Tecnológico de Monterrey, Campus Querétaro
Computer Science Department
Fernando Lobato Meeser
A01207873

Document: Architecture Evaluation
Software Architecture

“IndieCoin architecture evaluation”

The complete application can be found in the following repository:

https://github.com/fernandolobato/indiecoin

This document will serve as guide to explain the architecture, the designs decisions made and everything relevant to understanding the system. Finally we will have an evaluation of the general architecture. For any questions regarding the context of the project or the initial scope envisioning please refer to the HLD or FRD template. They can be found inside the repository as pdf documents in the docs/ path.

For a detail description of how to obtain, deploy, test and run the system refer to the README file inside the repository. User guide and installation steps can also be found there.

The system is composed of different modules that interact amongst each other. All of them, except the node module can work as independent modules. All modules can be found under de indiecoin/ folder. All code is written in python 2.7.

All the code inside the repository was written for this project except for the folder inside indiecoin/util/ecdsa this is an existing implementation of elliptic curve cryptography digital signatures written in pure python from Brian Warner.

There is also one file inside indiecoin/node/bt_peer.py which was taken from professor Nadeem Abdul. This file has basic functionality to implement a P2P network. Basically creates wrappers around native socket objects and has functionality implemented to communicate amongst them.

The remaining files were all written for the purpose of this project.

The modules are:

- Blockchain

  Takes care of blockchain operations, blocks, transactions and database operations. Contains the following files:

  The __init__.py file for blockchain has simple operations for the blockchain to query blocks and transactions based on hashes and height.

  - database.py

    In this file we have database class which performs direct operations into the database. Reads and writes blocks and transactions. Both transaction and block will have database classes which will inherit from this class. Also creates the initial database. The first time the project is ran it creates the folder ~/.indiecoin/data/ in this folder database will be stored.

- transaction.py

  Has the classes for handling transactions, transaction inputs and transaction outputs in run time. Also has a database class that inherits from database.py This classes in this file allow us to validate and manipulate transactions in runtime. The database class in this file serves as interphase for building transaction objects from the database. It does not query directly to the database, it uses methods in the database.py class.

- block.py

  The same as out previous file but to handle block objects at runtime. Also has a database inherited object to turn database information into objects we can handle and validate in runtime.

All the objects here have serialization methods to be sent over the network and can be constructed from python dictionaries and json objects in a nested fashion.

- Miner

Has file miner.py, this file handles the mining portion of our blockchain. It runs a separate thread that can be controlled by whoever start it. It receives a list of transaction objects from blockchain.transaction and creates a new block with this object referencing the last block in our blockchain. Once it creates a new block it can start mining. Mining is just the process of finding a sha256 digest of the block whose decimal value is less than or equal to a value that we have set called difficulty. Hence creating a proof of work. If the block is found it stops till the caller released the object to begin mining again. This way the network portion of the application can mine, broadcast and mine again. It can also be interrupted, the block discarded and begin mining again, in the case of an incoming block.

- Wallet

This module handles the address and safety features of this blockchain. It has one file called address.py. This file is basically a wrapper over the ecsa library. We can generate an Address object which with no parameters generates a new key pair. We can sign and verify messages with this key pair and get the key pair in string hexadecimal representation. We can also provide a private key, calculate the public key and do the same. In the case we only provide a public key, we can’t sign but we can verify. This module is used by transactions to verify authenticity.

- Node

This module has the following files:

  - bt_peer.py

    Establishes the basics for a p2p network. Has a class that serves as a wrapper over a socket object to handle connections amongst peers and a class that uses this objects for p2p communication. This class has a main loop that is the applications main loop. In this loop we listen in a socket for incoming requests. When a requests comes, we create a new socket for the request, serve it in a new thread and then close the thread an socket. To serve it we have a dictionary that maps type of responses to handler functions.

  - ic_peer.py

    This file has the basic handler functions for peers to communicate, request peers and connections. Inherits from bt_peer.py.

  - ic_node.py

    This is the actual node implementation, inherits from ic_node.py. This class has the communication handlers for communication and relaying blocks and transactions. It also has a thread that point to the main method in bt_peer.py from which it inherits and can start that thread through a public function. Receives a miner object and interacts with it to broadcast blocks and interrupt it if new blocks are received.

Inside this folder we also have the protocol folder, it has the following files:

  - protocol.py

    This file has all the definitions for the constants that our program matches to request handlers. Basically all the network communications.

  - response.py

    When we receive communications through a socket we just receive a tuple with the type of request and some text with all the data sent. This class turns what we received into an object where we can easily math that request type through different methods and turns the data into a python dictionary for easy handling.

- Util

This module has useful things that could not be fitted elsewhere. This module has the ecsa python implementation, has a couple of helper functions and wrappers over hash functions to make it easier to interact.

Documentation

All the code has been documented using the numpy guide for documenting. The numpy guide is very heavy, but we did our best to implement it. Every function, method and class has comments explaining what it does, who it calls, what are its attributes, what exceptions it can raise and what it returns in the case it returns something. To check this browse to any code file inside the repository and check the doc string in each method. This will make easy to integrate sphinx in the future which could take all this docstrings and generate a documentation portal. For the moment however it was a very complex project and am glad it's documented and well structured.

Code Style

All the code was written following pep8 which is the official python's style guide. To test this, a linter called flake8 was used to ensure the quality of the code. If flake8 is not installed in the system, we have included it in the requirements.txt file in the repository. We also have .flake8 configuration file in the repository which sets the values for flake8 to run pep8. To test this run the command flake8 in the root of the project. It will display all styling mistakes inside the project, if there are none, then there are no styling mistakes. We have removed ecdsa from the flake8 configuration file to avoid rewriting it.

Testing

Unit testing was performed on the different modules created to ensure the quality of the code. Unit test files can be found inside the tests/ folder in the root of the project. For steps on how to run them refer to the README file in the root of the project's repository. Each file test a different module and all test have been documented.

Architecture Patterns Used

Layer Pattern

Logic inside the blockchain module use this logic so that high level functions such as blocks have no direct interaction with the database, the have interaction with a database class which inherits from the class that has direct interaction with the database class. There are no calls skipping this process.

Peer-to-Peer pattern

The application is a peer to peer system. Each peer has a list of its peers, it can query more peers from its known peers if it loses contact with other peers. Each peer has a complete copy of the blockchain and can listen for transactions and blocks, validate and relay them to other peers on best effort basis.

SOA pattern

The communication protocol in the system allows for building very decoupled interphases on top of the protocol. All the information can be queried with protocol information and is sent in JSON format. So this protocol could easily work as an API for simplified payment verifications systems as the bitcoin protocol. Although the implementation of a JSON API for an SPV system is out of the scope of this project it could be easily implemented because of the architecture being followed.

Design Patterns Used

Dependency Injection

To keep the system as clean as possible there are many classes that do not directly create object but received them through a constructor. This was principally done for database objects to inhibit testing. Where any object can receive the database instance that it will use and pass it down into the inherited objects. The same happens with the interaction amongst miner and node. Node receives a Miner object with which it will interact.

Open / Close Principle

Classes keep their main functionality private but allows to be extended and implement new functionality. Also keeping the important things encapsulated. The wrapper of ecdsa python uses this concept so we don't interact with binary data, just hexadecimal string representations of this data.

Single Responsibility Principle

Keeping the functionality and responsibility of each class as much as possible. As we saw in the description of the modules and the classes amongst the modules, we can see they all have a very simple explicit responsibility. If something else was needed, we would create a new class or extend a previous class.

KISS principle

This was very easy to do since python by philosophy adheres a lot to this principle. The restriction of the scope of the project and the purpose of making a simple and understandable representation helped us a lot to achieve very simple code.

References

The following references are listed in order of how much knowledge someone has about the underlying system. The first to resources provide a very basic introduction into many of the concepts that will be used in all the documents. The other resources are lengthy, detailed and technical. However, they are useful as reference for specific subjects.

A swift introduction onto what bitcoin is, what it means and how it works. This videos show all the necessary prior knowledge needed to understand how a decentralized currency like the one that will be built works.

Concepts such as digital signatures, hashes, proof-of-work and certain cryptographic principles. While the explanations are not very technical they are great reference.

Khan Academy

https://www.khanacademy.org/economics-finance-domain/core-finance/money-and-banking/bitcoin/v/bitcoin-what-is-it

Created by: Zulfikar Ramzan

The initial bitcoin whitepaper which outlines the architecture for a distributed cryptocurrency.

Bitcoin: a peer to peer electronic cash system

https://bitcoin.org/bitcoin.pdf

By: Satoshi Nakamoto

Detailed explanation of all bitcoin features and their interactions.

Mastering Bitcoin

https://uplib.fr/w/images/8/83/Mastering_Bitcoin-Antonopoulos.pdf

By: Andreas M. Antonopoulos

The actual bitcoin open source repo implementation

https://github.com/bitcoin/bitcoin

A complete database schema for a bitcoin.

https://webbtc.com/api/schema

- BTPeer

  Description: A Pure-Python implementation of a peer to peer framework for general purpose.

  Author: Nadeem Abdul Hamid

Original Source can be found at:

http://cs.berry.edu/~nhamid/p2p/btpeer.py

- pyaes

  Description: Pure-Python implementation of AES block-cipher and common modes of operation.

  Author: Richard Moore (pyaes@ricmoo.com)

  License: MIT

- python-ecdsa

  Description: This is an easy-to-use implementation of ECDSA cryptography (Elliptic Curve Digital Signature Algorithm), implemented purely in Python, released under the MIT license. With this library, you can quickly create key pairs (signing key and verifying key), sign messages, and verify the signatures. The keys and signatures are very short, making them easy to handle and incorporate into other protocols.

  Author: Brian Warner (warner-github@lothar.com)

  License: MIT

  Original source can be found at:

https://github.com/warner/python-ecdsa