CRYPTO-SPATIAL : AN OPEN STANDARDS SMART CONTRACTS LIBRARY FOR BUILDING GEOSPATIALLY ENABLED DECENTRALIZED APPLICATIONS ON THE ETHEREUM BLOCKCHAIN

BENAHMED DAHO Ali[1]

[1] Geodetic Sciences and Topographic Works Engineer, Ain Temouchent, Algeria - bidandou@yahoo.fr

Commission VI, WG VI/6

KEY WORDS: Ethereum Blockchain, Decentralized Applications, Smart Contracts, IPFS, OrbitDB, Land Administration, OGC Open Standards, FOAM.

ABSTRACT:

Blockchain is an emerging immature technology that disrupt many well established industries nowadays, like finance, supply chain, transportation, energy, official registries (identity, vehicles, ...). In this contribution we present a smart contracts library, named Crypto-Spatial, written for the Ethereum Blockchain and designed to serve as a framework for geospatially enabled decentralized applications (dApps) development. The main goal of this work is to investigate the suitability of Blockchain technology for the storage, retrieval and processing of vector geospatial data. The design and the proof-of-concept implementation presented are both based on the Open Geospatial Consortium standards: Simple Feature Access, Discrete Global Grid Systems (DGGS) and Well Known Binary (WKB). Also, the FOAM protocol concept of Crypto-Spatial Coordinate (CSC) was used to uniquely identify spatial features on the Blockchain immutable ledger. The design of the Crypto-Spatial framework was implemented as a set of smart contracts using the Solidity object oriented programming language. The implemented library was assessed toward Etheruem's best practices design patterns and known security issues (common attacks). Also, a generic architecture for geospatially enabled decentralized applications, combining blockchain and IPFS technologies, was proposed. Finally, a proof-of-concept was developed using the proposed approach which main purpose is to port the UN/FAO-SOLA to Blockchain techspace allowing more transparency and simplifying access to users communities. The smart contracts of this prototype are live on the Rinkeby testnet and the frontend is hosted on Github pages. The source code of the work presented here is available on Github under Apache 2.0 license.

1. INTRODUCTION

Geospatial technology is nowadays in the heart of major socio-economical processes giving experts and casual users valuable insights to improve there performances, optimize there daily tasks and make informed decisions. However, as for any industry sector, a growing number of deeptech technologies are emerging and disrupting well known workflows and established practices. In fact, the permeation of technologies such as IoT, Big Data Analytics, Cloud Computing, Artificial Intelligence, etc. have also greatly aided the spurt in adoption of Location Intelligence solutions (Geospatal Media and Communications, 2019). Nevertheless, we noted that in the recent years, Blockchain technology has not been intensively investigated for its suitability to leverage geospatial applications, and it's just in july 2019 that the OGC announces the creation of a new Domain Working Group for Blockchain and Distributed Ledger Technologies (BDLT/DWG) (OGC, 2019).

In addition, despite the existence of many initiatives to develop standardized protocols for geospatial technology on the blockchain, like (FOAM, 2019), (XYO, 2019), (Helium, 2020), we notice that all those projects focus mainly on proof-of-location wireless networks and not on geospatial data structures and applications. To fill this gap, we investigate in this contribution the suitability of Blockchain technology for the storage, retrieval and processing of vector geospatial data. Also, a generic architecture for geospatially enabled decentralized applications, is proposed and a proof-of-concept is developed using the proposed approach.

2. DECENTRALIZED APPLICATIONS

2.1 Ethereum blockchain

Ethereum blockchain can be viewed as a transaction-based state machine: we begin with a genesis state and incrementally execute transactions to morph it into some final state. It is this final state which we accept as the canonical "version" of the world of Ethereum. The state can include such information as account balances, reputations, trust arrangements, data pertaining to information of the physical world; in short, anything that can currently be represented by a computer is admissible. Transactions thus represent a valid arc between two states; the 'valid' part is important. A valid state transition is one which comes about through a transaction (Wood et al., 2014). Formally:

$$\boldsymbol{\sigma}_{t+1} \equiv \Upsilon(\boldsymbol{\sigma}_t, T) \tag{1}$$

where Υ is the Ethereum state transition function. In Ethereum, Υ, together with $\boldsymbol{\sigma}$ are considerably more powerful than any existing comparable system; Υ allows components to carry out arbitrary computation, while $\boldsymbol{\sigma}$ allows components to store arbitrary state between transactions.

Transactions are collated into blocks; blocks are chained together using a cryptographic hash as a means of reference. Blocks function as a journal, recording a series of transactions together with the previous block and an identifier for the final state. They also punctuate the transaction series with incentives for nodes to *mine*. This incentivisation takes place as a state-transition function, adding value to a nominated account

(Wood et al., 2014). Formally, we expand to:

$$\boldsymbol{\sigma}_{t+1} \equiv \Pi(\boldsymbol{\sigma}_t, B) \tag{2}$$
$$B \equiv (..., (T_0, T_1, ...), ...) \tag{3}$$
$$\Pi(\boldsymbol{\sigma}, B) \equiv \Omega(B, \Upsilon(\Upsilon(\boldsymbol{\sigma}, T_0), T_1)...) \tag{4}$$

Where Ω is the block-finalization state transition function; B is this block, which includes a series of transactions amongst some other components; and Π is the block-level state-transition function.

This is the basis of the blockchain paradigm, a model that forms the backbone of not only Ethereum, but all decentralized consensus-based transaction systems to date.

In term of implementation, there are many choices of blockchains: over 200 Bitcoin variants, Ethereum and other permissioned blockchains. To meaningfully compare them, (Dinh et al., 2017) identified four abstraction layers found in all of these systems. (1) The consensus layer contains protocols via which a block is considered appended to the blockchain. (2) The data layer which contains the structure, content and operations on the blockchain data. (3) The execution layer includes details of the runtime environment support blockchain operations. Finally, (4) the application layer which includes classes of blockchain applications.

The Crypto-Spatial framework, described in this contribution, is designed for the Ethereum Blockchain and propose a set of smart contracts for the execution layer and a cheap geometry storage solution on IPFS for the application layer.

2.2 IPFS and OrbitDB

IPFS is a distributed file system which synthesizes successful ideas from many peer-to-peer systems, including DHTs, BitTorrent, Git, and SFS. The contribution of IPFS is simplifying, evolving, and connecting proven techniques into a single cohesive system, greater than the sum of its parts. IPFS presents a new platform for writing and deploying applications, and a new system for distributing and versioning large data. IPFS could even evolve the web itself. IPFS is peer-to-peer; no nodes are privileged. IPFS nodes store IPFS objects in local storage. Nodes connect to each other and transfer objects. These objects represent files and other data structures (Benet, 2014).

OrbitDB. It is a distributed, peer-to-peer database that is built on top of IPFS. OrbitDB supports various kinds of databases including key-value and log databases. This makes OrbitDB an excellent choice for the decentralized prototype (OrbitDB, 2020).

In the solution we present in this contribution, the geometry of spatial features are stored in an OrbitDB IPFS database as OGC Well Known Binary objects for simple parsing and visualization. The databases also have listeners implemented that triggers when the databases are replicating. Thereafter, the listeners trigger the user interface to update. This ensures that the users will always have the most recent geometry available.

2.3 Decentralized applications development

Developing applications for the Blockchain is very similar to developing applications with traditional web technologies with some key differences. In fact knowing if an application need

blockchain technology is as important as knowing blockchain technology itself. To do that a developer must asses its needs using the following key questions :

1. Is the system defining digital relationships? if yes, Blockchain is suitable for this system.
2. Should data be dynamic and auditable? if yes, Blockchain is suitable.
3. Should data be managed by a central authority? if yes, Blockchain is not suitable in this case.
4. Is the speed of the network important? if yes, Blockchain is also not suitable here.

The figure 1 illustrate the development workflow for an Ethereum Blockchain decentralized application.
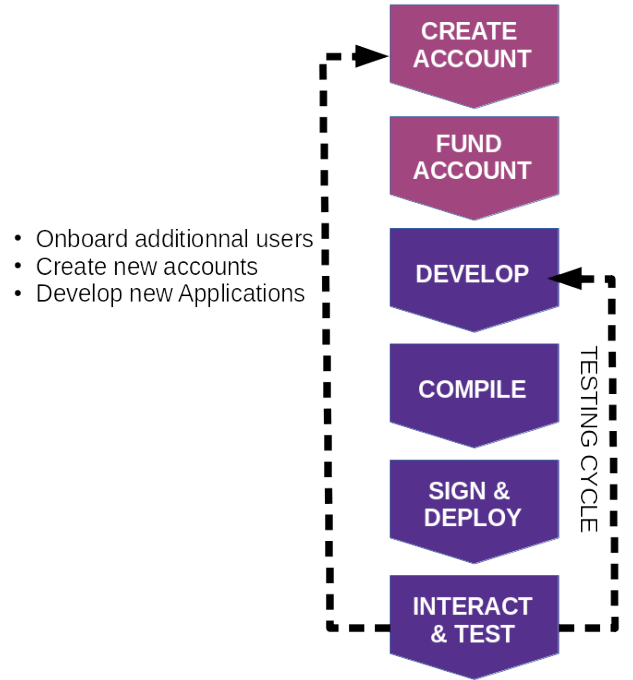

- Onboard additionnal users
- Create new accounts
- Develop new Applications

Figure 1. Blockchain development workflow

The proces start by creating an account using a wallet manager like Metamask (Metamask, 2020) or Uport (UPort, 2020) and fund it using a dedicated faucet for the selected testnet (Rinkeby, Ropsten, ...). If we develop locally we can just use ganache-cli (Truffle, 2020).

As we start to develop our decentralized application we can use one of the ethereum smart contracts development tools like (Truffle, 2020). Those tools allow us to compile, test and deploy our smart contracts on the testnet we select to use for our testing campaign before the ultimate deployment on the Ethereum mainnet which require real Ether (cryptocurreny of Ethereum).

For the frontend, we will need a web3 library corresponding to the programming language we use. Nevertheless, in general in the Etheruem world the ReactJS (ReactJS, 2020) framework is used to develop the User Interface (UI) with the web3.js (Web3.js, 2020) library.

3. THE CRYPTO-SPATIAL FRAMEWORK

The Crypto-Spatial Framework is a set of smart contracts written in Solidity (object oriented programming language for Ethereum) and serves as base classes that can be specialized and customized to meet business needs. The architecture of the Crypto-Spatial smart contracts is inspired from the OGC Simple Features specifications (Herring et al., 2011).

To uniquely identify spatial features on the Blockchain distributed ledger, the FOAM space concept of CryptoSpatial Coordinate (CSC) is used (Foamspace Corp, 2018). Nevertheless, some modifications was implemented to explore the alternatives suggested by (Gobe, Lathouwer, 2018) as the use of the H3 javascript library (H3, 2020), with an average Hexagon Edge Length of 0.5 km (resolution 15), which it is a partially conforming implementation of the Geodesic Discrete Global Grid Systems OGC standard (Purss et al., 2016).

3.1 Framework Design

The Crypto-Spatial smart contracts library, illustrated in the following class diagram, is designed and implemented using inheritance and interfaces to simplify its resusability.
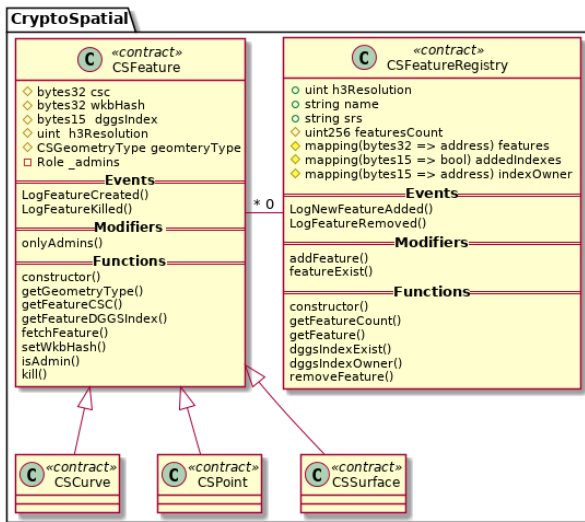

Figure 2. Crypto-Spatial Library Class Diagram

With this approach the developer should be able to :

1. inherit/use the Solidity components to build his custom contracts and develop more complex geospatial decentralized system.
2. use a deployed Crypto-Spatial smart contracts and libraries that suit his business needs from an ENS (ens.domains) resolvable Ethereum addresses.
3. integrate/download the solidity reusable components (contacts) from npmjs.com and/or ethpm.com as a standalone packages or included in a widely accepted library, like openZeppelin (OpenZeppelin, 2020)
4. visualize on a map all the geospatial features stored on the permanently deployed registry (as features index) to administer the features belonging to his registries.
5. access a fully featured dashboard displaying all useful informations about the permanently deployed features registries (the features index)

3.1.1 CSFeature The library design, inspired from OGC Simple Features, comprises a base abstract CSFeature smart contract to represent any type of spatial features. This smart contract is specialized to handle Points, Curves and Surfaces with the CSPoint, CSCurve and the CSSurface smart contracts respectively.

The CSFeature smart contract includes all necessary state variables, modifiers, events and functions to store and manipulate spatial features. Main members are :

bytes32 csc: the Crypto-Spatial Coordinate, which is the Keccak-256 hash of the DGGS index of the spatial feature and the owner address.
bytes32 wkbHash: the Well Known Binary Hash.
bytes15 dggsIndex: the Disrcete Global Geodetic System index.
uint h3Resolution: the H3 resolution used by the registry.
enum CSGeometryType: the geometry type (Point, Curve, Surface).
constructor: the constructor that initiate all state variables.
function getGeometryType: getter for geometry type.
function getFeatureCSC: getter for CSC.
function getFeatureDGGSIndex: getter for DGGS Index.
function fetchFeature: fetch all the state variables of the spatial feature.
function setWkbHash: setter for wkbHash.
function kill: to permanently remove the spatial feature from the blockchain ledger.

3.1.2 CSFeatureRegistry The second important abstract smart contract of the Crypto-Spatial core library design is the CSFeatureRegistry which serves as the spatial features collection. The CSFeatureRegistry smart contract includes all necessary state variables, modifiers, events and functions to store and manipulate spatial features. The main members of this smart contract are :

uint h3Resolution: the H3 resolution of the spatial feature registry (from 1 to 15) see [.....].
string name: the displayed name.
string srs: the Spatial Reference System code (EPSG or equivalent).
uint256 featuresCount: the Counter of the added features.
mapping features: an addresses mapping to handle the spatial features added to the registry.
mapping addedIndexes: a boolean mapping to keep trace of added indexes.
mapping indexOwner: a mapping to keep trace of DGGS indexes owners.
constructor: the constructor that initiate all state variables.
function addFeature: a modifier that must be called by the addFeature function of inherited smart contracts.
function getFeatureCount: getter of the featureCount.
function getFeature: getter for a designated spatial feature.
function dggsIndexExist: to confirm if an Index exist in the registry.
function dggsIndexOwner: returns the owner of a designated feature.
function removeFeature: permanently remove the spatial feature from the registry and the blockchain ledger.

3.2 Solidity Implementation

To demonstrate the suitability of the previous design, all the smart contracts library components has been implemented in

Solidity using the Truffle Suite (Truffle, 2020). Hereafter, a code snipet of the **addFeature** modifier demonstrating the ability to reuse business logic on the Etheruem smart contracts :

```
modifier
addFeature(bytes15 dggsIndex,
  bytes32 wkbHash,
  address _sender) {
 require(!paused(), "Contract is paused");
 require(dggsIndex.length != 0, "Empty dggsIndex");
 require(addedIndexes[dggsIndex] == false,
  "DGSS Index already exist");
 require(wkbHash.length != 0, "Empty wkbHash");

 _;

 addedIndexes[dggsIndex] = true;
 indexOwner[dggsIndex] = _sender;
 bytes32 csc = CSGeometryLib.computeCSCIndex(
  _sender, dggsIndex);
 emit LogNewFeatureAdded(name, csc, dggsIndex,
  wkbHash, _sender);
 featuresCount = featuresCount.add(1);
}
```

The complete implementation of the Crypto-Spatial framework can be found on the project github repository (BENAHMED DAHO Ali, 2020b).

3.3 Security issues and design patterns

3.3.1 Security issues mitigation In this section we synthesize the security vulnerabilities of Ethereum smart contracts and the vulnerabilities that can be (actually, most of them have been) exploited to carry on attacks. To avoid these common attacks a set of counter-measures have been implemented and some of them are described below.

Arithmetic Over/Under Flows To mitigate under/overflow vulnerabilities we use the openZeppelin 'SafeMath' mathematical library.

Reentrancy To mitigate the reentrancy vulnerability we place any code that performs external calls after the logic updating state variables.

Also, to avoid security vulnerabilities, the Crypto-Spatial smart contracts libray have been audited using common Ethereum security tools, formally : MythX/Mythril (MythX, 2020) and Slither (Slither analyzer, 2020). All the detected vulnerabilities have been fixed.

3.3.2 Design patterns As smart contracts are special immutable code executing on the Ethereum blockchain, a number of design patterns has to be applied to guarantee they are correctly prepared for all situations. This include checking the inputs as early as possible in the function body and throws an exception if the condition is not met, and restricting access to the smart contract functions that change the states using the (Open-Zeppelin, 2020) libraries 'Ownable' and 'Roles' .

4. DECENTRALIZED APPLICATIONS

Knowing that smart contracts takes only in charge the immutable part of the business logic of an application, it is also necessary to implement a frontend application to interact with the final user, set/get state variables and make calls to smart contracts functions. In the Ethereum techspace, this is generally done using ReactJS with the web3.js javascript library. To build a geospatially enabled decentralized application (GeodApp), integrating webmapping components with web3 enabled User Interface (UI) is necessary.

4.1 Geospatial dApps Architecture

To build Geospatially enabled dApps for the Ethereum blockchain the following 3-tiers architecture is recommended:

1. The smart contracts written in Solidity.
2. The IPFS/OrbitDB Storage.
3. The frontend web application (built with ReactJS or others).

The generic interaction between the previous components of a GeodApp can be described by the sequence diagram on figure 3.
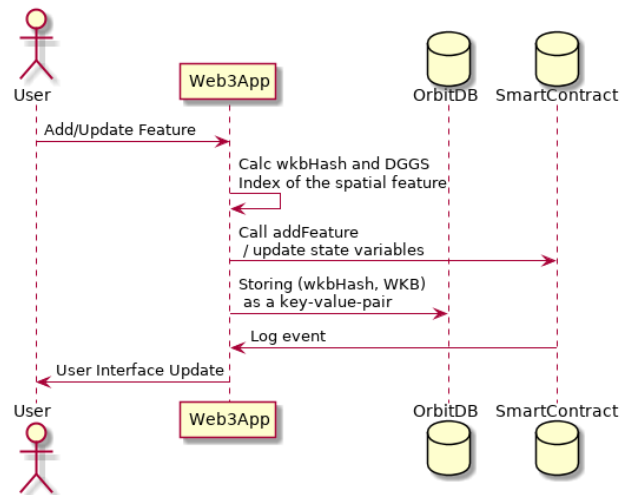

Figure 3. Geospatial dApps sequence Diagram

The choice to store the geometries of the spatial features on the IPFS (OrbitDB) is justified by the gas cost of the storage on the Ethereum Blockchain. It is cheaper to store the Well Known Binary (WKB) hash on the Blockchain and the WKB of the spatial feature geometry on OrbitDB/IPFS.

4.2 Decentralized land administration (DeLA)

As a proof-of-concept for the proposed architecture for a Geospatial Decentralized Application (GeodApp), we built DeLA (Decentralized Land Administration) which main objective is to implement on the Ethereum blockchain all the features of the the open source SOLA-FAO (Solutions for Open Land Administration) (FAO, 2017) which is a J2EE implementation that has many uses cases in Africa and Asia. Using SOLA allows us to incorporate international best practice and standards, namely the ISO 19152:2012 standard - Geographic information — Land Administration Domain Model (LADM)) (Van Oosterom et al., 2013).

One of the major goals for porting this land registry solution to the Etherum blockchain is the ability to use it as a crowd sourcing land registry platform (Seufert, 2013) to collect tenure relationships and as a tool for communities to assess and clarify their tenure regimes so to protect the individual and collective rights of their members.

The source code of this GeodApp is available on (BENAHMED DAHO Ali, 2020a) and a working version, deployed on the Rinkeby testnet, is live on (BENAHMED DAHO Ali, 2020b).

The class diagram on figure 4 illustrate the inheritance mechanism used to easily implement specific business logic with solidity smart contracts.
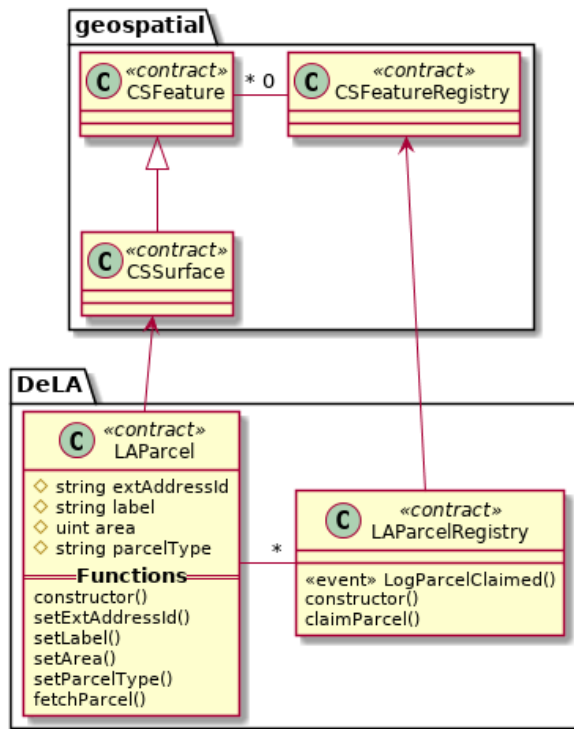

Figure 4. DeLA class Diagram

To illustrate the ability to reuse business logic on the Etheruem smart contracts, hereafter a code snippet of the **claimParcel** function reusing the **CSFeature addFeature** modifier.

```
function
claimParcel(bytes15 dggsIndex,
bytes32 wkbHash,
string memory extAddr,
string memory label,
uint area,
string memory landUseCode,
LAParcel.CadastralTypeCode cadastralType)
      public
      addFeature(dggsIndex, wkbHash, msg.sender)
      returns (bytes32) {
      ...
   return csc;
  }
```

5. BLOCKCHAIN BUSINESS MODELS

The adoption of the proof-of-concept described in this contribution can open many business opportunities as those described bellow and inspired from (Nitish Singh, 2018).

5.1 Developement plateform

The main goal of Crypto-Spatial is to deliver a framework of geospatially enabled smart contracts and libraries for secure GeodApps development. It will provide implementations of standards like OGC Simple Features Access, OGC Discrete Global Grid Systems, ISO 19107 Geographic information - Spatial schema (ISO, 2008) and the FOAM protocol.

The smart contracts and libraries can be deployed, as-is or extended to suit business needs, as well as Solidity components to build custom contracts and more complex decentralized systems.

After reaching certain maturity, this framework can be submitted as a candidate to OpenZeppelin, or as an EIP (Ethereum Improvement Proposal) for standardization by the community.

5.2 Blockchain as a service (BaaS)

To fully operate the DeLA platform, a set of permanently deployed components are required. In addition to the geospatially enabled smart contracts and libraries deployed on the Etheruem network (test in the development phase and mainnet after) the platform will require :

1. a mapping server with its geospatial database to store the spatial features (The Feature Index). The final platform can be implemented using PostgreSQL database, PostGIS middleware and Geoserver webmapping server.
2. a frontend web server to store and publish the platform Dashboard Application.
3. (optionally) a backend web server to manage business logic tasks, mainly the smart contracts events handling needed to catch the ledger recorded spatial features and populate the Features Index (database)

The components 1 and 3 are by design a shared services that can be easily monetized for further integration in custom blockchain geospatially enabled applications, without the need to redeploy them.

5.3 Blockchain based Software products

The DeLA (Decentralized Land Administration) platform presented in this contribution is the first built prototype of a GeodApp.

This prototype will serve as a fully operational demonstration of the proposed approach and could therefore be used in a variety of tastes, including :

1. Permissionless/Permissioned application for governmental agencies responsible for land administration and lacking critical resources to undertake there missions as described in (McKay et al., 2016)
2. An open data crowd sourcing platform, like OpenStreetMap, delivering useful land parcel informations where no authoritative or commercial data are available.
3. A building block for specific business case GeodApps using land information like Real estate, Investments valuation, Social responsibility, Environmental protection, Disaster management...

6. CONCLUSION

In this paper we investigate the suitability of Blockchain Technology to serve as a data layer for geospatial features. We then propose an open architecture for geospatially enabled decentralized applications (GeodApps) based on Ethereum blockchain and IPFS/OrbitDB peer-to-peer storage.

The main objective of this investigation have been reached and the proposed design have been successfully implemented in a proof-of-concept GeodApp which is live on the Ethereum Rinkeby testnet and available for further test and improvement as an open source project.

Main implications arising from those findings are summarized on the Business models section and further works should be done to port this solution to other type of protocols supporting smart contracts but not using solidity, like Hyperledger fabric.

ACKNOWLEDGEMENTS

REFERENCES

BENAHMED DAHO Ali, 2020a. DeLA live on Rinkeby. allilou.github.io/onchain-land-administration.

BENAHMED DAHO Ali, 2020b. DeLA source code repository. github.com/allilou/onchain-land-administration.

Benet, J., 2014. Ipfs-content addressed, versioned, p2p file system. *arXiv preprint arXiv:1407.3561*.

Dinh, T. T. A., Wang, J., Chen, G., Liu, R., Ooi, B. C., Tan, K.-L., 2017. Blockbench: A framework for analyzing private blockchains. *Proceedings of the 2017 ACM International Conference on Management of Data*, ACM, 1085–1100.

FAO, 2017. SOLA-FAO. github.com/SOLA-FAO.

FOAM, 2019. Foamspace Corp. foam.space.

Foamspace Corp, 2018. FOAM Whitepaper. tiny.cc/n5lpiz.

Geospatal Media and Communications, 2019. Geobuiz 2019. Geospatial Industry Outlook and Readiness Index.

Gobe, H., Lathouwer, B. D., 2018. Geospatial Standardization of Distributed Ledger Technologies. tiny.cc/36lpiz.

H3, 2020. Uber Technologies Inc. uber.github.io/h3.

Helium, 2020. Helium Systems Inc. www.helium.com.

Herring, J. et al., 2011. OpenGIS® Implementation Standard for Geographic information-Simple feature access-Part 1: Common architecture [Corrigendum].

ISO, I., 2008. ISO 19107-Geographic information–Spatial schema.

McKay, B., Rodriguez, F. B., Fajardo, D., 2016. The voluntary guidelines on the responsible governance of tenure of land, fisheries and forests in the context of national food security in colombia: Towards democratic land-based resource control. *Colloquium Paper No*, 19.

Metamask, 2020. ConsenSys AG. metamask.io.

MythX, 2020. CONSENSYS. mythx.io.

Nitish Singh, 2018. Top 7 Blockchain Business Models That You Should Know About. 101blockchains.com/blockchain-business-models.

OGC, 2019. The Open Geospatial Consortium. tiny.cc/bfkpiz.

OpenZeppelin, 2020. OpenZeppelin. openzeppelin.com/contracts.

OrbitDB, 2020. github.com/orbitdb/orbit-db.

Purss, M. B., Gibb, R., Samavati, F., Peterson, P., Ben, J., 2016. The ogc® discrete global grid system core standard: A framework for rapid geospatial integration. *2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, IEEE, 3610–3613.

ReactJS, 2020. Facebook Inc. reactjs.org.

Seufert, P., 2013. The FAO voluntary guidelines on the responsible governance of tenure of land, fisheries and forests. *Globalizations*, 10(1), 181–186.

Slither analyzer, 2020. crytic.io. github.com/crytic/slither.

Truffle, 2020. Truffle Blockchain Group. www.trufflesuite.com.

UPort, 2020. ConsenSys AG. www.uport.me.

Van Oosterom, P., Lemmen, C., Uitermark, H., 2013. ISO 19152: 2012, land administration domain model published by ISO. *FIG Working Week 2013 in Nigeria–Environment for Sustainability, Abuja, Nigeria, 6–10 May 2013*.

Web3.js, 2020. github.com/ethereum/web3.js.

Wood, G. et al., 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014), 1–32.

XYO, 2019. XYO Foundation. xyo.network.