Architect a Blockchain Supply Chain Solution

# Introduction

This document explains the implementation of a solution to identify and track coffee through its supply chain. For the front-end, a combination of HTML and javascript is used. For the back-end, a series of smart contracts written in Solidity have been deployed on the Ethereum test network Rinkeby.

# UML diagrams

This section explains the structure of the back-end (the Solidity smart contracts). In a nutshell, the contract “SupplyChain” implements all the functionalities that the front-end interacts with. It inherits from 5 other contracts:

* Ownable: a contract that makes it possible that only the address that deployed the contract can execute some operations
* FarmerRole, DistributorRole, RetailerRole, ConsumerRole: contracts that implement the roles for the different participants in the supply chain.

## Sequence diagram

The sequence diagram illustrates how the 4 different roles (farmer, distributor, retailer, consumer) interact in the supply chain, in a normal use case.

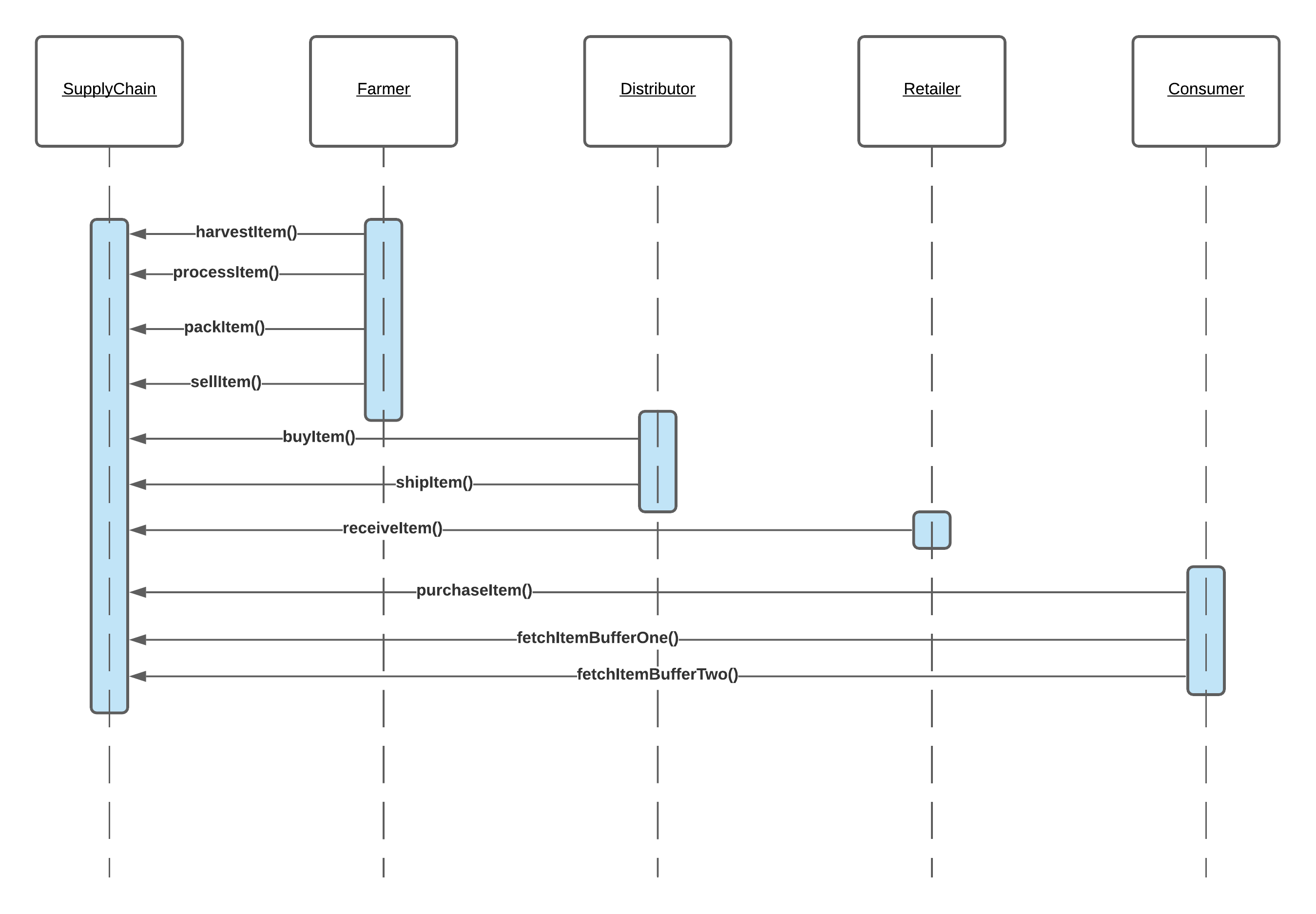


Figure - Supply Chain Sequence Diagram

## Activity diagram

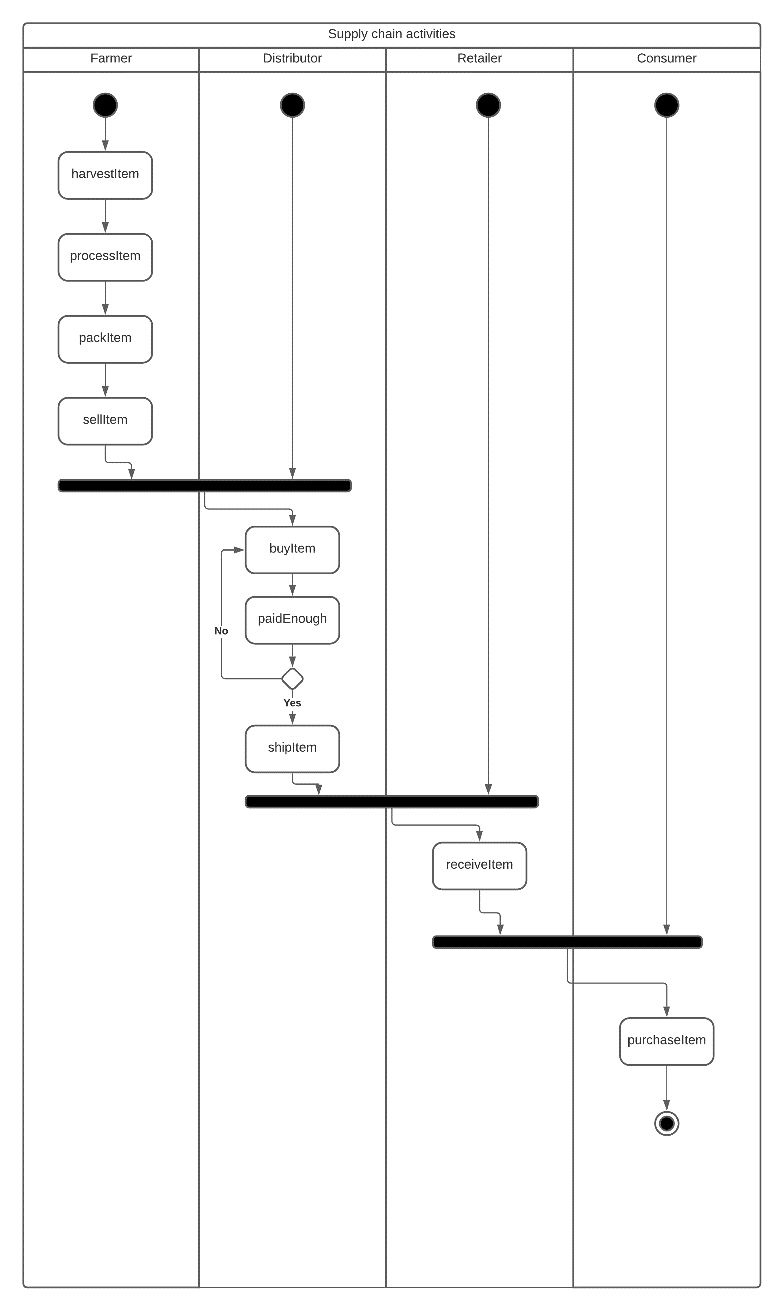


Figure - Supply Chain Activity Diagram

## State diagram

Note that compared to the boiler plate initially provided, the state “Uninitialized” has been added. The reason is to prevent an existing item in the supply chain to be harvested twice.

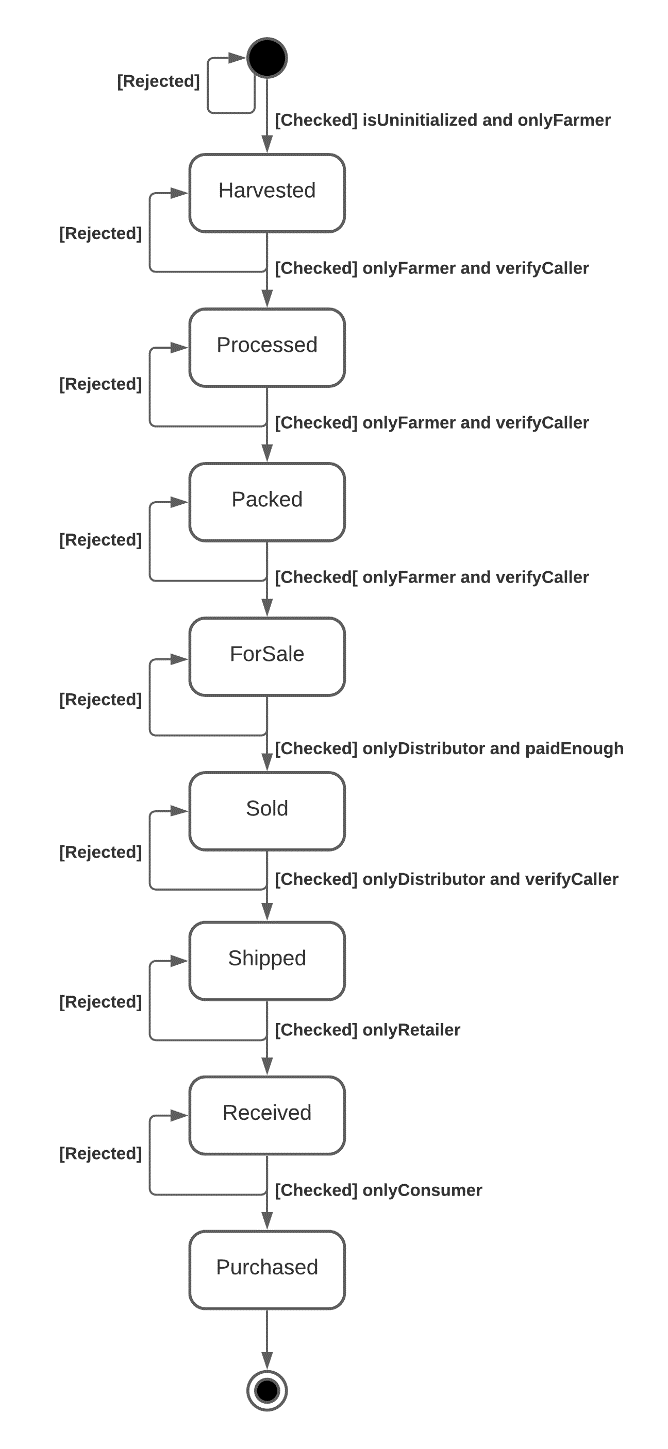


Figure - Supply Chain State Diagram

## Class diagram

Note that even though “Roles” is a library and not a contract, it is represented as a base class for the 4 roles. The reason is that Roles only has internal member functions, so it will be embedded in the contract using it, which is similar to the inheritance mechanism.

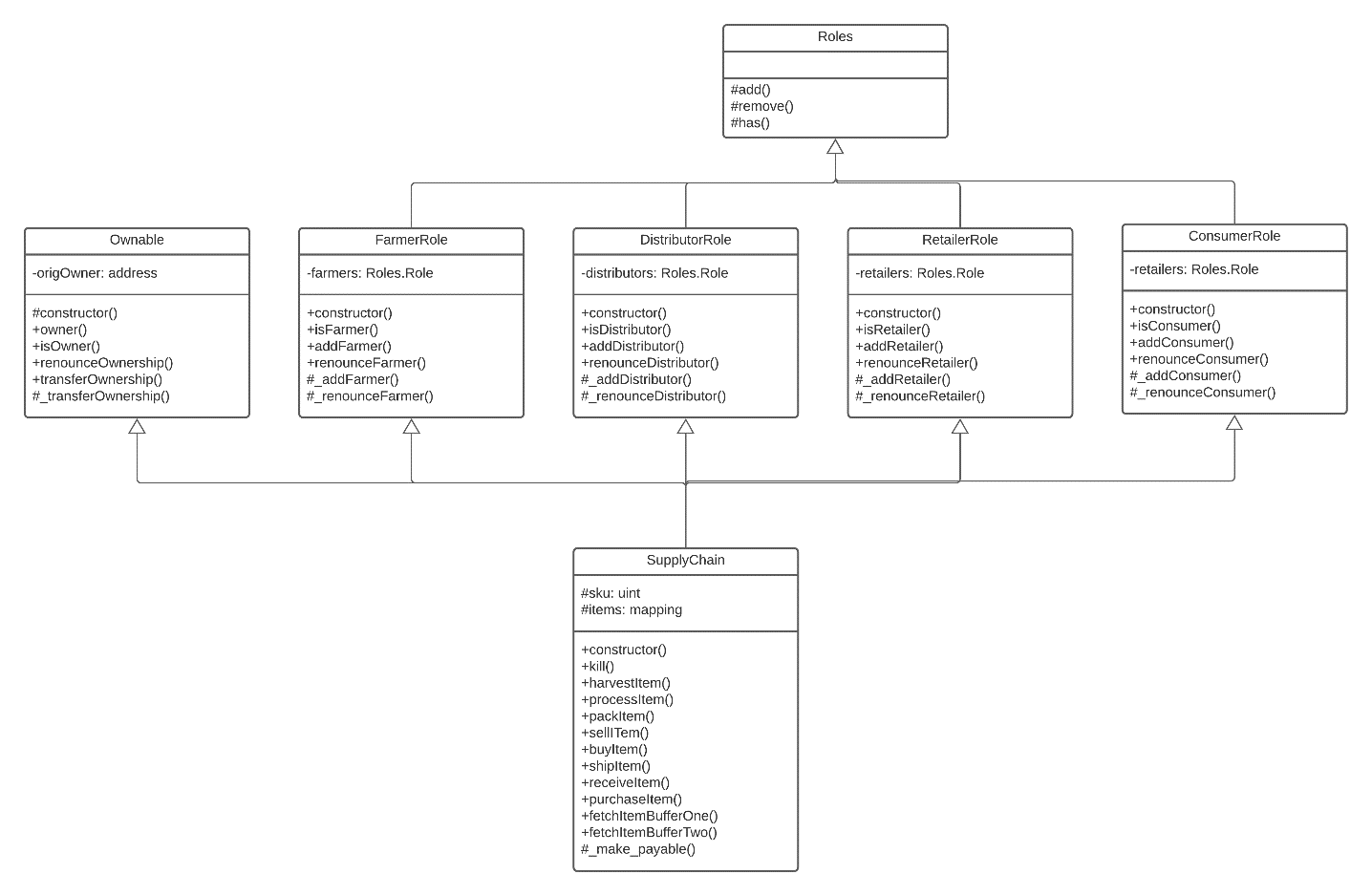


Figure - Supply Chain Class Diagram

# Libraries

No library was used (although Roles and Ownable could have been obtained from openzeppelin).

For dependencies, see “package.json” file.

# IPFS

IPFS was not used.