Designing future-proof smart contract systems

Jorge Izquierdo
Devcon3
Aragon

- **Decentralized organizations** platform built on Ethereum.
- Usable by **non-technical** users.
- Allow **extendability** of the system with third party on-chain modules.
Future-proof smart contracts

- **Dumb contracts** are the best smart contracts
- EVM is expensive, optimize for **gas savings** (deploy and usage)
- Contracts need to be **upgraded**

- **Goal:** cheap, upgradeable, yet very simple contracts
The case against upgradeability

- Changing the rules on a live contract
- **Trust** required on the entity that can upgrade
- **Front-running** with an upgrade
Why upgradeable contracts

- Extremely young technology
- Solving unanticipated bugs that can result in irreversible loss of funds at the contract level
- Need to add new features based on user feedback
Doing upgrades right

- Not rely on just one entity
- Time delayed to allow for *vetting* and *auditing*
- **Governance** process
Solidity libraries

- `delegatecall` under the hood to linked library
- Using semantics simulates calling methods on an object
- Library is deployed once and securely used by many contracts
- Separation of logic domains in a contract
- Allows for bigger contracts

Aragon Blog: [Library Driven Development](https://library drivendev.org)
Upgradeable libraries

github.com/maraoz/lib

Zeppelin + Aragon
Upgradeable libraries

Pros:
- Transparent for developer
- Allows to 'modify' the linked library

Cons:
- Main contract ABI cannot be modified
- Data structures are fixed
Delegate Proxy

- Run another contract's logic in the context of a contract.
Implementation (EIP 211)

```solidity
contract DelegateProxy {
    function delegatedFwd(address _dst, bytes _calldata) internal {
        assembly {
            switch extcodesize(_dst) case 0 { revert(0, 0) }

            let result := delegatecall(gas, _dst, add(_calldata, 0x20), mload(_calldata), 0, 0)
            let size := returndatasize

            let ptr := mload(0x40)
            returndatacopy(ptr, 0, size)

            switch result case 0 { revert(ptr, size) }
            default { return(ptr, size) }
        }
    }
}
```

aragon-core: `common/DelegateProxy.sol`
Delegate Proxy flavors

- Static delegate proxy: *forwarders*
- *Upgradeable* proxies
Static forwarders

- Very cheap to deploy 'clone contracts'
- Useful for contracts with a low number of interactions

Original idea: Vitalik’s DELEGATECALL forwarders
Solidity forwarders

```solidity
import "./DelegateProxy.sol";

contract Forwarder is DelegateProxy {
    address constant target = 0xbeef;

    function () payable {
        delegatedFwd(target, msg.data);
    }
}

contract ForwarderFactory {
    function clone() returns (address) {
        return address(new Forwarder());
    }
}
```
Gas overhead

- Added gas per call
  - 700 `delegatecall`
  - $3 + 3 \times \left(\frac{\text{returned datar size}}{32}\right)$ `returndatacopy`

- Deploy gas
  - ~66k gas `deploy` for barebones version
  - ~97k gas `deploy` Solidity (~87k using factory)

- 1M gas for contract deploy, ~1.3k calls break even
ENS Deed case study

- Deed create + parametrization = 620,741 gas
- Forwarder create (solidity) + setup call = 173,697 gas
- 340,565 found deed contracts
- Total gas saved = 152,247,539,860 gas
- Average 20 gwei gas price = 3044.95 ETH
Upgradeable Proxies

```
contract ProxyStorage {
    address target;
}

cell contract UpgradeableProxy is ProxyStorage, DelegateProxy {
    function UpgradeableProxy(address _target) {
        target = _target;
    }

    function () payable {
        delegatedFwd(target, msg.data);
    }
}

cell contract UpgradeableContract is ProxyStorage {
    function upgrade(address _newCode) {
        // do some checks here
        target = _newCode;
    }

    function foo() {
        // interesting upgradeable logic
    }
}
```

Original idea: Nick Johnson’s upgradeable.sol
Storage in Delegate Proxies

- Storage layout must respect Proxy contract's layout.
  - Inherit Proxy's storage

- Data structures must be designed thinking on upgradeability
Solidity Storage slots review

- Storage counter starts at 0
- Reverse inheritance graph order
- Solidity packs contiguous smaller types to 32 bytes
- Structs are stored as inline types
  TODO: addresses slots
Arrays

• Static length arrays behave just like normal types
  • `uint256[2] == uint256, uint256`

• Dynamic length arrays:
  • Store length in `p`
  • Array item position at `sha3(p) + arrayP`
  • Structs stored as inline items, adding to `arrayP`
  • Arrays in arrays follow same property

```solidity
contract StorageArray {
  uint256[3] a; // slots 0, 1, 2
  uint256[] b;  // slot 3 (array length)

  function storeUint() {
    b.length = 10; // slot 3 = 10
    b[0] = 1;  // slot sha3(3) = 1
    b[5] = 2;  // slot sha3(3) + 5 = 2
  }

  struct C {
    uint256 a;
    uint256 b;
    uint256[] c;
  }

  C[] cs;

  function storeStruct() {
    cs.length = 2; // slot 4 = 2
    cs[0].a = 1;    // slot sha3(4) = 1;
    cs[1].a = 1;    // slot sha3(4) + 3 = 1
    cs[1].b = 2;    // slot sha3(4) + 4 = 2
    cs[1].c.length = 1; // slot sha3(sha3(4) + 5) = 1
    cs[1].c[0] = 2; // slot sha3(sha3(4) + 5) = 2
  }
}
```
Mappings

- Values stored $\text{sha3}(\text{key}, p)$
- Nested mappings: $p$ is where the mapping would have been stored if it was a normal value

```solidity
contract StorageMapping {
    mapping (address => uint256) m;
    mapping (address => mapping (address => uint256)) mm;

    struct Map {
        mapping (address => uint256) m;
    } Map[] maps;

    function StorageMapping() {
        m[0x12] = 10; // slot Sha3(0x12, 0) = 10
        mm[0x12][0x34] = 11; // slot Sha3(0x34, Sha3(0x12, 1)) = 11
        maps.length = 1;
        maps[0].m[0x12] = 12; // slot Sha3(0x12, Sha3(2) + 0) = 12
    }
}
```
Mappings

- Values stored $\text{sha3(key, p)}$
- Nested mappings: $p$ is where the mapping would have been stored if it was a normal value

```solidity
contract StorageMapping {
    mapping (address => uint256) m;
    mapping (address => mapping (address => uint256)) mm;

    struct Map {
        mapping (address => uint256) m;
    } Map[] maps;

    function StorageMapping() {
        m[0x12] = 10; // slot sha3(0x12, 0) = 10
        mm[0x12][0x34] = 11; // slot sha3(0x34, sha3(0x12, 1)) = 11
        maps.length = 1;
        maps[0].m[0x12] = 12; // slot sha3(0x12, sha3(2) + 0) = 12
    }
}
```
Warnings

• Adding just one storage value anywhere will increase $p$ and break all storage.

• Failure will be silent, storage will be randomized.

• Always append new storage at the end.
Upgrade example

- Deploy Payroll
- Create 2 employees
- Upgrade to Payroll2
- Employee 1 joinDate = Employee2 salary
- Employee2 salary = 0
Upgrade example

- Deploy PayrollM
- Create 2 employees
- Upgrade to Payroll2M
- Salaries are correct
- Join date before update is 0
AragonOS

- Tiny kernel
- Upgradeable business logic on edges: apps
Kernel

- Context dependent **ACL**
- **Upgradeable** apps
ACL

- Usability/security balance
  - Can't rely on just one superuser or owner
  - Protect users from destructive actions
  - Different governance mechanism for different actions

- Purely address based whitelist
  - Apps as entities
  - Complex authentication on a second layer
  - New governance systems can be plugged in
ACL

- App defined roles for capabilities
- Granted permissions have a different parent
  - Parent can revoke the permission
  - If grantee is parent, grantee can re-grant it
import "@aragon/core/contracts/apps/App.sol";

contract Counter is App {
    /// Events
    event LogIncrement();
    event LogDecrement();

    /// State
    uint256 public value;

    /// ACL
    bytes32 constant public INCREMENT_ROLE = sha3("increment");
    bytes32 constant public DECREMENT_ROLE = sha3("decrement");

    function increment() auth(INCREMENT_ROLE) external {
        value += 1;
        LogIncrement();
    }

    function decrement() auth(DECREMENT_ROLE) external {
        value -= 1;
        LogDecrement();
    }
}
Upgrading the app

import "@aragon/core/contracts/apps/App.sol";

contract Counter is App {
    // Events
    event LogIncrement();
    event LogDecrement();

    // State
    uint256 public value;

    // ACL
    bytes32 constant public INCREMENT_ROLE = sha3("increment");
    bytes32 constant public DECREMENT_ROLE = sha3("decrement");

    function increment() auth(INCREMENT_ROLE) external {
        value += 1;
        LogIncrement();
    }

    function decrement() auth(DECREMENT_ROLE) external {
        value -= 1;
        LogDecrement();
    }
}
Putting everything together
Thanks!

wiki.aragon.one

github.com/aragon

aragon.chat