Aggregation

Unter Aggregation wird bei SusScope2 die Zusammenfassung von mehreren Belegquellen für Treibhausgasemissionen zu einer Identität (Konto). Belegquellen können einzelne GHG-Wallets sein, wobei jede Aggregation für sich eine Identität ist. Dies hat zur Folge, dass die Emissionen aktiv mit einem Transfer von der Quelle an die Aggregation übertragen werden müssen.

Aggregationen finden ausschließlich in der Distributed Ledger Technologie statt, weshalb benötigte Metadaten auf einem anderen Weg übertragen werden müssen.

Solidity Smart-Contract - Aggregation

```
pragma solidity ^0.8.6;

import "@openzeppelin/contracts/access/Ownable.sol";
import "./GHGNFT.sol";
import "./GHGERC20.sol";
import "./GHGTOKEN.sol";

contract GHGAggregation is Ownable {

    GHGToken ghgToken;
    GHGERC20 ghgSavings;
    GHGERC20 ghgEmissions;
    GHGNFT ghgCertificates;

    uint256 public savings=0;
    uint256 public emissions=0;
    uint256 public cntNFTs=0;
    uint256 public cntAggregations=0;

    mapping (uint256 => uint256) public idToNft;
    mapping (uint256 => address) public adToNft;

    mapping (address => uint256) public approvedContributors;
```

```
    constructor(GHGToken _ghgToken) {
        ghgToken = _ghgToken;
        ghgSavings = _ghgToken.ghgSavings();
        ghgEmissions = _ghgToken.ghgEmissions();
        ghgCertificates = _ghgToken.ghgCertificates();
    }

    function approveContributor(address _contributor) public onlyOwner {
        approvedContributors[_contributor] = 1;
    }

    function declineContributor(address _contributor) public onlyOwner {
        approvedContributors[_contributor] = 0;
    }

    function addNFT(uint256 _tokenId) public {
        if ((msg.sender != owner()) && (approvedContributors[msg.sender] == 0)) {
            revert();
        }
        if(ghgCertificates.ownerOf(_tokenId) == address(this)) {
            for(uint256 i=0;i<cntNFTs;i++) {
                if(idToNft[i] == _tokenId) revert();
            }
            idToNft[cntNFTs] = _tokenId;
            cntNFTs++;

            address hash = nftTokenHolder(_tokenId);
            savings += ghgSavings.balanceOf(hash);
            emissions += ghgEmissions.balanceOf(hash);
        } else {
            revert();
        }
    }

    function addAggregation(GHGAggregation _aggregation) onlyOwner public {
        if(_aggregation.owner() == address(this)) {
            for(uint256 i=0;i<cntAggregations;i++) {
                if(adToNft[i] == address(_aggregation)) revert();
            }
```

```
            adToNft[cntAggregations] = address(_aggregation);
            cntAggregations++;
            savings += _aggregation.savings();
            emissions += _aggregation.emissions();
        } else {
            revert();
        }
    }

    function transferAggregation(address to,GHGAggregation _aggregation) onlyOwner public
{
       if(_aggregation.owner() == address(this)) {
            bool found=false;
            for(uint256 i=0;i<cntAggregations;i++) {
                if(adToNft[i] ==  address(_aggregation)) {
                    adToNft[i]=address(0);
                    found=true;
                }
            }
            if(found) {
                 savings -= _aggregation.savings();
                 emissions -= _aggregation.emissions();
                 _aggregation.transferOwnership(to);
            } else {
                revert();
            }
        } else {
            revert();
        }
    }

    function transferNft(address to,uint256 _tokenId) onlyOwner public {
        if(ghgCertificates.ownerOf(_tokenId) == address(this)) {
            bool found=false;
            for(uint256 i=0;i<cntNFTs;i++) {
                if(idToNft[i] == _tokenId) {
                    idToNft[i]=0;
                    found=true;
                }
```

```
        }
        if(found) {
            address hash = nftTokenHolder(_tokenId);
            savings -= ghgSavings.balanceOf(hash);
            emissions -= ghgEmissions.balanceOf(hash);
            ghgCertificates.transferFrom(address(this),to,_tokenId);
        } else {
            revert();
        }
    } else {
        revert();
    }
}

function nftTokenHolder(uint256 _tokenId) public view returns (address) {
            string memory hash = ghgCertificates.tokenURI(_tokenId);
             uint endIndex =99;
             uint startIndex =57;

             bytes memory strBytes = bytes(hash);
             bytes memory result = new bytes(endIndex-startIndex);

             for(uint j = startIndex; j < endIndex; j++) {
                 result[j-startIndex] = strBytes[j];
              }
            address did = toAddress(string(result));
            return did;
}

function fromHexChar(uint8 c) public pure returns (uint8) {
    if (bytes1(c) >= bytes1('0') && bytes1(c) <= bytes1('9')) {
        return c - uint8(bytes1('0'));
    }
    if (bytes1(c) >= bytes1('a') && bytes1(c) <= bytes1('f')) {
        return 10 + c - uint8(bytes1('a'));
    }
    if (bytes1(c) >= bytes1('A') && bytes1(c) <= bytes1('F')) {
        return 10 + c - uint8(bytes1('A'));
    }
```

```
        return 0;
    }

    function hexStringToAddress(string memory s) public pure returns (bytes memory) {
        bytes memory ss = bytes(s);
        require(ss.length%2 == 0); // length must be even
        bytes memory r = new bytes(ss.length/2);
        for (uint i=0; i<ss.length/2; ++i) {
            r[i] = bytes1(fromHexChar(uint8(ss[2*i])) * 16 +
                        fromHexChar(uint8(ss[2*i+1])));
        }

        return r;

    }

    function toAddress(string memory s) public pure returns (address) {
        bytes memory _bytes = hexStringToAddress(s);
        require(_bytes.length >= 1 + 20, "toAddress_outOfBounds");
        address tempAddress;

        assembly {
            tempAddress := div(mload(add(add(_bytes, 0x20), 1)),
0x1000000000000000000000000)
        }

        return tempAddress;
    }
}
```

Revision #1
Created 10 January 2023 14:03:52 by Thorsten Zoerner
Updated 10 January 2023 14:14:25 by Thorsten Zoerner