18th International Conference on Renewable Energies and Power Quality (ICREPQ'20)
Granada (Spain), 1st to 2nd April 2020
Renewable Energy and Power Quality Journal (RE&PQJ)
ISSN 2172-038 X, Volume No.18, June 2020

A novel emulation concept for the test of smart contracts in the energy economy

A. Notholt[1], D. Coll-Mayor [1]

[1] School of Engineering
Reutlingen University
Alteburgstr. 150 72762 Reutlingen (Germany)
Phone/Fax number: +49 (7121) 271-7139,
e-mail: Antonio.Notholt@Reutlingen-University.de, Debora.Coll-Mayor@Reutlingen-University.de

Abstract
This paper presents a novel emulation concept for the test of smart contracts and Distributed Ledger Technologies (DLT) in distribute control or energy economy tasks and use cases. The concept uses state of the art behavioral modeling tools such as Matlab Simulink but presents a possible way to solve the shortfall of Simulink in communicating to DLT-Nodes directly. This is solved through a middleware solution. After this, an example used in verifying the test bed is presented and the target demonstration object is described. Finally, the possible expansion of the system is discussed and presented.

Keywords

Distributed markets, distributed ledger technologies, blockchain, distributed energy resources, off-grid systems.

1. Introduction

The European Commission has proposed in its Clean Energy Package[1] a new role for the end customer. Basically, the end costumer changes from a passive player in the energy economy to assuming a key role. In order to facilitate that, the energy economy has to support small-scale interactions, for example by using distributed IT structures.

In this changing panorama, DLT (i.e. Blockchain) offers an interesting solution by making very small transactions economically viable, enabling direct trading between market players. Blockchain removes the necessity of an intermediary to validate a transaction.

Smart contracts provide an interesting technological platform, in which not only data but also code is auditable by any participant and thus any user can exactly know what to expect. The largest smart-contract supporting DLT is the *Ethereum Blockchain*.

The promising ideas of using this technology must however be validated somehow. Simulation programs may emulate some latency incurred when submitting a transaction but cannot account for many other effects. For this, Reutlingen University has developed a simulation-based test infrastructure to emulate not only the electrical system but also the coupling between the Blockchain and the electrical system.

2. Requirements

A useful test and validation system shall comply with the following requirements:

- The different roles (producers, consumers, prosumers) should be implemented as runnable models, each having a defined behavior
- The electrical grid should be simulated at least as a power balancing model
- The communication infrastructure should resemble real conditions
- The Blockchain node architecture shall be comparable to the real application

3. Selected architecture

The architecture is divided into several subsystems: Hardware, Behavioral models, Network and Blockchain node testing. Figure 1 shows an overview of the developed architecture.

A. Hardware

In order to emulate real networking conditions, the models and nodes must be physically separated. For this a set of one-board computers (Raspberry Pi) is used. Each of the one-board computers will independently run the role model and a blockchain *light* node.

[1]https://ec.europa.eu/energy/en/topics/energy-strategy-and-energy-union/clean-energy-all-europeans

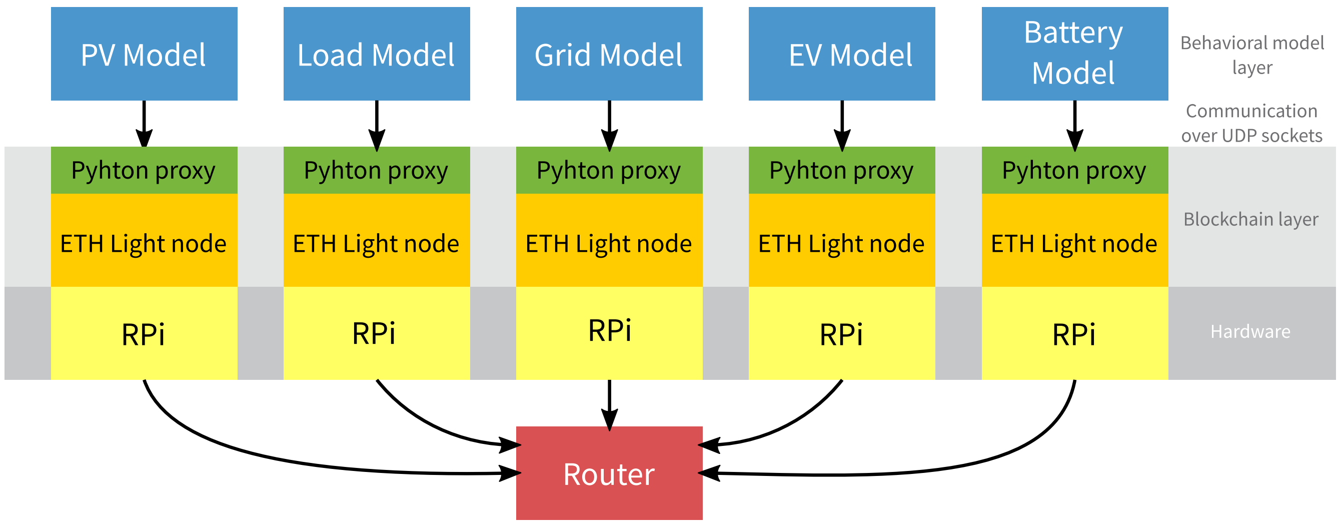

Fig. 1: Test bench architecture

B. *Behavioral models*

Due to its flexibility and widespread use, the software used for modeling actors, their roles and behaviors is Matlab Simulink. Since Matlab Simulink does not implement any communication protocol known to interact with an ethereum node, the Simulink model will interact via UDP packets through a proxy written in Python.

C. *Communication infrastructure*

The communication infrastructure will resemble an actual system by coupling all one-board computers through a router with the Internet. In this case a communication loss in any given node can be easily simulated.

D. *Node architecture*

The Raspberry PI hardware does not have the processor power to run a full node. However, they are capable of running light nodes in order to send, transactions and receive blocks. By running a distributed node network, the resemblance to real-world conditions is met. These nodes are connected to the Görli test network, which is one of the newer Proof-of-Authority test networks.

4. Implementation

In order to test the development test bench, a simple use case of energy sharing in a building was implemented. For this, only PV and load models were implemented. The use case allows a given load to *purchase* PV energy supplied by a neighboring PV system.

In this scenario, the load will offer to buy a specific amount of energy and pays in advance for it. If the PV produces the amount requested, the transferred funds equivalent to the produced energy are deposited in the PV system's wallet. If the amount of PV produced is larger than the requested, the PV does not get any compensation for the excess energy.

PV is able to read the requested energy and can independently react on such information, such as reducing PV production output.

A. *Smart contract*

Listing 1 below shows the source code of the smart contract implementing the described behavior. Since all state variables are declared public, any participant may know (read) the involved actors (through their addresses) as well as the current offer and demand figures. With this information a distributed control system may react (ether in the production or on the demand side) to match offer and demand.

```
pragma solidity ^0.4.24;

contract EtherControl {
  address public pvAddr;
  address public gridAddr;  // in this case the
      grid operator is the contract owner
  address public loadAddr;

  uint public gweiPerKwh_Pv;

  uint public loadReqWh;
  uint public pvOfferWh;

  constructor() public{
    gridAddr = msg.sender;  // Sets the owner of
      this contract
    gweiPerKwh_Pv = 10000;  // Assume 1Gwei = 1ct
  }

  function participatePv() public
  {
    require(pvAddr == 0 || pvAddr == msg.sender,
      "There is already one participant");
    pvAddr = msg.sender;
  }

  function offerPv(uint Wh) public
  {
    require(pvAddr == msg.sender, "Not entitled
      to do this");
    pvOfferWh = Wh;
  }


  function setProducedPV(uint Wh) public payable
      returns (bool success) {
```

```
    require(msg.sender==pvAddr, "You are not the
        right one");
    require(Wh <= loadReqWh, "Nobody needs so
        much energy");
    pvAddr.transfer(Wh * gweiPerKwh_Pv * 1000000)
        ;
    loadReqWh = loadReqWh - Wh;
    return true;
  }


  function participateLoad() public
  {
    require(loadAddr == 0 || loadAddr == msg.
        sender, "There is already one participant
        ");
    loadAddr = msg.sender;
  }

  function demandPvEnergy(uint Wh) public payable
      returns (bool success)
  {
    require(loadAddr == msg.sender, "Not entitled
         to do this");
    require(msg.value >= Wh*gweiPerKwh_Pv
        *1000000, "Please transfer enough ehter
        ...");
    loadReqWh = loadReqWh + Wh;
    return true;
  }
}
```

Listing 1: Implemented smart contract used in test case

B. *Middleware*

The middleware has as objective the transfer of information between the behavioral model and the ethereum node. This was written in python and used the library **web3**. The program would process any new UDP package into a trigger for a smart contract function. After the transaction has been cleared, the program sends the values of the contract's state variables back to the model.

Listing 2 shows an excerpt of the middleware software used for interacting between the UDP sochet data and the GETH client.

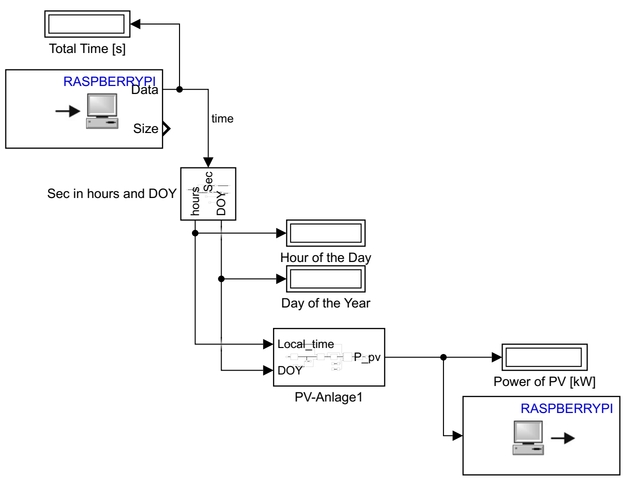

Fig. 2: Matlab Simulink model of the PV system (example

```
from web3 import Web3, HTTPProvider, IPCProvider
import time
import socket
import struct
import threading
from web3.middleware import geth_poa_middleware

# Address of published contract
CONTRACTADDRESS = 'Address of smart contract'
CONTRACTABI = 'ABI of SMart Contract'
# IP-Address of consumer
IP ='10.1.0.12'
# IP-Address of Simulink simulation
IP_SIMULINK = '127.0.0.1'

def producer_balance_thread(contract, socket):
  while True:
  #socket.sendto(get_sc_consumer_balance(contract
      ), (IP_SIMULINK, 20194))
  time.sleep(1)

def set_sc_consumer(contract, power,
    consumerAccount):
  contract.functions.demandPvEnergy(egy).transact
      ({'from': consumerAccount})

def main():
  # web3.IPCProvider for connecting to ipc socket
       based JSON-RPC servers. Path to geth.ipc
  web3 = Web3(IPCProvider('/home/ethereum/.
      ethereum/goerli/geth.ipc'))
  web3.middleware_onion.inject(
      geth_poa_middleware, layer=0)
  # get consumer account and unlock Account
      permanently
  consumerAccount = '0
      xaa17DC616D262A37a35089Ab59Cda44de8234b63'
  # check if account was successfully unlocked
  isUnlocked = web3.geth.personal.unlockAccount('
      acct', "passwd", 0)
  if isUnlocked == True:
    print('Account is unlocked')
  # setup variables of smart contract
  # to use functions use call() to manipulate the
       blockchain via function use transact
  contract = web3.eth.contract(address=web3.
      toChecksumAddress(CONTRACTADDRESS), abi=
      CONTRACTABI)
  # init udp receive connection for power
      parameter
  powerSocket = socket.socket(socket.AF_INET,
      socket.SOCK_DGRAM)
  addr_powerSocket= (IP, 20138)
  powerSocket.bind(addr_powerSocket)
  # setup udp send connection
  sendBalanceSocket = socket.socket(socket.
      AF_INET, socket.SOCK_DGRAM)
  # get initial value of consumer power
  consumerPowerOld = contract.functions.
      getAppartment1().call
  # main while-loop
  while True:
    data, addr = powerSocket.recvfrom(64)
    data = struct.unpack('d', data)
    consumerPower= int(data[0])
    if consumerPower != consumerPowerOld:
      print("new Data" + str(consumerPower))
      set_sc_consumer(contract, consumerPower,
          consumerAccount)

if __name__ == '__main__':
  main()
```

Listing 2: Implemented smart contract used in test case

C. Behavioral models

The models implemented in Matlab Simulink were specifically designed to run on near real time on the Raspberry Pi. It makes use of the target support package for the aforementioned platform. Figure 2 shows the main structure of the behavioral model. In order to have a synchronized simulation step, the simulation time is implemented as an UDP broadcast. Run by the principal model.

D. Physical implementation

The test bench was implemented at the laboratory for control engineering at Reutlingen University. As a use case example, a small house was constructed and the different loads and PV were set up. Figure 3 shows the physical demonstration object.

5. Further work

This test bed will be extended to accommodate different distributed ledger technologies available. For example IOTA, OByte or the Energy Web Foundation Chain. The use cases to be represented will be updated and enhanced as well.

6. Conclusions

This work has presented a method for emulating blockchain-based processes with the help of traditional modeling tools. Since current modeling tools have no way to interact with DLT-Nodes it was imperative to develop a method and software which could pair the input/output of a behavioral model and the blockchain. This was done through a python script, which then communicated to the blockchain node via IPC communication. The different nodes would run autonomously on a one-board computer, which in turn makes it easier to test scenarios like node separation (communication loss).

Tests like network resilience, reaction of distributed control to exceptions can now be easily reproduced and functionalities expanded.

The test bed must however be expanded to accommodate other DLTs in order to evaluate differences in performance, data integrity and other potential issues, challenging a wider adoption of DLTs in Energy Systems.

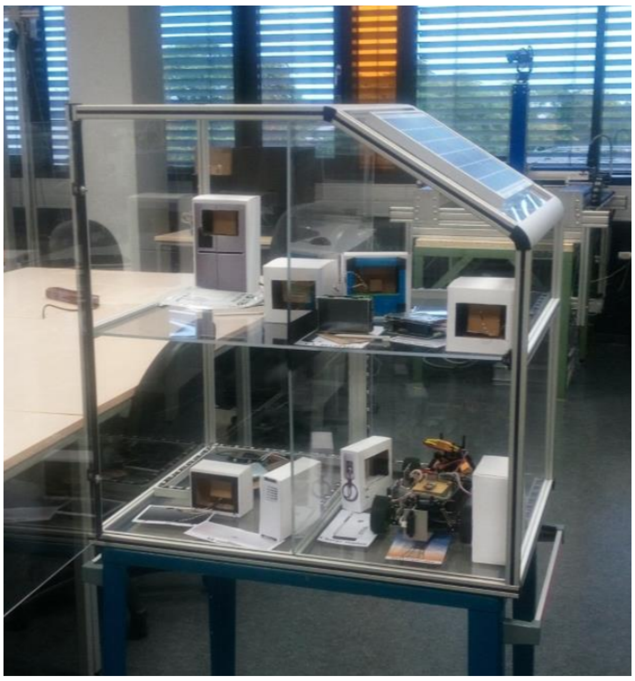

Fig. 3: Physical demonstration object

7. Acknowledgements

This paper presents the first results of the project "Virtuelles Kraftwerk der zweiten Generation" (VK_2G) funded by the DBU (Deutsche Bundesstiftung Umwelt) under number 34798. The authors would like to thank the DBU for its support in the development and implementation of the different use cases, included in the project VK_2G.

References

[1] S. Nakamoto. (2009) Bitcoin: A peer-to-peer electronic cash system. online. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[2] V. Buterin, "Ethereum: A next-generation cryptocurrency and decentralized application platform," *Bitcoin Magazine*, Jan. 2014.

[3] K. Yeow, A. Gani, R. W. Ahmad, J. J. P. C. Rodrigues, and K. Ko, "Decentralized consensus for edge-centric internet of things: A review, taxonomy, and research issues," *IEEE Access*, vol. 6, pp. 1513–1524, 2018.

[4] S. Popov. (2017) Iota whitepaper: The tangle. web. Visited on 29.1.2018. [Online]. Available: https://iota.org/IOTA_Whitepaper.pdf

[5] BDEW Bundesverband der Energie- und Wasserwirtschaft e.V., "Blockchain in der energiewirtschaft – blockchain in the energy sector," BDEW, techreport K08, Oct. 2017, page 63.

[6] Burger, A. Kuhlmann, P. Richard, and J. Weinmann, "Blockchain in the energy transition. a survey among decisionmakers in the german energy industry," Deutsche Energie-Agentur GmbH (dena) - German Energy Agency Energy Systems and Energy Services, Tech. Rep., 2016.

[7] Bundesnetzagentur, "Digitale transformation in den netzsektoren: Aktuelle entwicklungen und regulatorische herausforderungen," Bundesnetzagentur, techreport, May 2017.

[8] DENA, "Blockchain in der integrierten energiewende," Deutsche Energieagentur, techreport, Jan. 2019. [Online]. Available: https://www.dena.de/fileadmin/dena/Publikationen/PDFs/2019/Studie_Blockchain_Deutsches_Executive_Summary.pdf

[9] N. R. Jennings and S. Bussmann, “Agent-based control systems: Why are they suited to engineering complex systems?” *IEEE Control Systems Magazine*, vol. 23, no. 3, pp. 61–73, June 2003.

[10] M. R. B. Khan, R. Jidin, and J. Pasupuleti, “Multi-agent based distributed control architecture for microgrid energy management and optimization,” *Energy Conversion and Management*, vol. 112, pp. 288 – 307, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0196890416000273

[11] N. Wang, “Transactive control for connected homes and neighbourhoods,” *Nature Energy*, vol. 3, no. 11, pp. 907–909, Nov. 2018.

[12] H. Breuer. (2017, Jan.) A microgrid grows in brooklyn. newsletter. [Online]. Available: https://www.siemens.com/innovation/en/home/pictures-of-the-future/energy-and-efficiency/smart-grids-and-energy-storage-microgrid-in-brooklyn.html

[13] M. Saghaleini and A. Mazloomzadeh, “Agent based control scheme for a smart power system including renewable energy sources,” in *2011 10th International Conference on Environment and Electrical Engineering*, May 2011, pp. 1–4.

[14] A. M. O. Haruni, A. Gargoom, M. E. Haque, and M. Negnevitsky, “Dynamic operation and control of a hybrid wind-diesel stand alone power systems,” in *2010 Twenty-Fifth Annual IEEE Applied Power Electronics Conference and Exposition (APEC)*, Feb 2010, pp. 162–169.

[15] VDE, *Generators connected to the low-voltage distribution network. Technical requirements for the connection to and parallel operation with low-voltage distribution networks*, VDE Std. VDE-AR-N 4105, Rev. 2018-11, 2018.

[16] C. Rieger, Q. Zhu, and T. Basar, “Agent-based cyber control strategy design for resilient control systems: Concepts, architecture and methodologies,” in *2012 5th International Symposium on Resilient Control Systems*, Aug 2012, pp. 40–47.

[17] G. Polaków, P. Laszczyk, and M. Metzger, “Agent-based approach to model-based dynamically reconfigurable control algorithm,” in *2015 20th International Conference on Process Control (PC)*, June 2015, pp. 375–380.