Glossar

Begriffe aus der SusScope2 Implementierung und Dokumentation

- Aggregation
- Beleg
- Eigentümer(schaft)
- Emissionen
- GHG Wallet
- Mengen und Einheiten
- Non Fungible Token (NFT)
- Zertifizierungsdienst
- Vermiedene Emissionen (Savings)

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;
    }
}
```

Beleg

Meist ein Zertifikat für den Nachweis eines Vorgangs. Im Kontext der Bilanzierung von Treibhausgasemissionen werden Belege über einen Sachverhalt ausgestellt und enthalten alle Daten zur Nachvollziehbarkeit des bilanzierten Fakts. Zu jedem Beleg wird ein NFT ausgestellt, mit einer eindeutigen Eigentümerschaft; d.h. unabhängig davon, ob es Kopien des Beleges gibt, kann über die Distributed Ledger Technologie festgestellt werden, welche Identität (Wallet) der Eigentümer ist. Im Rahmen eines Geschäftsvorfalls wird zum Beispiel nach der Ladung eines E-Auto durch den Energie-Service-Anbieter ein Beleg an den Ladenden ausgestellt.

Eigentümer(schaft)

Als Eigentümer wird eine Identität bezeichnet, der in der Distributed Ledger Technologie (DLT) ein digitaler Wert zugeordnet ist. Im Rahmen von SusScope2 sind jedem Beleg ein Eigentümer zugeordnet. Die Eigentümerschaft ist mithilfe einer `transferOwnership()` Transaktion des entsprechenden Smart Contracts übertragbar. Die Übertragung der Eigentümerschaft ist für alle ersichtlich, da diese Bestandteil des Konsens ist.

Emissionen

Innerhalb von SusScope2 werden als Emissionen nur tatsächlich freigesetzte Treibhausgasemissionen bezeichnet. Entsprechend der Liste Mengen und Einheiten erfolgt die Aufzeichnung in Gramm-CO_2-Äquivalente (gCO_2eq). Der Weltklimarat (IPCC) hat Werte für Treibhausgase und ihre CO_2-Äquivalente veröffentlicht, mit deren Hilfe auch andere Klimagase berücksichtigt werden können, ohne diese einzeln bewerten zu müssen.

Beispielrechnung

Würde zu einem Zeitpunkt sämtlicher Strom in Deutschland aus den vorhandenen Steinkohlekraftwerken stammen, so würde die Emission je Kilo-Watt-Stunde (kWh) bei 868 gCO_2eq liegen (Stand 2019). Stammt jedoch zum selben Zeitpunkt die Hälfte des Stroms aus Photovoltaik (0 gCO_2eq/kWh), so würde lediglich 434 gCO_2eq als Emission je Kilo-Watt-Stunde angesetzt werden.

Bilanzierung und Berechnung

Die Zusammensetzung der Kraftwerke, welche in das Netz einspeisen, schwankt nach Tageszeit, Jahreszeit, Wetter und Verbrauch. Entsprechend des Erzeugungsmix verändert sich auch die spezifischen Emissionen einer Kilo-Watt-Stunde Strom. Im Rahmen von SusScope2 wird mit den zur Verfügung stehenden Daten der Energiewirtschaft gearbeitet, um die Nutzung von Durchschnittswerten weitestgehend zu vermeiden. Die Belege des Energieserviceanbieter werden im Rahmen des Zertifizierungsdienstes den feingranularen Verbrauchswerten zugeordnet, die jeweilige Berechnungsgrundlage ist im Beleg aufzuführen.

Abgeleitet von den Emissionen sind vermiedene Emissionen. Hier handelt es sich um Emissionen, die zum Beispiel durch eine zeitliche Verlagerung des Stromverbrauchs nicht entstanden sind. Hierbei erfolgt die Berechnung der vermiedenen Treibhausgase aus der Differenz zwischen dem Jahresschnitt und der zum Zeitpunkt des Strombezuges (Ableseperiode) vorhandenen Emissionswertes.

GHG Wallet

Eine digitale Brieftasche, mit der Hauptaufgabe einen privaten Signaturschlüssel zu speichern, welcher für die Erstellung von digitalen Signaturen notwendig ist und für sich eine digitale Identität besitzt. Zusätzlich bietet die GHG Wallet Methoden und Werkzeuge zur Verarbeitung und Verwaltung von Treibhausgasemissionen (Belege/Zertifikate). Belege können immer nur einen Eigentümer haben, d.h. einer Identität/Wallet zugeordnet sein.

Die GHG-Wallet ist eine eigenständige Node JS Bibliothek, welche Open-Source durch die STROMDAO GmbH bereitgestellt wird.

Node-RED "Nodes" der GHG-Wallet:



Mengen und Einheiten

Messgröße	Einheit	Beispielverwendung
Treibhausgasemissionen	Gramm (CO_2-Äquivalente)	`savings`, `actual`
Stromverbrauch	Wattstunden (Wh)	`grid`, `eco`
Zeit	Millisekunden (ms)	`time`, `timestamp`

Non Fungible Token (NFT)

Ein Non-Fungible Token (kurz NFT) ist ein „kryptografisch eindeutiges, unteilbares, unersetzbares und überprüfbares Token, das einen bestimmten Gegenstand, sei er digital oder physisch, in einer Blockchain repräsentiert“. Im Rahmen des SusScope2 Frameworks für die Verwaltung von Treibhausgasemissionen wird ein NFT für einen Beleg erstellt.

Zertifizierungsdienst

Dienstleister, der als Online-Service die Bestätigung der Treibhausgasemissionen für Stromverbrauch vornimmt. Ein Zertifizierungsdienst kann zum Beispiel durch einen Energieserviceanbieter oder durch den Messstellenbetreiber zur Verfügung gestellt werden. Die Kommunikation mit dem Dienst erfolgt über eine REST-Schnittstelle (https) und folgt dem Ablauf, welche für die Blockchain basierte Nachweisführung der THG-Emission und Minderung definiert ist.

Die Open-Source Bibliotheken sind vorkonfiguriert, die **STROM**DAO GmbH als Zertifizierungsdienst zu nutzen.

Vermiedene Emissionen (Savings)

SusScope2 unterstützt die Protokollierung von vermiedenen Treibhausgasemissionen. Festgehalten werden diese im Datenfeld `savings`. Entsprechend der allgemeinen Bilanzierungsregeln für die Nachhaltigkeitsberichterstattung werden diese nach folgenden Verfahren ermittelt und analog zu den Emissionen in gCO_2eq verarbeitet:

Einsparung durch zeitliche Verlagerung des Strombezugs

Auf der Grundlage, dass die Erzeugungsart von Strom einem ständigen Wechsel im Tagesverlauf unterzogen ist, wird für einen belegten Verbrauch die zu diesem Zeitpunkt entstandene Emission mit dem Jahresschnitt verglichen. Aus der Differenz der beiden Werte ergibt sich die Einsparung für einen dedizierten, beleghaften Verbrauch.

Mit einem zunehmenden Ausbau der Stromerzeugung aus regenerativen Quellen, treten Zeiten, in denen die vermiedene Emission die tatsächliche Emission übersteigen, mit steigender Häufigkeit auf.

Vermeidung durch eigene Stromerzeugung (PV)

Erfolgt die Bilanzierung von Stromerzeugung, die zur Eigenstromnutzung verwendet wird (zum Beispiel durch eine Photovoltaikanlage), so hat die dadurch erzeugte Strommenge verhindert, dass Strom aus dem Netz bezogen wurde. In diesem Fall wird die volle Höhe als vermiedene Emission ausgegeben (s.h. Konzept: THG neutrale Einspeisung)