SECR

Software Engineering Conference Russia 2018

October 12-13
Moscow

An Agile Software Engineering Method to Design Blockchain Applications

Michele Marchesi, Lodovica Marchesi, Roberto Tonelli
University of Cagliari

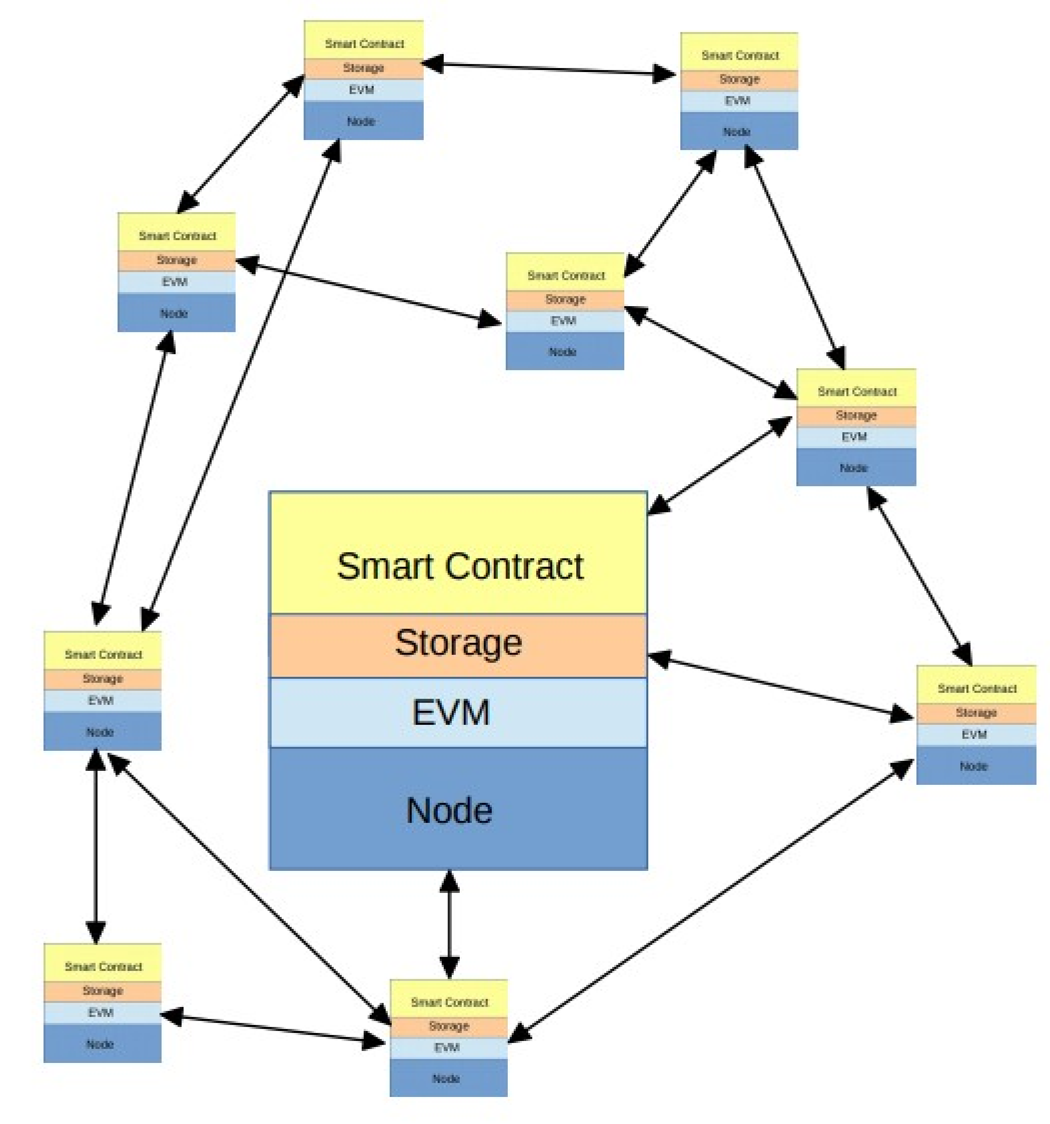
Blockchain

- The Blockchain was a technology whose first application was to run the Bitcoin cryptocurrency in a decentralized and secure way

- It is a distributed data structure characterized by:
  - data redundancy
  - check of transaction requirements before validation
  - recording of transactions in sequentially ordered blocks
  - ownership based on public-key cryptography
  - immutability
  - a transaction scripting language, associated to the transactions – the corresponding program is executed by all nodes

Smart Contracts (SC)

- The software associated to transactions and running on the Blockchain
- The SC run in every node
- **All executions must produce the same result**
- The calls and the storage modifications are recorded
- A SC cannot access any device or network
- The figure outlines the Ethereum approach for SC



Software Engineering for dApps

- In the past few years, there has been a strong increase of interest in cryptocurrencies, in Blockchain applications and in Smart Contracts
- This led to a huge inflow of money and of startup ideas
- Many projects were born and quickly developed software
- The scenario is that of **a rush to be the first on the market**, fearing of missing out
- This **unruled and hurried** software development does not assure neither software quality, nor that the basic concepts of software engineering are taken into account

Goals

- We propose a **software development process** to:
  - Gather the requirements
  - Analyze, Design
  - Develop, Test
  - Deploy **Blockchain applications**
- The process is based on Agile practices
- It makes also use of more formal notations, modified to represent specific concepts found in Blockchain development

BOS Design Method — Main Steps

- Steps 1-3: **Gather requirements** (without assuming the use of a blockchain)
- Step 4: Divide the system in **two subsystems**:
  - Step 5: the **blockchain** system (SC)
  - Step 6: the **external** system (server, client, GUI)
- Step 7: **Test** the two subsystems
- Step 8: Integrate and deploy

Steps 1 and 2

1. **Define** in one or two sentences **the goal of the system**. For instance: *To create a simple crowfunding system, managing various projects that can be financed using Ethers*

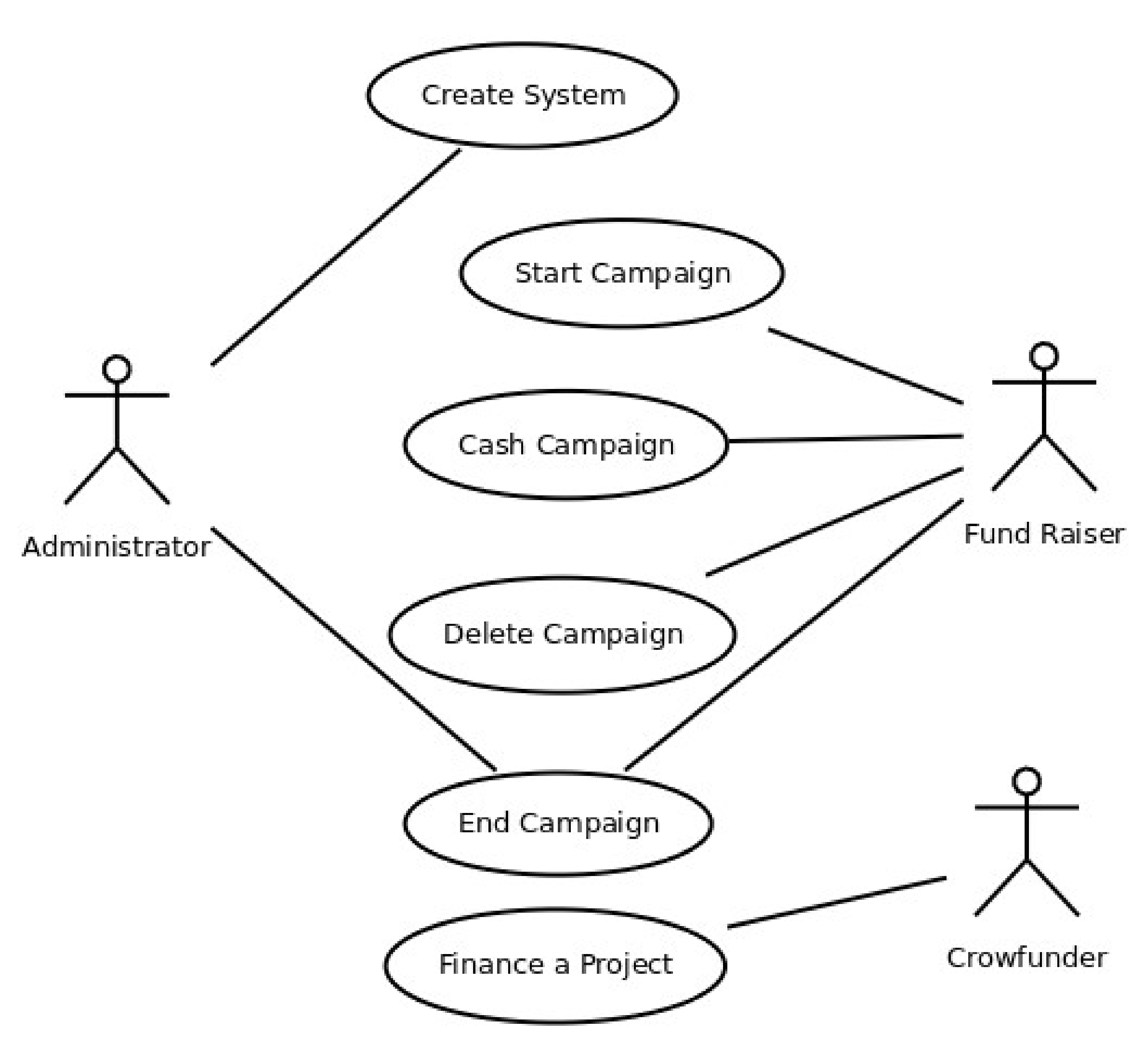
2. **Identify the actors** (human and external systems/devices). For instance:
   1. ***System Administrator****: s/he accepts the projects and their property; takes action in the case of problems*
   2. ***Fund Raiser****: they give the crowfunding project data, including the address receiving the money*
   3. ***Crowfunder****: they finance projects sending Ethers*

Step 3 – User Stories

- Write the system requirements in term of **user stories** or features:
  - Create System: The Administrator creates the contract, that register his address
  - Start Campaign: A Fund Raiser activates a CF project, giving its data: soft and hard cap, end date, address where to send money to
  - Cash Campaign: The Fund Raiser, if the time of the CF has expired, or if the hard cap has been reached, cashes out the Ethers given to the project

Step 3 – User Stories

- Delete Campaign: The Fund Raiser cancels the project; the Ethers are given back to Crowfunders
- End Campaign: The Administrator, or the Fund Raiser, if the time of the CF has expired and the soft cap has not been reached, ends the project; the Ethers are given back to Crowfunders
- Finance a Project: a Crowfunders sends Ethers to a project

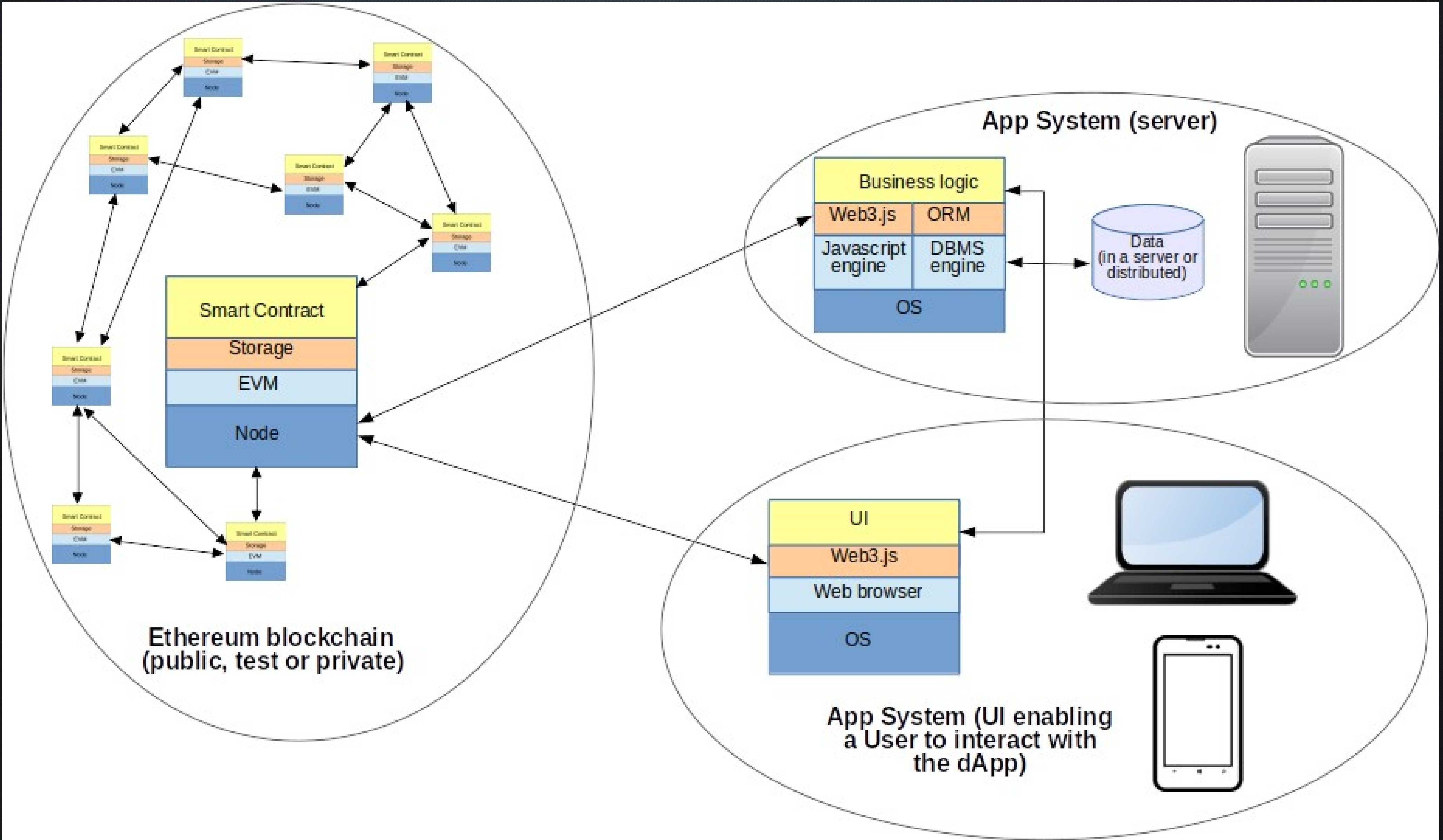


UML Use Case Diagram (User Stories)

Step 4 - Divide into SC system and external system

- Divide the system in two separate systems:
  - The Blockchain system, composed by the SCs
  - The external system that interacts with the first, sending transactions to the Blockchain and receiving the results
- The SC system interacts with the outside exclusively through blockchain transactions.
  - It has actors, recognized by the respective address
  - It can use libraries and external contracts
  - It can generate transactions to other contracts, or can send Ethers
- The client / server system is the one described in the previous steps
  - But it adds the interface to the SCs

A Typical dApp Architecture



Step 5 - Design of the SC subsystem

- **Redefine** the actors and the user stories
- Define the **decomposition** in SCs (one or more)
- For each SC, define the structure, the flow of messages and Ether transfers, the state diagram (if needed), the data structure, the external interface (ABI), the events, the modifiers...
- Define the tests and the **security assessment practices**

Step 6 – Design of the external subsystem

- Redefine the actors and the user stories, adding the new (passive) actors represented by the SCs
- Decide the architecture of the system
- Define the decomposition in modules, and their interfaces
- Define the User Interface of the relevant modules
- Perform a detailed design of the subsystem
- Perform a security assessment

BOS Design Method – Steps 7 and 8

7. Code and test the systems; in parallel:
   - Write and test the SCs, starting from their data structure and functions;
   - Implement the USs of external subsystem with an agile approach (Scrum or Kanban);
8. Integrate, test and deploy the overall system.

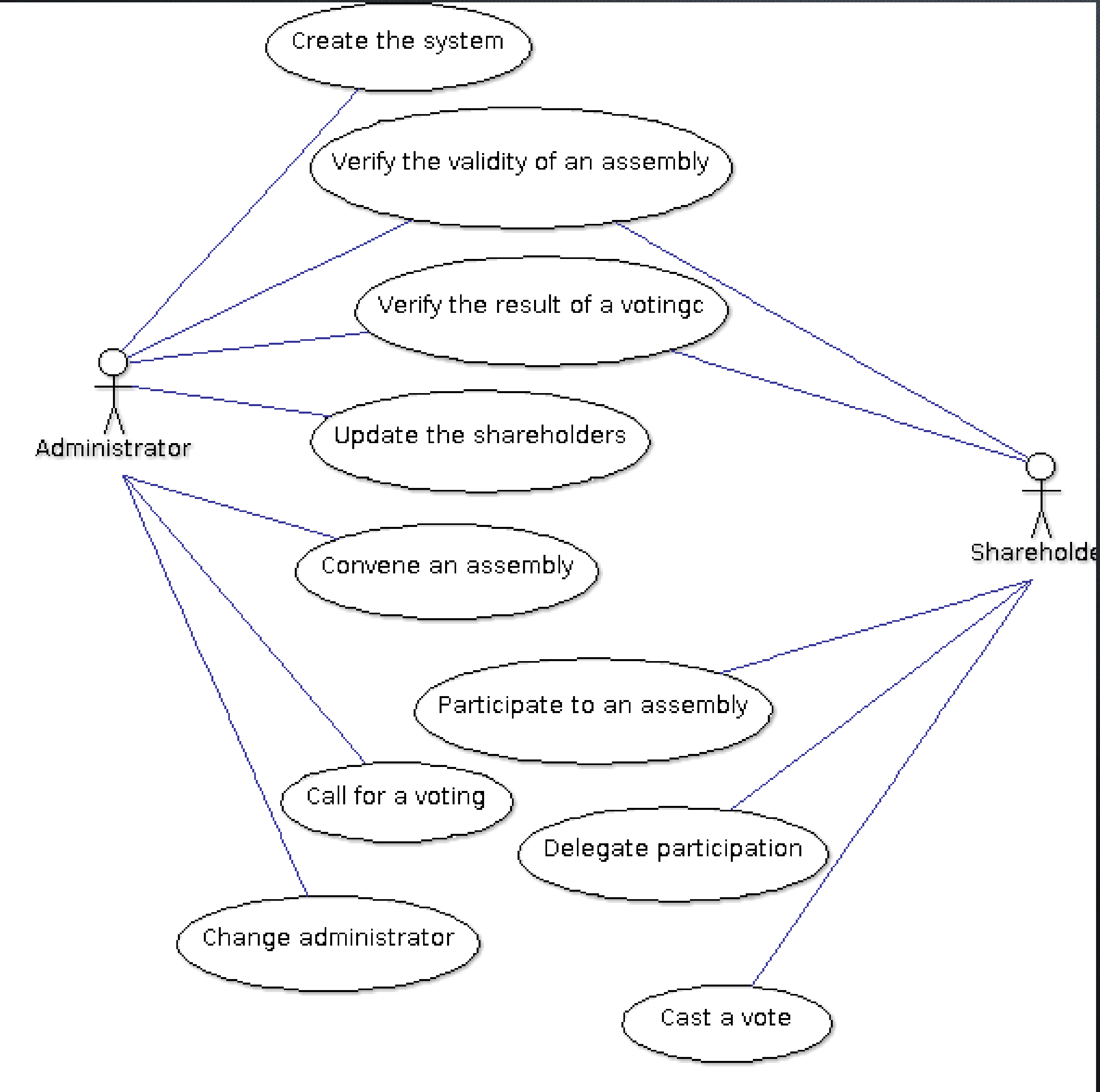
A Case Study: Corporate voting management

1. GOAL OF THE SYSTEM:

- To manage in a simplified way voting in corporate assemblies

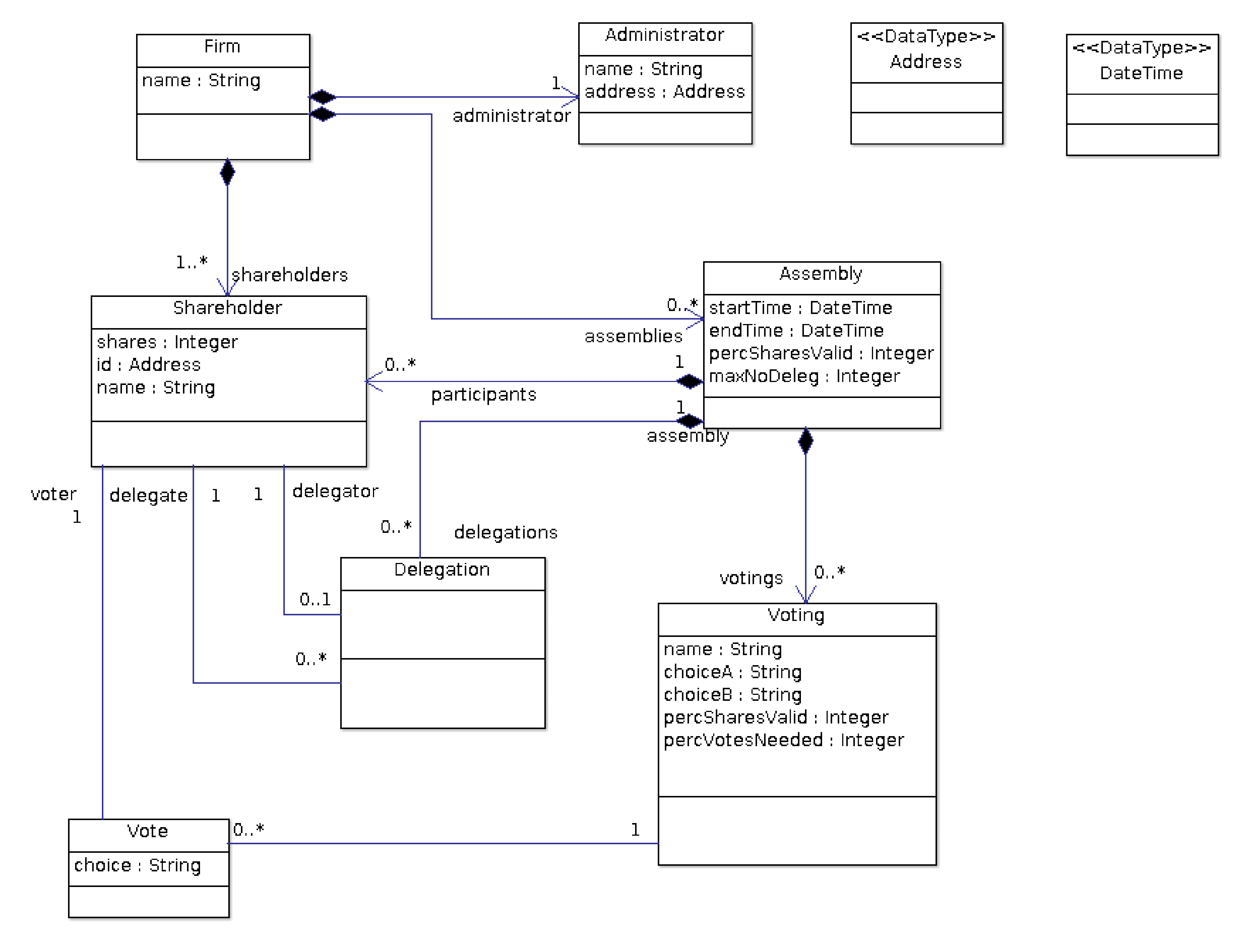
2. IDENTIFY ACTORS:

- **Corporate administrator**: manages the system, manages the shareholders and their shares, convenes assemblies, calls for votings
- **Shareholder**: participates to assemblies, casts his votes, delegates participation to assemblies



SECR

Step 3. User Stories



Step 3. The data structure representing this system shown using a UML class diagram

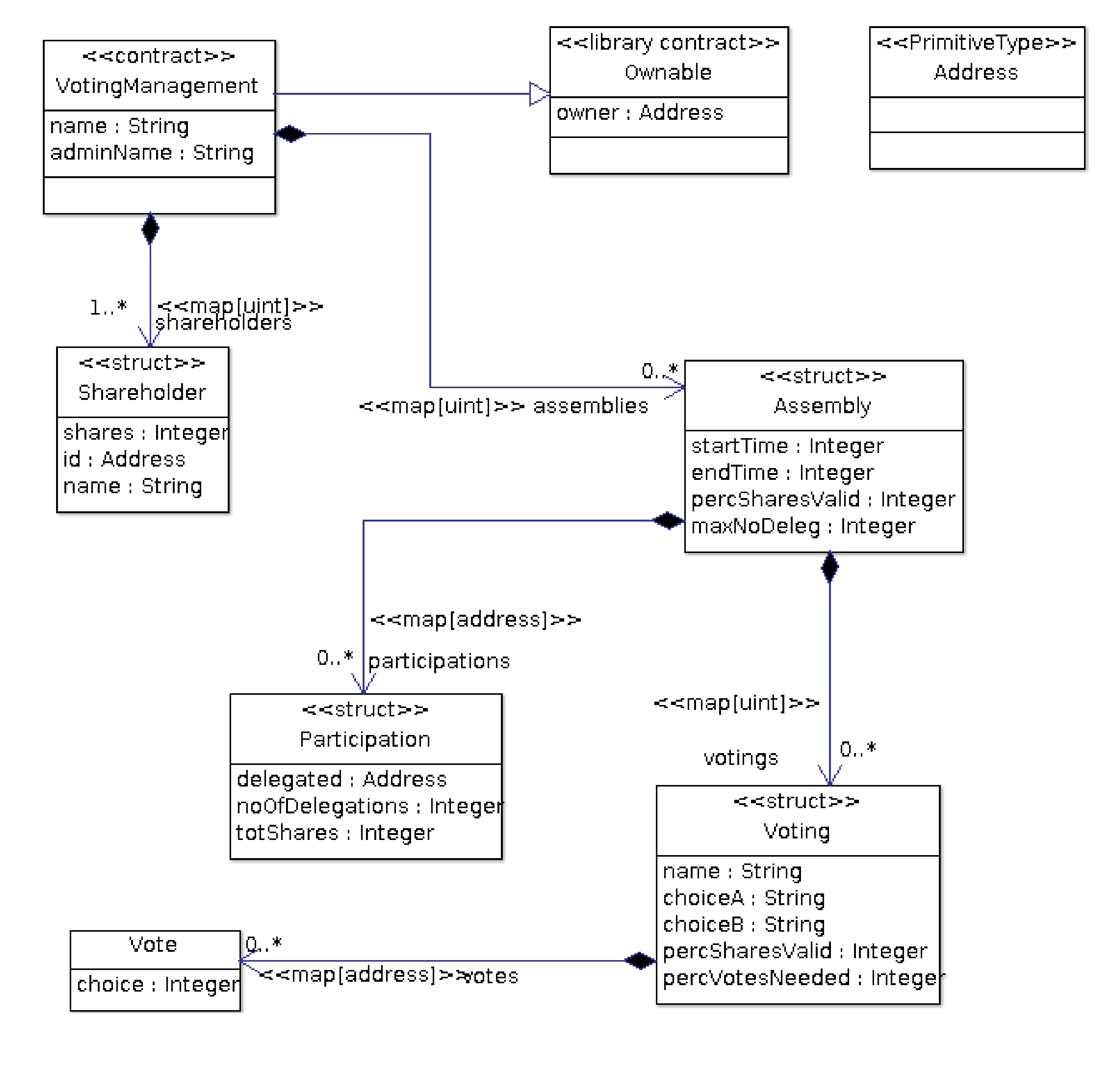
Step 4. Divide the system

- In this case the subdivision is trivial, because all US make use of Smart Contracts.
- The DApp subsystem US are the same. Each includes the Blockchain as further Actor.
- The Blockchain subsystem US are the same. The identifiers of the Actors are their unique adresses:
  - **Corporate administrator**: her/his address is at first the address that creates the contract, and then possibily a further address set by the *Change administrator* US
  - **Shareholder**: their addresses are specified and managed by the Administrator.

Step 5. Design of the SC subsystem

- The system is quite simple, so a single SC is the best option
- Following a SC standard, the "Ownable" standard contract is used to manage the ownership of the SC, held by the Administrator, who creates the SC

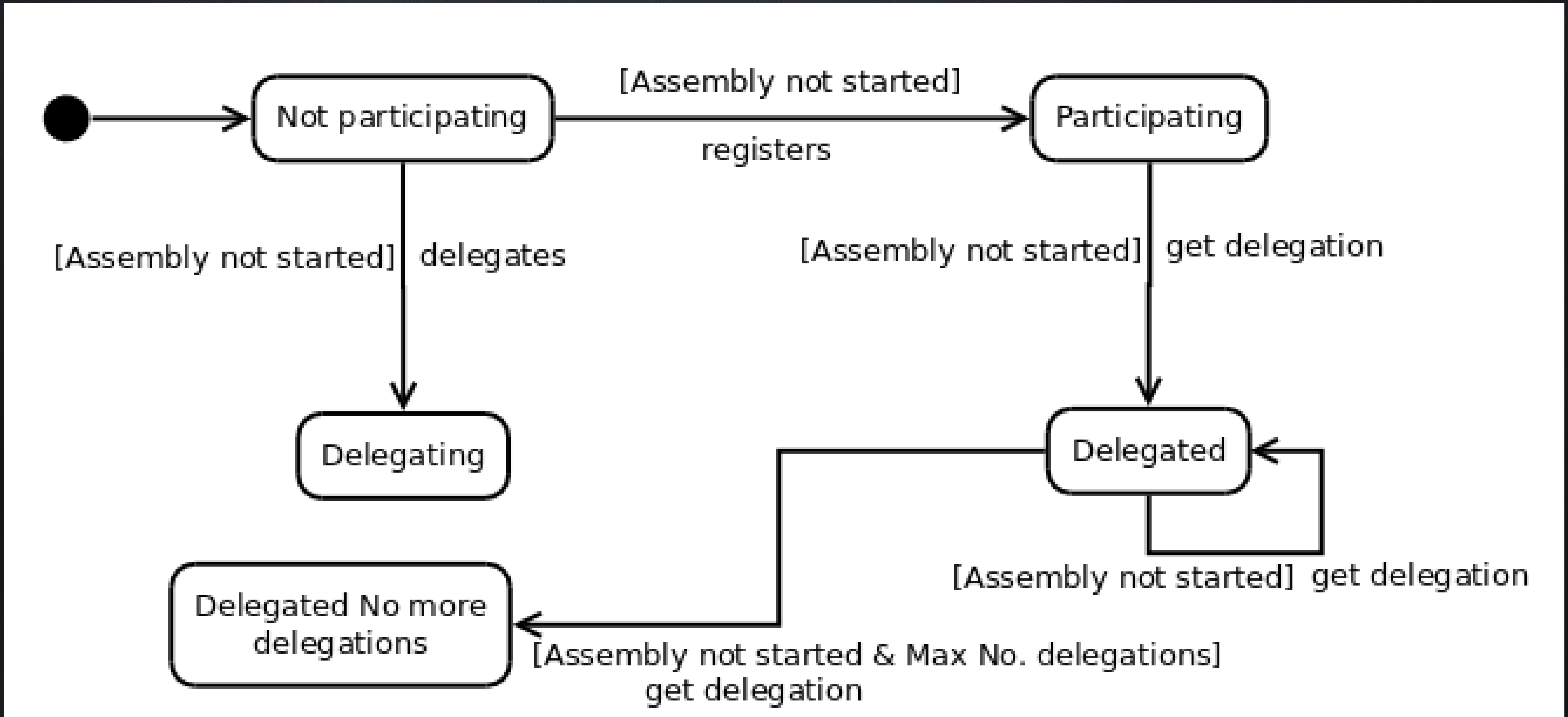
```
contract Ownable {
    address public _owner;
    modifier onlyOwner() {
        require(msg.sender == _owner);
        _;
    }
    constructor() public {
        _owner = msg.sender;
    . . .
}
```



Step 5. Design of the SC Data structure of the SC shown using a modified UML class diagram

UML State diagram of a Shareholder

- showing the possible ways of her/his participation to an assembly:



Step 5. The Dynamic model of the SC subsystem

- **Modifiers**:
  - onlyOwner()
  - onlyShareholder()
  - onlyOwnerOrShareholder()
  - assemblyRunning() – enforces that there is actually an assembly running at the time of the call
  - assemblyNotRunning() – enforces that there is no assembly running at the time of the call
- **Functions**:
  - AndSoOnAndSoOn...()

Step 6. Design of the external subsystem (ESS)

- Actors of the ESS:
  - Administrator
  - Shareholder
  - SC subsystem
- Architecture:
  - A responsive application for managing the system
  - An app for the shareholders (voting and delegating)
- The app GUIs are designed
- The apps are developed using the Ethereum API web3.js library and a dev environment of choice

Steps 7 and 8: coding, testing, deploying the system

- Here we give some details of SC security assessment
- We apply a checklist to SC design and code, to assess their security against known attacks:
  - Minimize external calls and check for reentrancy
  - Follow the "checks-effects-interactions" pattern
  - Check the proper use of assert(), require(), revert()
  - Check if there are ways to make the SC permanently stuck due to gas consumption above the limit
  - Have some way to update the contract in the case some bugs will be discovered
  - . . .

Conclusions

- Despite the huge effort presently ongoing in developing DApps, software engineering practices are still poorly applied
- A sound software engineering approach might greatly help in overcoming many of the issues plaguing blockchain development:
  - Security issues
  - Software quality and maintenance issues
- Researchers in software engineering have a big opportunity to start studying a field that is very important and brand new
- Blockchain firms, including ICO startups, could develop a competitive advantage using SE practices since the beginning

Contact Me

- Michele Marchesi
- Email: marchesi@unica.it