Interreg
Euregio Meuse-Rhine
BC4P

EUROPEAN UNION
European Regional Development Fund

Smart Contracts for the Blockchain for Prosumers Project D.T2.2.1

Version	Description	Author	Date
0.1	Initial draft.	Prakash Gupta (OUNL), Pietro Piccini (OUNL), Stefano Bromuri (OUNL), Florian Maurer (FH-Aachen), Lukas Walk (FH-Aachen)	11/01/2023
1.0	Finalization	Florian Maurer	21/02/2023

PXL
LIÈGE université
FH Aachen University of Applied Sciences
Open Universiteit
CAP CONSTRUCTION
howest
Tiorc
ZU YD
CLIMATE CITIES

Table Of Contents

Technical words	Explanation
Gsy-e framework	Framework provides a decentralized platform for energy trading and management, allowing energy producers, consumers, and other market participants to trade energy in a secure and transparent manner.
Gsy-e sdk	Provide the functionality to do asset trading and grid management
Redis	Redis is an open-source, in-memory data structure store used for caching, database, and message broker purposes.

Abstract

In this report, we explain the implementation and working of the smart contract with the Grid singularity exchange framework (gsy-e). The gsy-e framework enables prosumers and consumers organized in energy communities, with the support of local aggregators, to simulate and implement peer-to-peer and community energy trading by creating an active local energy market. We aim to integrate the blockchain with gsy-e to enable traceability, security, and other advantages discussed later. To enable blockchain functionality, we have created a set of smart contracts that handle the transactions of the assets and energy market on the blockchain. The utility of smart contracts is to trade energy between consumers and producers and to enable an increase in small-scale transactions. The smart contracts are connected with Redis and get events based on creation of asset trades, bids, and offers.

Introduction

The energy market in Europe is a complex system that involves the production, distribution, and consumption of energy. This market is regulated by a number of different bodies, including the European Union and national governments[1].

Several types of energy sources are used in Europe, including fossil fuels (such as coal, oil, and natural gas), nuclear power, and renewable sources (such as wind, solar, and hydroelectric). These sources are used to generate electricity, which is then transmitted through a network of transmission and distribution lines to homes, businesses, and other consumers[2].

A number of factors, including the availability of different energy sources, the cost of production, and energy demand, shape the energy market in Europe. There are also a number of initiatives in place to encourage the use of renewable energy sources and to reduce the region's reliance on fossil fuels.

Overall, the energy market in Europe is constantly evolving, and efforts are being made to ensure that it is sustainable, affordable, and reliable for consumers.

Blockchain technology has the potential to revolutionize the energy market by enabling the creation of decentralized networks for the production and distribution of energy.

One way the blockchain can help the energy market is by enabling the creation of peer-to-peer (P2P) energy trading platforms[3,4]. These platforms allow individuals and businesses to buy and sell excess energy that they generate from renewable sources, such as solar panels or wind turbines. This can increase the amount of renewable energy used and reduce reliance on fossil fuels.

[1] Pepermans, Guido. "European energy market liberalization: experiences and challenges." *International Journal of Economic Policy Studies* 13.1 (2019): 3-26.

[2] https://publicgoods.eu/comparison-production-and-consumption-energy-european-union

[3] Andoni, Merlinda, et al. "Blockchain technology in the energy sector: A systematic review of challenges and opportunities." *Renewable and sustainable energy reviews* 100 (2019): 143-174.

[4] M. Abdelwahed, T. A. Boghdady, A. Madian, and R. Shalaby, "Energy Trading Based on Smart Contract Blockchain Application," *2020 International Conference on Innovation and Intelligence for Informatics, Computing and Technologies (3ICT)*, 2020, pp. 1-6, doi: 10.1109/3ICT51146.2020.9312010.

The blockchain can also be used to create smart contracts that automate buying and selling energy5. These contracts can be programmed to execute automatically when certain conditions are met, such as when the energy price reaches a certain level or when a certain amount of energy has been generated. This can help to make the energy market more efficient and reduce the need for intermediaries.

In addition, the blockchain can be used to create more transparent and secure energy supply chains. It can be used to track the production and distribution of energy, ensuring that it is being used sustainably and ethically. This can help to build trust in the energy market and increase confidence in the renewable energy sector.

In this report, we explain the implementation and working of the smart contract with the Grid singularity exchange framework (gsy-e)[6]. The gsy-e framework enables prosumers and consumers organized in energy communities, with the support of local aggregators, to simulate and implement peer-to-peer and community energy trading by creating an active local energy market. We aim to integrate the blockchain with gsy-e to enable traceability, security, and other advantages discussed later. To enable blockchain functionality, we have created a set of smart contracts that handle the transactions of the assets and energy market on the blockchain. The utility of smart contracts is to trade energy between consumers and producers and to enable an increase in small-scale transactions. The smart contracts are connected with Redis and get events based on assets, trades, bids, and offers.

The rest of this report is structured as follows: Section 2 discusses the structure of the BC4P project as a whole and how the smart contracts are supported in the platform by extending gsy; Section 3 goes into the details of the Solidity smart contracts implemented for BC4P; Section 4 discusses an example of interaction between prosumers in the BC4P blockchain; Section 5 concludes this report.

BC4P: The total project

The project has two main components: the prosumers infrastructure represented by the grid singularity framework (gsy-e)[7] and the blockchain infrastructure represented through solidity smart contracts[8]. In this report, we are mainly focusing on blockchain infrastructure. The project uses gsy to manage the energy market and trading activity of consumers and producers. One of this project's main challenges is providing locally generated energy to other consumers and users without using

[5] *Leonhard, Robert, Developing Renewable Energy Credits as Cryptocurrency on Ethereum's Blockchain (December 14, 2016). Available at SSRN: https://ssrn.com/abstract=2885335 or http://dx.doi.org/10.2139/ssrn.2885335*

[6] *Grid Singularity. (August 27, 2020). Modelling study to assess the potential benefits of trading in and between local energy communities in Germany, https://gridsingularity.medium.com/modelling-study-to-assess-the-potential-benefits-of-trading-in-and-between-local-energy-d721395ddd4b*

[7] https://gridsingularity.github.io/gsy-e/documentation/

[8] *Dannen, C. (2017). Solidity Programming. In: Introducing Ethereum and Solidity. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-2535-6_4*

Interreg Euregio Meuse-Rhine BC4P European Union European Regional Development Fund

intermediaries such as energy stock exchanges, energy supply companies, or energy traders and trading with each other. To overcome the challenge, we have introduced a blockchain with smart contracts that manage the monetary aspects of the energy exchange in a P2P fashion. The blockchain with smart contracts makes it possible to trade locally generated energy to other consumers and users without compromising authenticity, credibility, and privacy.

The Smart contracts are implemented in the programming language Solidity[9]. The system consists of modules that make it possible to do the energy business and trading, such as Grid singularity Exchange, Software development kit –SDK (gsy-e-SDK), energy market package, and b4p-contracts. Figure 1 shows the coupling between the modules used for the BC4P market simulation. A brief explanation of each component shown in the layout is given in the following section.

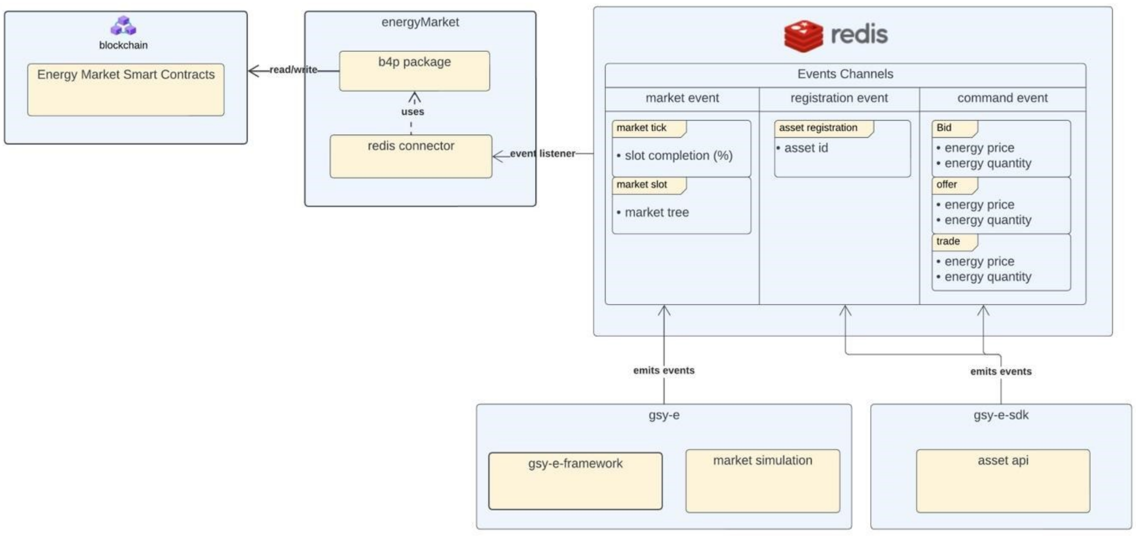


Figure 1: modules used in the BC4P simulation

Grid Singularity Energy Exchange

Grid Singularity provides a facility to consume, trade, or share energy based on the consumer's preferences of an energy type, location source, price, or trading partner. As a result of Grid Singularity's application interface, households and distributed energy assets are digitally represented by trading agents, and grid operators can integrate to facilitate a bottom-up market design.

Grid singularity exchange enables prosumers and consumers organized in energy communities, with the support of local aggregators, to simulate and implements peer-to-peer and community energy trading through the creation of active local energy markets.

[9] Dannen, Chris. *Introducing Ethereum and solidity*. Vol. 1. Berkeley: Apress, 2017.

Grid Operator

It can register to manage different markets, across different grid levels, on different grid levels in collaboration. The grid operator role is explicitly designed for Distribution System Operators (DSOs), Distribution Network Operators (DNOs), Independent System Operators (ISOs), and Transmission System Operators (TSOs). By optimizing specific metrics, like peak percentage, they can manage grid congestion in the local energy market by changing grid fees once they are registered and approved by the EO.

- ### Asset API

The Grid Singularity Asset API is designed for aggregators (energy service providers) who wish to provide their customers (individuals and communities) with the benefits of peer-to-peer and community trading facilitated by the Grid Singularity Exchange. Asset API can create agents that follow custom trading strategies to buy and sell energy in the energy market on behalf of managed energy assets. The agent can request and receive information through the Asset API, feed that information into an algorithm, and post bids or offers on the exchange.

- ### Grid API

The Grid Singularity Grid Operator API is designed for grid operators (notably Distribution System Operators or DSOs) to manage congestion and grid constraints in different markets across the grid. The structure is designed to manage multiple markets by a single agent (digital representation of the grid operator preferences), allowing information to be aggregated for integrated grid management.

It allows users to create agents that can dynamically change the grid fee in different markets. These agents can request and receive information through the Grid Operator API, feed that information into a tariff model, and submit grid fees through the Exchange SDK to change grid fees on the exchange.

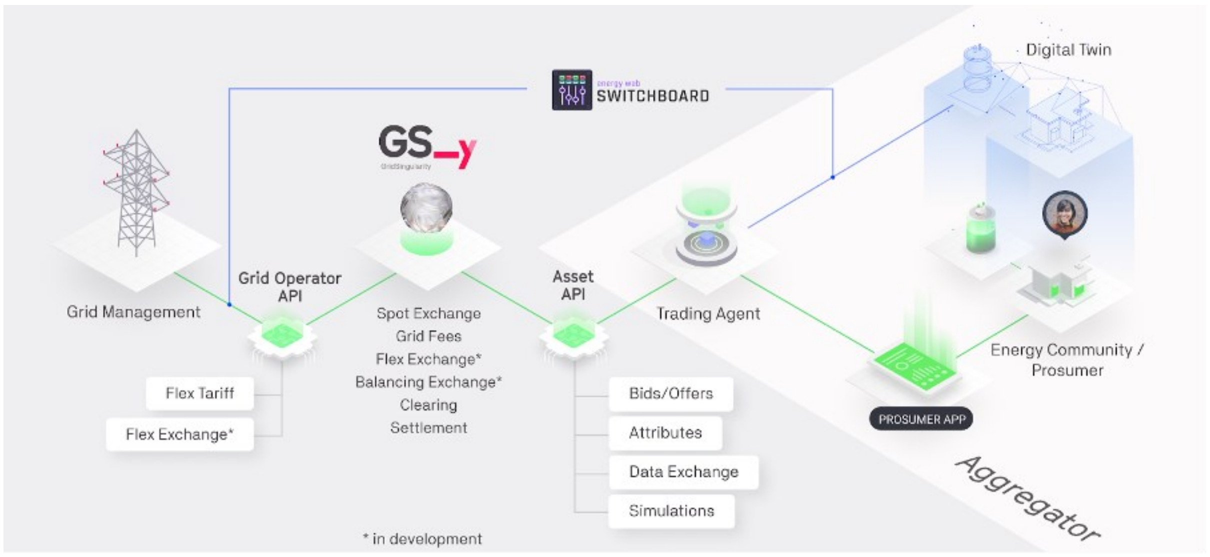


Figure 3: Overview of the Grid Singularity Exchange: The flow of information from energy resources is channeled through intermediaries and trading agents using API technology for pairing purposes. Energy Web Switchboard will serve as a decentralized register of assets and a decentralized tool for managing access to data.

REDIS event bus and BC4P REDIS Event Connector

REDIS is an open-source, in-memory data structure store used as a database, cache, and message broker used as the backbone event bus by GSY-e. In our use case, Redis is used for the message broker functionality or as an event bus, which facilitates communication between the GSY-e simulation, the asset, and grid API, and a component called the “BC4P REDIS Event Connector”, which performs smart contract operations. The communication follows a publisher/subscriber pattern where the simulation publishes market events and markets and listens for market commands emitted by the asset client. The Redis connector listens for all combined events and transforms these events into blockchain transactions. Therefore, the interaction between the assets in terms of energy is decoupled from the monetary aspects, avoiding potential bottlenecks caused by blockchain technology. A different architecture may be possible in which the blockchain may be strongly coupled with the interaction at the energy level, depending on the ability of the blockchain to handle it. In such a case where the blockchain technology can handle a high throughput of events, the only thing that needs to be changed is how the events are managed at the BC4P REDIS Event Connector to ensure high synchronicity with the events taking place in the GSY-e energy exchange. A figure of how the layers is related to each other in BC4P is shown below.

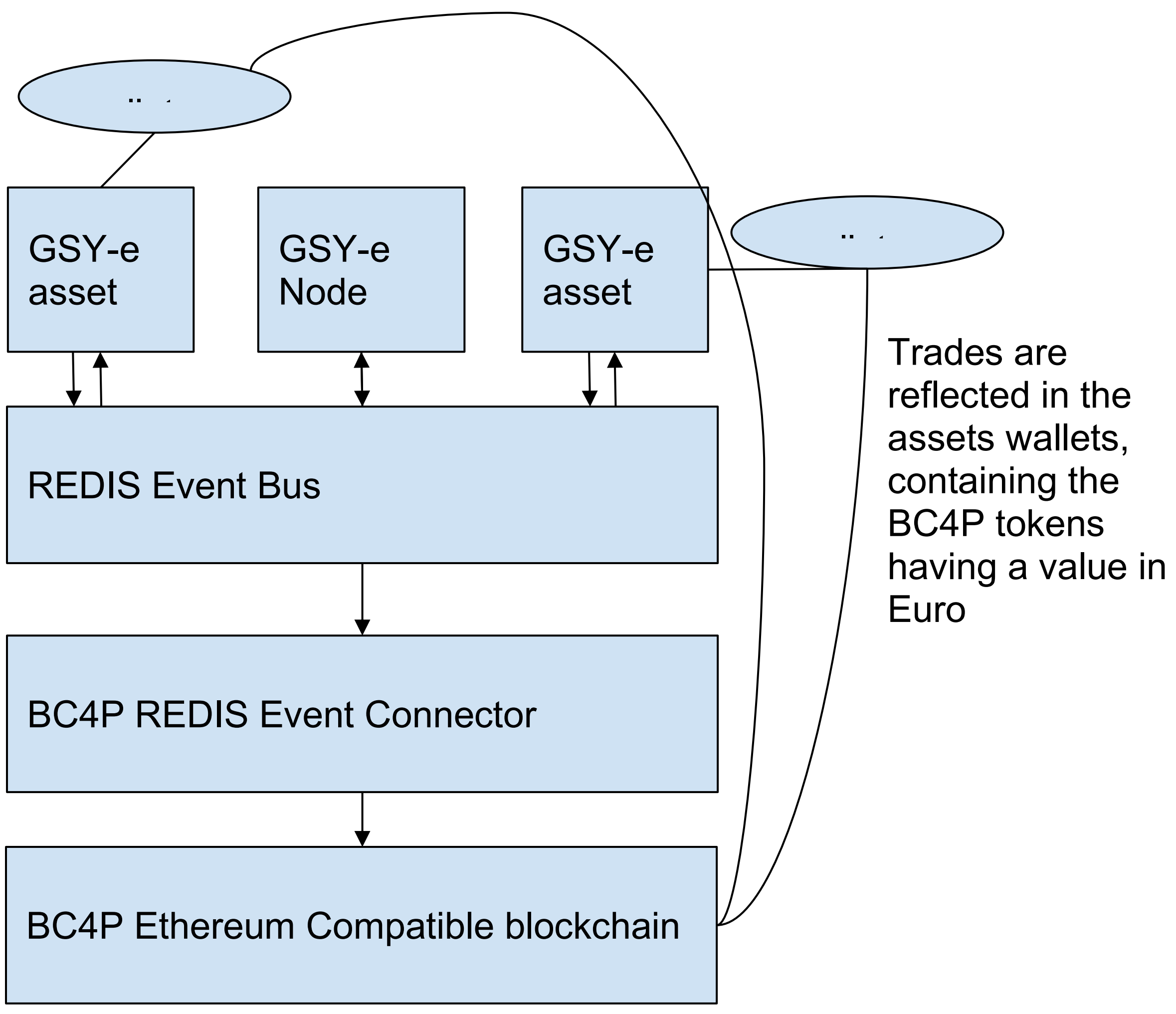

GSY-e asset
GSY-e Node
GSY-e asset
REDIS Event Bus
BC4P REDIS Event Connector
BC4P Ethereum Compatible blockchain
Trades are reflected in the assets wallets, containing the BC4P tokens having a value in Euro

Figure 4: Overview of the Integration between Smart Contract trades, redis data store, Ethereum Blockchain and GSY-e simulation

Smart Contracts for BC4P

Given that the events happening in GSY-e concerning energy trades are transformed into trade events in the blockchain through the BCP4 REDIS Event Connector, a mechanism is now needed to transform these into blockchain events. For this purpose, in BC4P, we use "Solidity **smart contracts**". Solidity is a statically typed programming language that runs on Ethereum. It is a tool for creating machine-level code and compiling it within Ethereum Virtual Machine. In addition, it has a rich ecosystem of tools, libraries, and frameworks that can make developing and deploying contracts more effortless. Smart Contracts are a way to allow **decentralized** networks to execute code securely without any central authority[10]. They are contractual clauses that implement and

[10] *Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. 2016. Making Smart Contracts Smarter. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16). Association for Computing Machinery, New York, NY, USA, 254–269. https://doi.org/10.1145/2976749.2978309*

enforce themselves through code. Smart contracts will enable the bid, offers, and trading of energy-based conditions.

Energy Market Module

The energy market module consists of the BC4P package[11], a package created for facilitating the use of the energy market smart contracts and exposing their function to a python client.

The energy market module uses the BC4P REDIS Event Connector, which listens for events emitted by the market simulation and performs the corresponding smart contract operations using the b4p package.

There are three event types that the BC4P REDIS Event Connector listens to

- Asset Registration
- Market Command
- Market Event

Asset registration tells the BC4P REDIS Event Connector that the asset API registered a new asset; the event includes information about the asset being created, such as its id and name. Once such an event has been captured, the Redis connector deploys a new producing asset smart contract and a consuming asset smart contract.

Market commands include the creation of bids, offers, and trades being executed. These three events will cause the BC4P REDIS Event Connector to trigger the corresponding functions within the smart contracts.

Market events are emitted every market tick and include summary information regarding the trading activity during the market cycle and the current state of the various assets and market areas. Such information can inform smart contracts of the market structure and grid fees.

The figure below shows the main components of the BC4P package. In the rest of this section, we present each of the components, including the tokens exchanged.

[11] https://github.com/bc4p/b4p-contracts

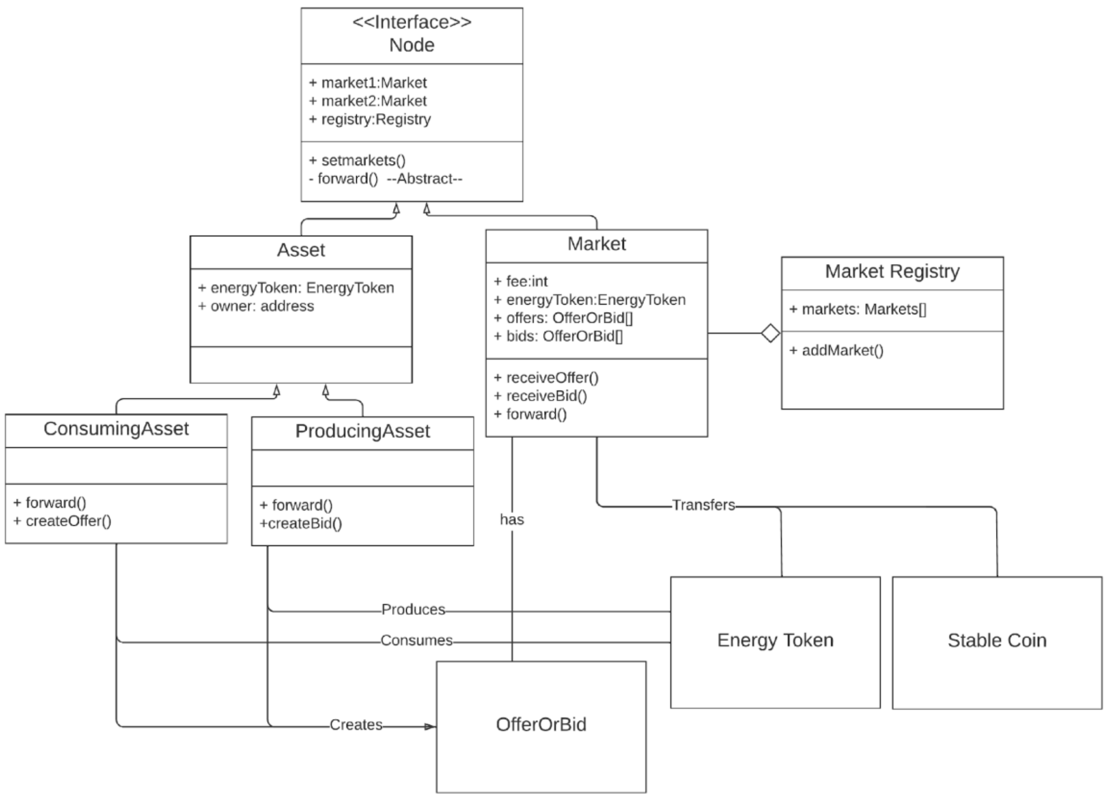


Figure 5: Entity-Relationship diagram of the involved classes

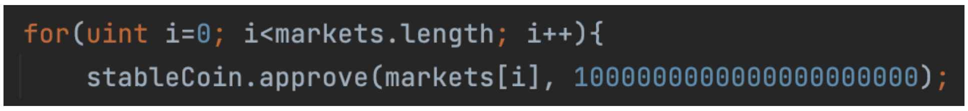
ERC20, IERC20, and ERC1155 tokens in BC4P

To manage the exchange of value in the blockchain, we use ERC-20 tokens and IERC20 stable coins specifications.

ERC-20 is a defined token standard that describes how a fungible token can be held and transferred between accounts. The ERC20 interface exists to standardize the implementation of tokens to promote interoperability between different blockchain applications. In our case, we need to have an ERC20 token that works as a utility token for the purpose of using the smart contracts, as each transaction consumes part of these tokens. Being an Ethereum-compatible blockchain, such a token has the same denomination as the Ethereum token in our private blockchain for BC4P.
In addition to the utility token needed to run the smart contracts' operations, we also need a stablecoin that would allow us to exchange value between prosumers.

Specifically, an IERC20 is a specific implementation of an ERC20 token where the token's value is pegged to the value of some other asset (most commonly a fiat currency such as the euro or the dollar). An IERC20 stablecoin can be useful as its value is relatively stable in price. In BC4P, we use a mock stablecoin pegged to the euro for the purpose of exemplifying how the money transfer would take place in BC4P. Finally, an ERC1155 token is an extension of the ERC20 standard. The main difference is that whereas the ERC20 tokens are completely fungible, i.e., they are all interchangeable, with the ERC1155 standard, it is possible to implement different types of tokens with different attributes. This result can be useful for our use-case in implementing the energy token as it is possible to distinguish between different energy tokens depending on where the energy was created and how. For the moment, we considered this at the level of the architecture and implementation, but in the current simulation, we do not distinguish in the type of energy; as part of future work, we will consider extensions to this model that may therefore allow us to prefer energy generated in a green way to energy generated with fossil fuels when performing a transaction.

The energy token contract and the EUR token contract are two implementations of the IERC20 stable coin and the ERC1155 protocols, respectively. The energy Token contract traces the energy being traded between market entities. Each producer has the ability to produce energy tokens; market nodes can receive these tokens and send them to energy consumers, which will destroy them when the energy is consumed.
In the future, the creation and destruction of these tokens will be directly connected to the readings of smart meters. The stablecoin is used to transact monetary value between market participants. The token is produced when purchased and pegged 1:1 with the euro. The market node takes responsibility for transferring these tokes from producers to consumers when a price is found. The markets are given an allowance to transfer tokens.



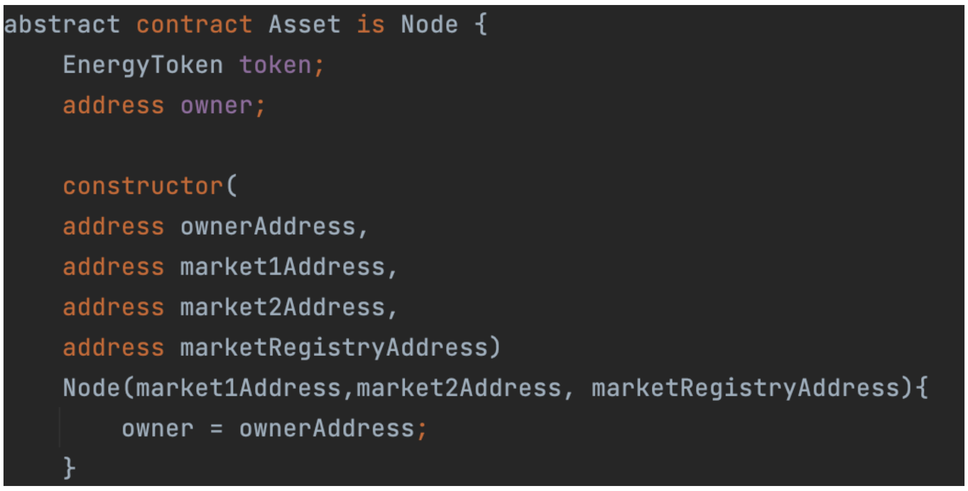
```
for(uint i=0; i<markets.length; i++){
    stableCoin.approve(markets[i], 1000000000000000000000);
```

Node

A node is an abstract representation of a market participant and is the base class from which other specific market participants extend their functionalities. Any class that extends the node class has to specify two market connections and forward functions. The market connections represent the market nodes that receive or send bids and offers, and the specific details of how they work depend on the node. Additionally, each node specifies a market registry that is used to track all nodes being created.

Asset

An asset is the base class containing the common functionalities of consumer and producer assets. Each asset has an owner that is specified as a public address. Creating bids and offers are commands that are only available to the owner of the asset. Finally, each asset specifies a trading token it uses to trade energy; every asset should trade with the same token.
An asset only uses one market connection as it cannot receive bids/offers but only send them.

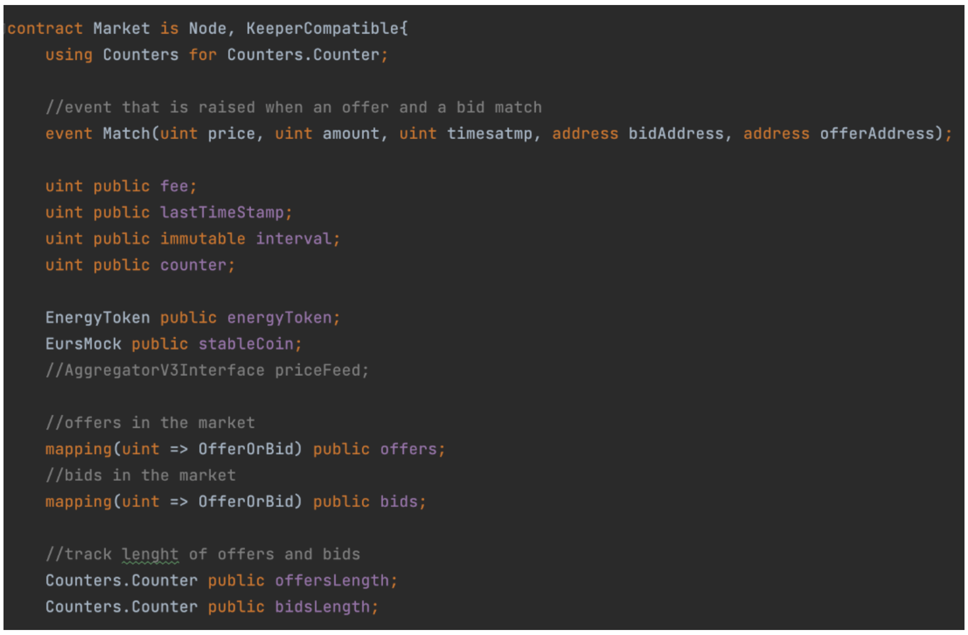


```
abstract contract Asset is Node {
    EnergyToken token;
    address owner;

    constructor(
    address ownerAddress,
    address market1Address,
    address market2Address,
    address marketRegistryAddress)
    Node(market1Address,market2Address, marketRegistryAddress){
        owner = ownerAddress;
    }
```

Market

A Market node represents a community that can receive and forward bids/offers. A market is responsible for holding the sets of all the bids and offers and the fee for that specific market.



```
contract Market is Node, KeeperCompatible{
    using Counters for Counters.Counter;

    //event that is raised when an offer and a bid match
    event Match(uint price, uint amount, uint timesatmp, address bidAddress, address offerAddress);

    uint public fee;
    uint public lastTimeStamp;
    uint public immutable interval;
    uint public counter;

    EnergyToken public energyToken;
    EursMock public stableCoin;
    //AggregatorV3Interface priceFeed;

    //offers in the market
    mapping(uint => OfferOrBid) public offers;
    //bids in the market
    mapping(uint => OfferOrBid) public bids;

    //track lenght of offers and bids
    Counters.Counter public offersLength;
    Counters.Counter public bidsLength;
```

It is essential to know that a “Market,” in this case, can be any market node responsible for holding and forwarding bids and offers; this includes community aggregators, grid operators, and Distribution System Operators.

When a Market receives an Offer, it is matched against all of the bids that are stored in that market. If there is a match, a new transaction is created.

- The offer’s owner receives stablecoins equal to the agreed energy price times the energy quantity from the bid’s owner. This is the transaction representing the purchase of the energy.

```
bool transferred_1 = stableCoin.transferFrom(bid._address, offer._address, bid.price - fee);
```

- The bid's owner receives energy tokens equal to the amount of energy purchased. This represents the energy being transferred to the bid's owner for traceability and accountability purposes.

```
energyToken.transferFrom(offer._address, bid._address, bid.amount);
```

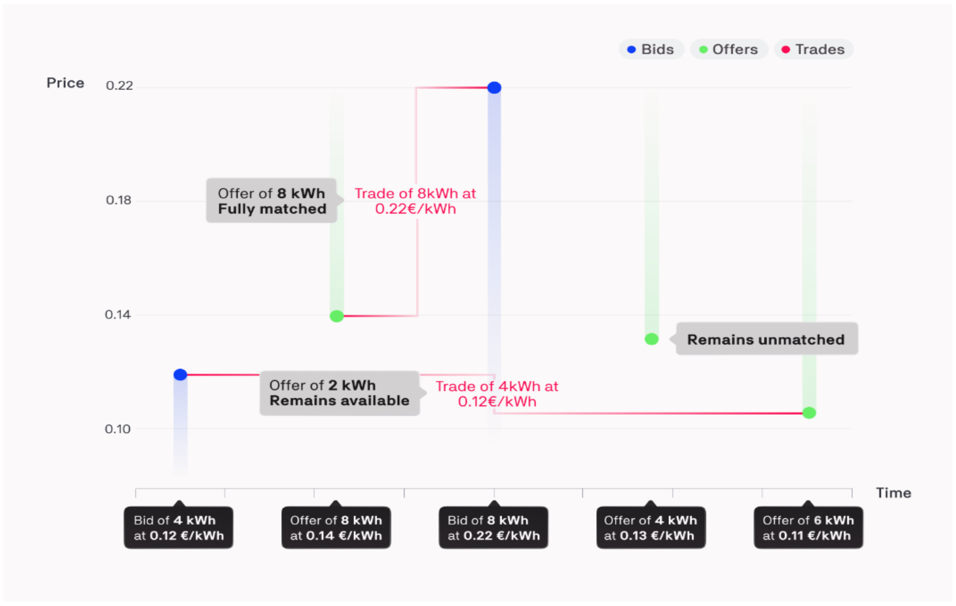
- The market's owner receives stablecoins equal to the fee price. This is the Market fee that the Market owner sets.

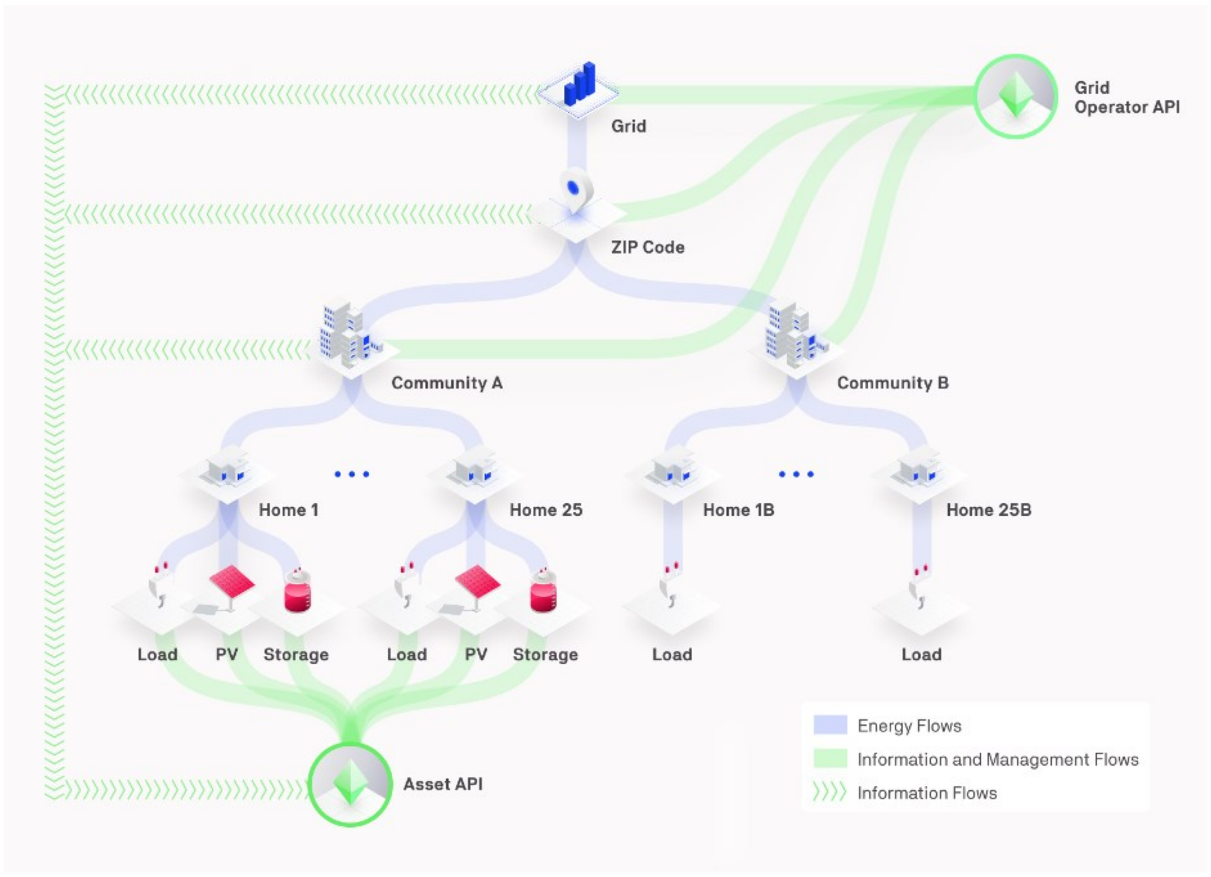
```
bool transferred = stableCoin.transferFrom(bid._address, address(this), fee);
```

Current functionality of the free transaction based on the offer acceptance. The fee transaction happens when they find a match, as it is a fee for forwarding the bid or offer to the market.

The Market contract allows for two mechanisms for matching bids and offers and selecting an agreed price: the two-sided pay-as-bid market and the One-Sided pay-as-offer mechanism.

With the two-sided pay-as-bid mechanisms, a bid and an offer match when the bid's price is higher than the offer's price, in which case the offeror will be paid the bid's price. This way, the offeror is always guaranteed a price equal to or lower than what it offered, whereas the bidder is guaranteed the exact price it is bidding.





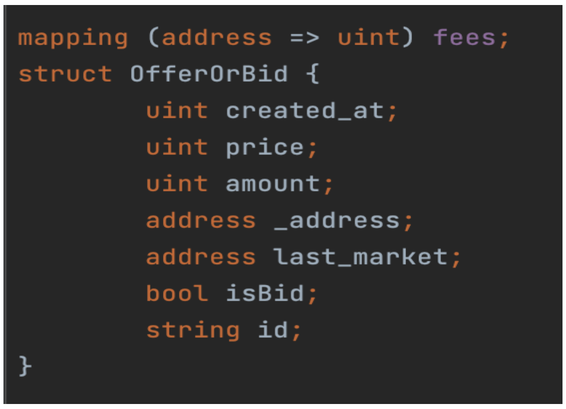
With the one-sided pay-as-offer mechanism, producers can select a bid and purchase it directly without giving an offer to the market.

Market Registry

A Market registry represents market registration if the market does not exist and checks for the existing one.

OfferOrBid

OfferOrBid is a “struct” that holds information about an offer or a bid. In solidity, a struct is a data type composed of multiple fields, and it is usually used to represent a complex object. The OfferOrBid struct is defined in the Node class and is, therefore, available for all Node descendants.



```
mapping (address => uint) fees;
struct OfferOrBid {
        uint created_at;
        uint price;
        uint amount;
        address _address;
        address last_market;
        bool isBid;
        string id;
}
```

Workflow of the blockchain from Gsy-e events to Blockchain transaction

In this section, we explain the working of the gsy-e with blockchain and the actions taken.
Our Approach is to use the REDIS server to get all the events and actions associated with the gsy-e framework and sdk-gsy-e. The energy market module consists of a BC4P package and event listener using a REDIS connector. The energy market module provides the
moderator between the blockchain and gsy-e. Once a market event has been detected, the Redis connector triggers a function that performs the required blockchain operation.

We can divide Redis events into three categories:

- market events
- registration events
- commands events

Market events

Market events are triggered periodically when the market simulation reaches a tick or starts a new market slot. These two events are helpful as they work as a market clock for the whole system. The market slot event communicates data related to the current state of the market, such as the structure of the market nodes, their fees, and the trades executed during the market's previous market cycle. The Redis connector uses this event to initiate and update market contracts.

Registration events

Registration events are triggered when the asset-API registers a new asset. The Redis connector uses such events to deploy the ConsumingAsset and ProducingAsset Contracts.

Command events

Command events are sent by the asset API when it gives the market a new bid or a new offer. The events specify the asset id, the price of the bid/offer, and the market where the bid/offer is sent. The Redis connector can use these events to give bids/offers to the appropriate market contract or to trigger a direct purchase.

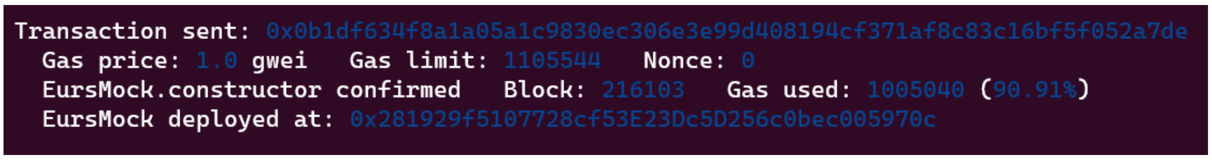
Simulation Example

We run a full simulation example to illustrate the functionalities of the system and the integration of smart contracts.
For this simulation, we illustrate the trading between two entities: an energy-producing entity and an energy-consuming entity that exchange money by trading energy in a market.

Setup

Before starting the simulation, the blockchain must be set up with the required smart contracts and accounts. The setup is automatically handled when the Redis connector script is run before anything else. During the setup, three smart contracts are deployed:

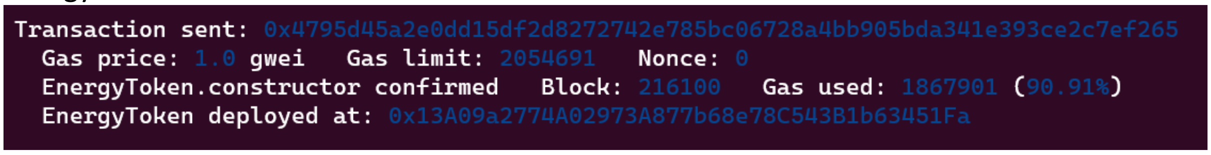
- EursToken



```
Transaction sent: 0x0b1df634f8a1a05a1c9830ec306e3e99d408194cf371af8c83c16bf5f052a7de
  Gas price: 1.0 gwei   Gas limit: 1105544   Nonce: 0
  EursMock.constructor confirmed   Block: 216103   Gas used: 1005040 (90.91%)
  EursMock deployed at: 0x281929f5107728cf53E23Dc5D256c0bec005970c
```

blockchain explorer link

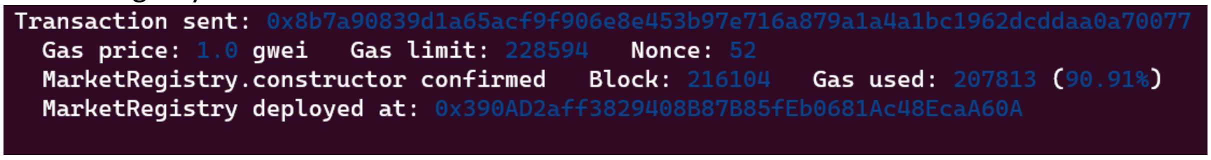
- EnergyToken



```
Transaction sent: 0x4795d45a2e0dd15df2d8272742e785bc06728a4bb905bda341e393ce2c7ef265
  Gas price: 1.0 gwei   Gas limit: 2054691   Nonce: 0
  EnergyToken.constructor confirmed   Block: 216100   Gas used: 1867901 (90.91%)
  EnergyToken deployed at: 0x13A09a2774A02973A877b68e78C543B1b63451Fa
```

blockchain explorer link

- MarketRegistry

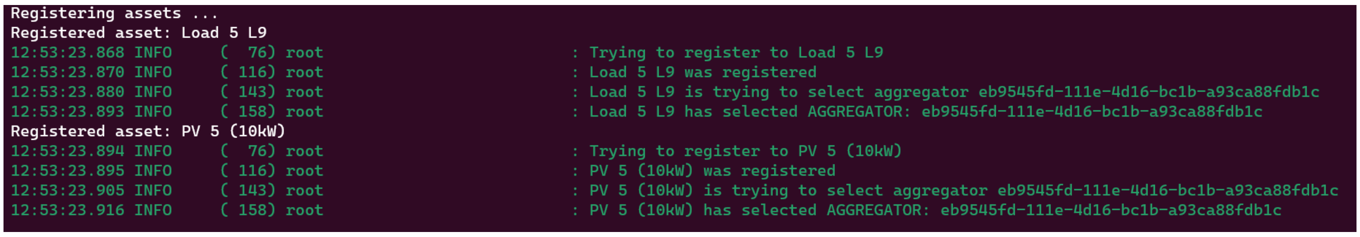


```
Transaction sent: 0x8b7a90839d1a65acf9f906e8e453b97e716a879a1a4a1bc1962dcddaa0a70077
  Gas price: 1.0 gwei   Gas limit: 228594   Nonce: 52
  MarketRegistry.constructor confirmed   Block: 216104   Gas used: 207813 (90.91%)
  MarketRegistry deployed at: 0x390AD2aff3829408B87B85fEb0681Ac48EcaA60A
```

blockchain explorer link.

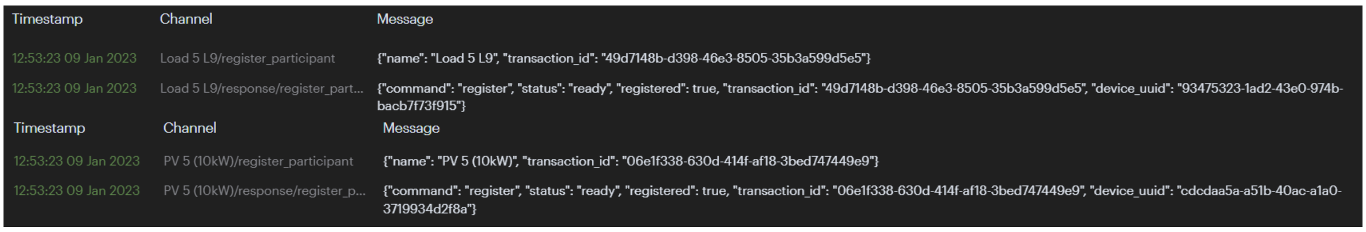
Asset Registration

Once the setup is completed and the simulation has started, the guy-e-SDK registers two assets. The first asset is called "Load 5 L9" and represents an energy-consuming asset that will publish bids for energy later. The second asset, called "PV 5 (10kw)," will produce energy and offer it to the market.
The registration happens inside the guy-e-SDK client, which its output can confirm:



```
Registering assets ...
Registered asset: Load 5 L9
12:53:23.868 INFO     (  76) root                              : Trying to register to Load 5 L9
12:53:23.870 INFO     ( 116) root                              : Load 5 L9 was registered
12:53:23.880 INFO     ( 143) root                              : Load 5 L9 is trying to select aggregator eb9545fd-111e-4d16-bc1b-a93ca88fdb1c
12:53:23.893 INFO     ( 158) root                              : Load 5 L9 has selected AGGREGATOR: eb9545fd-111e-4d16-bc1b-a93ca88fdb1c
Registered asset: PV 5 (10kW)
12:53:23.894 INFO     (  76) root                              : Trying to register to PV 5 (10kW)
12:53:23.895 INFO     ( 116) root                              : PV 5 (10kW) was registered
12:53:23.905 INFO     ( 143) root                              : PV 5 (10kW) is trying to select aggregator eb9545fd-111e-4d16-bc1b-a93ca88fdb1c
12:53:23.916 INFO     ( 158) root                              : PV 5 (10kW) has selected AGGREGATOR: eb9545fd-111e-4d16-bc1b-a93ca88fdb1c
```

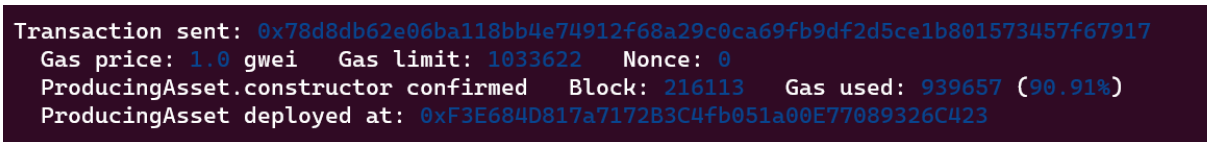
The registration also produces a Redis event as seen from a Redis GUI explorer:



Timestamp	Channel	Message
12:53:23 09 Jan 2023	Load 5 L9/register_participant	{"name": "Load 5 L9", "transaction_id": "49d7148b-d398-46e3-8505-35b3a599d5e5"}
12:53:23 09 Jan 2023	Load 5 L9/response/register_part...	{"command": "register", "status": "ready", "registered": true, "transaction_id": "49d7148b-d398-46e3-8505-35b3a599d5e5", "device_uuid": "93475323-1ad2-43e0-974b-bacb7f73f915"}

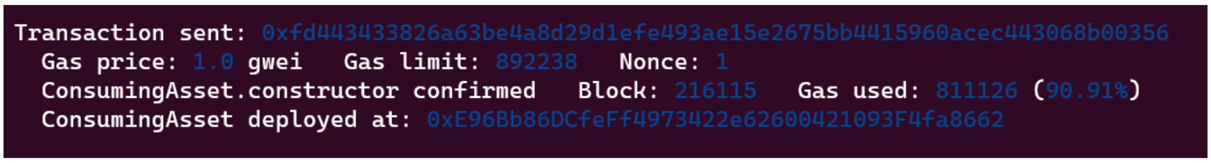
Timestamp	Channel	Message
12:53:23 09 Jan 2023	PV 5 (10kW)/register_participant	{"name": "PV 5 (10kW)", "transaction_id": "06e1f338-630d-414f-af18-3bed747449e9"}
12:53:23 09 Jan 2023	PV 5 (10kW)/response/register_p...	{"command": "register", "status": "ready", "registered": true, "transaction_id": "06e1f338-630d-414f-af18-3bed747449e9", "device_uuid": "cdcdaa5a-a51b-40ac-a1a0-3719934d2f8a"}

For every asset that is registered, two events are emitted: The first event is the actual registration event, and it signifies that an asset is being registered. The second event is the response, which describes whether the registration was successful. The Redis connector captures this second event, which creates and deploys new smart contracts to the blockchain.



```
Transaction sent: 0x78d8db62e06ba118bb4e74912f68a29c0ca69fb9df2d5ce1b801573457f67917
  Gas price: 1.0 gwei   Gas limit: 1033622   Nonce: 0
  ProducingAsset.constructor confirmed   Block: 216113   Gas used: 939657 (90.91%)
  ProducingAsset deployed at: 0xF3E684D817a7172B3C4fb051a00E77089326C423
```

Blockchain explorer link



```
Transaction sent: 0xfd443433826a63be4a8d29d1efe493ae15e2675bb4415960acec443068b00356
  Gas price: 1.0 gwei   Gas limit: 892238   Nonce: 1
  ConsumingAsset.constructor confirmed   Block: 216115   Gas used: 811126 (90.91%)
  ConsumingAsset deployed at: 0xE96Bb86DCfeFf4973422e62600421093F4fa8662
```

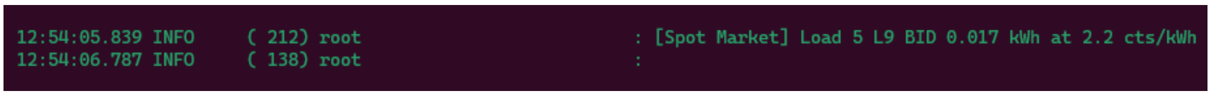
blockchain explorer link

Since this is a simulation and a proof of concept, these two smart contracts are funded by a faucet that transfers 50 EursTokens to their balance and makes trading possible.

Bidding and Trading

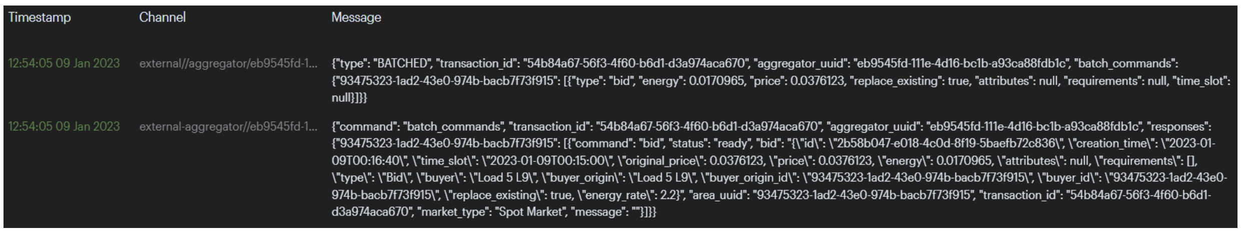
After creating such assets, the simulation proceeds, and the Load 5 L9 is required to purchase 0.017 KWh. Its strategy is to start bidding low and increase its bid until a producing asset is willing to sell it at that price.

Here's an example of one of the first bids placed by the consuming asset seen from the guy-e-SDK logs. Load 5 L9 In this case, tries to purchase the 0.0170965 kwh it needs at 2.2 cents/kwh. (The logs cut the precision of the energy to 3 significant digits at 0.17 kwh)



```
12:54:05.839 INFO    ( 212) root                          : [Spot Market] Load 5 L9 BID 0.017 kWh at 2.2 cts/kWh
12:54:06.787 INFO    ( 138) root                          :
```

This bid, like every bid, emits Redis events containing the following information:

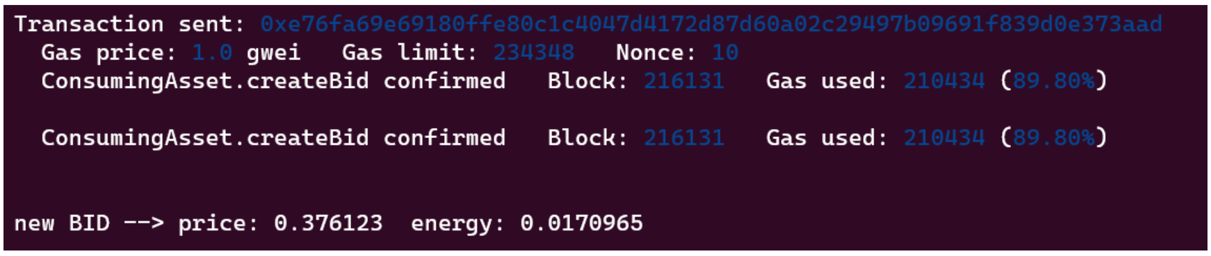


Timestamp	Channel	Message
12:54:05 09 Jan 2023	external//aggregator/eb9545fd-1...	{"type": "BATCHED", "transaction_id": "54b84a67-56f3-4f60-b6d1-d3a974aca670", "aggregator_uuid": "eb9545fd-111e-4d16-bc1b-a93ca88fdb1c", "batch_commands": {"93475323-1ad2-43e0-974b-bacb7f73f915": [{"type": "bid", "energy": 0.0170965, "price": 0.0376123, "replace_existing": true, "attributes": null, "requirements": null, "time_slot": null}]}}
12:54:05 09 Jan 2023	external-aggregator//eb9545fd-1...	{"command": "batch_commands", "transaction_id": "54b84a67-56f3-4f60-b6d1-d3a974aca670", "aggregator_uuid": "eb9545fd-111e-4d16-bc1b-a93ca88fdb1c", "responses": {"93475323-1ad2-43e0-974b-bacb7f73f915": [{"command": "bid", "status": "ready", "bid": "{\"id\": \"2b58b047-e018-4c0d-8f19-5baefb72c836\", \"creation_time\": \"2023-01-09T00:16:40\", \"time_slot\": \"2023-01-09T00:15:00\", \"original_price\": 0.0376123, \"price\": 0.0376123, \"energy\": 0.0170965, \"attributes\": null, \"requirements\": [], \"type\": \"Bid\", \"buyer\": \"Load 5 L9\", \"buyer_origin\": \"Load 5 L9\", \"buyer_origin_id\": \"93475323-1ad2-43e0-974b-bacb7f73f915\", \"buyer_id\": \"93475323-1ad2-43e0-974b-bacb7f73f915\", \"replace_existing\": true, \"energy_rate\": 2.2}", "area_uuid": "93475323-1ad2-43e0-974b-bacb7f73f915", "transaction_id": "54b84a67-56f3-4f60-b6d1-d3a974aca670", "market_type": "Spot Market", "message": ""}]}}

The two events follow the same mechanism. The first event signals a bid was emitted, whereas the second event signals that the bid was successfully published.

The event contains a lot of information about the bid but, most importantly, its price, which in this case was 0.0170965*2.2 = 0.0376123.

At this point, the Redis connector captures the second event and can publish a bid by calling the createBid function of the ConsumingAsset smart contracts. The appropriate ConsumingAsset smart contracThreeevent contains the bidder id; during the setup, the b4p package links to the corresponding smart contract. The bid price and energy amount are also contained in the event.



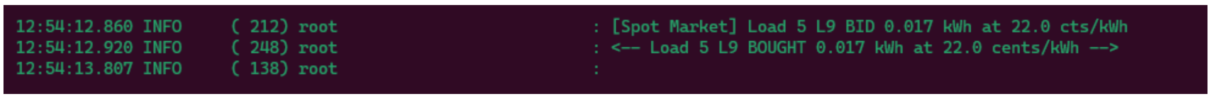
```
Transaction sent: 0xe76fa69e69180ffe80c1c4047d4172d87d60a02c29497b09691f839d0e373aad
  Gas price: 1.0 gwei   Gas limit: 234348   Nonce: 10
  ConsumingAsset.createBid confirmed   Block: 216131   Gas used: 210434 (89.80%)

  ConsumingAsset.createBid confirmed   Block: 216131   Gas used: 210434 (89.80%)


new BID --> price: 0.376123  energy: 0.0170965
```

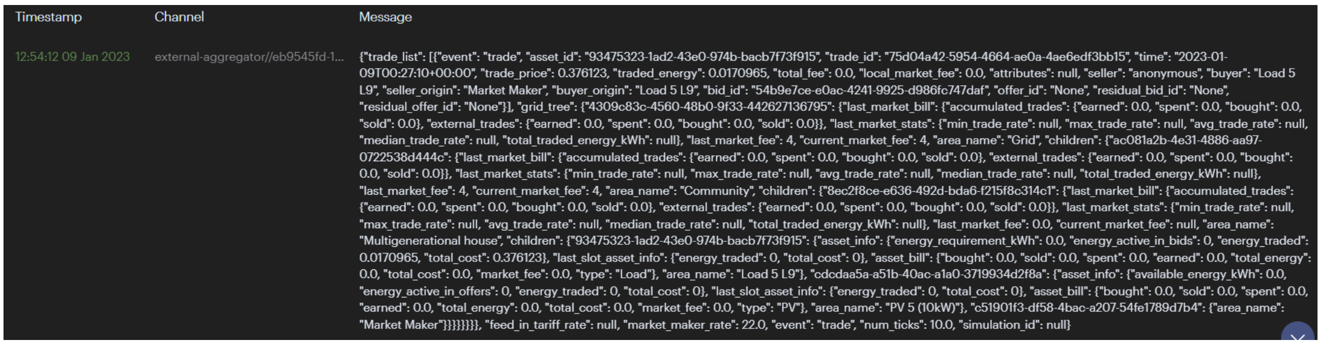
blockchain explorer link,

In this case, no one offers energy at that price. As the simulation continues, Load 5 L9 increases the bid price until finally, it posts a bid at 22 cents/kwh, and the producing asset PV (10kwh) can accept the bid and sell the energy to Load 5 L9.



```
12:54:12.860 INFO     ( 212) root                        : [Spot Market] Load 5 L9 BID 0.017 kWh at 22.0 cts/kWh
12:54:12.920 INFO     ( 248) root                        : <-- Load 5 L9 BOUGHT 0.017 kWh at 22.0 cents/kWh -->
12:54:13.807 INFO     ( 138) root                        :
```

In this case, the producing asset does not post an offer to the market but accepts the bid directly. The main difference here is that these two actions generate three events. The first two are for the creation of the bid, as sent before. A third event is a market event where information about the new purchase can be found.



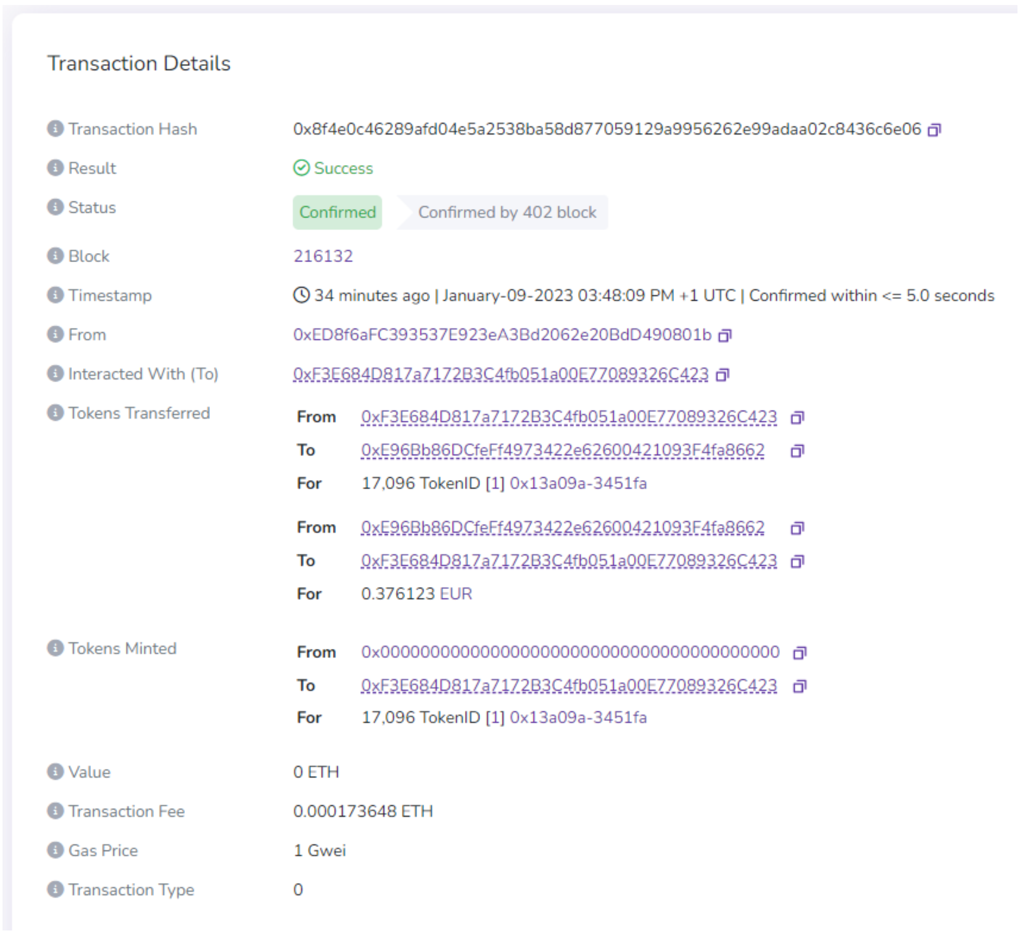
```
Timestamp              Channel                        Message

12:54:12 09 Jan 2023   external-aggregator//eb9545fd-1...
```

{"trade_list": [{"event": "trade", "asset_id": "93475323-1ad2-43e0-974b-bacb7f73f915", "trade_id": "75d04a42-5954-4664-ae0a-4ae6edf3bb15", "time": "2023-01-09T00:27:10+00:00", "trade_price": 0.376123, "traded_energy": 0.0170965, "total_fee": 0.0, "local_market_fee": 0.0, "attributes": null, "seller": "anonymous", "buyer": "Load 5 L9", "seller_origin": "Market Maker", "buyer_origin": "Load 5 L9", "bid_id": "54b9e7ce-e0ac-4241-9925-d986fc747daf", "offer_id": "None", "residual_bid_id": "None", "residual_offer_id": "None"}], "grid_tree": {"4309c83c-4560-48b0-9f33-442627136795": {"last_market_bill": {"accumulated_trades": {"earned": 0.0, "spent": 0.0, "bought": 0.0, "sold": 0.0}, "external_trades": {"earned": 0.0, "spent": 0.0, "bought": 0.0, "sold": 0.0}}, "last_market_stats": {"min_trade_rate": null, "max_trade_rate": null, "avg_trade_rate": null, "median_trade_rate": null, "total_traded_energy_kWh": null}, "last_market_fee": 4, "current_market_fee": 4, "area_name": "Grid", "children": {"ac081a2b-4e31-4886-aa97-0722538d444c": {"last_market_bill": {"accumulated_trades": {"earned": 0.0, "spent": 0.0, "bought": 0.0, "sold": 0.0}, "external_trades": {"earned": 0.0, "spent": 0.0, "bought": 0.0, "sold": 0.0}}, "last_market_stats": {"min_trade_rate": null, "max_trade_rate": null, "avg_trade_rate": null, "median_trade_rate": null, "total_traded_energy_kWh": null}, "last_market_fee": 4, "current_market_fee": 4, "area_name": "Community", "children": {"8ec2f8ce-e636-492d-bda6-f215f8c314c1": {"last_market_bill": {"accumulated_trades": {"earned": 0.0, "spent": 0.0, "bought": 0.0, "sold": 0.0}, "external_trades": {"earned": 0.0, "spent": 0.0, "bought": 0.0, "sold": 0.0}}, "last_market_stats": {"min_trade_rate": null, "max_trade_rate": null, "avg_trade_rate": null, "median_trade_rate": null, "total_traded_energy_kWh": null}, "last_market_fee": 0.0, "current_market_fee": null, "area_name": "Multigenerational house", "children": {"93475323-1ad2-43e0-974b-bacb7f73f915": {"asset_info": {"energy_requirement_kWh": 0.0, "energy_active_in_bids": 0, "energy_traded": 0.0170965, "total_cost": 0.376123}, "last_slot_asset_info": {"energy_traded": 0, "total_cost": 0}, "asset_bill": {"bought": 0.0, "sold": 0.0, "spent": 0.0, "earned": 0.0, "total_energy": 0.0, "total_cost": 0.0, "market_fee": 0.0, "type": "Load"}, "area_name": "Load 5 L9"}, "cdcdaa5a-a51b-40ac-a1a0-3719934d2f8a": {"asset_info": {"available_energy_kWh": 0.0, "energy_active_in_offers": 0, "energy_traded": 0, "total_cost": 0}, "last_slot_asset_info": {"energy_traded": 0, "total_cost": 0}, "asset_bill": {"bought": 0.0, "sold": 0.0, "spent": 0.0, "earned": 0.0, "total_energy": 0.0, "total_cost": 0.0, "market_fee": 0.0, "type": "PV"}, "area_name": "PV 5 (10kW)"}, "c51901f3-df58-4bac-a207-54fe1789d7b4": {"area_name": "Market Maker"}}}}}}}}, "feed_in_tariff_rate": null, "market_maker_rate": 22.0, "event": "trade", "num_ticks": 10.0, "simulation_id": null}

The Redis connector captures this market event, which triggers the acceptBid function of the ProducingAssert smart contract. When this function is executed, the producing asset creates the energy tokens that are then transferred to the consuming asset smart contract representing the energy purchased.
The Eurs token is also transferred from the consuming asset to the producing asset.

This transaction can be seen from a blockchain explorer:



Transaction Details

Transaction Hash	0x8f4e0c46289afd04e5a2538ba58d877059129a9956262e99adaa02c8436c6e06	
Result	Success	
Status	Confirmed	Confirmed by 402 block
Block	216132	
Timestamp	34 minutes ago | January-09-2023 03:48:09 PM +1 UTC | Confirmed within <= 5.0 seconds	
From	0xED8f6aFC393537E923eA3Bd2062e20BdD490801b	
Interacted With (To)	0xF3E684D817a7172B3C4fb051a00E77089326C423	
Tokens Transferred	**From**	0xF3E684D817a7172B3C4fb051a00E77089326C423
	To	0xE96Bb86DCfeFf4973422e62600421093F4fa8662
	For	17,096 TokenID [1] 0x13a09a-3451fa
	From	0xE96Bb86DCfeFf4973422e62600421093F4fa8662
	To	0xF3E684D817a7172B3C4fb051a00E77089326C423
	For	0.376123 EUR
Tokens Minted	**From**	0x0000000000000000000000000000000000000000
	To	0xF3E684D817a7172B3C4fb051a00E77089326C423
	For	17,096 TokenID [1] 0x13a09a-3451fa
Value	0 ETH	
Transaction Fee	0.000173648 ETH	
Gas Price	1 Gwei	
Transaction Type	0	

blockchain explorer link

Conclusion and Future Work

This report investigates blockchain technology's implementation in the energy trading realm. Specifically, it seeks to explore how decentralized energy trading platforms utilizing smart contracts can be facilitated by the blockchain in monetary transactions between producers and consumers without third-party authorization and by simply calling functions in smart contracts. Implementing such systems holds the potential to revolutionize the energy market by providing a secure and transparent means of conducting trade.

On the one hand, a limitation of the proposed infrastructure can be seen in the fact that we necessitate an exchange like GSY-e to run the necessary operations concerning the management of bids, offers, and fees, making the interaction partially centralized. The proposed infrastructure is still P2P regarding how the bids and offers happen, allowing any asset to trade with any other asset in the network. This approach has the advantage of separating engineering concerns from monetary concerns.

On the other hand, in the case in which a completely decentralized solution is wished, as an energy exchange must operate in real-time to be able to keep the balance of the network, a consequence for a solution that is built solely on the blockchain would be that the blockchain would need to be able to handle many events in parallel, which is a difficult feature to implement with respect to smart contracts.

In addition to considering the possibility of moving part of the energy grid logic to the blockchain, future work may imply the implementation of future predictions of the production or consumption of energy in the form of tokens to be stored in the blockchain and traded by means of smart contracts. The actual benefit of such an approach would be to preallocate energy and create a different channel of revenue for the producer and saving for the consumer.
Given that prediction algorithms are getting more accurate, the risk associated with a wrong prediction may become orders of magnitude less relevant than the benefit of preallocating energy to sure trades.

This deliverable is naturally connected to the WP3 deliverable about digital identities. Such a deliverable will specify how assets can be created with a wallet to trade in the BC4P blockchain solution. In this deliverable, we assumed that all the assets are created with a pre-defined account for each producing or consuming asset in the simulation.