token estate

the future of real estate investing

From a non upgradable smart contract to an evolutive smart contract!

CONTACT

CHRISTIAN BROILLET

@cbroillet

MEETUP

Geneva-DevChain-UserGroup

@GenevaDUG

geneva-devchain

DevchainUserGroup

tokenestate.io

ADDRESS

Ch. du Pré-Fleuri 3
1228 Geneva
Switzerland

@token_estate

tokenestate

tokenestate

token-estate

SITE WEB

https://tokenestate.io/

Smart Contract Upgrade

tokenestate.io

- It's not easy to get Upgradeable Smart Contracts right.
- You should be able to fix bugs or enhance features, obviously.
- But not at the cost of losing immutability. It's this element of trust that nobody can change anything. Thinking of censorship, government bans, and so on...

Smart Contract Upgrade

tokenestate.io

- The problem is, no matter the way of the implementation, there is always some advantage and some drawback.
- In general there is a good reason for and against being able to upgrade contracts.

Smart Contract Upgrade Cons

tokenestate.io

- The most important part of immutability: It makes sure that nobody can make changes afterwards.
- This element of trust is what makes Ethereum and Smart Contracts so incredibly powerful.
- You release your code - and bam - it's there. On chain. Nobody can change it or take it down.

Smart Contract Upgrade Pros

tokenestate.io

- But on the other hand, recent hacks were mostly based on very simple programming errors.
- A lot of those bugs could be fixed very easily, if it was possible to upgrade these contracts in one way or another.
- Think about Parity Multisig Wallet hack...

Some solutions

tokenestate.io

1st level

- Make your code simple and put only what realy need to be on chain.
- Make your code modular.
- Have good testing strategies and tactics.
- Have an emergency stop to stop all operations during migration.

Some solutions

tokenestate.io

2nd level

- Smart contracts as services.
- Proxy Contracts.
- Separate Logic and Data Contracts.
- Partially Upgradeable Strategies

Some solutions | Monax

tokenestate.io

Smart contracts as services - The five types model:

1. Database contracts
2. Controller contracts
3. Contract managing contracts
4. Application logic contracts
5. Utility contracts

This approach is for larger scale architecture. It was not part of this study.

Some solutions | Zeppelin

tokenestate.io

Proxy Contracts:

- Instead of linking the main, user facing contract directly with the address of the deployed library, it is linked to a ‘Dispatcher’.

- When a transaction comes in, the main contract thinks it is making a delegatecall to the library it is linked with. But this delegatecall will instead be made to the dispatcher.

Some solutions | Zeppelin

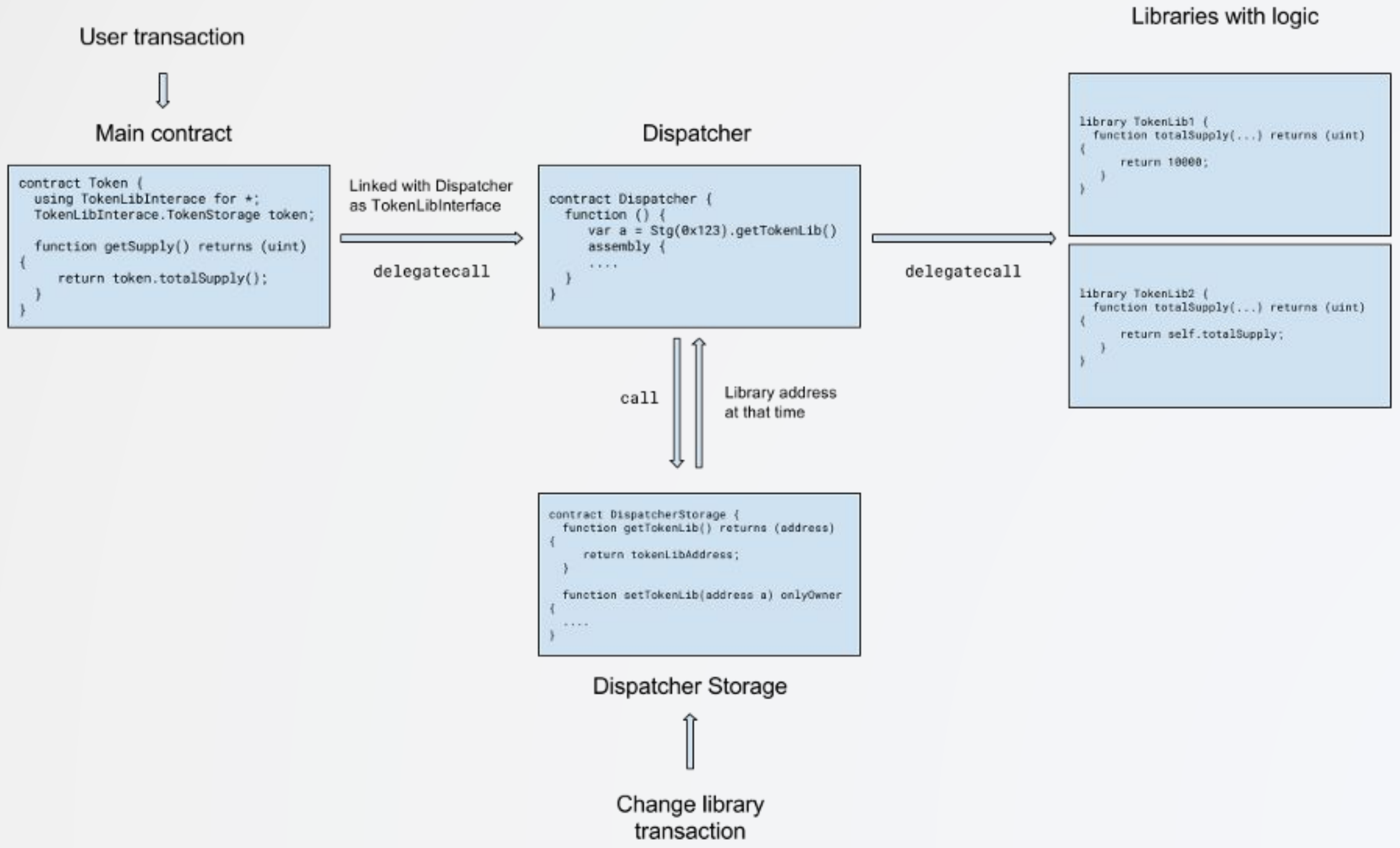
tokenestate.io

Proxy Contracts:

- Once the dispatcher catches the delegatecall in its fallback function it figures out what the correct version of the library code is, and redirects the call once again with a delegatecall.

- Once the library returns, the return will go all the way back to the main contract.

Some solutions | Zeppelin



```
contract Token {
  using TokenLibInterace for *;
  TokenLibInterace.TokenStorage token;

  function getSupply() returns (uint)
{
     return token.totalSupply();
  }
}
```

```
contract Dispatcher {
  function () {
     var a = Stg(0x123).getTokenLib()
     assembly {
     ....
  }
}
```

```
library TokenLib1 {
  function totalSupply(...) returns (uint)
{
      return 10000;
   }
}
```

```
library TokenLib2 {
  function totalSupply(...) returns (uint)
{
      return self.totalSupply;
   }
}
```

```
contract DispatcherStorage {
  function getTokenLib() returns (address)
{
     return tokenLibAddress;
  }

  function setTokenLib(address a) onlyOwner
{
  ....
}
```

Some solutions | Colony

tokenestate.io

Separate Logic and Data Contracts:

- There is one smart contract called "EternalStorage" which is a pure storage-contract without the logic.
- Another contract which can access the EternalStorage contract.
- You can have as many contracts accessing the EternalStorage contract as you want, and hence update the logic part without sacrificing the storage.

Some solutions | Rocket Pool

tokenestate.io

Separate Logic and Data with Data as Key-Value pairs:

- They improved the initial version of the EternalStorage, which can now save almost anything you like.
- Instead of using the final desired data structures that your contract would normally use (structs, mappings etc), all data is abstracted down and stored in primitive key-value pairs.

Some solutions

tokenestate.io

Partially Upgradeable Strategies:

- In this strategy, core features of your smart contract can be left as non-upgradable to retain trust.
- Other components that may be less-integral or more complex (and hence have high probability of requiring upgrade) are implemented with an upgradeable strategy.

References

tokenestate.io

- Smart contracts as services - The five types model | Monax
- Proxy Contracts | Zeppelin
- Separate Logic and Data Contracts | Colony
- Separate Logic and Data with Data as Key-Value pairs | Rocket Pool

References

tokenestate.io

This presentation is inspired by Upgrade Smart Contracts on Chain article from Thomas Wiesner.

If you want to go deeper, you can find a lot of interesting R&D references on the “Research References” part of this blog post: Summary of Ethereum Upgradeable Smart Contract R&D from Jack Tanner.

hello@tokenestate.io