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 at time T_0.

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 set-up a proxy contract, that will *simulate* a direct upgrade to the original smart contract.

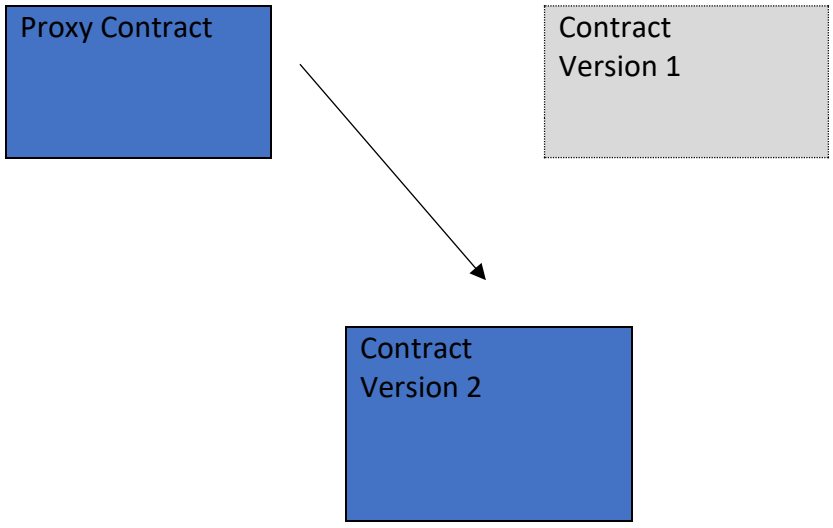
How can this be accomplished?

User → Proxy Contract C_P → Deployed Smart Contract C_1

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

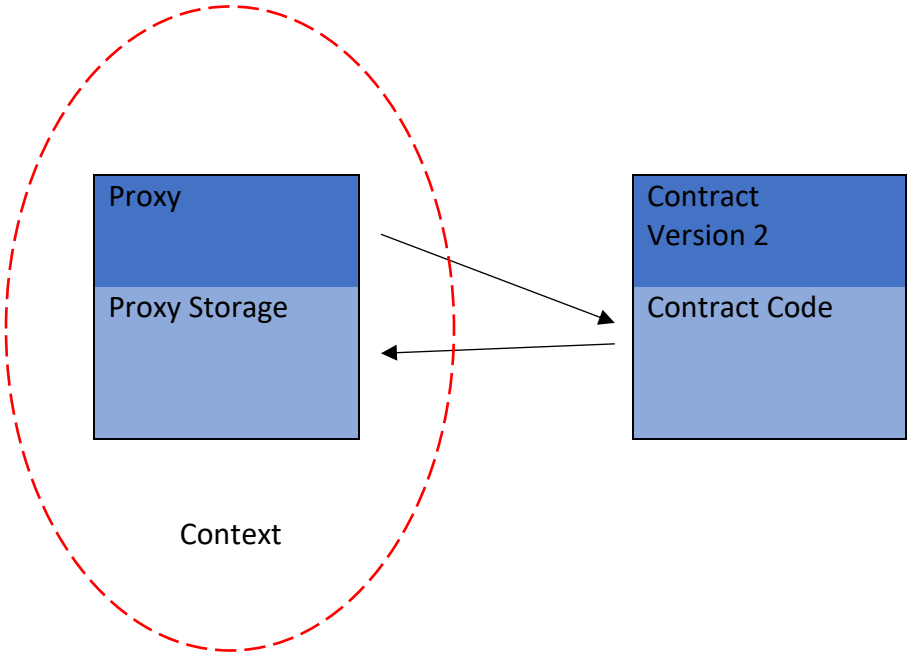
User → deploys a new version of smart contract C_2

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, thee 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: *Greeting_v1.sol*, *Greeting_v2.sol* and *ProxyContract.sol*.

Say we have deployed *Greeting_v1*, but determine that the code is incorrect and we want to upgrade it to *Greeting_v2*. We can accomplish this using the following steps:

1. Deploy the proxy contract ProxyContract.sol.
2. Deploy version 1 of the smart contract Greeting_v1.sol.
3. Call the function *updateAddress* in the Proxy Contract to point to *Greeting_v1.sol*.

4. Call the function *setGreeting* in *Greeting_v1.sol*, but not directly: call *setGreeting* in the proxy. But this function does not exist in the proxy, which triggers the fallback function performing a deletagecall to the address of *Greeting_v1.sol* saved in the *addr* variable.
5. The proxy contract will load the code of the *setGreeting* function from *Greeting_v1.sol* and execute it. *Greeting_v1.sol* does not store any data, the proxy contract does.
6. Calling g*etGreeting* through the proxy will return the newly set greeting. Although calling g*etGreeting* directly in *Greeting_v1.sol* will return the previous value in *Greeting_v1.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: *Greeting_v2.sol*.
8. Call the update function in the Proxy Contract to point to the newly deployed Smart Contract version 2, by calling *updateAddress* in the Proxy Contract to point to *Greeting_v2.sol*.

Gas Estimates


```
{
        "Creation": {
                "codeDepositCost": "47000",
                "executionCost": "20334",
                "totalCost": "67334"
        },
        "External": {
                "": "infinite",
                "updateAddress(address)": "20451"
        }
}
```

Code

Key Findings

Visual Data

Insert any data tables/charts/graphs/infographics etc.

Conclusion

Time to wrap it up. What is your conclusion? How would you synthesize all the information into something even the busiest CEO wants to read? What are the key takeaways? How does your product/service/methodology uniquely address the issues raised by your study?

Key Takeaways

- Takeaway #1
- Takeaway #2
- Takeaway #3