Nota: 9,5

Sistema de trazabilidad basado en blockchain

Blockchain-based product traceability system

Trabajo Fin de Grado

Curso 2024-2025

Autor

Clara Sotillo Sotillo

Directores

Pablo Gordillo Alguacil y Jesús Correas Fernández

Grado en Ingeniería del Software

Facultad de Informática

Universidad Complutense de Madrid

Agradecimientos

Quiero agradecer a mi familia por inspirarme siempre a continuar trabajando, por darme la oportunidad de aprender sobre lo que me gusta y por escucharme incansablemente hablar de ello, incluso si a veces no lo comprenden del todo.

También agradecerle a Edu por animarme y aguantarme en los momentos difíciles y por creer en mi incluso cuando yo no lo hacía.

Y principalmente, agradecerles a mis tutores Jesús y Pablo por permitirme hacer un proyecto tan interesante, por guiarme durante todo este proceso haciéndolo más ameno, y por su esfuerzo y ayuda durante el desarrollo de este proyecto.

Resumen

Sistema de trazabilidad de productos basado en blockchain

La trazabilidad es un elemento crucial de la producción de productos y servicios, ya que permite mantener un registro de todo lo que ocurre durante el proceso de producción industrial y permite realizar un análisis de los puntos de mejora en los procesos y detectar los problemas que puedan surgir durante la producción.

Este proyecto se centra en el análisis de los sistemas de trazabilidad y las ventajas que la tecnología blockchain podría ofrecer para la implementación de estos sistemas. Con esa finalidad se desarrolla un prototipo de un sistema de trazabilidad implementando esta tecnología.

Durante el proceso de análisis, se ha llegado a la conclusión de que una parte vital de la trazabilidad es la representación de las relaciones entre productos o procesos industriales. Se ha encontrado que la mejor forma de representar estas relaciones es mediante un grafo acíclico.

Este prototipo es un estándar de token, siendo esta una manera idónea de representar productos en el sistema, dotándolo además de las herramientas necesarias para representar una estructura de grafo necesaria para mostrar las relaciones entre productos o procesos. Con este prototipo se pretende crear un sistema de trazabilidad de productos que pueda ser empleado en diversas industrias, y que proporcione una manera fiable de certificar la veracidad de los datos proporcionados por el sistema, evitando la necesidad de intermediarios que podrían manipular u obviar información relevante a la trazabilidad de los productos.

Palabras clave

Trazabilidad, blockchain, token, ERC1155, smart contract, grafos, NFT.

Abstract

Blockchain-based product traceability system

Traceability is a crucial element of product production and services, since it allows for a record of everything that happens during the industrial production process and allows to perform an analysis of the point of improvement and detect problems that may arise during the production process.

This project focuses on the analysis of traceability systems and the advantages blockchain technology could offer for the implementation of these systems. With that in mind, a traceability system prototype using this technology is developed.

During the analysis process, it was concluded that a vital part of traceability is the representation of product or process relationships. It was determined that an acyclic graph is the most effective way to represent those relationships.

This prototype is a token standard, this being an ideal way to represent products in the system, providing the token with the necessary tools to represent a network structure needed to show the relationships between products or processes. The goal of this prototype is to create a product traceability system that could be implemented into various industries and delivers a reliable way to certify the integrity of the data provided by the system, avoiding the need for third parties that could manipulate or omit relevant information for the traceability of products.

Keywords

Traceability, blockchain, token, ERC1155, smart contract, graph, NFT.

ÍNDICE DE CONTENIDOS

1. INTRODUCCIÓN

1.1. MOTIVACIÓN

Vivimos en una sociedad plenamente industrializada, donde todos los productos que se emplean en el día a día han pasado por una serie de procesos comenzando con la extracción de las materias primas, pasando por diversos procesos de fabricación y ensamblaje hasta llegar al consumidor final. Como no existe nada perfecto, estos procesos son susceptibles a errores y manipulaciones maliciosas. Por ello es importante tener un registro de todos los estados por los que ha pasado el producto, así como los componentes implicados en su construcción. Esto ayuda a identificar las áreas problemáticas, acercándolo un poco más a esa perfección inalcanzable y evitando que afecte negativamente al consumidor final.

La trazabilidad está presente en la mayor parte de los sectores productivos y es empleada al menos para alguna de las actividades que desarrollan. La trazabilidad suele aplicarse para mejorar la actividad, como pueden ser mejorar la eficiencia de procesos, reducir costes ante fallos en la producción o mejorar el servicio ofrecido, por ello es importante que sea lo más fiable y eficiente posible, maximizando así los beneficios que proporciona. [1]

La naturaleza inmutable de la tecnología blockchain la hace una propuesta interesante aplicada a los sistemas de trazabilidad, considerándose una solución más segura y transparente para llevar un registro de la trazabilidad de los productos. El uso de esta tecnología aplicada a los sistemas de trazabilidad mejora la confianza que estos sistemas ofrecen.

La idea principal de este Trabajo de Fin de Grado consiste en crear un “mapa” donde se muestre el recorrido de un producto desde la obtención de sus materias primas hasta su venta, indicando todos los productos implicados en su

composición y las modificaciones y problemas derivados de su elaboración que han surgido a lo largo del proceso. Con este “mapa” se podrían identificar con facilidad los puntos problemáticos en la cadena de producción, reduciendo costes, tiempo y desperdicio de materiales, además de mejorar la seguridad y confianza de los destinatarios.

1.2. OBJETIVOS

El objetivo principal de este proyecto es crear un prototipo que permita gestionar y mantener un registro de la trazabilidad de los procesos de producción de productos en diversas áreas. Para lograr este objetivo se han identificado una serie de objetivos específicos que se detallan a continuación:

- Realizar un estudio de las características principales que debe tener un sistema de trazabilidad de productos.

- Buscar documentación relevante sobre sistemas existentes de trazabilidad implementados con tecnología blockchain, identificando tanto sus puntos fuertes como sus puntos de mejora.

- Diseñar un prototipo basado en blockchain que cubra todos los requisitos de un sistema de trazabilidad con el objetivo de hacerlo más seguro y eficiente.

- Evaluar la viabilidad del sistema con un ejemplo realista.

1.3. PLAN DE TRABAJO

Para la organización y el seguimiento continuado del trabajo se realizaron reuniones semanales, donde se revisaba el trabajo realizado durante la semana y se tomaban decisiones sobre el rumbo general del proyecto y sobre tareas específicas a realizar en la siguiente semana.

El primer paso consistió en un proceso de análisis de los sistemas de trazabilidad, donde el objetivo es obtener las características principales que debe tener un sistema de trazabilidad.

Seguidamente, se investigó el estado actual de la tecnología blockchain en el ámbito de la trazabilidad. El objetivo de este paso es identificar las necesidades y puntos de mejora en los sistemas de trazabilidad actuales.

Una vez identificadas las necesidades se planteó un diseño del prototipo que cumpla con las características de un sistema de trazabilidad y además cubra las necesidades identificadas en el paso anterior.

Tras la implementación del sistema se realizan una serie de pruebas para garantizar el correcto funcionamiento del prototipo, realizando ajustes cuando sea necesario.

1. INTRODUCTION

1.1. MOTIVATION

We live in a fully industrialized society, where all products used on a day-to-day basis have gone through a series of processes from the extraction of raw materials, through various manufacturing and assembly processes, until the products reach the consumer. Since nothing is perfect, these processes are susceptible to errors and malicious manipulation. It is therefore important to have a record of all the stages the products have gone through, as well as all the components involved in their construction. This helps identify problematic areas, bringing the processes a little closer to that unattainable perfection and preventing them from negatively affecting the end consumer.

Traceability is present in most production sectors and is used in at least some of the activities they carry out. Traceability is usually applied to improve activities, such as improving the efficiency of processes, reducing costs in the event of production failures or improving the service offered, so it is crucial to make a traceability system as reliable and efficient as possible, thus maximizing the benefits it provides. [1]

The immutable nature of blockchain technology makes it an interesting proposal applied to traceability systems, making it a more secure and transparent solution for keeping a record of product traceability. The use of this technology applied to traceability systems improves the trust these systems offer.

1.2. OBJECTIVES

The main objective of this project is to create a prototype to manage and maintain the traceability records of production processes in different areas. In order to achieve this objective, a series of specific objectives have been identified, which are detailed below:

- Conduct a study on the main characteristics every traceability system should have.

- Search relevant information on existing traceability systems implemented with blockchain technology, highlighting both their strengths and improvement areas.

- Design a blockchain-based prototype that covers all the requirements of a traceability system with the goal of making it more secure and efficient.

- Evaluate the feasibility of the system with a realistic example.

1.3. ACTION PLAN

Weekly meetings were held to review the work done during the week and to make decisions on the general project direction and on specific tasks that were going to be carried out the following week.

The first step consisted of an analysis of traceability systems, where the goal was to obtain the main characteristics that a traceability system should have.

Next, an analysis of the current state of blockchain technology on the field of traceability was carried out. The main goal of this step is to identify the needs and points of improvement of the current traceability systems.

Once the needs were identified, the design for the prototype that meets the characteristics of a traceability system was proposed covering the needs identified in the previous step.

After implementing the system, a series of tests were conducted to ensure the prototype was working correctly and making adjustments when necessary.

2. ESTUDIO DE LOS SISTEMAS DE TRAZABILIDAD EXISTENTES BASADOS EN TECNOLOGÍA BLOCKCHAIN

Debido a la novedad de la tecnología blockchain, no existe mucho contexto en su aplicación para sistemas de trazabilidad. Las pocas empresas que promocionan el uso de esta tecnología, en el contexto de la trazabilidad, no ofrecen información sobre su implementación, ni tecnologías concretas empleadas. La información de la que se dispone en estos proyectos es principalmente publicitaria o grandes proyectos que buscan financiación para mejorar su trazabilidad pero que aún no están en desarrollo.

2.1. INDUSTRIA ALIMENTARIA Y FARMACÉUTICA

El uso de la tecnología blockchain en la trazabilidad de productos alimenticios y farmacéuticos es esencial ya que permite garantizar la calidad y seguridad de los productos. En estos casos es importante poder identificar con rapidez los problemas que puedan surgir en el proceso de fabricación, esencial en casos de retirada de productos. Además, permite asegurar que los fabricantes cumplan con las regulaciones y brinda transparencia a los clientes.

La implementación de la trazabilidad en estas industrias tiene una serie de beneficios que se amplifican con el uso de la tecnología blockchain. A continuación, se listan algunos de los beneficios de la trazabilidad en estas industrias tan cruciales y que requieren de una buena trazabilidad [2]:

GARANTÍA DE CALIDAD

Al tener un registro de todos los procesos en la producción de un producto, la identificación de cualquier problema surgido en su desarrollo se vuelve más sencilla y permite tomar medidas correctivas, mejorando el proceso de fabricación

y como consecuencia la calidad del producto. La tecnología hace que este proceso sea más eficaz y evita el falseo de datos debido a la característica de inmutabilidad de datos propia de esta tecnología.

SEGURIDAD DEL CLIENTE

Cuando se localiza un problema con un producto o lote de productos, permite lograr medidas preventivas para paliar los efectos negativos que el uso de estos productos defectuosos pueda ocasionar en el cliente. La retirada de productos se vuelve más sencilla con el uso de la tecnología blockchain ya que permite localizar los lotes dañados con rapidez y evitando que se desperdicie producto al tener los identificadores de los productos exactos que necesitan ser retirados.

Existen organismos reguladores como el *FDA* (Food and Drug Administration) [3] en Estados Unidos, que indican la necesidad de usar nuevas tecnologías como blockchain para mejorar la trazabilidad de los productos con el objetivo de obtener los beneficios ya mencionados. Su objetivo es que las empresas implementen sistemas digitales internos, que permitan recibir eventos de seguimiento críticos y datos clave de socios de la industria y reguladores.

Para promover el desarrollo de sistemas que identifiquen y tracen los medicamentos con receta, se creó el *DSCSA Pilot Project Program*, donde se proponen distintos proyectos.

Existen empresas más pequeñas que emplean el blockchain para realizar la trazabilidad de sus productos. Como *BumbleBee* [4] , una empresa pesquera que, mediante un código localizado en la lata, permite a sus clientes obtener toda la información relevante de su pescado.

2.2. SECTOR SANITARIO

Con el objetivo de mejorar la trazabilidad de los dispositivos médicos, un equipo de la universidad de Khalifa plantea una solución basada en NTF [5], lo que proporciona un acceso y almacenamiento inmutable de la información de los dispositivos médicos, proporcionando anonimidad de la información médica siendo este un requisito fundamental en este tipo de sistemas.

Esta solución tiene el objetivo de realizar la trazabilidad y gestión de dispositivos médicos reacondicionados, asegurando la integridad y autenticidad de estos. Este sistema proporciona un mecanismo de gestión de la pertenencia del dispositivo y certificación de la autenticidad de este durante el proceso de reacondicionamiento.

En este sistema cada pieza de recambio es un NFT que se relaciona con un dispositivo mediante su ID único. Cada dispositivo almacena un registro de sus piezas de recambio en un array de hijos.

El problema de este sistema es que limita la trazabilidad a un solo sentido, es decir sólo puedes conocer las piezas de recambio empleadas en un dispositivo, pero no puedes conocer a qué dispositivo pertenece una pieza de recambio. Además, permite únicamente trazar las piezas de recambio, no proporciona una trazabilidad completa del dispositivo. Estas piezas de recambio no ofrecen información trazable sobre las piezas que lo componen, haciendo que no sea transferible a la trazabilidad de productos fuera de los dispositivos médicos reacondicionados.

Este sistema es el que más se aproxima al propuesto en este Trabajo de Fin de Grado en el cual se intenta hacer un sistema derivable a otros sectores y expandible a todos los componentes del sistema, no solo a los recambios. Además de proporcionar la capacidad de realizar la trazabilidad desde cualquier componente,

no solo desde el producto final, permitiendo observar ambos sentidos de la trazabilidad.

2.3. INDUSTRIA MANUFACTURERA

El uso de la trazabilidad empleando tecnología blockchain en la industria manufacturera proporciona también una serie de beneficios propios como pueden ser:

PREVENCIÓN DE FALSIFICACIONES

Gracias a que esta tecnología puede certificar de manera inequívoca la procedencia de un producto, se pueden garantizar los orígenes de un producto dando mayor validez a las denominaciones de origen y a los certificados de autenticidad. Esto evita también problemas con la calidad de los productos en el caso de la producción industrial, como el uso de productos con calidad menor a la esperada, pudiendo crear problemas en los procesos posteriores.

MENOR IMPACTO ECOLÓGICO

Dado que todos los procesos de producción y productos participantes en el proceso de producción quedan registrados en la blockchain, las certificaciones que indican que el proceso de producción de un producto cumple los estándares ecológicos se vuelven más fiables

Existen ejemplos de uso como la startup *TrusTrace* [6] que emplea la tecnología blockchain para realizar la trazabilidad de productos en la industria textil. *TrustTrace* tiene una colaboración con la marca inglesa *NewLook* para permitirle ser más transparente con sus clientes, certificando una producción ética de sus productos.

Adidas [5] es una de las primeras empresas en el sector en emplear *TrusTrace* para realizar la trazabilidad de sus productos, con el objetivo de certificar la sostenibilidad de sus productos.

Por último, en la industria automovilística Renault [7] tiene el proyecto *XCEED* [8] en colaboración con *IBM* [9], para realizar la gestión de la cadena de producción, introduciendo transparencia y trazabilidad.

2.4. INDUSTRIA DEL LUJO

Esta industria tiene el foco de la trazabilidad centrado en la certificación de autenticidad y la proveniencia de los materiales, para certificar un producto libre de conflictos éticos por ejemplo en el caso de los diamantes de sangre.

El mayor ejemplo de uso de la tecnología blockchain en esta industria es el de la empresa *De Beers Group* [10], siendo la primera en documentar la trazabilidad de un diamante desde su extracción hasta su venta.

En el mundo de las marcas de lujo se creó *Aura Blockchain Consortium* [11, 12] como un sistema de blockchain para la autenticación y la trazabilidad de sus productos de lujo.

2.5. INDUSTRIA LOGÍSTICA

En esta industria existe el caso de *Maersk* y su proyecto *TradeLens* [13] , para la trazabilidad de mercancías con tecnología blockchain. *TradeLens* ofrecía una propuesta muy interesante ya que esta industria tiene un número de participantes muy elevado, extendiéndose por diversos sectores y países, abarcando desde los productores hasta la venta de los productos, pasando por aduanas y transportistas. Este proyecto cerró en 2023 ya que no fue capaz de convencer a todas las partes

implicadas de compartir información, haciendo así inviable tener una trazabilidad completa de las mercancías

2.6. ESTUDIOS CIENTÍFICOS

Del *DCSCSA Pilot Program* mencionado en la sección 2.1 nace *TrialChain* [14], un estudio de la universidad *Cornell* centrado en la validación de datos recogidos en estudios biomédicos. Este es el estudio más completo sobre el uso de la tecnología blockchain en el ámbito de la trazabilidad, ya que proporciona datos sobre su implementación.

TrialChain emplea un modelo mixto de blockchain público-privada, emplea una blockchain privada implementada con *MultiChain*, donde se almacenan todos los datos relacionados con los estudios y Ethereum como blockchain pública para añadir otro nivel de seguridad a sus datos.

Para garantizar esta mayor inmutabilidad de los datos, se guarda periódicamente el último blockhash de la blockchain privada como una transacción en Ethereum de forma similar a como funcionan las arquitecturas de rollups para proporcionar escalabilidad al sistema. Si todo transcurre según lo planeado, este nuevo bloque contendrá el tiempo actual, la dirección de la cartera de Ethereum utilizada para la aplicación, el hash de la transacción y el último blockhash de la blockchain privada.

Una de las aplicaciones que se mencionan en este proyecto, es su implementación en la *NDSP* (National Center for Cardiovascular Disease Data Science Platform). En este caso utilizan una API basada en la librería Falco que proporciona un método seguro de introducir información desde distintas aplicaciones.

3. CONCEPTOS FUNDAMENTALES

El siguiente apartado proporciona una breve introducción a los conceptos fundamentales necesarios para el desarrollo del sistema de trazabilidad propuesto.

3.1. SISTEMA DE TRAZABILIDAD

Según la Real Academia Española (RAE) [15] la trazabilidad es la posibilidad de identificar el origen y las diferentes etapas de un proceso de producción y distribución de bienes de consumo.

El objetivo de un sistema de trazabilidad es proporcionar una manera de conocer y registrar tanto los orígenes como todos los procesos por los que ha pasado un producto hasta llegar a estar finalizado o incluso hasta la propia venta de éste y los componentes que los forman.

Existen diversas razones por las que una empresa querría realizar la trazabilidad. Empresas como *Coca-Cola* [16] lo hacen para asegurar que no existen infracciones de los derechos humanos en su cadena de producción. Otras como *BASF* [17] lo hacen para certificar la autenticidad de sus productos. También existen empresas como *PepsiCo* [18] *que* realizan la trazabilidad de sus productos para evitar la deforestación y conservar la biodiversidad de las zonas donde obtienen sus materias primas. Además, la Unión Europea está trabajando en un sistema de trazabilidad de medicamentos con el objetivo de combatir problemas con las falsificaciones, desabastecimiento y retirada de productos.

Todo sistema de trazabilidad debería cumplir una serie de características que lo identifiquen como tal. A continuación, se definen las características que se consideran necesarias para un sistema de trazabilidad:

Trazable en todos sus puntos.

El sistema debería poseer un registro del estado del producto en todos los procesos de la producción de este. De esta manera se puede garantizar que no se han producido errores no registrados y evitando posibles cambios maliciosos.

Privacidad de los datos.

El sistema debería proteger los datos de las empresas colaboradoras, para que estas puedan mantener la privacidad de la información de terceros. De esta manera, más empresas estarán dispuestas a participar y compartir su información con el sistema.

Transparente.

El sistema debería ser transparente con los clientes, proporcionando información relevante de los productos. Esto les permitirá tomar decisiones de compra informadas de sus productos, conociendo así su procedencia y métodos de producción.

Datos inmutables.

Los datos introducidos en el sistema no deberían cambiar a lo largo del proceso de producción. Las modificaciones de los datos del sistema deberían estar restringidas a solamente algunos elementos de forma controlada, y debería quedar registrado cualquier cambio realizado. Esto permite evitar la posible modificación de datos con fines maliciosos o de forma no intencionada, perjudicando también la transparencia en el proceso.

Seguro.

El sistema debería ser resistente a ataques y modificaciones. De no serlo pondría en peligro tanto la privacidad como la transparencia e inmutabilidad de los datos.

Rápido.

La información debería ser accesible en todo momento de manera casi inmediata. Esto permitirá una rápida actuación en el caso de retirada de productos.

Confiable.

La información debería ser correcta. El sistema debería promover la confianza entre sus participantes, así como la confianza del cliente en la procedencia del producto.

Información completa.

El registro de toda la información relevante del producto y su estado durante el proceso de producción es importante para la identificación de problemas durante el transcurso del proceso de producción y para evitar obviar datos que puedan afectar a la decisión de compra del cliente.

3.2. BLOCKCHAIN

Una blockchain es un tipo especial de base de datos. Es un libro mayor o "ledger" digital descentralizado. Sus datos están almacenados en una serie de ordenadores conectados en red.

Los datos y el estado se almacenan en bloques o lotes secuenciales. Cada transacción tiene que ser añadida para poderse realizar con éxito. Además, se dice que es una cadena de bloques ya que cada bloque hace referencia criptográficamente a su antecesor encadenándolos. Esta estructura hace que no se puedan cambiar los datos de un bloque sin modificar todos los anteriores. Para poder realizar este cambio, se requerirá del consenso de toda la red.

Cada ordenador es un "nodo" de la red y deben aceptar los nuevos bloques y la cadena en su conjunto. Esto garantiza que todas las personas en la red tienen los mismos datos, para que los nodos puedan llegar a un acuerdo sobre cuál es el estado de la cadena de bloques.

MECANISMOS DE CONSENSO

Ethereum emplea actualmente como mecanismo de consenso "Proof of Stake", que requiere que los validadores [19] apuesten capital en forma de ETH en un contrato inteligente en Ethereum [20]. Estos validadores son los responsables de verificar la validez de los nuevos bloques propagados por la red y, en ciertas circunstancias, de crear y propagar nuevos bloques.

Al participar como validador, existe la posibilidad de que ciertos usuarios ataquen la red en busca de beneficios propios. Para evitar problemas, se les retirará la recompensa en ETH cuando no participen tras haber sido asignados y antes malas conductas se podrá destruir el ETH de sus participaciones.

SMART CONTRACT

Existen diversos tipos de blockchain, siendo la más conocida Bitcoin. Ethereum es una blockchain que al igual que Bitcoin, utiliza dinero digital sin proveedores de pago o bancos. Además, a diferencia de Bitcoin, Ethereum es programable, así que también puede utilizarse para construir y mantener aplicaciones descentralizadas sobre su infraestructura.

Para la programación dentro de Ethereum, se emplean "smart contract". Son un tipo de cuenta de Ethereum, lo que significa que tienen saldo en ETH y pueden ser el objetivo de transacciones. Estos contratos no están controlados por un usuario, sino que se implementan en la red y se ejecutan automáticamente. Las cuentas de usuarios pueden interactuar con un smart contract, enviando transacciones que ejecuten una función definida.

Estos programas viven en la blockchain y se ejecutan cuando una transacción los desencadena. Una vez publicado un contrato estará operativo, sin que ni siquiera el autor pueda eliminarlo.

Para crear aplicaciones en la blockchain de Ethereum se emplea el lenguaje de programación Solidity.

GAS

Cada operación en la red de Ethereum requiere un esfuerzo computacional para ejecutarse. Este esfuerzo se mide con una unidad llamada "gas" [21]. Los recursos informáticos necesarios para llevar a cabo una transacción, tienen que ser pagados para poder evitar vulnerabilidades que hagan que quede atascado en un lapso computacional infinito. Este pago se realiza en forma de "tarifa de gas".

La tarifa de gas es la cantidad de gas usado para realizar una operación, multiplicado por el coste unitario del gas. Esta tarifa se paga, independientemente de que la operación sea o no exitosa. Esta tarifa tiene que pagarse en ETH, es decir la moneda nativa de Ethereum. El precio del gas viene expresado en gwei.

$$1gwei = 10^{-9}ETH$$

Se puede establecer la cantidad de gas que se está dispuesto a pagar cuando se realice una transacción.

$$cantidad\_total\_gas = tarifa_{base} + tarifa_{prioritaria}$$

Dependiendo de la cantidad de gas que se pague la transacción se incluirá antes o después en un bloque. Si la cantidad es muy baja los validadores estarán menos incentivados a incluirla en el siguiente bloque, por lo que es posible que se ejecute más tarde o ni se ejecute. En el caso de que sea demasiado alta, se puede desperdiciar ETH.

Existe un protocolo que establece la tarifa base, esto indica la cantidad mínima necesaria para que la transacción se considere como válida. La tarifa prioritaria, es la "propina" añadida para incentivar a los validadores a incluir la transacción, por ello una transacción que solo pague la tarifa base, es poco probable que se incluya.

$$tarifa_{total} = \text{unidades\_gas\_usadas} * (tarifa_{basica} + tarifa_{prioritaria})$$

3.3. TOKENS

Token es el termino empleado para referirse a la representación digital de activos, ya sean fungibles como las criptomonedas o no fungibles (NFT) [22] . Estos tokens mantienen un registro de la dirección de su propietario actual, dándoles así la noción de propiedad. Además, permiten la transferencia de esta propiedad entre los usuarios del sistema de blockchain.

Las EIP o propuestas de mejora de Ethereum [23, 24], son estándares que especifican nuevas características o procesos potenciales para Ethereum. Contienen especificaciones técnicas para los cambios propuestos. Las EIP pueden ser creadas por cualquier usuario dentro de la comunidad de Ethereum. El autor y la comunidad deben llegar a un consenso y documentar opciones alternativas.

Un ERC o “Ethereum Request for Comments” [25] es un tipo de EIP que define ciertas reglas y características que los tokens y smart contracts deben seguir.

Permiten la interoperabilidad entre diferentes aplicaciones y contratos en la blockchain de Ethereum.

Todos los tokens que se crean en Ethereum deben seguir los estándares establecidos por estas ERC. Estos tokens pueden representar virtualmente en Ethereum cualquier elemento.

3.3.1. ERC-20

El token ERC-20 [26] es un estándar que permite crear tokens fungibles, es decir, todos los tokens son iguales entre sí, como ETH o Bitcoin. Permite la transferencia de tokens, obtención del saldo de tokens en una cuenta y el suministro total disponible en la red, además de permitir aprobar los tokens de una cuenta que pueden gastarse con una cuenta de terceros.

Aunque es uno de los tokens más empleados, sigue teniendo problemas que se arreglan en estándares posteriores. Uno de estos problemas es la transferencia de tokens a contratos que no están diseñados para manejar este tipo de tokens. En este caso ya que el contrato receptor no posee ninguna funcionalidad para reconocer o responder a estos tokens entrantes, puede ocasionar que estos tokens se pierdan de forma permanente.

3.3.2. ERC-721

Los tokens no fungibles o NFT se emplean para identificar algo, como puede ser un objeto del mundo real, ya sea físico como el mencionado en la sección 2.2 o conceptual como lo puede ser un contrato, de una manera única, es decir tokens con un valor único y que actúan como objetos digitales verificables únicos que no pueden cambiarse entre sí, al contrario que los ERC-20. El ERC-721 [27] proporciona las mismas funcionalidades que el ERC-20, como la noción de propiedad y su

transferencia entre usuarios, pero enfocadas a objetos específicos. En este cada token tiene un ID que lo distingue del resto.

3.3.3. ERC-1155

Es una interfaz para contratos que administran múltiples tipos de tokens, tanto fungibles como no fungibles. Los tokens ERC-1155 [28] pueden realizar las mismas funciones que los ERC-20 y los ERC-721, incluso ambas al mismo tiempo. Mejora ambos estándares haciéndolos más eficientes.

El ERC-1155 permite la transferencia de lotes, es decir, permite transferir más de un token en una sola llamada. De la misma manera permite conocer saldos en lote y la creación de tokens en lote. Además, posee un conjunto de reglas para realizar las transferencias de forma segura.

Dada su compatibilidad con el EIP-165, que crea un estándar para publicar y detectar qué interfaces implementa un smart contract, el ERC-1155 admite hooks de recepción. La función hook debe devolver un valor mágico predefinido. Cuando el contrato de recepción muestra este valor, asumimos que el contrato acepta la transferencia y sabe cómo manejar estos tokens ERC-1155. Esto arregla el problema descrito anteriormente en el ERC-20, evitando pérdidas permanentes de tokens.

3.4. TRAZABILIDAD BASADA EN BLOCKCHAIN

Un sistema de trazabilidad tiene como objetivo hacer accesible toda la información referente a un producto o proceso de producción.

La trazabilidad de productos ha sido realizada tradicionalmente en papel o con sistemas informáticos anticuados. Esto ralentiza el acceso a la información y lo hace más propenso a la pérdida de datos. Otro de sus puntos débiles es la

posibilidad de que los datos sean modificados durante el proceso, lo cual haría que la trazabilidad fuera imprecisa y poco fiable. Todos estos problemas producen desconfianza tanto en las entidades implicadas, como en los clientes, ya que no tienen una idea clara sobre los orígenes del producto.

En la Figura 3.1. se muestran los principales problemas de los sistemas de trazabilidad tradicionales. En este caso menciona los problemas éticos que conlleva no conocer los orígenes de los productos, como pueden ser el incumplimiento de los derechos humanos, o problemas de deforestación. Además, no hay un control verificable del impacto ambiental que se produce durante la producción. El control por parte de terceros tiene que estar más programado y es menos frecuente. Además, el cliente no tiene seguridad sobre los orígenes de sus productos.

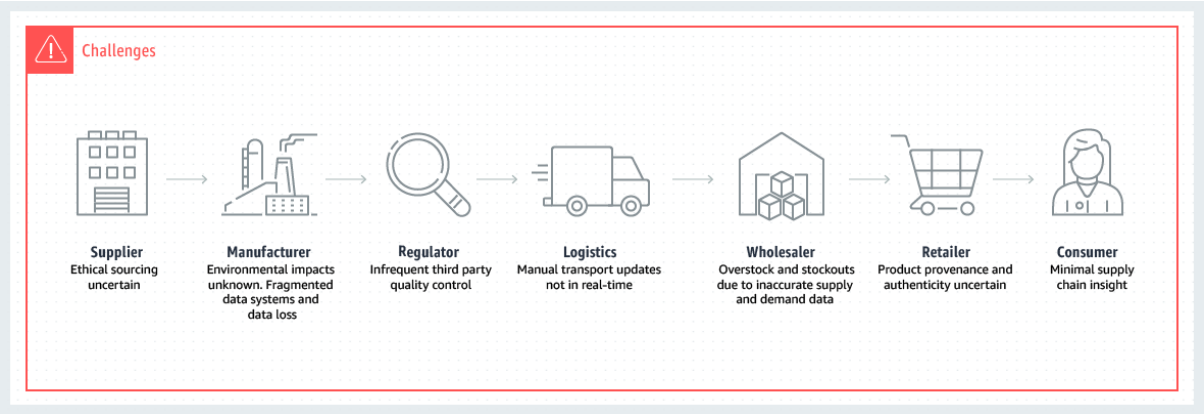

Figura 3.1 Problemas en los sistemas de trazabilidad tradicionales. [29]

Con el fin de paliar estos problemas se propone el uso de la tecnología blockchain en este ámbito. La tecnología blockchain tiene como ventaja la inmutabilidad de sus datos, lo que permite que una vez introducidos, no puedan ser modificados, por lo menos sin que quede registro de ello. Esto genera más confianza en todas las partes involucradas en el proceso.

Al tener todos los datos en la blockchain, el acceso a estos es rápido, facilitando una pronta actuación en caso de ser necesario. También garantiza que en caso de que

se produzcan pérdidas o desperfectos del producto, se pueda indicar rápidamente en qué parte del proceso se produjo y cuántos productos fueron afectados.

La Figura 3.2 muestra los principales beneficios de un sistema de trazabilidad implementado con tecnología blockchain. Gracias a esta tecnología, se conocen más datos sobre los orígenes exactos de los productos, de esta manera los problemas éticos como los mencionados anteriormente se minimizan. El impacto medioambiental tiene mayor visibilidad a lo largo de la cadena de producción. Con toda la información disponible y teniendo en cuenta que los datos son inmutables, el cliente tiene una mayor seguridad de los orígenes de sus productos.

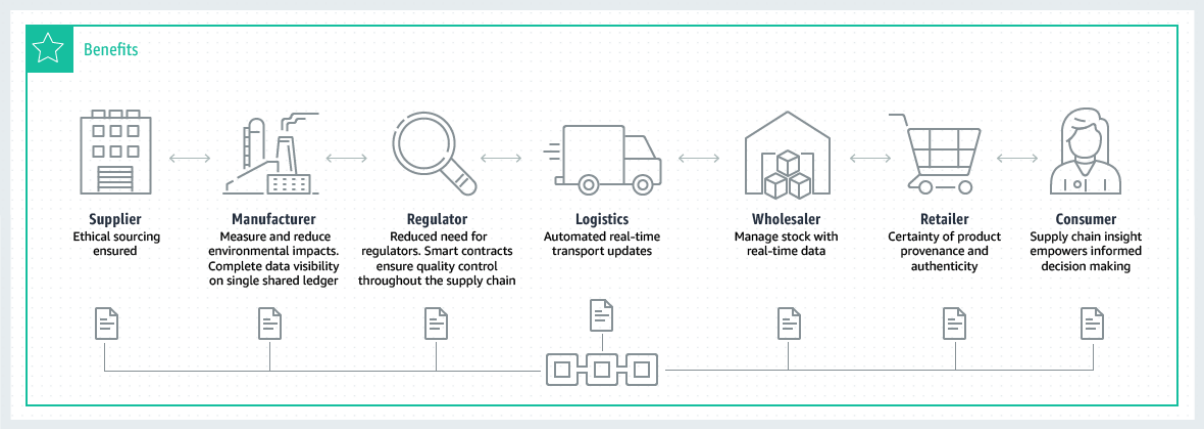

Figura 3.2. Beneficios del uso de sistemas de trazabilidad basados en blockchain. [29]

Por las características propias de la tecnología blockchain, un sistema implantado con ella podría cumplir todas las características descritas anteriormente para hacer un sistema de trazabilidad completo y eficiente.

4. DISEÑO DEL PROTOTIPO

El sistema de trazabilidad de productos propuesto se centra en la creación de grafos que describan la composición del producto y el uso de tokens para la representación del propio producto.

En este grafo, cada vértice se representará mediante un token ERC-1155. Este tipo de tokens es una gran opción para un sistema de trazabilidad, ya que permite crear tokens tanto fungibles como no fungibles en un solo contrato, lo que es ideal para un sistema complejo que posee productos tanto divisibles como unitarios. Además, las operaciones principales como minting (creación del token) y transferencias son más eficientes que en el ERC-721 o ERC-20 en términos de gas, sobre todo cuando se quieren crear o transferir varios tokens a la vez.

El uso de tokens para la elaboración de un sistema de trazabilidad de productos se puede justificar con las propiedades únicas propias de la tecnología blockchain. La inmutabilidad de los datos permite que quede un registro de todas las actividades del sistema, sin posibilidad de alterar ni borrar información sin que quede rastro. Los tokens permiten la trazabilidad individualizada de los productos, o procesos permitiendo situarlos y controlarlos a lo largo de la cadena de producción. Al encontrarse on-chain, facilitan exportar la trazabilidad a las distintas empresas implicadas en el proceso de producción del producto, aunque estas se encuentren en países distintos. La propiedad de los productos queda registrada, por lo que se pueden certificar los orígenes de producto con mayor seguridad, esto es especialmente valioso en productos con denominación de origen, certificaciones ecológicas o productos de lujo. Además, estos sistemas permiten que distintos actores en la cadena de producción interactúen en una única plataforma, común y verificable, sin necesidad de intermediarios.

Es fundamental poder representar la composicionalidad de los distintos productos del sistema, las relaciones entre componentes forman el grafo de productos. Para poder representar estas relaciones, cada token tiene que guardar el ID de los

tokens con los que se relaciona directamente. Estas relaciones forman las aristas del grafo.

Los tokens de productos también almacenarán toda la información referente al producto, como pueden ser su nombre, características, la categoría de producto a la que pertenecen o si es un recambio o añadido. Además, deberá guardar información sobre si el producto sigue o no en uso.

Para simplificar la creación de productos, se plantea un grafo general que recoja el nombre de todos sus componentes y las relaciones entre ellos. Este grafo también estará, al igual que el grafo de productos, formado por tokens ERC-1155, y da las pautas de creación de un producto, indicando cuáles son sus componentes.

El objetivo principal de este sistema es facilitar y optimizar la búsqueda de componentes dentro de distintos productos, así como permitir la transparencia dentro del proceso de producción.

4.1. GRAFOS

Para la visualización de la estructura composicional que posee un producto finalizado, se propuso el uso de grafos dirigidos acíclicos. Estos grafos están compuestos por vértices, que representan productos y aristas que indican relaciones padre-hijo o producto-componente.

Los productos menos elaborados son considerados los hijos, mientras que los productos de los que son componentes son considerados los padres.

Si no tienen relación directa, los productos más elaborados son considerados antecesores. Y en el caso contrario los productos menos elaborados sin relación directa son considerados predecesores.

Una estructura en forma de grafo dirigido acíclico permite que se conozca desde cualquier vértice del grafo o bien los componentes que tiene un producto, o los productos de los que es componente hasta llegar al producto final, dependiendo de la orientación de las aristas.

Una pieza fundamental del prototipo es el almacenamiento del grafo en la propia blockchain, como un conjunto de relaciones entre distintos tokens, donde cada token representa un producto concreto.

Los vértices del grafo son tokens ERC-1155, en los cuales se almacena toda la información referente al producto, así como el propietario del producto. En este caso el concepto de propiedad del producto hace referencia al distribuidor y al fabricante del producto, de esta manera se tiene registro de la procedencia del producto, lo cual es un punto clave si se quiere mantener la trazabilidad.

Cada token contiene también todas sus relaciones directas con los otros tokens. Estas relaciones son las que forman las aristas del grafo.

Al tener el grafo en la blockchain, se garantiza que la información es inmutable y completamente trazable, certificando que la información recogida en este es correcta y generando confianza en los usuarios del sistema.

Cuando sea necesario listar los componentes de un producto se realizará un recorrido en anchura de todos los hijos del vértice que representa el producto en cuestión. Este recorrido refleja nivel a nivel los vértices del grafo y su relación entre sí.

Por el contrario, cuando se quiera conocer en qué productos se encuentra un componente, se realizará también un recorrido en anchura de todos los padres del vértice que representa el producto, ya que se quiere obtener el grafo nivel a nivel.

Para que estos recorridos se puedan realizar correctamente, es necesario evitar la aparición de ciclos. Conceptualmente la existencia de ciclos en el sistema carece de lógica, ya que un producto no puede ser componente y a la vez estar compuesto

de sí mismo. Por ese motivo es importante evitarlos a la hora de implementar la estructura de grafos en el sistema.

Si existiesen ciclos, indicaría que un producto compuesto es componente de otro menos compuesto, lo cual tampoco es posible. En Figura 4.1 se muestra un ejemplo simplificado de un producto donde existe un ciclo. La fecha de color rojo indica una arista que produce un ciclo. En este caso no tendría nunca sentido que el producto finalizado fuese componente de uno de sus propios componentes, en este caso el bolígrafo no puede ser componente de la tinta.

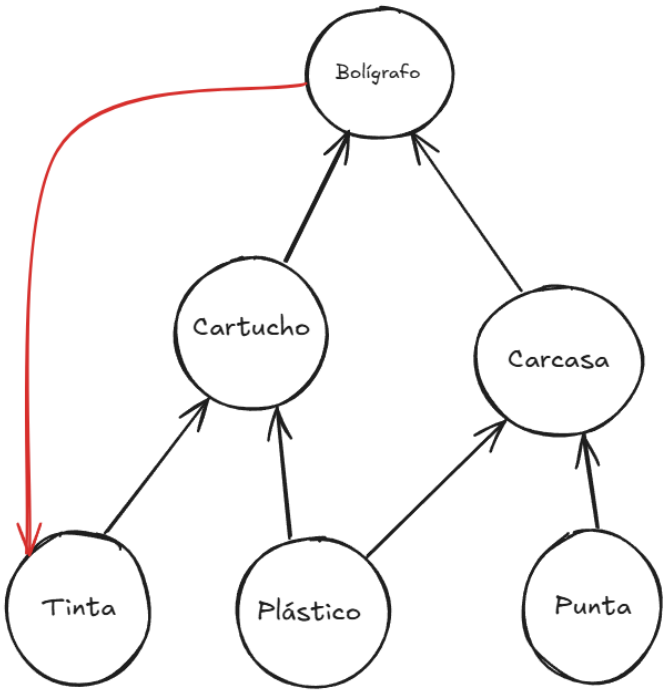

Figura 4.1 Ejemplo de ciclos en un grafo de productos.

Debido al alto coste que conlleva realizar recorridos dentro de la blockchain, estos se realizan off-chain. Esta decisión se tomó ya que el coste de ejecutar funciones recursivas sin saber la cantidad de llamadas necesarias sería demasiado grande. En este caso el coste se mide en gas, al no saber el número de llamadas, no se puede calcular el gas necesario para ejecutar la función y recorrer todo el grafo, por

lo que no se puede establecer un límite de gas suficiente con anterioridad y la función podría detenerse a mitad de ejecución para evitar ataques de denegación de servicio en la red blockchain.

En la estructura propuesta se forma un grafo, que contiene información tanto de los antecesores como de los predecesores de un producto. Este grafo sólo puede recorrer en un sentido cada vez, es decir, sólo puedes recorrer los antecesores o los predecesores, nunca ambos a la vez. Esta manera de recorrer el grafo evita que se produzcan ciclos y permite la trazabilidad completa de este en cualquier dirección y evitando confusiones. Se podría decir que cada arista está duplicada, invirtiendo su sentido. Es fundamental que las aristas sean dirigidas ya que se quiere evitar que se puedan realizar recorridos simultáneos en ambas direcciones, pudiendo en el proceso generar bucles.

Cabe destacar que este sistema de trazabilidad también es empleable para la trazabilidad de procesos. En este caso cada vértice es una etapa o tarea, aunque también puede ser una certificación o aprobación. En la Figura 4.2 se muestra un ejemplo de trazabilidad de procesos, donde el grafo no se centra tanto en la composicionalidad del producto, sino que está enfocado en mostrar los procesos por los que pasa el producto hasta su venta.

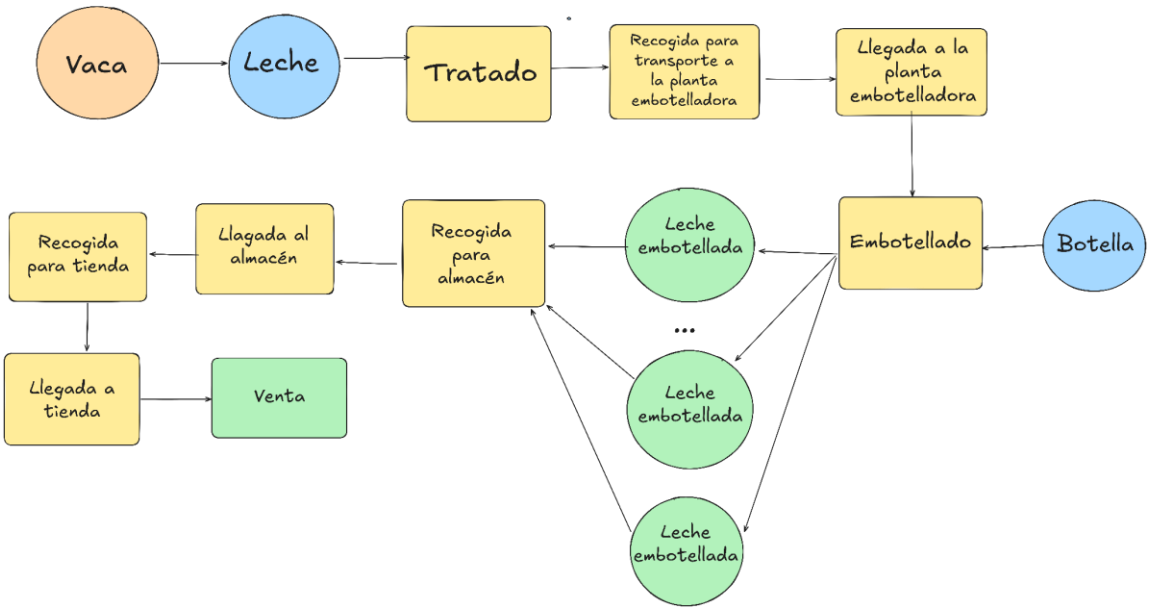

Figura 4.2 Ejemplo de un grafo de procesos.

No se contempla la eliminación completa de un nodo, ya que esto afectaría a la trazabilidad. Por ello, se realiza un borrado lógico, marcando el vértice como inactivo, pero permitiendo que siga existiendo dentro del grafo. Todo esto permite que se mantenga un registro de todos los cambios y componentes que han formado parte del producto. Es muy importante que la información se mantenga inmutable, ya que es una de las razones principales que hacen que el sistema de trazabilidad sea confiable, evitando que se pueda omitir información relevante al estado del producto, manteniendo un histórico de todos los cambios producidos.

Con el fin evitar los ciclos y reducir el número de comprobaciones y recorridos a realizar, se implementan grafos para dos conceptos dentro del sistema, categorías de productos y productos concretos.

4.2. CATEGORÍAS

En este grafo se detallan las categorías o tipos de productos que tiene el sistema y cómo se relacionan entre sí. No indica en ningún momento la cantidad de producto que es necesario, ya que solo indica el tipo de producto que es.

Cuando se genera el grafo de categorías, es necesario que se comience añadiendo el producto más elaborado y se continúen añadiendo sus componentes de más elaborados a menos elaborados. Esta forma de crear el grafo simplifica las comprobaciones de ciclos, reduciendo la necesidad de realizar recorridos y haciendo de esta manera que el sistema sea más eficiente.

Cuando se añade un vértice, el principal requisito a cumplir es que el vértice que se quiere añadir no tenga hijos asociados a él. De esta manera evitas que alguno de esos hijos sea padre del vértice al que se quiere añadir.

Como se explicó anteriormente, este grafo pretende dar las pautas necesarias para la creación de un producto, ya sean los procesos por los que tiene que pasar o los componentes que lo constituyen. Estas categorías no dan información concreta del producto, ya que pretenden ser sólo una guía sobre la que basar el grafo de productos.

4.3. PRODUCTOS

El componente más importante de este sistema de trazabilidad es el grafo de productos, siendo el que contiene la información de los productos que se quieren trazar. Sigue una estructura similar a la del grafo de categorías, con la peculiaridad de que puede haber varios elementos de una misma categoría en él.

En estos grafos existen dos tipos de productos, los divisibles y los no divisibles. Los productos divisibles o primarios son aquellos que pueden aparecer en más de un producto con el mismo id, teniendo más de una arista dirigida hacia distintos padres. Los no divisibles o unitarios, permiten únicamente un padre, teniendo de esta manera una única arista dirigida hacia los padres.

Por lo general los productos divisibles se encuentran en las materias primas del grafo, aunque en el caso de procesos podrían encontrarse en procesos intermedios. En cambio, los productos compuestos tienden a ser productos unitarios.

Cada vértice representa un producto concreto dentro del sistema que tiene almacenada toda la información necesaria de este.

5. IMPLEMENTACIÓN

Este capítulo se centra en el desarrollo del prototipo en Solidity, siendo este el lenguaje más utilizado para programar contratos de Ethereum. En los siguientes apartados se definirán todas las estructuras de datos y funciones que aparecen en el prototipo, además de describir algunas decisiones de diseño tomadas durante el desarrollo del prototipo.

5.1. ESTRUCTURAS DE DATOS

Para la implementación de un sistema de trazabilidad eficiente y seguro, son necesarias una serie de estructuras de datos que permitan organizar toda la información referente a los tokens y a los usuarios. En esta propuesta de token se emplean tres variables globales, una que identificará al administrador del sistema y otras dos que permitirán asignar identificadores únicos a los productos y a las categorías

Estas estructuras de datos se utilizarán en todas las funciones del token permitiendo mantener los datos actualizados en todo momento.

5.2. VARIABLES GLOBALES

El contrato necesita una serie de variables globales que permitan mantener el orden y la seguridad dentro del sistema.

La seguridad en los sistemas de trazabilidad es crucial para mantener los datos consistentes y correctos, evitando que usuarios externos al sistema tengan acceso a este. Para ello se emplea un administrador, encargado de añadir y eliminar usuarios en el sistema, evitando que usuarios externos sean capaces de modificar información en el sistema.

Para implementar el concepto de administrador se guarda el **address** del creador del contrato. Este **address** será utilizado más adelante para verificar que las modificaciones en los usuarios del sistema solo las realiza el administrador:

```
address private admin;
```

Para mantener la organización dentro del sistema se emplean dos contadores, estos contadores indican los IDs de los nuevos productos y categorías que se crean. Ambos comienzan en 0 y se van incrementando cada vez que se crea un nuevo producto o categoría. La declaración de los contadores es la siguiente:

```
uint256 private currTokenID;
uint256 private currCategoryID;
```

Estos contadores son de tipo **uint256**, permitiendo almacenar números enteros sin signo desde 0 a $2^{256} - 1$. Esto permite dar soporte a una gran cantidad de productos dentro del sistema. Este es el tipo de datos más habitual para valores numéricos en Solidity, ya que facilita los cálculos sobre operaciones criptográficas y ofrece un rango muy amplio de valores.

Todas estas variables son privadas ya que solo están destinadas para uso interno dentro del contrato.

5.3. STRUCTS

Cada categoría y producto tiene una serie de información asociada que no solo los describe, sino que facilita la trazabilidad de estos.

Estas estructuras también contienen los datos necesarios para la reconstrucción de los grafos, conteniendo también la información referente a los padres e hijos de los productos o categorías.

5.3.1. CATEGORÍA

Las categorías no necesitan almacenar mucha información, ya que distintos productos en la misma categoría pueden tener características muy distintas especificas a cada producto. Por ello en este caso la única información referente al tipo de producto al que representan es el nombre que aparece en la variable **`name`**. El código referente al struct de categoría es el siguiente:

```
struct category {
    uint256[] isAncestor;
    uint256[] parents;
    uint256[] children;
    string name;
}
```

Para la representación del grafo de categorías explicada en la sección 4.2 se emplean en cada uno de los vértices del grafo dos arrays **`parents`** y **`children`** que contienen los IDs de los vértices adyacentes padres e hijos respectivamente.

Como se explicó al principio de la sección 4.2, las categorías se deben crear en un orden específico para garantizar que el grafo de categorías sea acíclico y dirigido.

En el apartado 5.6.3 se explican con más detalle las operaciones de creación de categorías.

Con el fin de prevenir ciclos en el grafo, es necesario realizar comprobaciones en las categorías que se van a añadir considerando transitivamente todos los ancestros de cada vértice. El coste de realizar un recorrido completo del grafo es demasiado alto para implementarlo en el contrato del token, teniendo solo aplicación práctica en grafos muy sencillos. Para evitar estos recorridos las comprobaciones se realizan a través de un bitmap llamado **`isAncestor`**, que indica si una categoría con un ID determinado es o no ancestro de la categoría que contiene el bitmap.

`isAncestor` es un bitmap representado mediante un array, donde cada posición es un **`uint256`**, es decir cada posición tiene 256 bit disponibles. Cada bit representa un token, por lo que cada **`uint256`** puede representar el estado de 256 tokens distintos, donde 0 indica que no es ancestro y 1 indica que sí lo es. Este bitmap debe representar todos los nodos del grafo de categorías como bits. Esta representación permite determinar qué nodos son ancestros de uno dado. De esta forma, se utiliza una representación muy compacta evitando así recorridos en el grafo. En este caso las comprobaciones se hacen directamente sobre el array, lo que permite un acceso directo a cada elemento. Además, los cálculos de bits son muy eficientes, reduciendo el coste significativamente.

Para localizar un token concreto dentro del array, primero se divide el ID de la categoría entre 256, que obtiene la posición del array en la que se encuentra el bit referente a la categoría que se busca en el bitmap.

Dentro de la posición del array de tipo **`uint256`**, es necesario saber el bit exacto que se necesita, lo cual se determina mediante el resto de la división del ID entre 256. El número resultante es la posición del bit empezando desde la derecha. Un ejemplo del funcionamiento de un bitmap se puede observar en Figura 5.1.

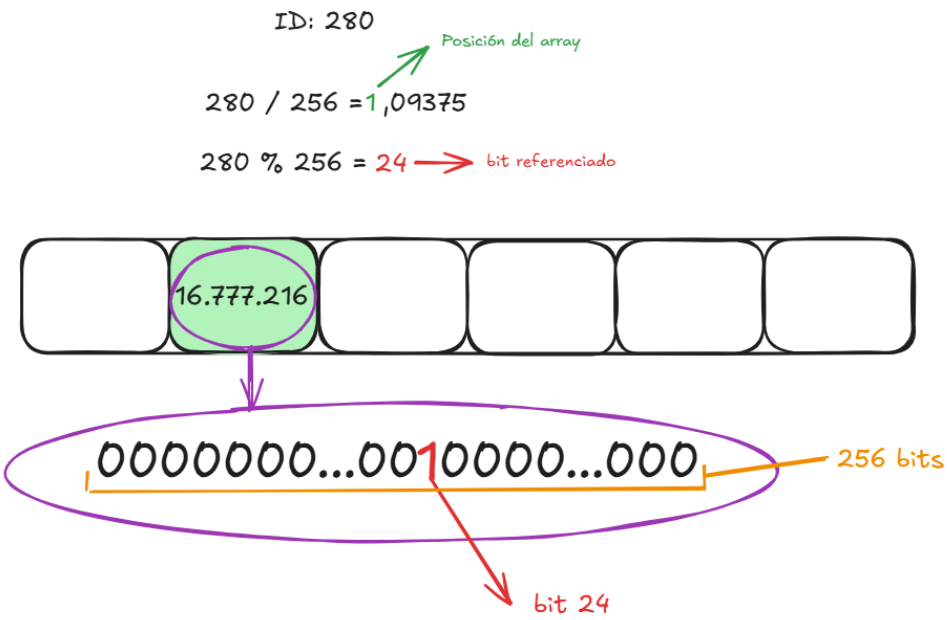

Figura 5.1 Representación de un bitmap.

5.3.2. PRODUCTO

Los productos tienen más información que las categorías dado que son más complejos, pudiendo tener varios tokens de la misma categoría o idénticos excepto por su ID.

El siguiente fragmento de código muestra el struct de producto:

```
struct product {
    bool active;
    bool isReplacement;
    bool isAddOn;

    address owner;

    uint256 productCategory;
    uint256 replacementFor;
    uint256 replacementPiece;
    uint256 amountLeft;

    address[] previousOwners;
    uint256[] parents;
    uint256[] children;
    uint256[] isDescendant;

    string name;
    string description;
}
```

Todos los productos tienen un ID que los distingue entre sí, un nombre que se almacena en la variable **`name`** y una descripción almacenada en la variable **`description`** para poder tener todos los detalles necesarios para identificar el producto. Tienen también un ID de categoría que los clasifica como parte de ésta que aparece en la variable **`productCategory`**.

Al ser tokens, existe la noción de propiedad de estos, en este caso la propiedad del token la tiene primero el fabricante y en caso de haberlo, los distribuidores. De esta manera se tiene constancia del origen del producto y por los procesos por los que ha pasado el producto hasta llegar a su destino. Para esto se emplean la variable **`owner`** que guarda el propietario actual y el array **`previousOwners`** que guarda todos los propietarios anteriores.

Además, tiene un contador con la cantidad de producto disponible llamado **`amountLeft`**. Si el producto no es divisible la cantidad inicial debe ser 1 y después de añadirlo al grafo pasará a ser 0. En cambio, en los productos divisibles la cantidad varía dependiendo de cómo se divida el producto, haciendo que, a la hora de

añadirlo a un grafo, sea necesario indicar la cantidad que se añade. Este producto no se podrá añadir a un grafo si no existe producto suficiente disponible.

También tiene varios indicadores representados con booleanos. El primero de ellos indica si el producto está activo o no, ya que no se realizan borrados lógicos, este booleano indica si el producto sigue formando parte del producto final o no y aparece con el nombre de **`active`**. Los siguientes booleanos indican si el producto ha sido añadido después de crear el grafo, es decir, sin seguir el orden normal de construcción del producto final. Existen dos booleanos de este tipo, uno indica si el producto es un recambio por lo que ha sido intercambiado con otro vértice del grafo en este caso tiene el nombre **`isReplacement`**, mientras que el otro indica si se ha añadido un vértice nuevo al grafo, **`isAddOn`**. Este nuevo vértice no aparece como predecesor de los vértices posteriores, sólo del padre directo, por lo que es necesario indicar que es un añadido.

Estos dos últimos indicadores pueden necesitar información adicional en el caso de ser ciertos. Para los recambios es necesario recoger por qué pieza se cambia, **`replacementFor`** y su padre indicará cuál es su recambio, **`replacementPiece`**.

Por último, los productos también contienen información sobre su posición en el grafo de productos, de manera similar a las categorías como se ha visto en el apartado anterior.

Al igual que las categorías, los productos poseen dos arrays que contienen todos los IDs de los padres y los hijos directos, facilitando así la reconstrucción del grafo. Estos arrays son **`parents`** y **`children`**.

En cambio, a diferencia de las categorías, en el caso de los productos se mantiene un bitmap que indica los productos que son descendientes del producto el cuál se está consultando el bitmap. El funcionamiento del bitmap **`isDescendant`** es el mismo que el explicado en el bitmap **`isAncestor`** de las categorías.

5.4. MAPPINGS

La agilidad en el acceso a los datos de los distintos elementos del sistema es importante para su eficiencia. El lenguaje de programación Solidity incluye de forma nativa estructuras de datos denominadas **`mappings`**, tablas asociativas muy eficientes para almacenar información. Los **`mappings`** funcionan como diccionarios utilizando un elemento como clave y devolviendo el valor establecido. En este sistema se emplean tres **`mappings`** para organizar los datos de los distintos elementos del sistema.

El primer **`mapping`** es de usuarios, donde se almacenan todos los usuarios del sistema. Tiene como clave el **`address`** del usuario y como valor un booleano que indica si es o no parte del sistema. Está declarado en el siguiente fragmento de código:

```
mapping (address => bool) private users;
```

Los otros dos emplean como clave los IDs y devuelven toda la información referente a las categorías y a los productos descrita anteriormente en los structs. Ambos aparecen en el siguiente fragmento de código:

```
mapping (uint256 => category) private categories;
mapping (uint256 => product) private products;
```

Estos **`mapping`** permiten que el acceso a la información se realice con coste constante mediante un único acceso a la memoria persistente del contrato, haciendo que sea mucho más eficiente que otras estructuras de datos.

5.5. MODIFICADORES

Los modificadores son elementos del leguaje Solidity que funcionan como restricciones que se aplican a las distintas funciones y que se repiten a lo largo de estas. En esta sección se listan todos los modificadores que se emplean en el proyecto:

onlyUser

Comprueba que el **`address`** que se introduce por parámetro es el de un usuario del sistema:

```
modifier onlyUser (address user) {
    require(users[user], "01"); // No eres usuario
    _;
}
```

sameAddress

Verifica que las dos **`addresses`** es que se proporcionan como parámetros son iguales:

```
modifier sameAddress(address add1, address add2){
    require(add1 == add2, "02");
    _;
}
```

validCategory

Valida que se haya introducido un nombre de categoría, comprobando que la longitud de la palabra es mayor a 0:

```
modifier validCategory (string memory name) {
    require(bytes(name).length > 0, "03"); // categoria no valida
    _;
}
```

isActive:

Indica si el producto con el ID proporcionado en los parámetros, se encuentra activo o no. En el caso de que el producto no se encuentre activo, no se ejecutará la función que aplique el modificador. El siguiente fragmento muestra su código:

```
modifier isActive (uint256 id) {

    require(products[id].active, "04"); // Producto no activo
    _;
}
```

enoughProduct

Comprueba si el producto con el ID indicado en los parámetros tiene una cantidad de producto suficiente para cubrir la demanda indicada por **`amount`**. Su implementación se muestra en el siguiente fragmento de código:

```
modifier enoughProduct (uint256 productID, uint256 amount) {
    require(products[productID].amountLeft >= amount, "05"); // No queda suficiente producto
    _;
}
```

canBeProductChild

Comprueba si un producto es hijo de otro. Para ello, con el ID del primer producto busca en el bitmap de ese producto la posición en bits correspondiente al ID del otro producto, tal y como explico anteriormente. Para evitar errores de acceso, comprueba primero si la longitud del bitmap permite realizar la comprobación y

en caso contrario añade todas las posiciones necesarias hasta llegar a la que indica el producto al final. Su código se muestra a continuación:

```
modifier canBeProductChild (uint256 product1, uint256 product2) {
    extendBitmap(products[product1].isDescendant, product2);
    require( and(products[product1].isDescendant[product2 / 256], product2) == 0, "06");
    _;
}
```

canBeCategoryParent

Este modificador tiene una función similar a "canBeProductChild", pero comprueba si una categoría es padre de otra. En este modificador también es necesario comprobar la longitud del bitmap. Su código es el siguiente:

```
modifier canBeCategoryParent (uint256 product1, uint256 product2) {
    extendBitmap(categories[product1].isAncestor, product2);
    require( and(categories[product1].isAncestor[product2 / 256], product2) == 0, "07");
    _;
}
```

isDirectChildCategory

Este modificador comprueba si las categorías del producto padre e hijo tienen una relación directa. Esto es importante ya que relaciona el grafo de categorías con el grafo de productos, obligando a que se cumplan en el grafo de productos las relaciones descritas en el grafo de categorías. En el siguiente fragmento de código se muestra su implementación:

```
modifier isDirectChildCategory (uint256 cat1, uint256 cat2) {
    bool isDirectChild = false;

    for(uint256 i = 0; i < categories[cat1].children.length; i++){
        if(categories[cat1].children[i] == cat2){
            isDirectChild = true;
        }
    }

    require(isDirectChild, "0");
    _;

}
```

El uso de estos modificadores permite simplificar el código y realizar verificaciones antes de comenzar la ejecución de las funciones.

5.6. FUNCIONES

Para construir y gestionar los grafos, son necesarias una serie de funciones que controlen la lógica del sistema y realicen las comprobaciones necesarias para mantenerlo seguro.

5.6.1. FUNCIONES DE USUARIO

Para garantizar la privacidad y seguridad del sistema, solo se permite que los usuarios designados por el administrador puedan interactuar con los productos. Las funciones de usuario tienen como objetivo permitir al administrador proporcionar y revocar permiso a los usuarios para ejecutar las funciones de producto.

AÑADIR USUARIO

Esta función permite al administrador añadir usuarios al sistema. Modifica en el mapping **users** la posición correspondiente al **address** del usuario que se quiere añadir, asignando valor true al booleano. De esta manera cuando se vuelva a consultar el mapping en esa posición, indicará que el usuario es parte del sistema.

El siguiente fragmento muestra el código de esta función:

```
function addUser (address userAddress) public sameAddress(msg.sender, admin) {
    require(!users[userAddress] , "12"); // Ya existe el usuario

    users[userAddress] = true;
}
```

Se emplea el modificador **sameAddress** para comprobar que es el administrador quien ejecuta la función.

Además, se comprueba si el usuario ya es parte del sistema, para evitar redundancias.

ELIMINAR USUARIO

La función eliminar funciona al contrario que la de añadir, marca en el mapping de usuario en la posición correspondiente al **address** del usuario que se desea eliminar, el booleano correspondiente a false. Su código aparece en el siguiente fragmento:

```
function deleteUser (address userAddress) public sameAddress(msg.sender, admin) onlyUser(userAddress) {
    users[userAddress] = false;
}
```

Esta función además de emplear el modificador **`sameAddress`** para verificar que la función es ejecutada únicamente por el administrador, emplea la función **`onlyUser`** cerciorándose así de que el usuario era parte del sistema antes de eliminarlo.

5.6.2. FUNCIONES AUXILIARES

OR

Esta función tiene como objetivo devolver el resultado de realizar un "|" lógico entre los bits de dos números. Para ello se pasan por parámetro un número y un token ID. Esta función se emplea para obtener el número que se genera al marcar con un uno el bit correspondiente al token ID en un bitmap. En el siguiente fragmento de código se muestra la función:

```
function or(uint256 num, uint256 tokenID) internal pure returns(uint256){
    return num | (1 << (256 - (tokenID % 256)));
}
```

Las operaciones entre paréntesis generan un número cuya representación en binario tiene un solo 1 en el bit que representa el token ID proporcionado en **`uint256`**, no se tiene en cuenta la posición del array porque eso se evaluará en otras funciones. Con el bit calculado, se realiza un "|" lógico que consiste en unir los bits de ambos números para que tengan los mismos unos marcados, es decir que deben tener las mismas posiciones marcadas con un 1 sin cambiar a 0 las que ya tenían marcadas.

AND

Esta función comprueba si en el número proporcionado en los argumentos, la posición en bits que ocupa el token ID proporcionado también en los argumentos está marcado con un 1 o no. Si no está marcada con un 1 devuelve 0, y en el caso contrario devuelve un número distinto de 0. En el código siguiente se muestra la función:

```
function and(uint256 num, uint256 tokenID) internal pure returns(uint256) {
    return num & (1 << (256 - (tokenID % 256)));
}
```

Funciona creando una máscara que hace que todos los bits pasen a ser 0 excepto el indicado tras el "**&**". Al igual que la función anterior, esta función se creó para reducir el espacio que ocupaba el código.

EXTENDER EL BITMAP

Para evitar accede a posiciones no existes en los bitmaps se emplea la función **`extendBitmap`**. Se le proporcionan como argumentos el array a extender **`arr`** y el valor que se necesita guardar en el bitmap **`val`**.

Se comienza comprobando si la posición para almacenar ese valor existe ya dividiendo val entre 256, encontrando así la posición del array en la que se almacenaría y comprobando si es menor o igual a la longitud del array.

En el caso de que la longitud del array sea menor que la posición necesaria para almacenar el valor, se ejecuta un bucle que añade ceros al final del array extendiendo sus posiciones hasta que alcance la longitud necesaria. En el siguiente fragmento de código se muestra su implementación:

```
function extendBitmap(uint256[] storage arr, uint256 val) internal {
    if((arr.length <= val/256)) {
        for(uint256 i = arr.length; i < val/256 + 1; i++){
            arr.push(0);
        }
    }
}
```

5.6.3. FUNCIONES DE CATEGORÍA

Para la creación del grafo de categorías son necesarias una serie de funciones que gestionen y realicen las comprobaciones necesarias para su correcta construcción.

CREAR CATEGORÍAS

La función de creación consiste en proporcionar un ID a la nueva categoría, así como añadir un 0 en el bitmap de ancestros, de esta manera lo inicializa y deja disponibles las primeras 256 categorías para añadirlas como ancestros de esta categoría. En el caso de existir más de 256 categorías, la extensión del bitmap se realizará con la función auxiliar **`extendBitmap`**. Por último, añade el nombre proporcionado en los argumentos e incrementa el contador de categorías. Su código es el siguiente:

```
function createCategory (string calldata name)
public
sameAddress(msg.sender,admin) validCategory(name)
returns (uint256 catID) {
    catID = currCategoryID;
    categories[currCategoryID].isAncestor.push(0);
    categories[currCategoryID].name = name;
   unchecked {++ currCategoryID; }
    return(catID);
}
```

Esta función tiene como modificadores **`sameAddress`**, donde se comprueba que el usuario que la ejecuta es el administrador, y **`validCategory`**, que comprueba que se ha proporcionado un nombre para la categoría.

UNIR CATEGORÍAS

Esta función es la encargada de generar el grafo de categorías, relacionando dos categorías existentes mediante una arista dirigida de categoría de producto más complejo a categoría de producto más simple.

Para unir categorías se necesita realizar varias comprobaciones, comenzando con el modificador **sameAddress** para comprobar que solo el administrador la ejecuta. También emplea el modificador **validCategory** para comprobar que existen categorías con los IDs proporcionados. El último modificador utilizado es **canBeCategoryParent** dónde se comprueba si la categoría que se quiere unir es padre de la otra.

El siguiente fragmento muestra su definición, parámetros y modificadores:

```
function addCategory (uint256 parentID, uint256 childID)      infinite gas
 public
 sameAddress(msg.sender, admin)
 validCategory(categories[parentID].name) validCategory(categories[childID].name)
 isCategoryParent(parentID, childID){
```

Después se añade el ID de la categoría hija en el bitmap **isDescendant** de la categoría padre, y el ID de la categoría padre en el bitmap **isAncestor** de la categoría hija:

```
categories[parentID].children.push(childID);
categories[childID].parents.push(parentID);
```

El siguiente paso consiste en comprobar si el bitmap de la categoría padre es menor o mayor que el bitmap de la categoría hija, de esta manera se evita que al unir los bitmaps se creen problemas de acceso.

En el caso de que el bitmap de la categoría hija sea mayor que el de la categoría padre, no habría problema, ya que se quiere fusionar en el bitmap del hijo las

categorías padre de ambos. Por lo que en este caso no se producirían errores de acceso.

En el caso contrario, se añaden al final del bitmap de la categoría hija las posiciones que le faltan para tener el mismo tamaño que el bitmap de la categoría padre, con los valores que tiene el de la categoría padre.

Esta lógica se muestra en el siguiente fragmento:

```
uint256 len;
if(categories[parentID].isAncestor.length < categories[childID].isAncestor.length){
    len = categories[parentID].isAncestor.length;
}
else {
    len = categories[childID].isAncestor.length;

    for(uint256 i = len; i < categories[childID].isAncestor.length; ++i){
        categories[childID].isAncestor.push(categories[parentID].isAncestor[i]);
    }
}
```

Para completar la unión de los bitmaps se juntan con ***or*** lógico todos los bits de cada posición. De esta manera, el bitmap de la categoría hija tendrá marcados con 1 todos sus padres y los padres que tenía almacenados en su bitmap la categoría padre:

```
for(uint256 i = 0; i < len; ++i){
    categories[childID].isAncestor[i] = categories[childID].isAncestor[i] | categories[parentID].isAncestor[i];
}
```

Finalmente se añade la categoría padre al bitmap del hijo. Comprobando primero si es necesario añadir una posición adicional el bitmap o no:

```
if(categories[childID].isAncestor.length <= (parentID/256)) {
    for(uint256 i = categories[childID].isAncestor.length; i < (parentID/256); ++i){
        categories[childID].isAncestor.push(0);
    }
    categories[childID].isAncestor.push(or(0,parentID));
}
else {
    categories[childID].isAncestor[parentID/256] = or(categories[childID].isAncestor[parentID/256], parentID);
}
```

VER CATEGORÍAS

Esta función permite ver todas las características de una categoría dado su ID, devolviendo su nombre y los arrays de padres e hijos directos, así como su bitmap:

```
function getCategory (uint256 id) public view onlyUser(msg.sender)     infinite gas
validCategory( categories[id].name)
returns (uint256[] memory, uint256[] memory, uint256[] memory, string memory)
{
    return (categories[id].parents, categories[id].children, categories[id].isAncestor, categories[id].name);
}
```

Para poder ver la información de una categoría, es necesario ser usuario del sistema, lo que se comprueba con el modificador **`onlyUser`**. Además, hay que comprobar que la categoría existe por lo que se emplea el modificador **`validCategory`**.

5.6.4. FUNCIONES DE PRODUCTO

La creación y gestión del grafo de productos son más complejas que en las categorías. La complejidad de los casos hace que sea necesario añadir funciones específicas para simplificar y optimizar las funciones principales.

CREAR PRODUCTO

Los tokens se crean con la función **`createToken`**, que es la encargada de definir todas las características del producto, así como su cantidad disponible en un inicio.

En el caso de querer crear más de un producto con las mismas características, se crean N tokens idénticos que solo se distinguen por su **`tokenID`**.

Esta función genera tantos tokens como se indiquen en los argumentos, teniendo todos como dueños al usuario que ejecutó la función.

Para la función de creación de productos, son necesarios dos modificadores **`onlyUser`** para evitar que usuarios ajenos al sistema ejecuten la función y **`validCategory`** para asegurar que la categoría del producto existe:

```
function createToken (uint256 categoryID,uint256 amountTokens,string calldata name, string calldata description, uint256 amountLeft)
public
onlyUser(msg.sender) validCategory(categories[categoryID].name)
returns(uint256[] memory)
{
```

No está permitido que se creen más de 100 productos a la vez, ya que generaría bucles muy grandes, pudiendo gastar todo el gas establecido antes de terminar su ejecución. Este requisito se muestra en la siguiente imagen:

```
require(amountTokens <= 100, "08" ); //demasiados token
```

El siguiente paso es crear en una variable local un producto cuyos datos se rellenan con los argumentos nombre, descripción, categoría y cantidad disponible, proporcionados en la función. Además, se marca el producto como activo y los booleanos de añadido y recambio a false. Por último, se guarda como dueño del token al usuario que ejecutó la función y se inicializa el array con el bitmap de descendientes con un 0 en la primera posición. Este proceso se puede observar en el siguiente código:

```
product storage newProduct = products[currTokenID];

uint256[] memory tokenIDs = new uint256[](amountTokens);

newProduct.active = true;
newProduct.isAddOn = false;
newProduct.isReplacement = false;
newProduct.owner = msg.sender;
newProduct.amountLeft = amountLeft;
newProduct.productCategory = categoryID;
newProduct.isDescendant.push(0);
newProduct.name = name;
newProduct.description = description;
```

Este producto que se ha creado sólo existe como variable local, por lo que es necesario añadirlo al mapping de productos. Para ello se añade en las posiciones correspondientes al ID de los productos que se va a crear con estas características el producto descrito en la variable local y se añaden sus IDs a un array de IDs que serán devueltos al final de la función. Esta inicialización se muestra en la siguiente imagen:

```
for(uint256 i=0; i < amountTokens; ++i) {
    tokenIDs[i] = currTokenID;
    products[currTokenID] = newProduct;
    unchecked { ++currTokenID; }
}
```

Finalmente se crean los tokens con la función **`_mint`** procedente del estándar ERC-1155. En el caso de que solo se quiera crear un token se emplea la función "**`_mint`**", tiene como argumentos, el dueño del token, su ID, la cantidad de tokens que se quieren crear que en este caso es 1 y datos adicionales que se pasan vacíos.

Por otro lado, si se van a crear más de un token con las mismas características pero distinto ID, se utiliza la función **`_mintBatch`**, a la cual se le pasan todos los IDs de los tokens que se van a crear.

Estas llamadas se muestran en el siguiente fragmento de código:

```
if(amountTokens >= 1) {
    _mintBatch(msg.sender, tokenIDs, new uint256[](amountTokens), "");
}
else {
    _mint(msg.sender, tokenIDs[0], 0,"");
}

return tokenIDs;
```

ELIMINAR PRODUCTO

La función eliminar producto emplea como muchas otras el modificador **`onlyUser`** para comprobar que el usuario es parte del sistema, además de emplear **`isActive`** para comprobar que el producto existe y no ha sido borrado ya, y **`sameAddress`** para comprobar que el usuario que está eliminando el producto es el dueño del token.

La función se muestra en el siguiente fragmento de código:

```
function deleteProduct (uint256 tokenID)
public
onlyUser(msg.sender)
isActive(tokenID)
sameAddress(msg.sender, products[tokenID].owner)
{
    products[tokenID].active = false;
}
```

Es una función muy simple, ya que su única función es marcar a false el booleano **`isActive`** indicando que el producto no está disponible.

VER PRODUCTO

Esta función permite ver toda la información del producto. Esta función no tiene modificadores, ya que por transparencia debería permitir a cualquier usuario ver la información referente al producto.

El siguiente fragmento muestra el código de esta función:

```
function getProduct (uint256 tokenID) public view  returns(product memory )  {

    return(products[tokenID]);
}
```

Se devuelve toda la información recogida en la estructura de producto para el producto indicado con el ID que se proporciona en los argumentos.

UNIR PRODUCTOS

Esta función es la encargada de generar el grafo de productos. Su función es unir dos productos en el grafo, con los IDs indicados en los argumentos **`childID`** y **`parentID`**, creando una arista entre ellos. Se añaden los IDs del padre y del hijo en los arrays correspondientes de cada token y uniendo en el bitmap del padre los bitmaps de predecesores de ambos. Esta función no tiene en cuenta cambios de piezas o complementos, estos serán tratados en otras funciones.

El argumento **`amount`** indica la cantidad de producto hijo que se necesita. Aunque la relación sea con el token completo, es necesario indicar la cantidad que se emplea del producto, para evitar añadir productos ya acabados, es decir que no se pueden emplear, al grafo de productos. Este **`amount`** se resta a la variable **`amountLeft`** del producto.

Para poder ejecutar esta función es necesario cumplir una serie de restricciones, comenzando por que el usuario debe pertenecer al sistema. Además, los productos que se quieren unir deben estar activos, debe de quedar suficiente cantidad de producto hijo para satisfacer lo pedido en los argumentos y deben tener una relación directa en el grafo de categorías. Para ello se emplean los modificadores **`onlyUser`**, **`isActive`**, **`enoughProduct`**, **`isDirectChildCategory`**. Esto se ejecuta en el siguiente fragmento de código:

```
function joinProduct (uint256 childID, uint256 parentID, uint256 amount)    infinite gas
 public
 onlyUser(msg.sender)
 isActive(childID) isActive(parentID)
 enoughProduct(childID, amount)
 isDirectChildCategory(products[parentID].productCategory, products[childID].productCategory)
 {
```

También es necesario comprobar que el producto padre no tiene a su vez padres, ya que esto complicaría la creación del grafo, haciendo necesario realizar recorridos añadiendo en todas las capas superiores al producto hijo como hijo en los mappings **`isDescendant`**. Esta limitación del sistema impone una restricción sobre el orden en el que se añaden los productos al grafo, pero evita la necesidad de realizar recorridos completos del grafo, que son muy costosos. Existen casos que no están contemplados por la implementación propuesta por ser excepcionales, pues añadir un producto a otro ya terminado es un caso raro. Este caso se contempla en la función **`addOn`**. En el siguiente fragmento de código se muestra cómo se realiza la comprobación de que el producto padre no tiene otros padres, es decir su array de padres **`parents`** está vacío:

```
require(products[parentID].parents.length == 0, "09");
```

Una vez cumplidos los requisitos, hay que unir los productos, para ello se reduce la cantidad disponible, almacenada en la variable **`amountLeft`** del producto, en la cantidad indicada en el argumento **`amount`**. Al restar estas cantidades, se asegura

que siempre sea posible conocer la cantidad disponible de un producto. También se añade el ID del producto padre en el array de padres del producto y viceversa. En el siguiente fragmento de código se muestran las operaciones descritas:

```
unchecked {products[childID].amountLeft -= amount; }
products[childID].parents.push(parentID);
products[parentID].children.push(childID);
```

Después se marca el producto como hijo en el bitmap del producto padre, de la misma manera que se hizo con las categorías. En este caso se comprueba si el bitmap del padre es de mayor o menor longitud que el del hijo, en caso de que el bitmap del hijo sea de mayor tamaño, se añaden tantas posiciones como diferencia haya entre las dos longitudes. Estas posiciones tendrán el mismo valor que sus equivalentes en el bitmap del producto hijo. Después de realizar está comprobación, se marcan en el bitmap del padre las posiciones que aparecen marcadas como hijo en el bitmap del producto hijo. De esta manera cada vértice del grafo tiene todos los hijos que le resultan relevantes.

En la siguiente imagen se muestra el código referente a la lógica descrita en el párrafo anterior:

```
uint256 len;
if(products[childID].isDescendant.length < products[parentID].isDescendant.length){
    len = products[childID].isDescendant.length;
}
else {
    len = products[parentID].isDescendant.length;

    for(uint256 i = len; i < products[childID].isDescendant.length; ++i){
        products[parentID].isDescendant.push(products[childID].isDescendant[i]);
    }
}
```

Para finalizar se marca como hijo en el bitmap del padre. De esta manera evita que se pueda volver a añadir, que se creen ciclos y facilita a los futuros vértices las comprobaciones de ciclos:

```
if(products[parentID].isDescendant.length < (childID/256)) {
    for(uint256 i = products[parentID].isDescendant.length; i < (childID/256); ++i){
        products[parentID].isDescendant.push(0);
    }
    products[parentID].isDescendant.push(or(0,childID));
}
else {
    products[parentID].isDescendant[childID/256] = or(products[parentID].isDescendant[childID/256],childID);

}
```

AÑADIR RECAMBIO

En el caso de que un producto se estropee o se quiera reemplazar por otro, se utiliza la función **`replaceProduct`**, encargada de indicar en el grafo que un producto ha sido cambiado por otro, sin eliminar el producto original del grafo.

En esta función igual que en la anterior es necesario indicar en el argumento **`amount`** la cantidad de producto que se va a emplear. En este caso además de los IDs de los productos padre e hijo indicados en los argumentos **`parentID`** y **`childID`**, es necesario indicar el id del producto que se va a añadir al grafo, es decir el recambio, este id se indica en el argumento **`replacementID`**.

Para añadir el recambio de un producto, se necesitan los mismos modificadores que para unir productos **`onlyUser`**, **`enoughProduct`** y **`isActive`**. Además, se comprueba si el producto que se quiere usar como recambio es ya hijo del padre del producto que se quiere reemplazar. En el siguiente código se muestran todos los argumentos y modificadores necesarios para esta función:

```
function replaceProduct (uint256 parentID, uint256 childID, uint256 replacementID,uint256 amount)
public
onlyUser(msg.sender)
isActive(replacementID) isActive(childID) isActive(parentID)
enoughProduct(replacementID, amount)
canBeProductChild(parentID, replacementID)
{
```

Para poder cambiar un producto, el recambio debe ser de la misma categoría que el producto que se quiere reemplazar, por lo que es necesario comprobar que la variable **`productCategory`** de ambos productos es igual:

```
require((products[childID].productCategory == products[replacementID].productCategory), "10");
```

Lo primero que se hace es restar la cantidad indicada en los argumentos a la cantidad restante de producto y marcar el producto que se quiere cambiar como inactivo. Además, el recambio cambia su booleano **`isReplacement`** a true para indicar que es un recambio. El producto a cambiar indica cuál es el ID de su reemplazo y el recambio indica cuál es el producto que está sustituyendo. Finalmente, el recambio añade a su array de padres el ID del padre del producto original. El código referente a esta lógica descrita aparece a continuación:

```
unchecked {products[replacementID].amountLeft -= amount;}
products[childID].active = false;
products[replacementID].isReplacement = true;
products[childID].replacementPiece = replacementID;
products[replacementID].replacementFor = childID;
products[replacementID].parents = products[childID].parents;
```

Añadiendo los recambios de esta manera se puede saber qué piezas han sido cambiadas, además de mantener la información de la pieza original en el grafo y se evita tener que recorrer el grafo añadiendo el recambio como hijo de los productos de las capas superiores.

AÑADIR COMPLEMENTO

En el caso de querer añadir un producto a otro ya finalizado, se emplea la función **addOn**. Esta función añade el producto al grafo, indicando que fue añadido con posterioridad a la finalización del producto.

Los complementos se añaden de una manera muy similar a los recambios, por ello comparte la mayoría de sus modificadores, añadiendo a estos el modificador **canBeCategoryParent** que comprueba si la categoría del producto que se quiere añadir es padre de la categoría del producto padre. En esta función también es necesario el argumento **amount** para indicar la cantidad de producto que se empleará. El siguiente fragmento de código muestra todos los argumentos y modificadores de la función:

```
function addOn (uint256 parentID, uint256 addOnPiece, uint256 amount)    infinite gas
public
onlyUser(msg.sender)
isActive(addOnPiece) isActive(parentID)
enoughProduct(addOnPiece, amount)
canBeProductChild(parentID, addOnPiece)
canBeCategoryParent(products[parentID].productCategory, products[addOnPiece].productCategory)
{
```

Para indicar que este producto es un añadido, se marca el booleano **isAddOn** a true, además se resta la cantidad indicada a la cantidad restante de producto. También se marca el producto como hijo en el bitmap de hijos del padre y se añade a su array de hijos directos. Finalmente se añade el ID del padre al array de padres directos del añadido. Esta lógica se puede observar en el siguiente fragmento de código:

```
unchecked { products[addOnPiece].amountLeft -= amount;  }
products[addOnPiece].isAddOn = true;
products[parentID].isDescendant[addOnPiece/256] = or(products[parentID].isDescendant[addOnPiece/256] ,addOnPiece);
products[parentID].children.push(addOnPiece);
products[addOnPiece].parents.push(parentID);
```

De esta manera se añade un producto que no existía originalmente en el producto final, solucionando el problema de añadir vértices una vez finalizado el grafo, además de indicar que el producto ha sido un añadido posterior.

5.7. DECISIONES DE DISEÑO

Durante el desarrollo del prototipo se tomaron una serie de decisiones de diseño con el objetivo de mejorar la eficiencia y asegurar la viabilidad del prototipo.

La primera decisión que se tomó fue la creación del grafo de categorías, el cual ayuda a mantener siempre la misma estructura cuando se crea un producto en el grafo de productos. Además, este grafo permite evitar bucles innecesarios cuando se añade un recambio, ya que obliga a los productos a compartir categoría, y comprueba cuando se introduce un añadido que tenga una colocación adecuada dentro del producto final.

La segunda decisión de diseño está relacionada con el tamaño del código ejecutable del contrato, que es superior al tamaño máximo permitido por Ethereum, tan solo 24KB. Este problema se podría haber solucionado dividiendo en dos el contrato, donde un contrato contendría la lógica del producto y el otro el de las categorías. El problema de esta solución es que requiere de un consumo de gas mayor, por ello en su lugar se activó la función de optimización en el compilador, con un número muy bajo de ejecuciones para que se redujera al máximo el tamaño del código ejecutable. Esta función permite que el despliegue inicial del contrato cueste menos gas, aunque las ejecuciones posteriores sean un poco más caras. Además, para ayudar a reducir el tamaño del contrato, se introdujeron las funciones “and” y “or”, que, al repetirse en otras funciones del contrato, redujeron el coste de este. También se eliminaron los textos descriptivos que aparecian en los mensajes de error, optando en su lugar por códigos de error.

Los códigos de error que aparecen actualmente se muestran en Figura 5.1 con sus mensajes asociados.

CÓDIGO	MENSAJE
01	No es usuario del sistema.
02	Los **`address`** no coinciden.
03	No es una categoría válida.
04	El producto no está activo.
05	No queda suficiente producto.
06	Los productos ya están relacionados.
07	Las categorías ya están relacionadas.
08	Las categorías de los productos no tienen relación directa.
09	Se están intentando crear demasiados tokens a la vez.
10	El producto ya está acabado.
11	La pieza que se intenta reemplazar no es compatible con la original.
12	Ya existe el usuario.

13	El usuario no es **`owner`** del producto.

Tabla 5.1. Códigos de error

También se intentó implementar el prototipo en una blockchain privada empleando *Hyperledger Fabric*, pero ese sistema de blockchain está dirigido a soluciones empresariales que utilizan contratos tan complejos, además de carecer de una documentación técnica avanzada para resolver los problemas que se pudieran encontrar en su implementación. Por estos motivos, se decidió en su lugar centrar el prototipo en crear un token más robusto en Ethereum.

5.8. ACCESO AL CÓDIGO FUENTE

El código del prototipo descrito en este capítulo se encuentra en el siguiente repositorio de GitHub: https://github.com/ClaraS-22/Sistema-de-trazabilidad-basado-en-tecnologia-blockchain

6. PRUEBAS DEL SISTEMA

Para probar que el sistema funciona correctamente se han realizado una serie de pruebas basadas en un caso de uso, que consiste en el proceso de producción de una batería de ion de litio. La idea es conseguir un registro de la trazabilidad de una batería, sus componentes y todos los procesos por los que pasa.

Se comienza desplegando el contrato, lo que definirá quien es el administrador. En la Figura 6.1. se muestra el **`address`** que tendrá el administrador, es decir el **`address`** con la que se despliega el contrato.

*Figura 6.1. **`Address`** del administrador.*

En la Figura 6.2. se muestra el resultado de desplegar el contrato con el **`address`** del administrador y muestra el **`address`** que tendrá el propio contrato.

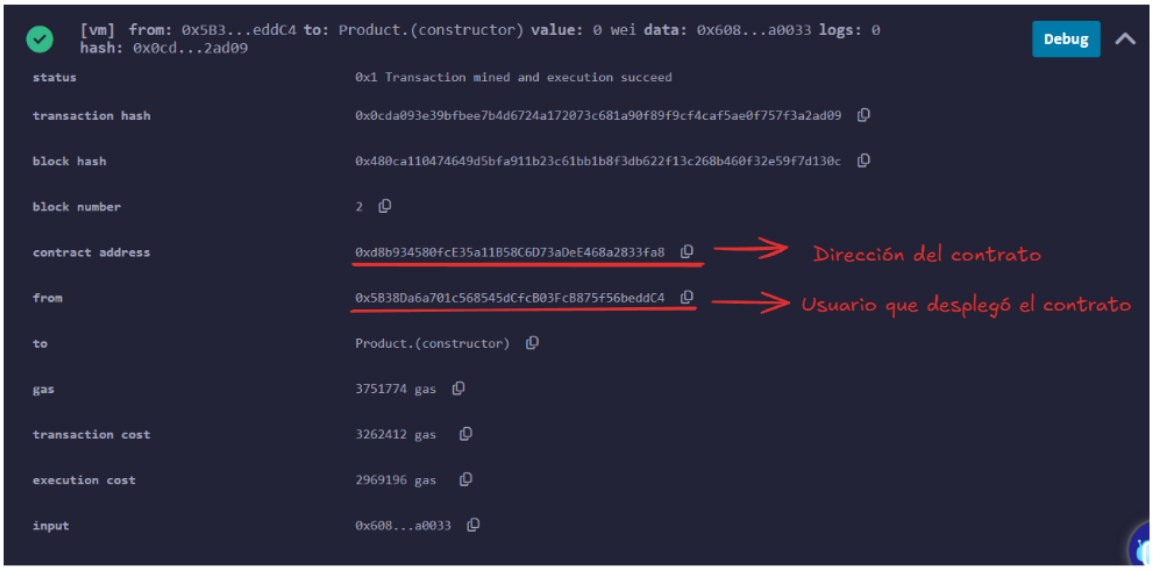

Figura 6.2. Despliegue del contrato.

El siguiente paso es añadir los usuarios del sistema con la función **addUser** como se muestra en la Figura 6.3. Esto les permitirá ejecutar las funciones del sistema necesarias para poder completar el grafo de productos.

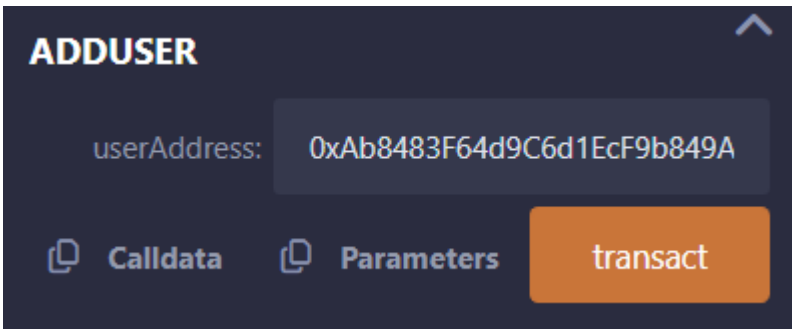

*Figura 6.3. Ejemplo de uso de la función **addUser**.*

En la Figura 6.4 se muestran los **address** escogidos para los usuarios. Para no complicar el ejemplo solo habrá cuatro usuarios que se repartirán los componentes y procesos siendo el usuario 1 el responsable del cátodo y todos sus componentes y procesos. El usuario 2 se encargará del ánodo y los nodos que se relacionan con él. El usuario 3 se encargará del electrolito y el separador. Finalmente, el usuario 4 será el propietario de la celda final.

Figura 6.4. Direcciones de los usuarios del ejemplo.

Se añadirán uno por uno los usuarios mencionados, mostrando en la consola un mensaje de éxito como el que aparece en la Figura 6.5.

Figura 6.5. Éxito al añadir un usuario.

A continuación, el administrador crea el grafo de categorías. Este grafo tiene como objetivo definir el tipo de componentes que deberá tener el grafo de productos. En la Figura 6.6 se muestra como debe ser el grado de categorías, indicando todos los procesos y componentes que formarán una batería. Los nodos que aparecen en azul hacen referencia a los productos físicos, mientras que los nodos amarillos hacen referencia a los procesos por los que pasan esos productos.

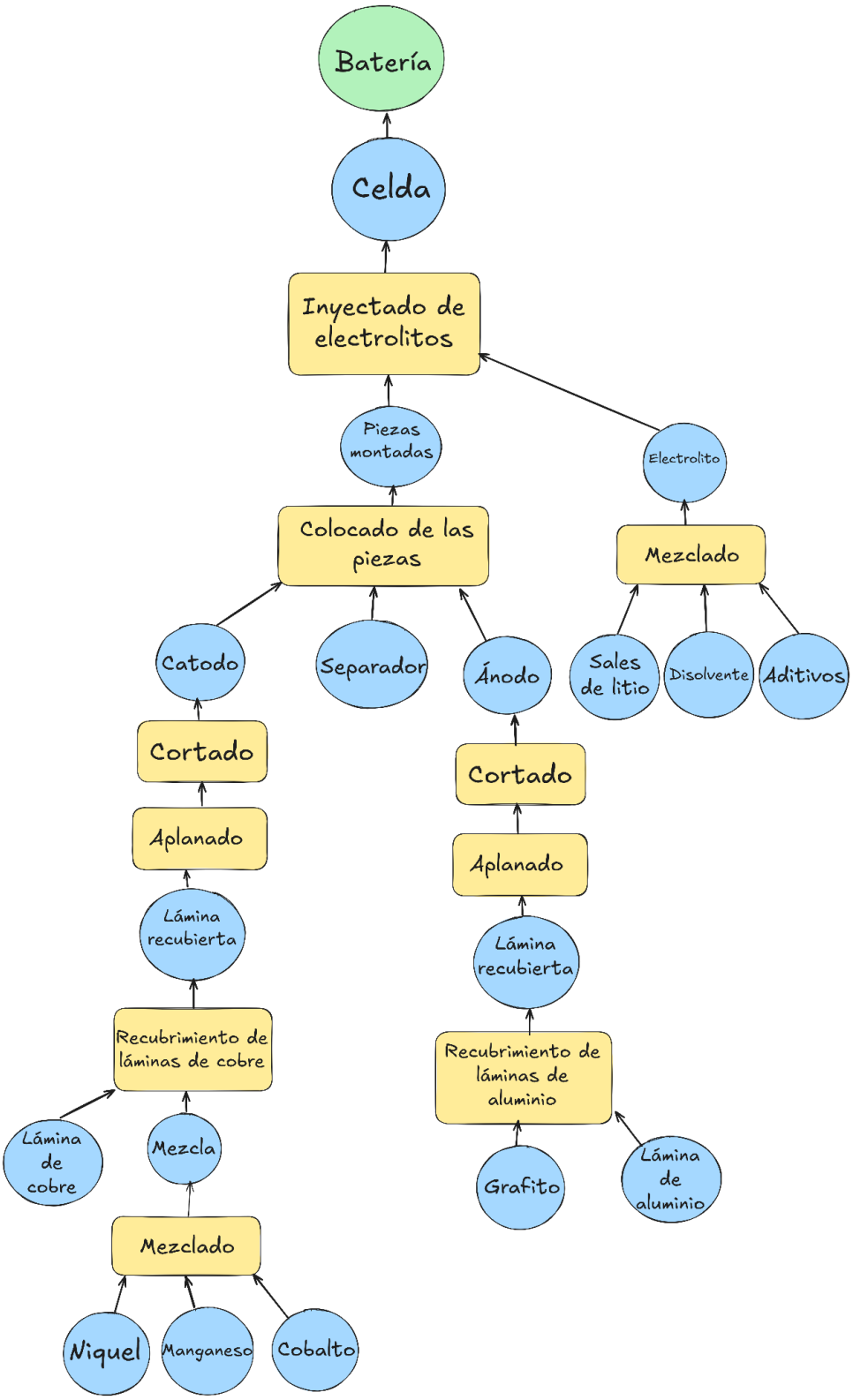

Figura 6.6. Grafo de productos y procesos de una batería de ion litio.

La creación del grafo de categorías comienza añadiendo los nodos que representarán los productos y procesos. En la Figura 6.7 se muestra un ejemplo de cómo crear un nodo para un producto.

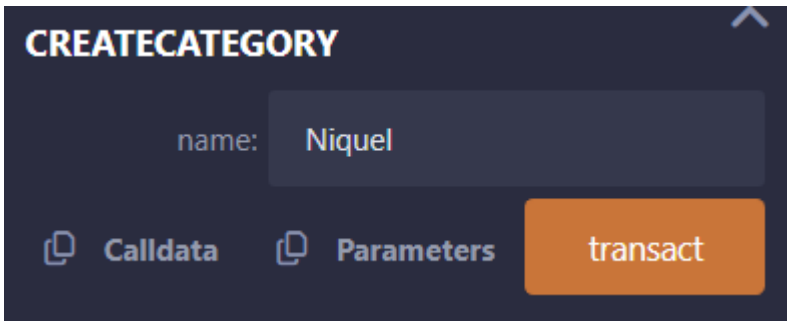

*Figura 6.7. Ejemplo de uso de la función **createCategory**.*

La Figura 6.8 muestra el resultado de ejecutar la función **createCategory**, la cual devuelve el ID del nodo en este caso 0.

*Figura 6.8. Resultado de ejecución de la función **createCategory**.*

Para unir categorías, es decir crear una relación entre ellas se emplea la función **addCategory** a la que se le proporcionan los ID de los productos padre e hijo, como se puede observar en Figura 6.9

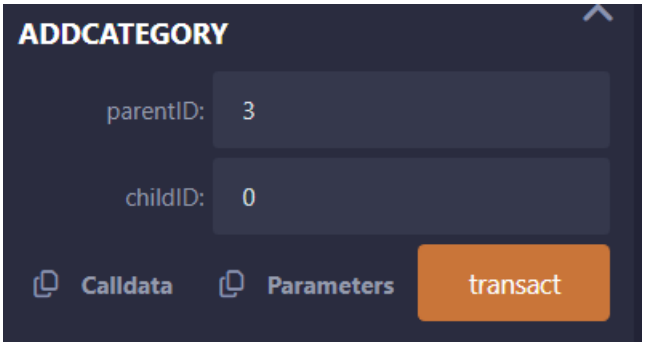

*Figura 6.9. Ejemplo de uso de la función **`addCategory`**.*

La Figura 6.10 muestra, para mejor claridad del test, los IDs de los productos y procesos del cátodo.

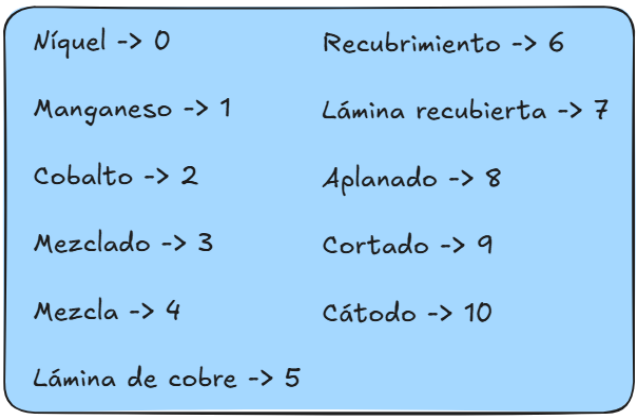

Figura 6.10. Lista de IDs de las categorías del cátodo.

La función **`getCategory`** permite ver todos los datos relacionados con la categoría indicada por el ID proporcionado. La Figura 6.11 muestra un ejemplo de una llamada a esta función con la intención de ver la información referente a la categoría “Mezclado”.

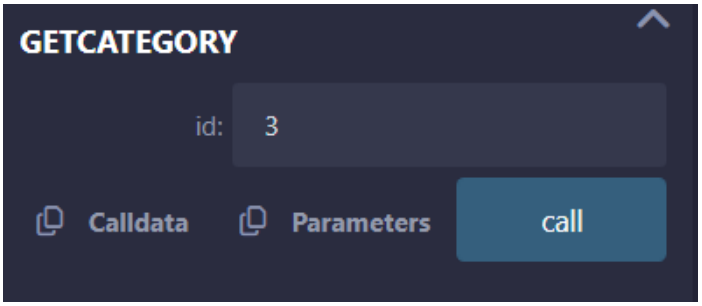

*Figura 6.11. Ejemplo de uso de la función **`getCategory`**.*

La Figura 6.12 muestra el resultado de ejecutar la función anterior. Esta función devuelve el array de padres, el array de hijos, el bitmap y el nombre de la categoría. En este caso el proceso de mezclado solo tiene un padre que es la mezcla y tres hijos, que son el níquel, manganeso y cobalto. El bitmap devuelve un número que cuando se descompone en bits indica los nodos de los que es hijo.

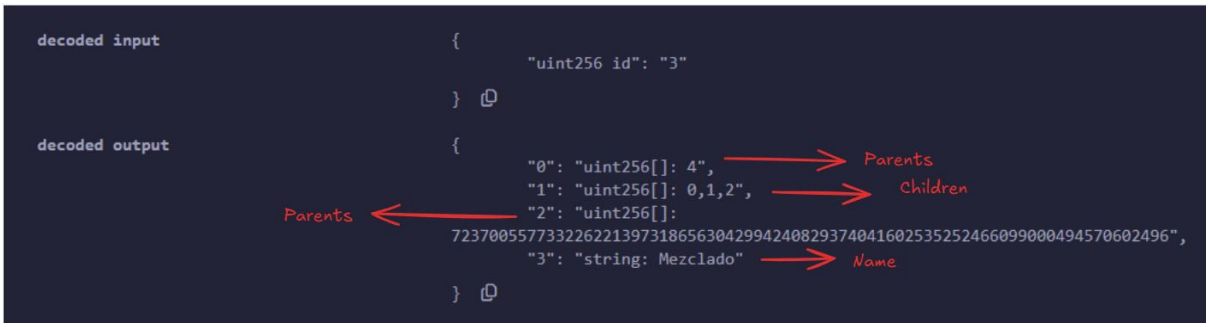

*Figura 6.12. Resultado de la llamada a la función **`getCategory`**.*

El proceso se repite para el resto de componentes hasta generar el grafo completo. La Figura 6.13 indica los IDs del resto de productos por mayor claridad de los test.

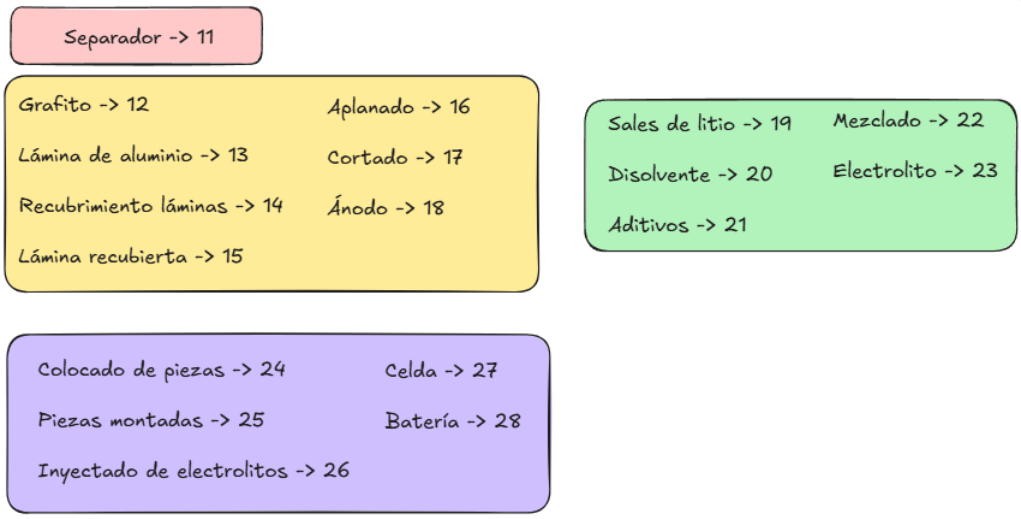

Figura 6.13. Lista de IDs de los nodos del grafo de categorías, excluyendo los del cátodo y sus componentes.

Tras la unión de todos los componentes, finaliza la creación del grafo de componentes, que servirá como guía para la creación de las baterías. El grafo de productos es muy similar al de categorías, con la distinción de que puede haber varios productos de la misma categoría. Todos los productos tendrán IDs distintos entre sí.

Comenzamos con la creación del cátodo, en la Figura 6.14 se muestra un ejemplo de cómo crear un token de producto. En el caso de este se generará un solo token que se puede dividir en dos.

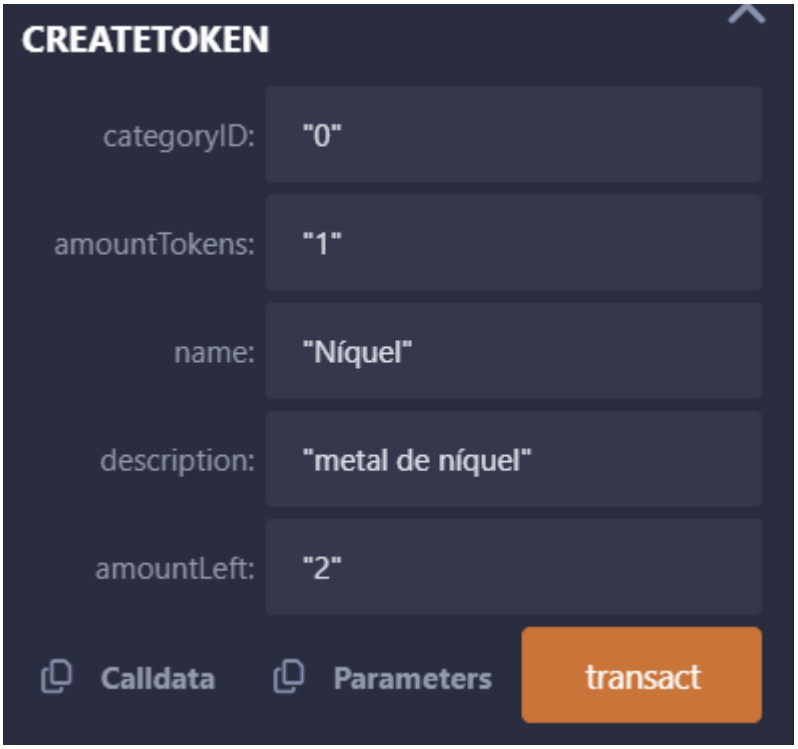

Figura 6.14. Ejemplo de uso de función ***`createToken`***.

En la Figura 6.15 se muestra el resultado de ejecutar la función **`createToken`**. En esta figura se muestran los datos introducidos y devuelve el ID del token generado.

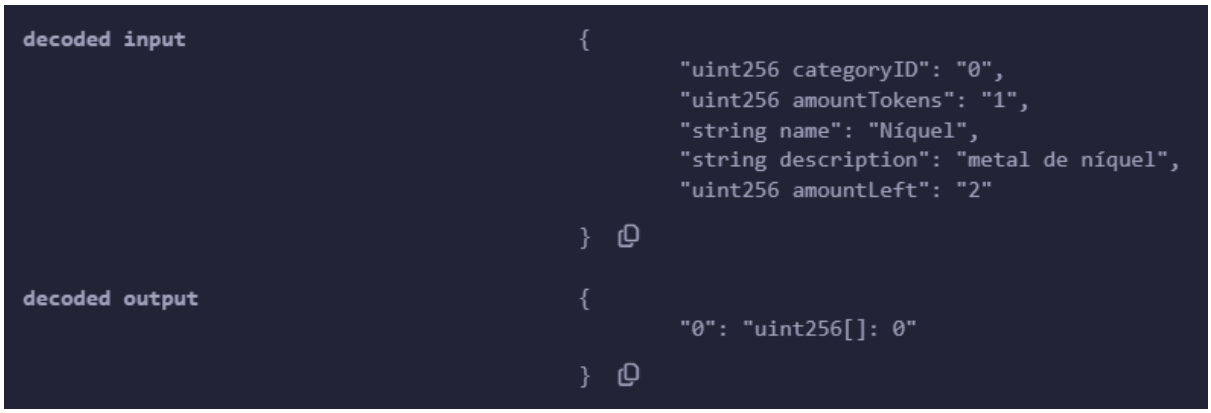

Figura 6.15. Resultado de ejecución de la función ***`createToken`***.

La Figura 6.16 también muestra un ejemplo de cómo generar tokens, en este caso se generan dos tokens con los mismos datos, pero distintos ID.

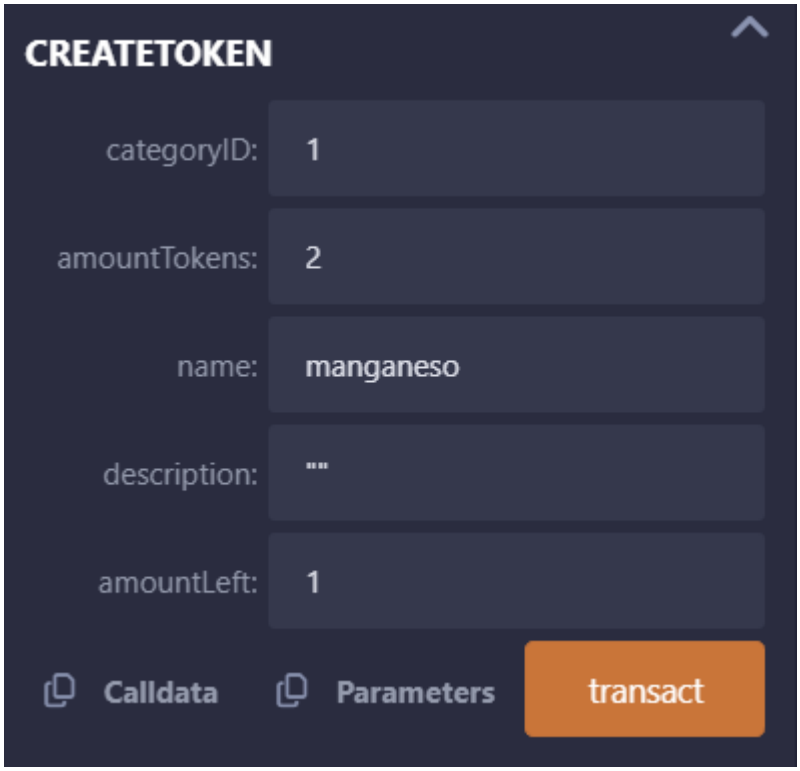

*Figura 6.16. Ejemplo de uso de la función **`createToken`** creando varios tokens en una sola llamada.*

Al igual que en la Figura 6.15, la Figura 6.17 también muestra el resultado de la ejecución del ejemplo anterior mostrando todos los datos introducidos y devolviendo un array con los ID generados.

*Figura 6.17. Resultado de ejecución de la función **`createToken`** devolviendo varios IDs*

Una vez generados los tokens, es necesario relacionarlos para generar el grafo de productos. En la Figura 6.18. se muestra un ejemplo de cómo emplear la función **`joinProduct`** encargada de unir dos productos en el grafo. Para ello se introducen los IDs del hijo y del padre y se indica la cantidad de producto hijo que se quiere unir. En este caso se une el producto “Níquel” con el proceso “Mezclar”.

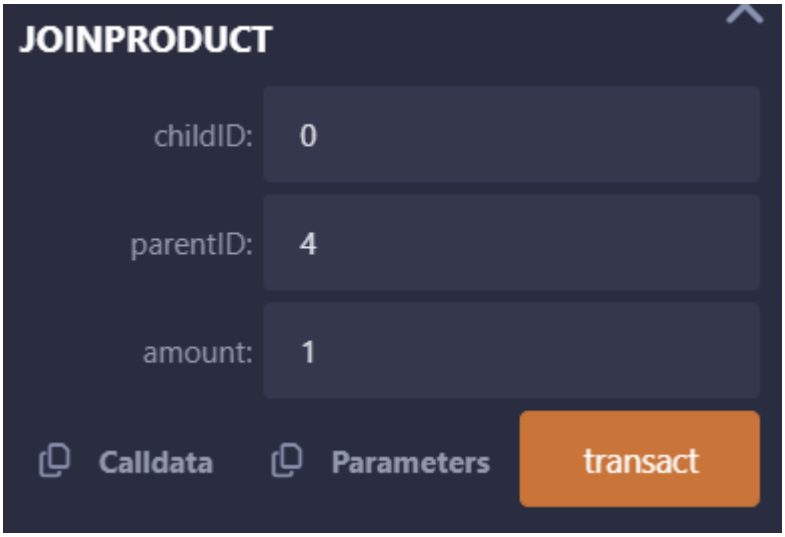

*Figura 6.18. Ejemplo de uso de la función **`joinProduct`**.*

Este proceso se repite con el resto de tokens para generar el grafo de productos. En este grafo cada nodo es un token que representa un producto o proceso en la cadena de producción del producto. Con este grafo se podría realizar la trazabilidad del producto recorriendo los arrays de padres e hijos, reconstruyendo el grafo. La Figura 6.19 muestra cómo quedaría el grafo final.

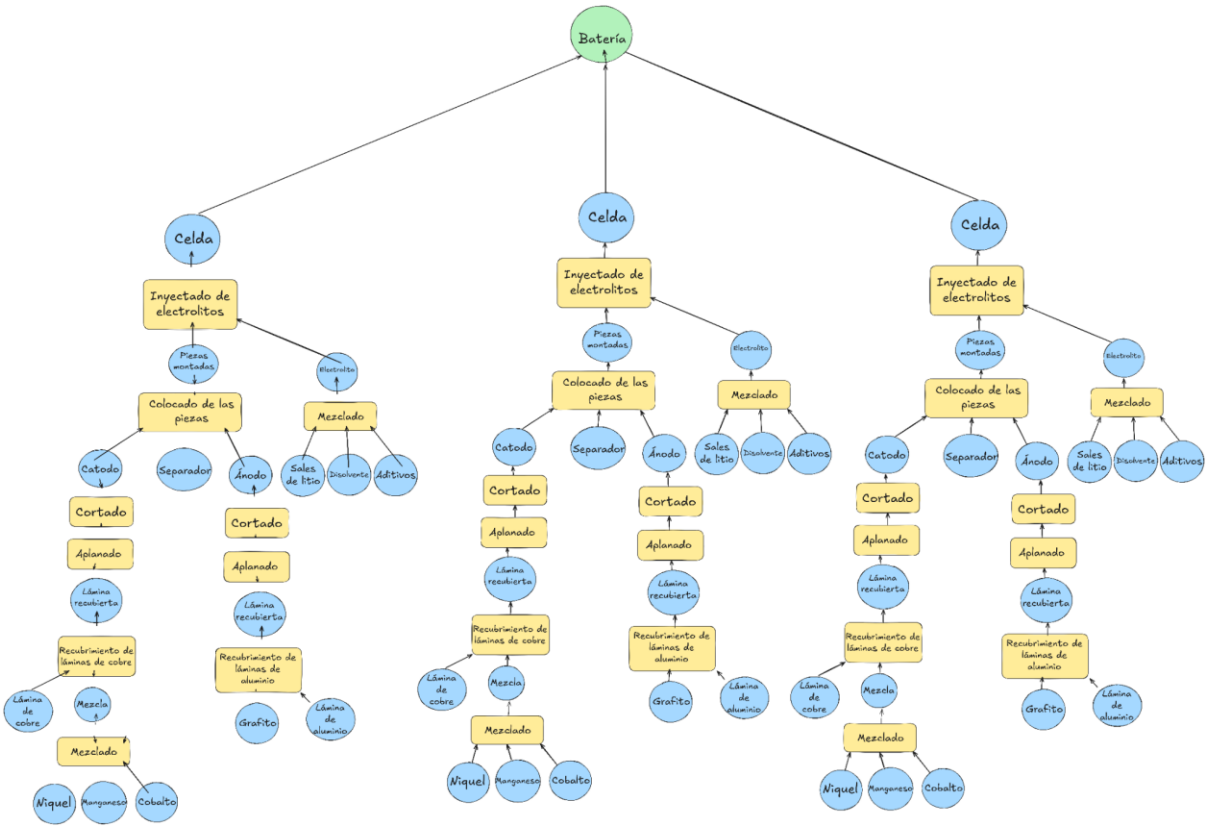

Figura 6.19. Grafo del producto final.

En este prototipo se planteó que la reconstrucción se realizase off-chain, por lo que dentro la blockchain no se podría mostrar el grafo reconstruido. Para realizar la reconstrucción on-chain, habría que realizar llamadas recursivas recorriendo los arrays de padres e hijos hasta llegar a los tokens que dependiendo del sentido del recorrido tengan el array de padres o el de hijos vacíos. El coste de ejecución de estos recorridos impediría emplearlo en grafos de tamaño medio.

Las funciones restantes se emplean una vez finalizado el grafo. Estas funciones añadirán un nodo, es decir un token al grafo de productos.

En la Figura 6.20 se muestra un ejemplo de cómo emplear la función **`replaceProduct`**, en este caso se quiere cambiar una celda defectuosa por otra que funciona.

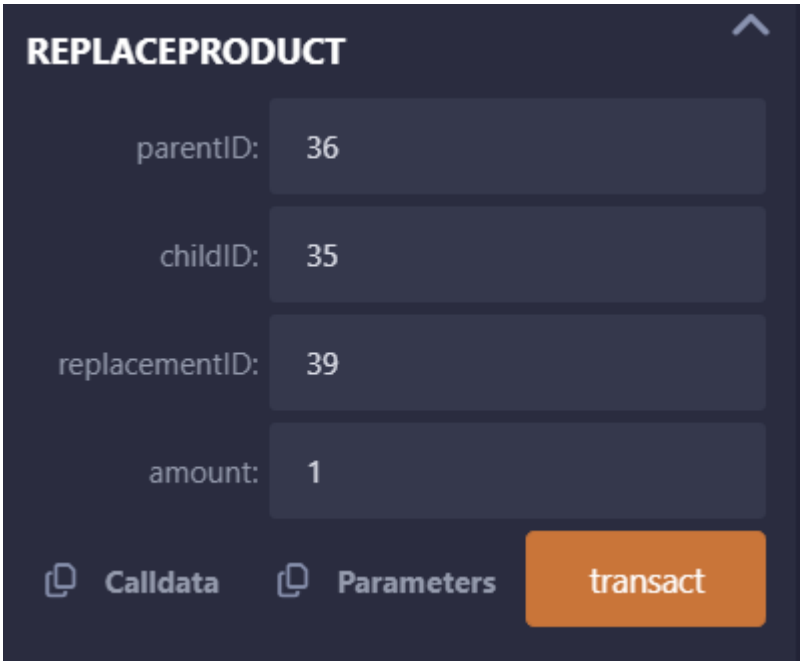

*Figura 6.20. Ejemplo de uso de la función **`replaceProduct`**.*

Al ejecutar la función **`replaceProduct`** para añadir el recambio al grafo, modifica sus datos marcando el booleano **`isReplacement`** a true, como se muestra en la Figura 6.21 siendo el segundo dato devuelto. Además, muestra el ID del producto que ha reemplazado, en este caso otra celda con ID 35.

```
true,true,false,0x5B38Da6a701c568545dCfcB03FcB875f56beddC4,27,35,
```

Figura 6.21. Estado del producto recambio tras su adición al grafo.

La última función es **`addOn`** que se encarga de añadir productos cuando ya se ha finalizado el producto. En la Figura 6.22 se muestra un ejemplo de empleo, donde se añade el ID de una batería como padre y el ID de una celda como añadido indicando la cantidad de producto que se añade.

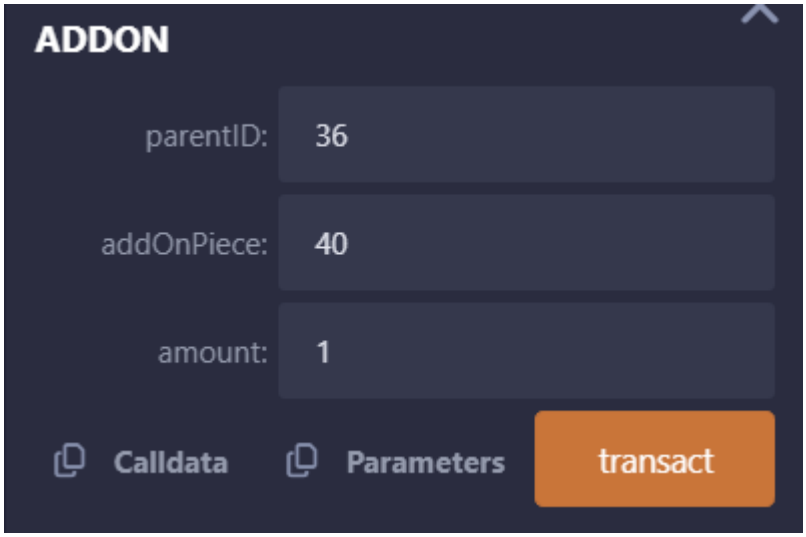

Figura 6.22. Ejemplo de uso de la función ***addOn****.*

En este caso también se modifican los datos del añadido, cambian el booleano **`isAddOn`** a true que en la Figura 6.23 se muestra como el tercer dato mostrado. En este caso no es necesario modificar más datos, aparte de los que siempre se modifican al unir dos productos en el grafo.

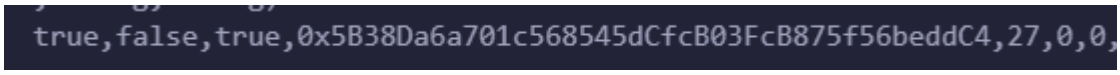

Figura 6.23. Estado del producto añadido tras su adición al grafo.

En la Figura 6.24. se puede observar cómo quedaría el grafo habiendo un recambio y un añadido. En esta figura la celda que aparece en rojo es la estropeada y la que aparece en verde sería el recambio. Además, la celda representada en color naranja representa una celda añadida una vez terminada la batería.

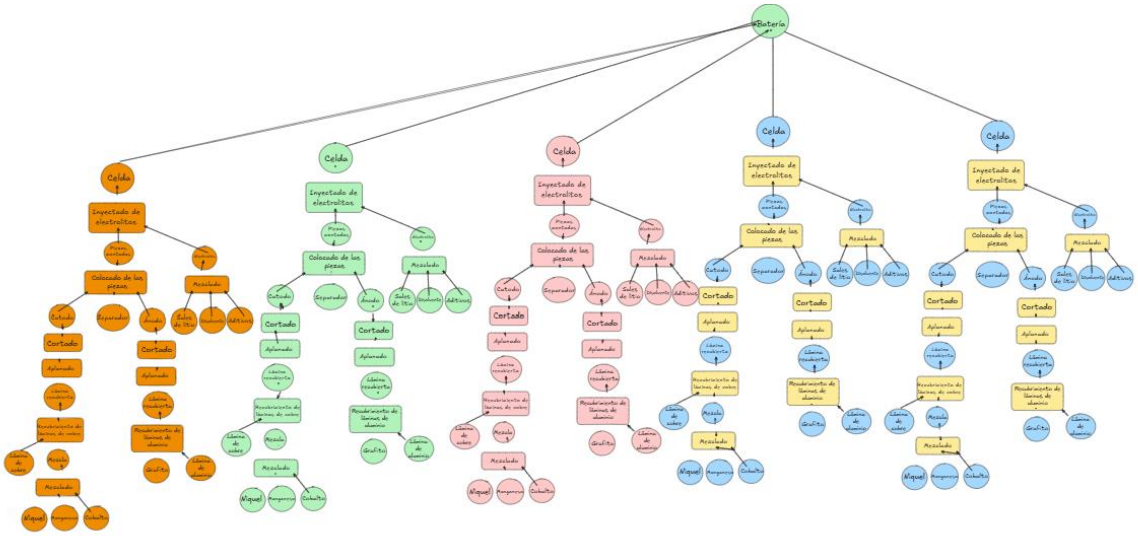

Figura 6.24. Grafo de producto finalizado con recambios y añadidos.

7. SEGURIDAD

Dada la naturaleza inmutable de la tecnología blockchain, es necesario hacer un estudio de la seguridad del contrato antes de que este sea publicado, ya que no podrá ser modificado después.

En primer lugar, en este prototipo al trabajar con contadores para asignar un ID a cada token, se utilizó una versión superior a la 0.8.0, ya que, a partir de esa versión, Solidity incorpora protecciones contra desbordamientos aritméticos. Esto hace que el contrato no sea susceptible a estas vulnerabilidades.

En segundo lugar, los ataques de reentrada ocurren cuando un contrato malicioso realiza llamadas a otro contrato externo, y antes de que la primera llamada haya finalizado, vuelve a llamar al contrato externo, permitiendo a este contrato malicioso modificar el estado del contrato original. En este contrato se evitan reduciendo la cantidad antes de actualizar los mappings de padres e hijos, aunque el objetivo principal de este tipo de ataque suele ser drenar un contrato de ETH.

Otro ataque más relevante a este contrato sería el ataque de suplantación de identidad, que permitiría al atacante usar tokens que no le pertenecen. Este tipo de ataque se produce por los conceptos de “tx.origin” y “msg.sender”, donde “msg.sender” hace referencia a la dirección que está llamando a la función directamente y “tx.origin” indica la dirección que inició la transacción. El uso de “tx.origin” en la lógica de autorización podría engañar al contrato para permitirle a un usuario malicioso realizar acciones privilegiadas para las que no está autorizado. En este caso, este tipo de ataque no es posible porque solo se utiliza “msg.sender”.

Existen ataques como el DoS (Denial of Service) que se producen cuando se introducen una gran cantidad de datos simultáneos, haciendo que los bucles que manejan estos datos consuman el limite máximo de gas por bloque. En este sistema se tiene en cuenta esta vulnerabilidad, evitando recorridos en los grafos e

introduciendo el concepto de categorías para facilitar comprobaciones en el reemplazo de productos, además de emplear bitmaps para crear una manera compacta para detectar ciclos en los grafos.

8. CONCLUSIONES Y TRABAJO FUTURO

El prototipo propuesto se ha centrado en cumplir con las características de un sistema de trazabilidad fiable, proporcionando una manera de realizar un seguimiento de los productos a lo largo de su proceso de producción. El uso de la tecnología blockchain y empleando tokens para identificar cada producto, hace que los datos sean inmutables, pudiendo así proporcionar una fiabilidad y confianza que otros sistemas más tradicionales no pueden asegurar. Los datos de los productos son accesibles casi al instante, vital en casos de retirada de productos.

Su estructura general hace que pueda ser adaptado a las distintas necesidades que pueden tener las industrias, permitiendo que no esté ligado a un único uso.

El proyecto ha dado como resultado un prototipo de token que permite gestionar y mantener un registro de la trazabilidad de los procesos de producción de productos, que es aplicable a múltiples áreas, cumpliendo así el objetivo principal del proyecto. Se comenzó el proyecto realizando un análisis de las características principales que debe tener un sistema de trazabilidad, mediante la búsqueda de artículos que definiesen qué es un sistema de trazabilidad y ejemplos de empresas que lo empleen analizando que ventajas les proporciona. El siguiente paso consistió en buscar ejemplos de sistemas de trazabilidad implementados con tecnología blockchain, con el objetivo de obtener un punto de partida para el desarrollo del prototipo, y buscando mejorar los sistemas existentes, en este caso haciéndolo exportable a distintos ámbitos de producción. Una vez implementado el prototipo se evaluó el sistema con un ejemplo realista basado en las baterías de ion de litio empleadas en los coches eléctricos. Con estas pruebas se evalúa el correcto funcionamiento del sistema en conjunto, haciendo uso de todas sus funciones. Con esto se cumplen todos los objetivos propuestos al inicio del proyecto dejando como resultado un prototipo de sistema de trazabilidad basado en la tecnología blockchain.

El prototipo se centra en la creación del smart contract que actuará como sistema de trazabilidad de los productos, permitiendo que en un futuro se pueda implementar una aplicación que realice las llamadas al contrato y muestre el grafo de productos y categorías asociado a un producto. Esta aplicación facilitaría el uso del sistema por parte de los usuarios implicados. Las llamadas al contrato se realizarán gracias a la librería de Web3, permitiendo una interacción con la blockchain desde la aplicación. El recorrido del grafo se podría realizar en esta aplicación, evitando así un consumo elevado de gas y por consiguiente de ETH, derivado de la ejecución de este recorrido.

Además, se podría plantear la creación de una blockchain privada o público-privada, que proporcione una capa más de seguridad al sistema. De esta manera quedaría un sistema de trazabilidad de productos más seguro y adaptable a diversas industrias.

8. CONCLUSIONS AND FUTURE WORK

The proposed prototype, focused on meeting the characteristics of a reliable traceability system, provides a way to track products throughout the production process. Using blockchain technology and employing tokens to identify each product, makes data immutable, consequently being able to provide reliability and trust that other more traditional systems cannot ensure. Product data is accessible almost instantly, which is vital in cases of recalls.

Its overall structure means that it can be adapted to the different needs that the industries may have, allowing it not to be tied to a single use.

The result of the project is a prototype of a token that allows to manage and maintain a record of the traceability of products during the production process, which can be applied across multiple fields, thus fulfilling the main objective of the project. The project began with an analysis of the main characteristics that a traceability system should have, by searching for articles that define what a traceability system is and examples of companies that use them and analyzing the advantages it provides. In the next step, a research about traceability systems implemented with blockchain technology was carried out, with the aim of obtaining a starting point for the development of the prototype, and seeking to improve existing systems, in this case making it applicable to different production areas. Once the prototype was implemented, the system was evaluated with a realistic example based on lithium-ion batteries used in electric cars. These tests evaluate the correct operation of the system as a whole, making use of all its functions. With this, all the objectives proposed at the beginning of the project are fulfilled, leaving as a result a traceability system prototype based on blockchain technology.

The prototype focuses on the creation of a smart contract that will act as a product traceability system, allowing the future implementation of an application that makes calls to the contract and displays the graph of products and categories

associated with a product. This application would make the use of the system easier for the users involved. The calls to the contract would be made using the Web3 library, allowing interactions between the application and the blockchain. The graph traversal could be performed in this application, thus avoiding a high consumption of gas and consequently of ETH, derived from the execution of this graph traversal.

In addition, the creation of a private or public-private blockchain could be considered, providing an additional layer of security to the system. This would result in a more secure product traceability system, adaptable to various industries.

9. BIBLIOGRAFÍA

[1] «Wikipedia,» 8 Abril 2025. [En línea]. Available: https://es.wikipedia.org/wiki/Trazabilidad.

[2] «FasterCapital,» 2 Mayo 2025. [En línea]. Available: https://fastercapital.com/es/contenido/Trazabilidad-en-productos-farmaceuticos--garantizar-la-calidad-y-seguridad-de-los-medicamentos.html#La-importancia-de-la-trazabilidad-en-los-productos-farmac-uticos.

[3] «FDA,» 12 Junio 2024. [En línea]. Available: https://www.fda.gov/drugs/drug-supply-chain-security-act-dscsa/dscsa-pilot-project-program.

[4] «Trace My Catch,» s.f.. [En línea]. Available: https://www.bumblebee.com/tracemycatch/?_gl=1*19yaqr6*_up*MQ.

[5] S. A. Gebreab, K. Salah, R. Jayaraman y J. Zemerly, «Trusted Traceability and Certification of Refurbished Medical Devices Using Dynamic Composable NFTs,» *IEEE,* 21 Marzo 2023.

[6] «Ledger Insights,» 14 March 2022. [En línea]. Available: https://www.ledgerinsights.com/adidas-adopts-blockchain-solution-trustrace-for-sustainable-materials/.

[7] «Renault Group,» 6 Julio 2021. [En línea]. Available: https://www.renaultgroup.com/en/magazine/energy-and-motorization/xceed-a-new-blockchain-solution-for-renault-plants-in-europe.

[8] «Blockpedia,» 4 Noviembre 2024. [En línea]. Available: https://hathor.network/renault-xceed-blockchain-project/.

[9] «IBM,» 9 Enero 2024. [En línea]. Available: https://www.ibm.com/es-es/case-studies/renault.

[10] «De Beers Group,» 5 Mayo 2022. [En línea]. Available: https://www.debeersgroup.com/media/company-news/2022/de-beers-group-introduces-worlds-first-blockchain-backed-diamond-source-platform-at-scale.

[11] «Aura Blockchain Consortium,» [En línea]. Available: https://auraconsortium.com/. [Último acceso: 26 Octubre 2024].

[12] «Fashion Network,» 21 Abril 2021. [En línea]. Available: https://es.fashionnetwork.com/news/Lvmh-prada-y-cartier-lanzan-aura-su-propio-blockchain,1296354.html#genny.

[13] «PierNext,» 26 Enero 2023. [En línea]. Available: https://piernext.portdebarcelona.cat/tecnologia/el-cierre-de-tradelens/.

[14] H. Dai, H. P. Young, T. J. Durant, G. Gong, M. Kang, H. M. Krumholz, W. L. Schulz y L. Jiang, *TrialChain: A Blockchain-Based Platform to Validate Data Integrity in Large, Biomedical Research Studies,* 2018.

[15] Real Academia Española. (s.f.). Lenguaje.

[16] «The Coca-Cola Company,» 11 Julio 2017. [En línea]. Available: https://www.coca-colacompany.com/policies-and-practices/california-transparency-in-supply-chain-act.

[17] «BASF,» 17 Febrero 2016. [En línea]. Available: https://www.basf.com/global/en/media/news-releases/2016/02/p-16-133.

[18] «PepsiCo,» 20 Septiembre 2024. [En línea]. Available: https://www.pepsico.com/our-impact/esg-topics-a-z/deforestation.

[19] L. Orenes-Lerma, «Ledger Academy,» 26 Agosto 2024. [En línea]. Available: https://www.ledger.com/es/academy/temas/blockchain/what-is-a-blockchain-validator.

[20] «Ethereum,» s.f. d. [En línea]. Available: https://ethereum.org/es/what-is-ethereum/.

[21] «Ethereum,» s.f. e. [En línea]. Available: https://ethereum.org/es/developers/docs/gas/#what-is-gas.

[22] A. Villanueva, «FINEC,» 7 Mayo 2024. [En línea]. Available: https://www.finect.com/usuario/avillanuevae/articulos/que-es-un-token-como-funciona-y-para-que-sirve-tipos-y-ejemplos.

[23] «Ethereum,» 14 Julio 2024. [En línea]. Available: https://ethereum.org/es/eips/.

[24] N. C. Reitwießner, F. Johnson, J. Vogelsteller, K. Baylina, W. Feldmeier y Entriken, «Ethereum,» 23 Enero 2018. [En línea]. Available: https://eips.ethereum.org/EIPS/eip-165.

[25] M. Musharraf, «Ledger Academy,» 29 Junio 2023. [En línea]. Available: https://www.ledger.com/es/academy/blockchain/que-son-los-tokens-erc-y-por-que-los-utilizamos.

[26] «Ethereum,» s.f. b. [En línea]. Available: https://ethereum.org/es/developers/docs/standards/tokens/erc-20/.

[27] «Ethereum,» s.f a. [En línea]. Available: https://ethereum.org/es/developers/docs/standards/tokens/erc-721/.

[28] «Ethereum,» s.f. c. [En línea]. Available: https://ethereum.org/es/developers/docs/standards/tokens/erc-1155/.

[29] «AWS,» [En línea]. Available: https://aws.amazon.com/es/web3/#:~:text=With%20blockchain%2C%20supply%20chain%20companies,at%20any%20point%20in%20time.

[30] Ledger, «Ledger Academy,» 26 Abril 2022. [En línea]. Available: https://www.ledger.com/es/academy/que-es-la-prueba-de-trabajo.