Course
Fog Computing for Smart Services

Development guide for Smart Contracts

University of Ljubljana
Faculty of Computer and Information Science

DESIGN PATTERNS

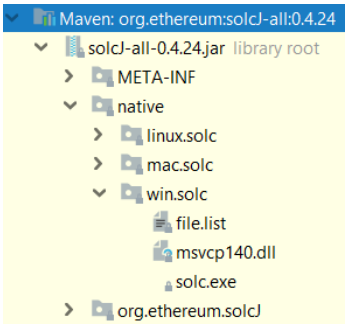
- Identified design patterns in Smart Contracts[1]
  - Check the pattern "**ORACLE (DATA PROVIDER) PATTERN**":
    Problem: An application scenario requires knowledge contained outside the blockchain, but Ethereum contracts cannot directly acquire information from the outside world. On the contrary, they rely on the outside world pushing information into the network.
    Solution**:** Request external data through an oracle service that is connected to the outside world and acts as a data carrier.
  - The Oracle pattern will be replaced later on during the course with a Smart Oracle approach, for example:
    - Centralized Smart Oracle: Provable (ex Oraclize), http://www.oraclize.it/
    - Decentralized Smart Oracle: ChainLink, https://chain.link/
- Secure and reusable Smart Contracts: https://openzeppelin.com/ (check Products -> Contracts)
- ConsenSys best practices: https://consensys.github.io/smart-contract-best-practices/
- Comprehensive guide: https://yos.io/2019/11/10/smart-contract-development-best-practices/
- Handy DApp design patterns: https://medium.com/@i6mi6/solidty-smart-contracts-design-patterns-ecfa3b1e9784

ESSENTIALS

- ETH Gas station:
  - Mainnet: https://ethgasstation.info/
  - Rinkeby: https://www.rinkeby.io/#stats
  - Ropsten: https://ropsten-stats.parity.io/
  - Kovan: N/A from the official Web page: https://kovan-testnet.github.io/website/
- Ethereum explorers:
  - Mainnet: https://ethplorer.io/ https://etherscan.io/
  - Rinkeby: https://rinkeby.etherscan.io/
  - Ropsten: https://ropsten.etherscan.io/
  - Kovan: https://kovan.etherscan.io/
- Web tools:
  - Remix IDE: https://remix.ethereum.org/
  - Oyente: https://oyente.melonport.com/
  - Truffle Suit: https://www.trufflesuite.com/
  - Metamask: https://metamask.io/
- How to compile *solidity* Smart Contracts on preferred OS?
  - Use Maven dependency https://mvnrepository.com/artifact/org.ethereum/solcJ-all
    1. Setup a Maven Project in a local IDE (e.g. Intellij Idea).
    2. Use the solcJ-all dependency for a preferred version.

[1] M. Wöhrer and U. Zdun, "Design Patterns for Smart Contracts in the Ethereum Ecosystem," 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), Halifax, NS, Canada, 2018, pp. 1513-1520.
doi: 10.1109/Cybermatics_2018.2018.00255, URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8726782&isnumber=8726472

3. In the external libraries you can find fully working compilers for Linux, Max and Windows as presented on the figure bellow.



4. You are able to compile it locally, for example in MS Windows:
   solc.exe --bin <mySmartContract.sol> --abi --optimize -o <path for the BIN and ABI output> --overwrite

- Ways to connect to the Ethereum node?
  - Run your own node, guides:
    1. https://dev.to/eric_khun/a-beginner-guide-to-setup-an-ethereum-full-node-4d9
    2. https://medium.com/quiknode/run-your-own-ethereum-node-5c3061925e6a
    3. By running a full node you are able to use filters and events: https://web3j.readthedocs.io/en/latest/filters_and_events.html
  - Use a faucet such as Infura has limitation with events (link). Notify me if this limitation is solved.
  - Run the node in Docker Container, for example: https://hub.docker.com/r/ethereumex/geth-node

EXAMPLES

Message.sol is basic smart contract that allows storing data on the blockchian. The user can trigger a function to store a message on the blockchain, which can be available on the blockchain for further verification whenever neccessary. Moreover, the user can always get the latest message that has been stored on the blockchain. The smart contract also allows to be destroyed by its user.

HappyMath.sol is a smart contract that allows the owner of the smart contract to perform simple mathematical calculation. The smart contract implements a modifier that checks whether the current address is allowed to execute the calculation.

Businessman.sol is a smart contract that facilitates P2P transactions between two entities (i.e. the person executing the transactions and the owner of the smart contract). The smart contract implements a modifier that checks whether the current address has sufficient resources. The smart contract provides three different practices for performing transactions, by using: *send*, *transfer*, *call*.

address.transfer()	***address.send()***	***address.call()***
throws on failure	returns false on failure	returns false on failure
forwards 2,300 gas stipend (not adjustable), safe against reentrancy	forwards 2,300 gas stipend (not adjustable), safe against reentrancy	forwards all available gas (adjustable), not safe against reentrancy
should be used in most cases as it's the safest way to send ether	should be used in rare cases when you want to handle failure in the contract	should be used when you need to control how much gas to forward when sending ether or to call a function of another contract

All smart contracts can be found on the Učilnica Web page. Feel free to adopt, fix and enhance the presented smart contracts.

Authors: Assist. Sandi Gec, sandi.gec@fri.uni-lj.si; Assist. Petar Kochovski, petar.kochovski@fri.uni-lj.si