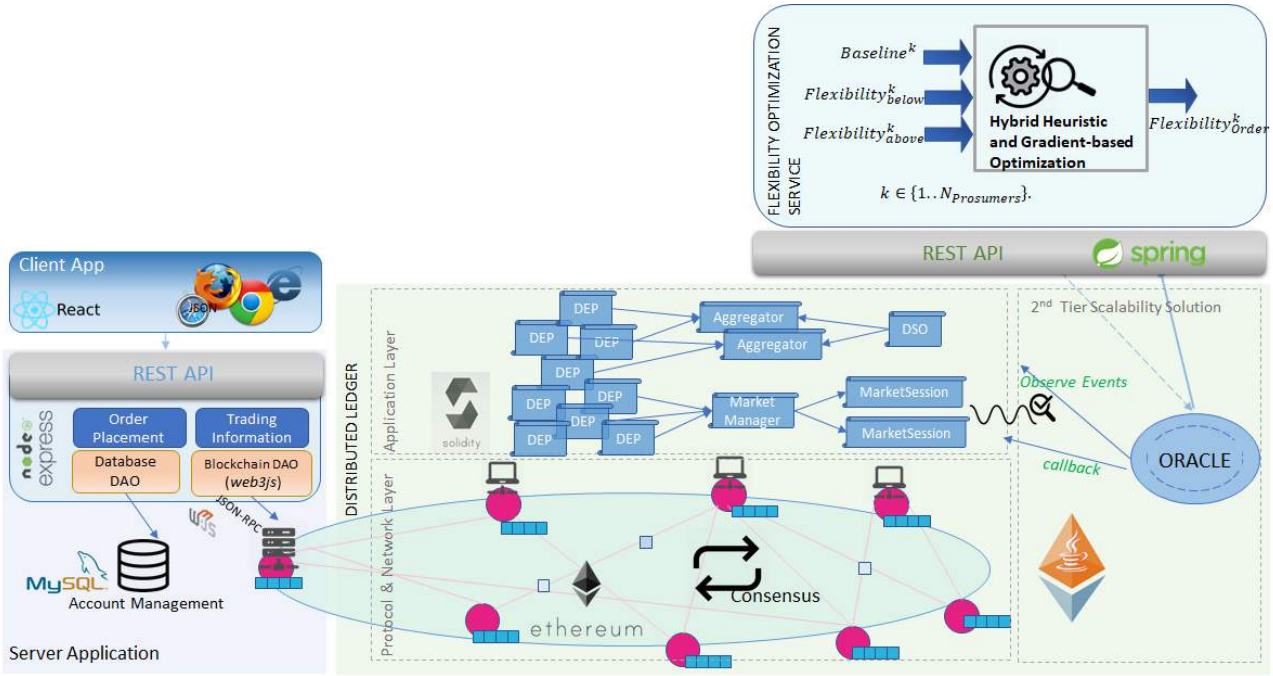


Figure 21. Conceptual architecture and technologies

The architecture is composed of five main components, as follows:

- **Ethereum [9] Network**– the blockchain network where smart contracts managing the energy-trading and flexibility services implemented using Solidity [10] are deployed.
- **Server Application exposing REST API**:
  - **MySQL Database** – this component represents the account management storage, having the goal of saving data about accounts.
  - **REST API** is implemented using NodeJS [11], and web3js [12]. The database DAO manages the access to the database and the blockchain DAO manages the smart contracts method calls.
- **Oracle –** a standalone component waiting for relevant events implemented using Java and web3j.
- **Flexibility Optimization Service -** implements an optimization algorithm that aims to compute the optimal subset of prosumers and their flexibility energy profiles exposing a REST API using Spring REST [13].
- **React Client App** – a web application that allows the user to interact with the prototype marketplaces by publishing bids/offers and asking for statistics about active and past sessions, and real-time energy-tracking.

The public defined API is presented in

Table 8.

Table 8. Public REST API exposed

Peer-to-peer Energy-trading related REST API	
Place Orders Based on Estimated Energy Profile	
Description	Through this interface, prosumers may place several orders based on the estimated energy profile and prices provided for the next period of time.
End-point URL	/market/order/{market_session}/{account}
Parameters	market_session: the market session where the order is to be published
	account: the account identification information about the prosumer that publishes the orders
Allowed HTTP Methods	**POST**
Request Body	"prosumerTradingEnergy": { "estimatedEnergy": [20,40,-30,40,50,70,30,45,45,-23,-56, 34,76,34,34,65,34,23,76, 34,54,45,23,34], "tradingPrices": [20,40,30,40,50,70,30,45,45,23,56,34,76,34,34,65,34,23,76,34, 54,45,23,34], "profileStartHor": 0, "energyType": "GREEN", "marketSessionType":"DAYAHEAD" }
Response	{"orderIds": ["0x9a37ac6ecdc856ea6e87698d889217803b82965e52f4af852dfcaf08bdd996b5", ..]}
Place Energy Order (BID or OFFER)	
Description	Through this interface, prosumers may place individual orders (bids or offers) on the open sessions on chain.
End-point URL	/market/order/{market_session}/{ account }
Parameters	market_session: the type of the market session where the order is to be published
	account: the account identification information about the prosumer that publishes the order
Allowed HTTP Methods	**POST**
Request Body	{ "orderSide": "BID", "prosumerAddress":"0xAA21803000499f1b58C67F4DA7083AFA2ee37090", "timestamp":"123453123", "tokenId":10, "metadata": { "startTimeToken": 0,

	"endTimeToken": 1, "energyType":"GREEN", "producer":"0xAA21803000499f1b58C67F4DA7083AFA2ee37090"; } "quantity":12321 "price":35, }
Response	{ "orderId": "0x9a37ac6ecdc856ea6e87698d889217803b82965e52f4af852dfcaf08bdd996b5"}
Register Matched Orders (BIDs and OFFERs)	
Description	Through this interface, the oracle can publish the matching orders to be settled.
End-point URL	/energy/market/trades/{market_session_contract}
Parameters	market_session_contract: the address that identifies the market session on which the trades are to be published
Allowed HTTP Methods	**POST**
Request Body	[{ "id": "0x309911d2e38ff1649f7b6d39ef0e3fb33c87ae5d4fda6fa065ec9821c77fe2e3", "buyOrderId": "0x1c817f45dd2349c07b100ac0c9204d1652cea73006c0b000c9c39c4c40710d78", "sellOrderId": "0x95c8b70fc368ad3a3d7544715461d189616114cca00cabb97d2665a84e5d8b54", "prosumerBuyingAddress":"0x04fb94f5e2555d1e860462060337aa62ec6e919d", "prosumerSellingAddress":"0xAA21803000499f1b58C67F4DA7083AFA2ee37090", "timestamp":"123453123", "tokenId":10, "quantity":12321 "price":35, }, …]
Response	
Flexibility Services Management-related REST API	
Register Prosumer Flexibility Potential	
Description	Through this interface, the prosumer can place his flexibility potential for the following period of time.
End-point URL	/prosumer/potential/{account}
Parameters	account: the account identification information of the prosumer that publishes the request

Allowed HTTP Methods	**POST**

Request Body	{
	"baseline": [20,40,30,40,50,70,30,45,45,23,56,34,76,34,34,65,34,23,76,34,54,45,23,34],
	"flexibilityBelow": [10,30,20,30,40,60,20,35,35,13,46,24,66,24,24,55,24,13,66,24,44,35,13,24],
	"flexibilityAbove": [30,50,40,50,60,80,40,55,55,33,66,44,86,44,44,75,44,33,86,44,64,55,33,44]
	}

Response	
Register DSO Request for Flexibility	
Description	Through this interface, the DSO account can place a flexibility request for the following period of time.
End-point URL	/dso/flexibility-request/{account}
Parameters	account: the account identification information of the DSO that publishes the request
Allowed HTTP Methods	**POST**

Request Body	{
	"reward": [20,20,20,20,20,20,20,20,20,20,20,20,20,20,20,20,20,20,20,20,20,20,20,20],
	"flexibilityRequest": [50733, 62244, 61116, 72334, 48948, 63998, 75964, 41910, 73644, 106410, 67629, 125160, 145908, 71520, 176958, 82218, 121033, 115099, 79313, 42571, 101089, 46595, 77595, 37843]
	}

Response	
Register Aggregator Flexibility Offer	
Description	Through this interface, the Aggregator account can place the optimal subset of the prosumers that can match the profile requested by the DSO. Based on the provided information, the aggregated profile is placed as a counter-offer to the DSO's request
End-point URL	/aggregator/optimal-prosumer-subset/{account}
Parameters	account: the account identification information of the aggregator that makes the request
Allowed HTTP Methods	**POST**

Request Body	[{
	"prosumerAddress": "Prosumer 1",
	"flexibilityOrder": [20,40,30,40,50,70,30,45,45,23,56,34,76,34,34,65,34,23,76,34,54,45,23]},
	{
	"prosumerAddress": "Prosumer 1",

	"flexibilityOrder": [20,40,30,40,50,70,30,45,45,23,56,34,76,34,34,65,34,23,76,34,54,45,23]} …]

Response	{
	"flexibilityAggregated": [50733, 62244, 61116, 72334, 48948, 63998, 75964, 41910, 73644,
	106410, 67629, 125160, 145908, 71520, 176958, 82218, 121033,
	115099, 79313, 42571, 101089, 46595, 77595, 37843]
	"evaluationScore": "1000",
	"riskFactor": "2"
	}

Get Prosumer's Flexibility	
Description	Through this interface, the prosumer interrogates the flexibility request set by the aggregator for him to follow.
End-point URL	prosumer/flexibility-order/{account}
Parameters	account*:* the account identification information of the prosumer that makes the request
Allowed HTTP Methods	**GET**
Request Body	["prosumerAddress": "Prosumer 1"]
Response	[20,40,30,40,50,70,30,45,45,23,56,34,76,34,34,65,34,23,76,34,54,45,23,34]

5 Experimental results

A series of experiments have been conducted on our private deployment, considering an Ethereum private network, having Parity nodes set to run using a Proof-of-Authority consensus mechanism.

5.1 Peer-to-peer energy-trading

For peer-to-peer energy-trading we considered a micro-grid with 14 prosumers: 7 energy producers and 7 energy consumers. There are registered with our local energy-trading marketplace. The objective was to locally match production with consumption, thus avoiding the escalation of potential imbalances. Each prosumer is publishing its energy bids and offers on the day-ahead market session according to their estimated (i.e. forecast) demand and production, respectively.

In Figure 22, presents the bids and the offers registered considering the price for the electricity of 0.2 euro per kWh and the price of Ether of 330 Euro, resulting in a reference price of 606 Gwei per Wh. The clearing price at which all accepted bids and offers are remunerated is computed at 600 Gwei per Wh as well as the overall quantity of energy traded.

Figure 22. Energy bids and offers registered, their associated price and calculated clearing price

The bids and offers of the first four producers and consumers are matched as follows: the consumers willing to pay at least the clearing price are matched with the producers willing to sell energy at a price lower than the clearing price (see Figure 23).

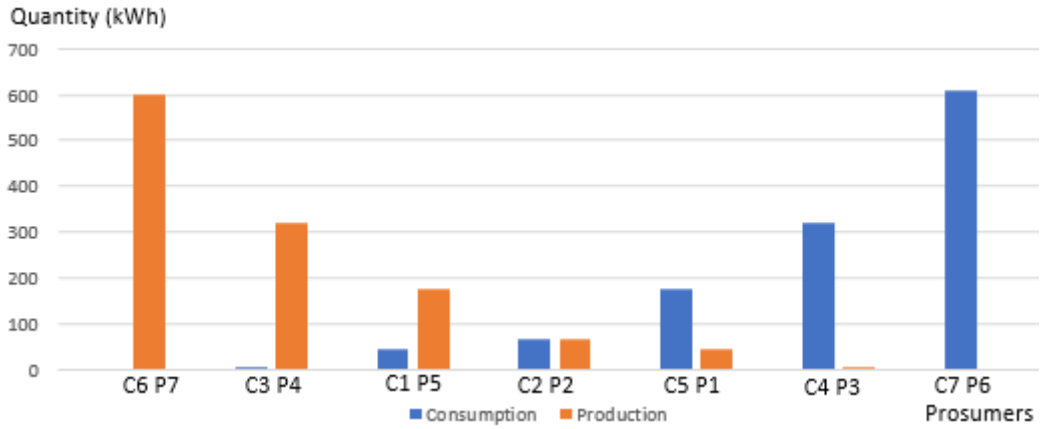


Figure 23. The energy bids and offers matched

Figure 24 shows Consumer 4 energy consumption monitoring in comparison with the energy brought from the marketplace.

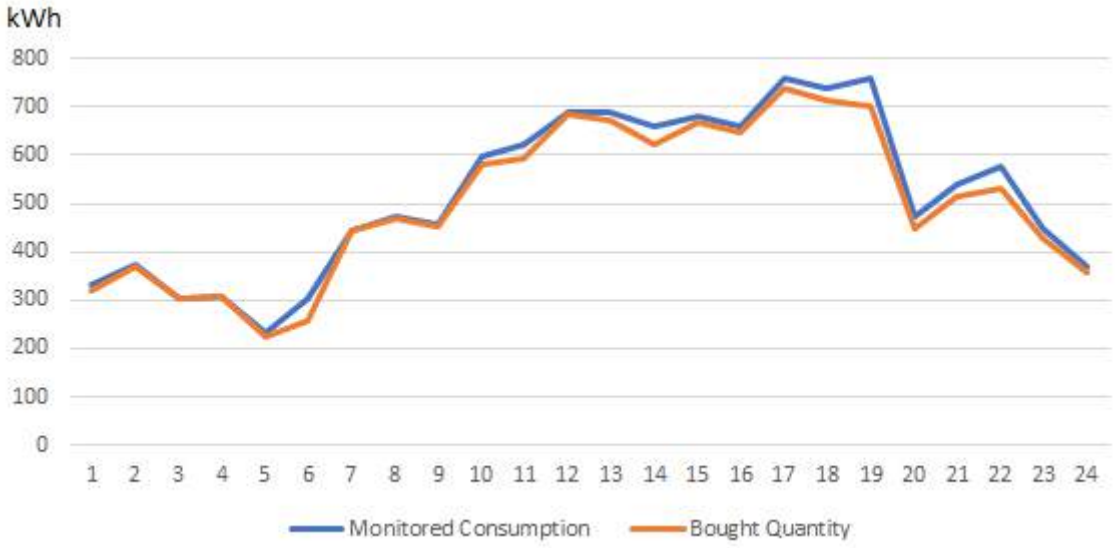


Figure 24. Consumer 4's actual energy consumption

In Figure 25 the payments of the Consumer 4 are presented. Firstly, the payments on the day-ahead market are registered once the bids/offers are published and matched, and then additional payments are registered during real-time energy-monitoring, each time the consumer exceeds the bought token quantity. The excess energy is being brought in near real time at higher energy price.

Figure 25. Consumer 4's actual energy payments

Similarly, in Figure 26 the matched energy producer (Producer 4) activity is depicted, presenting the comparison between the energy monitored values and the ones sold in the marketplace.

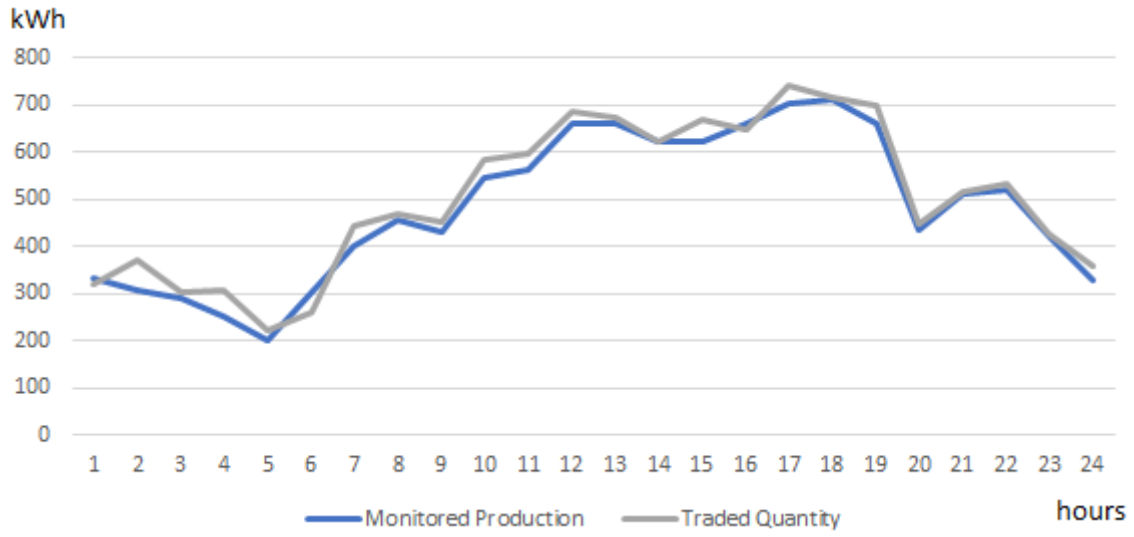


Figure 26. Producer 4's actual energy generation and the one traded

Once the producer registers in the peer-to-peer market, energy tokens are generated proportionally to the estimated energy quantity, in exchange for an initial deposit made by the producer. As depicted in**Error! Reference source not found.** Figure 27, these initial deposits are returned together with the value earned from the matching buyers as soon as proof of the energy delivered is registered on chain. If the producer fails to deliver the entire quantity, part of the producer's deposit will be used to acquire the missing quantity at the real time energy price, and the rest will be returned.

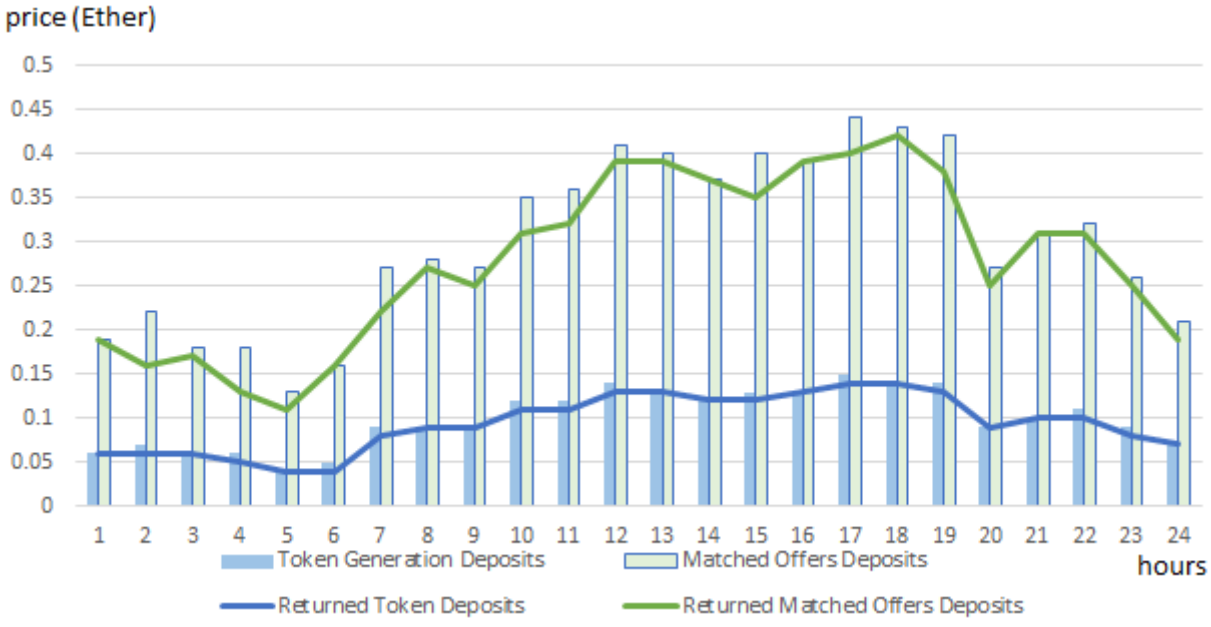


Figure 27. Producer 4's incomes and the ratio between deposit tokens and matched ones

5.2 Flexibility services management

For the scenario of flexibility management, we considered a micro-grid with 12 prosumers and 2 aggregators. Out of the 12 prosumers, 6 prosumers have been set up using historical data provided by the partner KIWI, and 6 prosumers have been set up using synthetically generated data.

Figure 28 presents the initial energy state of the simulated micro-grid. It can be noticed that there is an imbalance between the total energy demand and energy production in the time window between hours 9 and 20 generated by an unpredicted peak of renewable production.

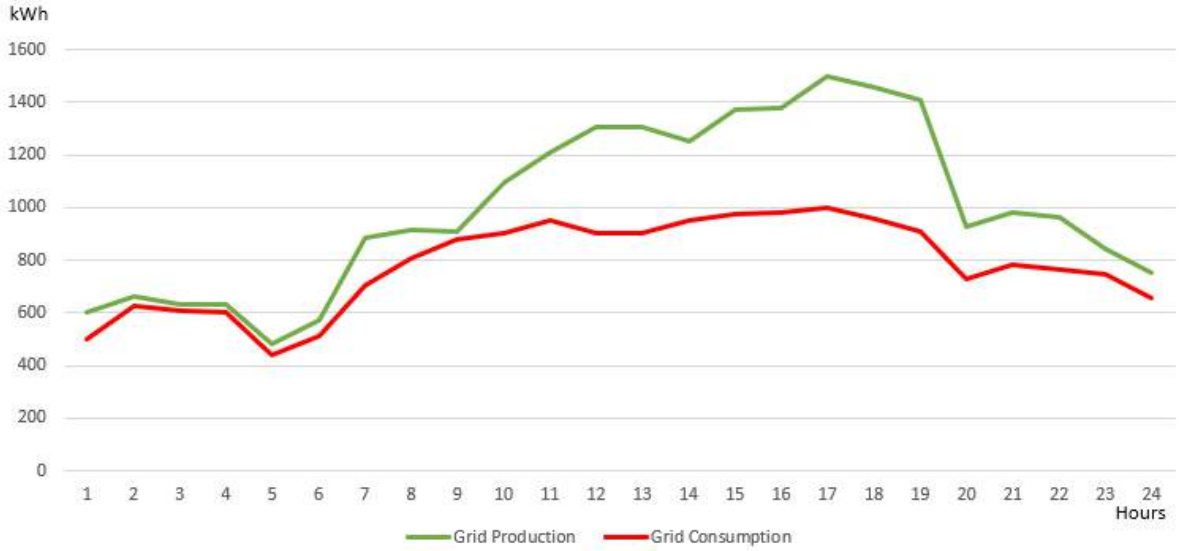


Figure 28. Initial state of the micro-grid and the congestion point

The DSO identifies a congestion point and the associated micro-grid aggregators are activated. More specifically, the DSO publishes a flexibility request on the flexibility marketplace asking for a specific amount of energy flexibility such that in the micro-grid the energy demand it is increased to match the production peak.

Figure 29. DSO flexibility request and aggregators flexibility offers

In Figure 29 the DSO flexibility request and the flexibility offers from two aggregators are presented. The DSO accepts one flexibility offer and then sends a flexibility order to selected aggregator who will adjust the load of their registered DEPs as to fulfil the flexibility need. Aggregator 1's offer, being the closest to the DSO flexibility request profile is selected by the DSO associated smart contract as the best offer.

Aggregator 1 get in contract with enrolled prosumers connected to this congestion point to offer and aggregate their flexibility. Upon prosumers flexibility availability their availability, the aggregator-associated

smart contract will inject individual flexibility control signals in the smart contracts regulating individual prosumers flexibility, thus requesting them to adapt their baseline energy profile by shifting flexible energy.

Figure 30 presents the baseline energy profiles of each of the 6 prosumers enrolled with Aggregator 1 considered in our testing scenario.

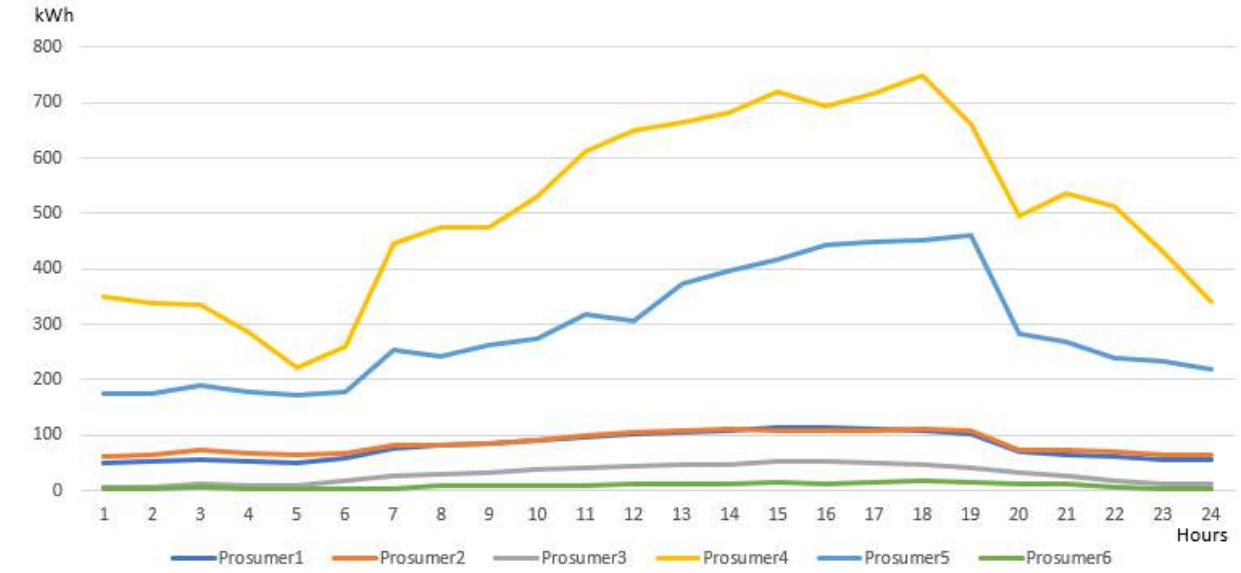


Figure 30. Baseline energy profile of prosumers enrolled with Aggregator 1

Figure 31 presents details on the flexibility service provided by Prosumer 1: the baseline energy demand profile, the flexibility potential upper and lower bounds, the flexibility control signal received from Aggregator 1 and the adapted energy profile as result of flexibility shifting.

Figure 31. Prosumer 1's adapted energy profile and flexibility potential

In Figure 32 presents the subset of prosumers whose aggregated flexibility is offered by the Aggregator 1. For each prosumer, the signal selected by the load disaggregation module is bounded by the prosumer's flexibility potential determined by the minimum and the maximum flexibility availability.

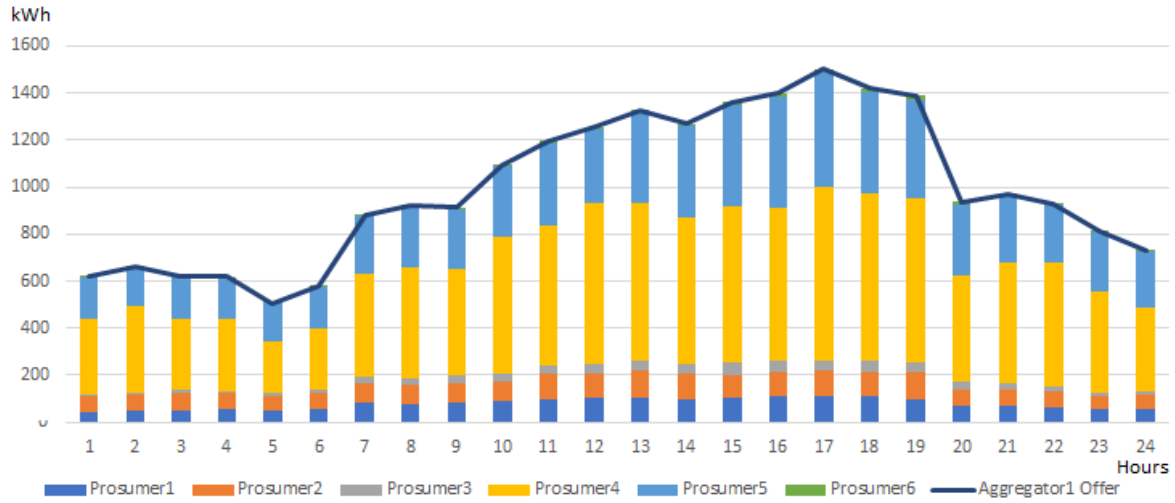


Figure 32 Subset of prosumers selected by Aggregator 1 to deliver the flexibility requested

Each prosumer's activity is tracked and validated in real-time against the flexibility request set by the Aggregator 1. The corresponding smart contracts evaluate the difference between the requested energy flexibility (i.e. a curve signal) and the flexibility actual delivered (as shown by monitored energy values registered in the distributed ledger). If relevant deviations are identified, specific actions will be taken to rebalance the energy demand with the energy production, thus, the smart contracts act as a decentralized control mechanism.

In Figure 33 the imbalances computed as differences between the monitored values and the requested values are depicted. Each time the imbalances pass the threshold minimum allowed (10% of the request), the prosumer is penalized for the imbalance created (for example hours 10 and 18); otherwise, the prosumer will receive incentives proportionally with the service delivered.

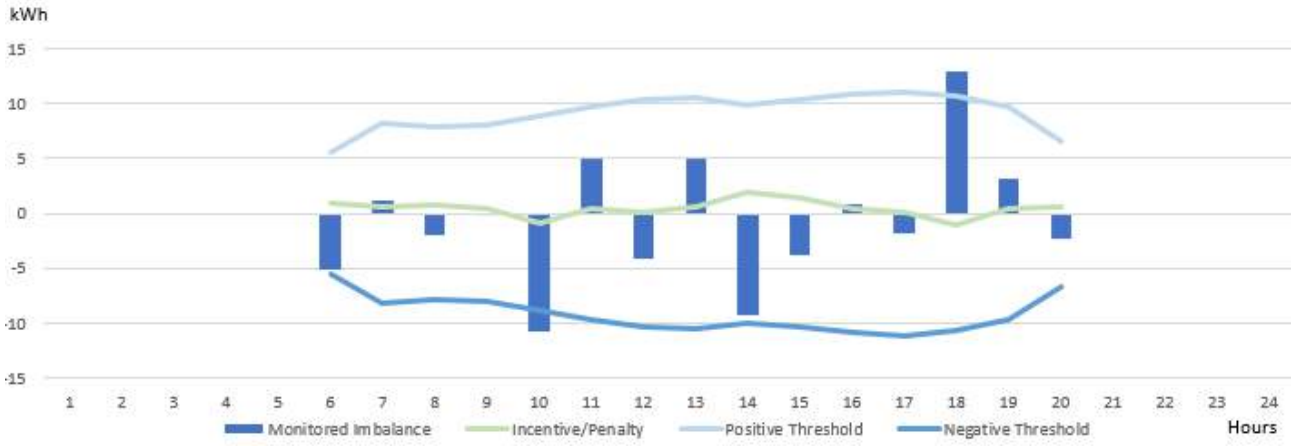


Figure 33. Prosumer 1's flexibility delivery tracking

The imbalances generated by the prosumers are further reported to the aggregator, the total value in the micro being presented in Figure 34. Whenever prosumers fail to follow the requested profile, ancillary services are triggered (batteries and generators) to balance the grid and provide the surplus or deficit of energy. Furthermore, new signals are created by the aggregator contract whenever the batteries have been charged

or discharged as a result of balancing the grid. By addressing these signals, the batteries can be brought back in the state they were before being triggered.

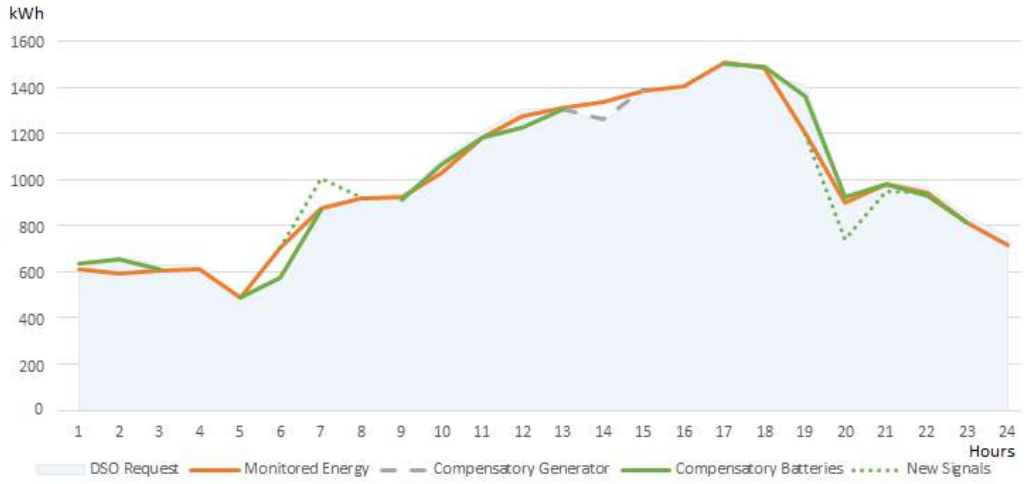


Figure 34. Micro-grid level flexibility delivery tracking

6 Conclusion

In this deliverable, we presented a blockchain-based platform for the distributed control and management of energy micro-grid leveraging on peer to peer energy trading and flexibility services management. The price-driven peer-to-peer energy marketplace allows the local trading and consumption of energy produced at the micro-grid while flexibility services distributed provisioning and management allows to assess and track in near real-time the share of flexible energy actually delivered.

The deliverable presents in detail self-enforcing smart contracts that are defined and used for the implementation of both micro-grid management approaches.

In our flexibility management scenario, the results are promising the flexibility requests signals are followed with high accuracy while the deviations in the share of flexibility actual delivered are identified and addressed in near real-time fashion. Energy trading results show the potential for balancing locally in a peer-to-peer fashion the energy demand and production. As result the escalation of potential micro-grid level imbalances could be avoided while the energy transactions are being tracked in real time to settle prosumers' wallets considering actual energy monitored data.

For future work we plan to build on top of and extend the blockchain based platform by implementing energy bid/offer-matching algorithms in the peer-to-peer energy marketplace (which is an NP-hard problem) taking into account also the capacity of the lines between prosumers and promoting the consumption of green energy and also by extending flexibility management to implement a price-driven flexibility marketplace that allows individual prosumers to participate.

References

[1] ERC721 open standard, http://erc721.org/

[2] Proof of Authority, https://medium.com/poa-network/proof-of-authority-consensus-model-with-identity-at-stake-d5bd15463256

[3] [Practical Byzantine Fault Tolerance, http://pmg.csail.mit.edu/papers/osdi99.pdf

[4] ERC721 Standard, Available online at https://github.com/ethereum/EIPs/blob/master/EIPS/eip721.md

[5] ERC721Metadata defined at https://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/token/ERC721/ERC721Metadata.sol

[6] J. Benet, "IPFS -Content Addressed, Versioned, P2P File System", arxiv.org, July 14th, 2014. Available: https://arxiv.org/pdf/1407.3561.pdf.

[7] D3.1: Electricity production/consumption forecasting techniques and tool V1

[8] Pop, C., Cioara, T., Antal, M., Anghel, I., Salomie, I., & Bertoncini, M. (2018). Blockchain based decentralized management of demand response programs in smart energy grids. *Sensors*, *18*(1), 162.

[9] Wood, Gavin. "Ethereum: A secure decentralised generalised transaction ledger." Ethereum project yellow paper 151 (2014): 1-32.

[10] Solidity, Available online at https://solidity.readthedocs.io/en/latest/

[11] NodeJS, https://nodejs.org/en/

[12] web3js, https://web3js.readthedocs.io

[13] Spring Rest , https://spring.io/guides/gs/rest-service/

[14] D3.2: Recommendations for baseline load calculations in DR programs V1