Proxy Smart Contract

Solidity with Inline Assembly

Andrew Starling | 100191710

Problem

What is the problem that a *proxy contract* is solving?

Consider a software developer who creates a smart contract and deploys it to the Ethereum blockchain.

Later it is determined that there is a bug in the smart contract, or perhaps a new feature must be included and a change to the code needs to be made. A dilemma arises, due to the immutable nature of the blockchain. It is impossible to upgrade the code of a smart contract which has already been deployed, however, it is possible to utilize a proxy contract, that will *simulate* a direct upgrade to a smart contract. This proxy solution must be implemented initially in order to make use of upgradeability. How can this be accomplished?

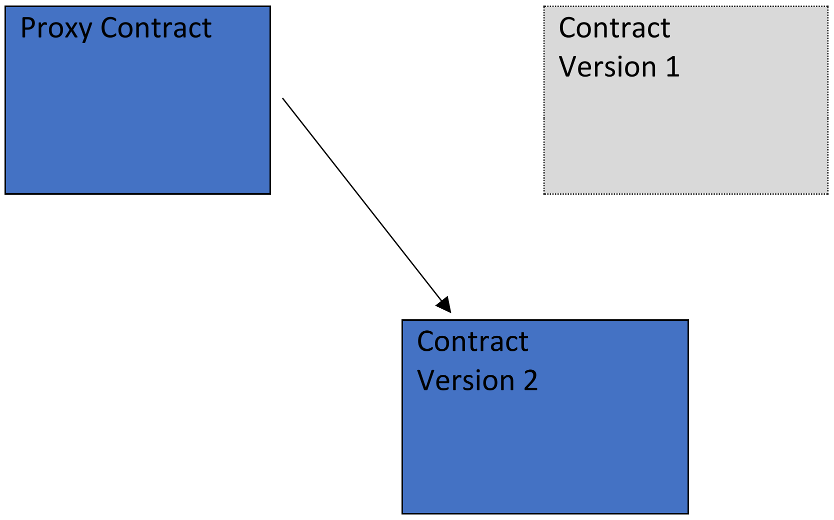
User → deploys Proxy Contract C_P

User → deploys Smart Contract C_1

User → function call → C_P → forwards call to → C_1

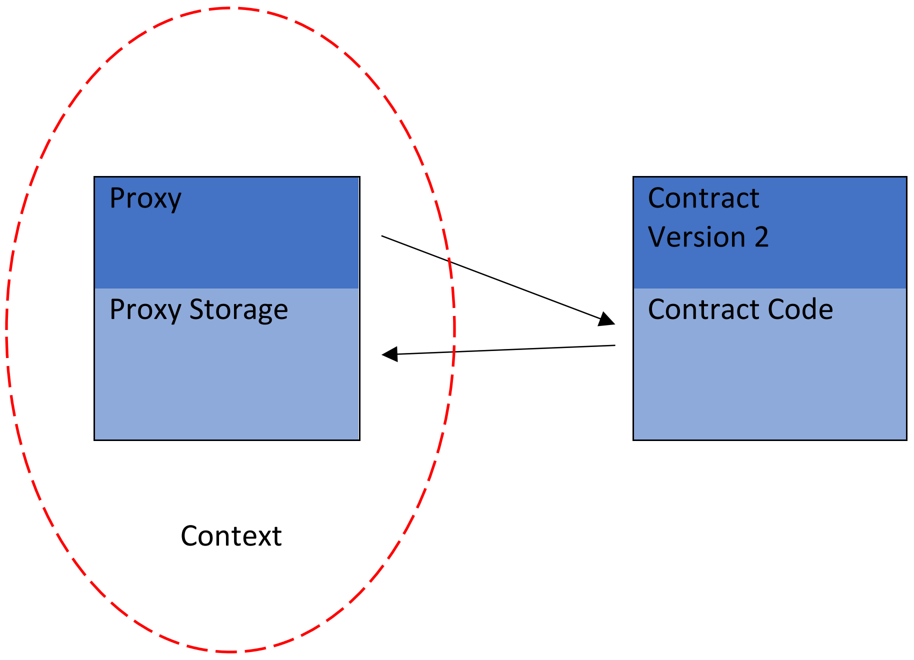
User → deploys a new version of smart contract C_2

User → function call → C_P → forwards call to → C_2



A Proxy contract will redirect to the latest deployed contract code. This process can be repeated, i.e. new versions of the deployed contract can be deployed and referenced, effectively allowing for a contract to be upgraded, bypassing the inherent immutability of the blockchain. The Proxy is updated, to reference the new contract address.

This is made possible by DelegateCall, which loads code from the contract being called. Storage is done in the calling contract. The proxy contract holds the state of the upgradeable contract. In other words, the code of the latest deployed contract is executed within the context of the proxy. In the following diagram, the code in Contract Version 2 will modify storage of the proxy.



Solution

Utilize a proxy which supports upgradeable smart contracts. In this example, there will be 3 contracts: *GreetingV1.sol*, *GreetingV2.sol* and *ProxyContract.sol*.

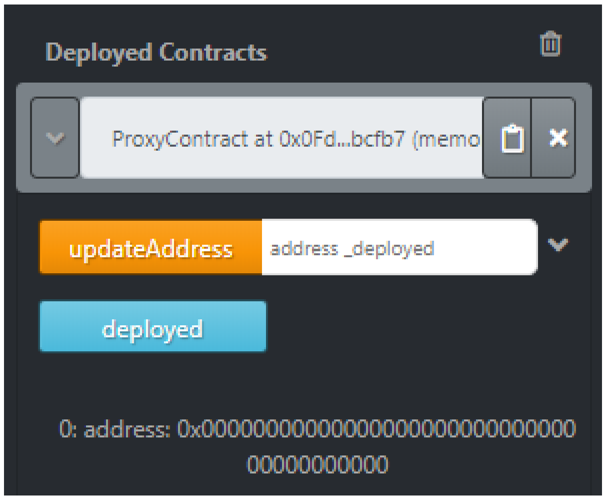
We can accomplish this using the following steps:

1. Deploy the proxy contract ProxyContract.sol.
2. Deploy version 1 of the smart contract GreetingV1.sol.
3. Call the function *updateAddress* in the Proxy Contract to point to *GreetingV1.sol*.
4. Call the function *setGreeting* in *GreetingV1.sol*, <u>but not directly</u>: call *setGreeting* in the proxy. <u>But this function does not exist in the proxy</u>, which triggers the fallback function performing a deletagecall to the address of *GreetingV1.sol* saved in the *addr* variable.

5. The proxy contract will load the code of the *setGreeting* function from *GreetingV1.sol* and execute it. *GreetingV1.sol* does not store any data, the proxy contract does.
6. Calling *getGreeting* through the proxy will return the newly set greeting. Although calling *getGreeting* directly in *GreetingV1.sol* will return the previous value in *GreetingV1.sol* because the updated value is stored in the proxy.
7. In the future, if an upgrade to the smart contract is necessary due to a bug fix or new functionality, deploy a new version of the smart contract: *GreetingV2.sol*.
8. Call function *updateAddress* in the Proxy Contract to point to *GreetingV2.sol*.
9. The proxy contract is delegating calls to the new version of the contract, and the state persists in the proxy across the updates.

Testing Walkthrough

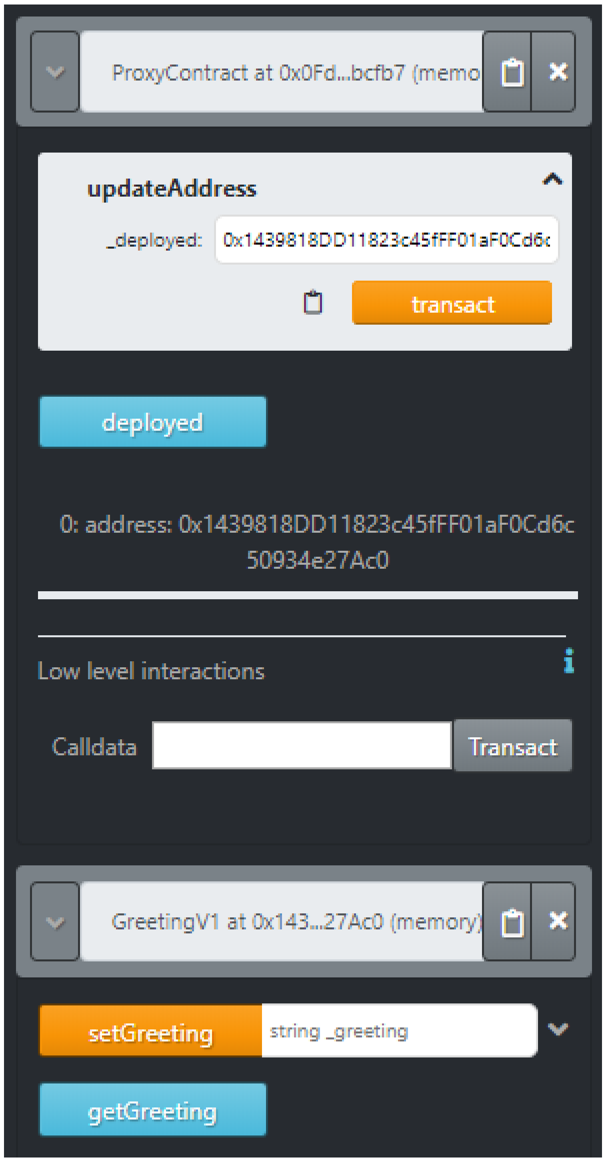
➔ Deploy the proxy contract.



```
0: address: 0x0000000000000000000000000000000000000000
```

Initially the proxy is not pointing to version 1 of the smart contract This will happen in the next step.

➔ Deploy version 1 of the smart contract and execute the function updateAddress in the proxy to point to it.


ProxyContract at 0x0Fd...bcfb7 (memo

updateAddress

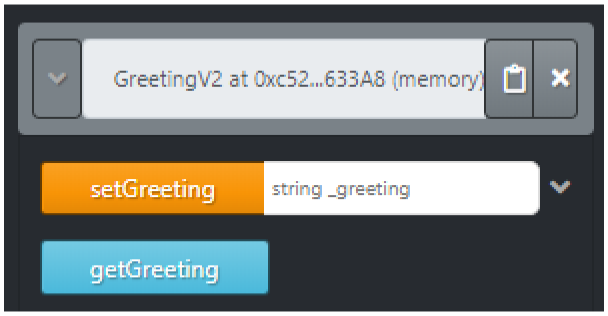
_deployed: 0x1439818DD11823c45fFF01aF0Cd6c

0: address: 0x1439818DD11823c45fFF01aF0Cd6c50934e27Ac0

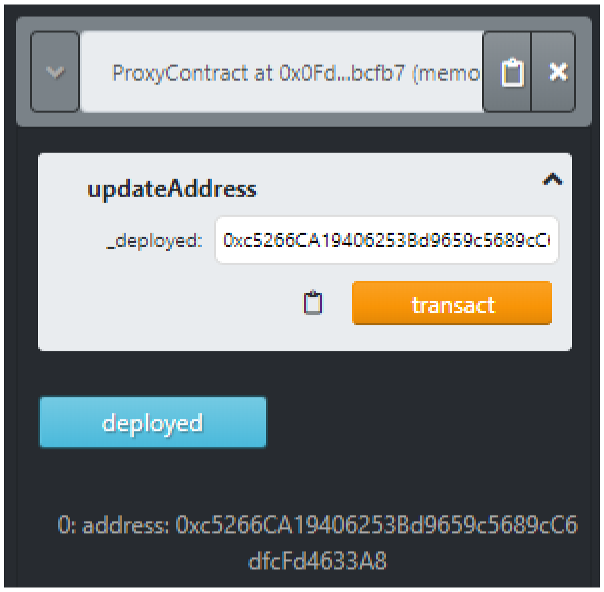
GreetingV1 at 0x143...27Ac0 (memory)

The proxy contract is now pointing to version 1 of the smart contract.

➔ In order to upgrade version 1 to version 2 of the smart contract, deploy version 2.


GreetingV2 at 0xc52...633A8 (memory)

➔ And execute the function updateAddress in the proxy to point to the upgraded version of the smart contract.



```
0xc5266CA19406253Bd9659c5689cC
0: address: 0xc5266CA19406253Bd9659c5689cC6
dfcFd4633A8
```

Gas Estimates

ProxyContract.sol

```
"Creation":
     "codeDepositCost": "67800",
     "executionCost": "20352",
     "totalCost": "88152"

 "External":
     "": "infinite",
     "deployed()": "459",
     "owner()": "415",
     "updateAddress(address)": "20473"
```

Security

It is a requirement that in order to point the proxy contract at a new upgraded version of the smart contract the owner of the contract must be the same as the deployer of the original proxy contract.

```
require(owner == msg.sender, 'must be owner to update address');
```