# 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, vehicules, ...). 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) developement. The main goal of this work is to investigate the suitability of Blockchain technology for the storge, 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, formally : Simple Feature Acces, Discrete Global Grid Systems (DGGS) and Well Known Binary (WKB). Also, the FOAM protocole concept of Crypto-Spatial Coordinate (CSC) was used to uniquely idenitify spatial features on the Blockchain immutable ledger. The desgin 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 developped using the proposed approach which main pupose is to port the UN/FAO-SOLA plateform to Blockchain techspace allowig 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 availaible on Github under Apache 2.0 licence.

## 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 [GeoBuiz-19 report]. 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)[opengeospatial.org press 3016].

In addition, despite the existance of many initiatives to develop standardized protocols for geospatial technology on the blockchain, like [foam.space], [xyo.network], [helium.com], 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 storge, retrieval and processing of vector geospatial data. Also, a generic architecture for geospatially enabled decentralized applications, is proposed and a proof-of-concept is developped 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. Formally:

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

where $\Upsilon$ is the Ethereum state transition function. In Ethereum, $\Upsilon$, together with $\boldsymbol{\sigma}$ are considerably more powerful than any existing comparable system; $\Upsilon$ 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. Formally, we expand to:

$$\boldsymbol{\sigma}_{t+1} \equiv \Pi(\boldsymbol{\sigma}_t, B) \quad (2)$$

$$B \equiv (..., (T_0, T_1, ...), ...) \quad (3)$$

$$\Pi(\boldsymbol{\sigma}, B) \equiv \Omega(B, \Upsilon(\Upsilon(\boldsymbol{\sigma}, T_0), T_1)...) \quad (4)$$

Where $\Omega$ is the block-finalisation state transition function; $B$ is this block, which includes a series of transactions amongst some other components; and $\Pi$ 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 decentralised 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, [Ji Wang and al.] 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 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 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 sytems, 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, J. (2014). IPFS-content addressed, versioned, P2P file system. arXiv preprint arXiv:1407.3561.]

OrbitDB. It is a distributed, peer-to-peer database that is built on top of IPFS [60]. OrbitDB supports various kinds of databases including key-value and log databases. This makes OrbitDB an excellent choice for the decentralized prototype.

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 visualisation. 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. [OrbitDB. OrbitDB. url: https://github.com/orbitdb/orbit-db/].

### 2.3 Decentralized applications developement

Developping 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 folowing key questions :

1. Is the system defining digital reationships? 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 [] illustrate the developement workflow for an Ethereum Blockchain decentralized application.

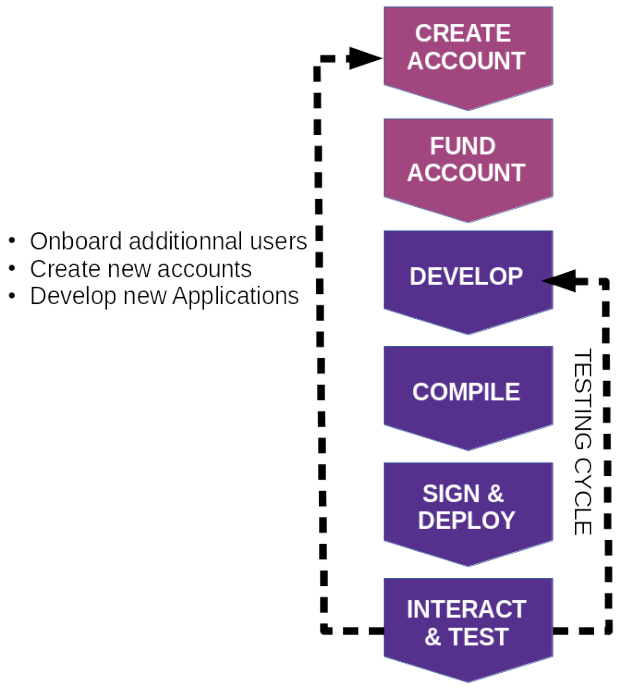


Figure 1. Blockchain developement workflow

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

As we start de developp our decentralized application we can use one of the ethereum smart contracts developpement tools like Truffle[]. Those tools allow us to compile, test and deploy our smart contracts on the testnet we select to use for our testing campaing before the ultimate deploiement on the Ethereum mainnet which require real Ether (cryptocurreny of Ethereum).

For the frontend, we will need a web3 library depending on the programming kanguage we use. Nevertheless, in general in the Etheruem world the ReactJS framework is used to develop the User Interface (UI) with the web3.js 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 specilized and customized to meet business needs. The architecture of the Crypto-Spatial smart constracts is inspired from the OGC Simple Features specifications [..].

To uniquely identify spatial features on the Blockchain distributed ledger, the FOAM space [foam.space] concept of Crypto-Spatial Coordinate (CSC) is used. Nevertheless, some modifications was implemented to explore the alternatives suggested by the [OGC discussion paper (7.5)-18-041r1.html] as the use of the H3 javascript library [uber.github.io/h3/], with the resolution 15 (Average Hexagon Edge Length of 0.5 km), which it is a partially conforming implementation of the [Geodesic Discrete Global Grid Systems-http://webpages.sou.edu/ sahrk/sqspc/pubs/gdggs03.pdf) OGC standard.

### 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.

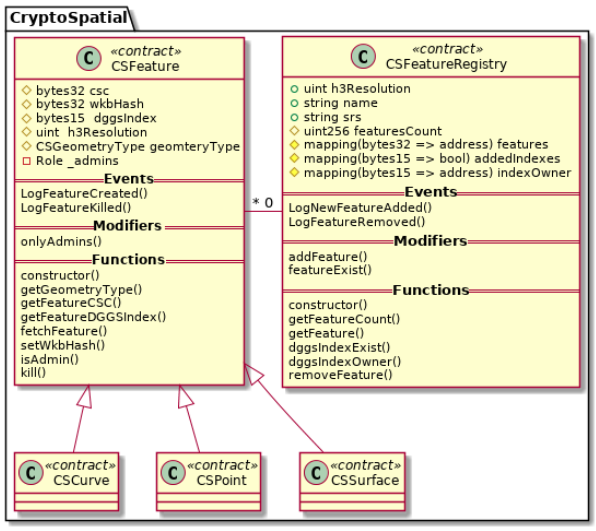


Figure 2. Crypto-Spatial Library Class Diagram

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 ENS (ens.domains) resolvable Ethereum an addresses.
3. integrate/donwload the solidity reusable components (contacts) from [npm](https://www.npmjs.com/) and/or [ethPM](https://www.ethpm.com/) as standalone packages or included in a widely accepted library, like [openZeppelin](https://openzeppelin.com/contracts/)
4. visualize on a map all the gospatial features stored on the permanently deployed registry (as features index) to administer the features belonging to his regitries.
5. access a fully featured dasboard displaying all useful information 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 hanldle 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 of GeoJSON geometry.
**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 smar contracts 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 contract.
**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 fetaure 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 [..]. Herafter, a code snipet of the **addFeature** modifier demonstrating the ability to resuse 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 following github repository.
https://github.com/allilou/onchain-land-administration/tree/master/solidity/contracts/geospatial

### 3.3 Security issues and design patterns

**3.3.1 Security issues mitigation** In this section we synthetize 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-mesures 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 any 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 and Slither. 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 acces to the smart contract functions that change the states using the openZeppelin 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 enabed dApps for the Ethereum blockchain the following 3-tiers architecture is recommanded:

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 [].

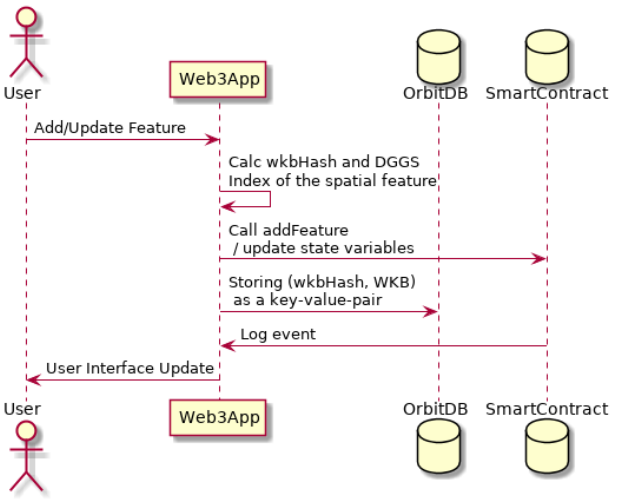


Figure 3. Geospatial dApps sequence Diagram

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 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 Decentrelized Application, we built DeLA GeodApp (Decentrelized Land Administration) which main objective is to implement on the Ethereum blockchain all the featires of the the open source SOLA-FAO (Solutions for Open Land Administration) [https://github.com/SOLA-FAO/] 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))](https://www.iso.org/standard/51206.html).

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 plateform](http://www.fao.org/tenure/voluntary-guidelines/en/) 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 dApp is available on [https://github.com/allilou/onchain-land-administration] and a working version, deployed on the Rinkeby testnet, is live on [https://allilou.github.io/onchain-land-administration/].

The class diagram on figure [] illustrate the inheritance mecanism used the easily implement specific business logic with solidity smart contracts.

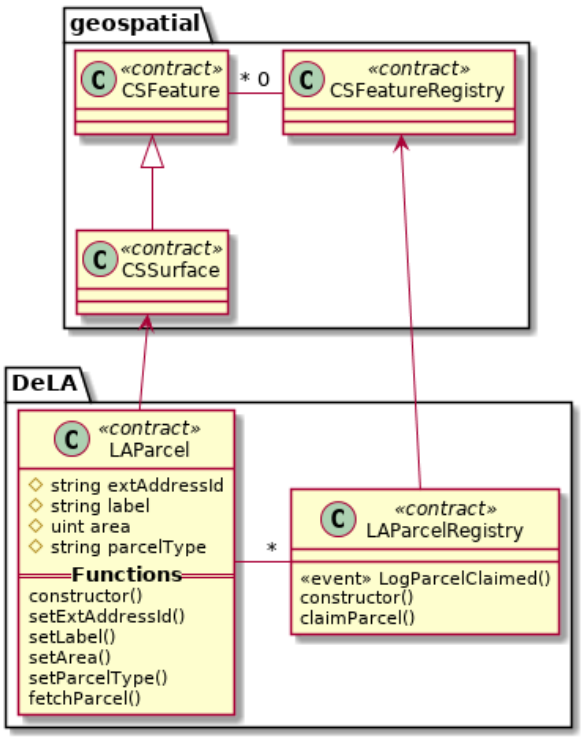


Figure 4. DeLA class Diagram

To demonstrate the ability to resuse business logic on the Etheruem smart contracts, herafter a code snipet 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 [https://101blockchains.com/blockchain-business-models/].

### 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 developement. It will provide implementations of standards like [OGC Simple Features Access], [OGC Discrete Global Grid Systems], [ISO 19107 Geographic information - Spatial schema] 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 developement phase and mainet after) the platform will require :

1. a mapping server with its geospatial database to store the spatial features (The Feature Index). The final plateform can be implemented using [PostgreSQL database], [PostGIS middleware] and [Geoserver] as webmapping server.
2. a frontend web server to store and publish the platform Dashboard Application.
3. (optionnaly) 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 montized for further integration in custom blockchain geospatially enabled applications, without the need to redeploy them. The deploiment diagram below illustrate this strategy.

### 5.3 Blockchain based Software products

The DeLA (Decentralized Land Administration) platform presented in this contributionis the first prototype built of a geospatially enabled application.

This prototype will serve as a fully operationnal demonstation 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 adminsitration and lacking critical ressources to undertake there missions as described in the [FAO Voluntary Guidelines on the Responsible Governance of Tenure of Land, Fisheries and Forests in the Context of National Food Security]

2. An open data crowd sourcing plateform, OpenStreetMap like, delivering useful land parcel informations where no autoritative or commerical data are available. [Open Land Data in the Fight Against Corruption - Discussion Report - landportal.org]
3. A building block for specfic business case GeodApps using land information like Real estate, Investments valuation, Social responsbility, Envirmental 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 avaibale for further test and improvement as an open source project.

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

## ACKNOWLEDGEMENTS

I would like to thanks Africa Blockchain Alliance to giving me the chance to enroll on the Africa Blockchain Developper Program supported by ConsenSys Academy. Also many thanks to my family members, particularly Ziad, for supporting me all the way.