Smart contracts

Blockchain y computación cuántica

CONTENIDO

Objetivos

- Entender las características de los *smart contract* y sus principales usos.
- Conocer la estructura de los *smart contracts* en *ethereum*.
- Aprender el funcionamiento de los *smart contracts* en *ethereum*.

Introducción

Antes de la aparición de *ethereum*, el concepto de *smart contracts* fue descrito en 1995 por Nick Szabo [1,2], un matemático y criptógrafo americano que público un documento llamado *Smart contracts glossary*. En 1997, desarrolló un documento más completo que explicaba los smart contracts. Estas ideas nunca llegaron a ponerse en práctica debido a las carencias de la tecnología de la época.

Los *smart contracts* están presentes en un gran número de soluciones *blockchain* y cada una proporciona una solución tecnológica diferente para implementar la funcionalidad de los smart contracts.

En este tema se usan como referencia los *smart contracts* de *ethereum*, por ser la primera solución en permitir la ejecución de código programado dentro de blockchain y por la relevancia que tiene esta red para proporcionar soluciones implementadas con *smart contracts*.

Los conceptos desarrollados en este tema, son aplicables a otras soluciones *blockchain* que incorporan la funcionalidad de *smart contracts*.

¿Qué es un *smart contract*?

Los contratos son acuerdos escritos donde se definen unos compromisos por las partes implicadas y una serie de penalizaciones si no se cumplen las acciones pactadas. Este documento en papel muchas veces puede ser interpretado de manera diferente por las partes, generando disputas que requieren de terceros para ser resueltas.

Un *smart contract* o contrato inteligente es un **código que se almacena en** *blockchain*, donde se **definen las reglas** que permiten **gestionar los datos almacenados** por el contrato inteligente. Los *smart contracts* se ejecutan en *blockchain* acorde a las reglas establecidas sin la necesidad de un tercero que controle el proceso.

El código de un s*mart contract* tras ser desplegado en *blockchain* **no puede ser modificado**. Esto implica que su comportamiento siempre será acorde al código programado.

Los *smart contracts* permiten guardar información en *blockchain*. Esta información puede ir variando en función de las reglas establecidas en el código del *smart contract*. Se puede decir que cada *smart contract* controla la información que guarda en la cadena de bloques.

Cada *smart contracts* tienen una **dirección o identificador único** asociado.

Para ejecutar un *smart contract* es necesario enviar una **transacción al identificador o dirección** del *smart contract.*

En *ethereum*, los **mineros son los encargados de la ejecución del código** y de actualizar la información en la cadena de bloques.

Estructura

Los *smart contracts* cuentan con una estructura *(figura 1)* definida que se compone de cuatro elementos:

- **Identificador:** es la dirección o nombre que permite determinar cada *smart contract* almacenado en *blockchain*.
- **Variables de estado:** son los datos asociados al *smart contract* que se guardan en *blockchain*.
- **Constructor:** es una función específica que permite inicializar las variables de estado del *smart contract* cuando se despliega en *blockchain*.
- **Funciones:** donde se define la lógica de negocio o reglas que manejan las variables de estado o criptomonedas asociadas al *smart contract*.

Figura 1. Estructura de un *smart contract*.

Características

Cuando se despliega un *smart contract* se le asocia una dirección que sirve para identificar a ese *smart contract*.

El código de los *smart contracts* desplegados ya **no puede modificarse**. El código queda almacenado en la cadena de bloques asociado a la dirección del *smart contracts*.

Los *smart contracts* **requieren de una transacción para ejecutarse**. La ejecución de un *smart contract* siempre está asociado a una transacción.

Un *smart contract* **pueden recibir y enviar criptomonedas** asociadas a su dirección.

Un *smart contract* puede llamar a otro *smart contract*.

Se pueden **establecer restricciones sobre las llamadas a las funciones**, como definir un periodo para la ejecución de una función o limitar una función para que solo puedan ser ejecutada por una o más direcciones.

Los *smart contracts* son predecibles, transparentes e inalterables.

Al funcionar en *blockchain* no tienen periodos de inactividad o caída del servicio, siempre están disponibles para ser ejecutados dentro de la red *blockchain*.

Funcionamiento

Para poder ejecutar un *smart contract* este ha de estar desplegado en la red *blockchain* y es necesario conocer la dirección del *smart contract* y las funciones que este tiene disponibles para ser ejecutadas.

Es importante publicar las funciones de un *smart contract* si se quiere que los usuarios de la red u otros *smart contracts* puedan interactuar con dichas funciones.

Despliegue de un *smart contract*

El despliegue de un *smart contract* se realiza creando una transacción donde el **campo datos** contiene el **código del** *smart contract* y la **dirección destino no se rellena** [3].

Esta acción permite desplegar el *smart contract* inicializando las variables de entorno con el contenido definido en el constructor del contrato inteligente.

Al desplegar un *smart contracts* se obtiene la dirección del *smart contract*, que es necesaria para poder realizar operaciones sobre el *smart contract* desplegado.

Ejecución de un *smart contract*

Para poder ejecutar un *smart contract* es necesario enviar una transacción a la dirección del *smart contract* indicando [3]:

- La **dirección** del *smart contract*.
- La **función** que se quiere ejecutar.
- Los **datos** necesarios para ejecutar la función.
- La cantidad de **criptomoneda** que se quiere enviar (opcional).

También es necesario añadir la cantidad de criptomoneda necesaria para ejecutar la función indicada.

La transacción *(figura 2)* se firma y se envía a uno o varios nodos como el resto de las transacciones.

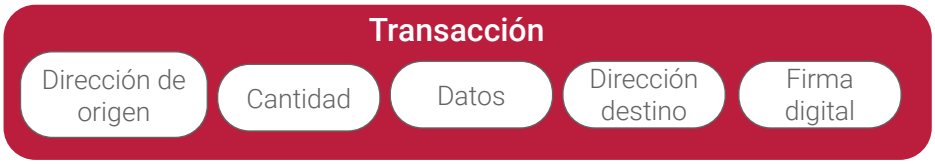

Figura 2. Esquema campos transacción.

Cuando un minero procesa una transacción de un *smart contract*, este ha de obtener el código del *smart contract*, esto lo hace buscando en la cadena de bloques el código desplegado utilizando el identificador del contrato inteligente.

Al recuperar el código también obtiene el valor de las variables de estado. Esta información se carga en la máquina virtual de *ethereum* (EVM) encargada de ejecutar el código con los datos cargados de la transacción y la información de las variables de entorno.

Cuando termina la ejecución, el minero establece los cambios en la cadena y continúa ejecutando otra transacción, hasta que puede crear un bloque y minarlo.

Los cambios no son efectivos hasta que el bloque es validado e incluido en la cadena de bloques.

En *ethereum* los smart contracts pueden generar unos eventos que permiten comunicar a otros dispositivos o aplicaciones fuera de *blockchain* que se ha completado la ejecución de cierta transacción.

Funcionamiento de la máquina virtual de *ethereum* (EVM)

La EVM [4] proporciona un entorno aislado que permite la ejecución de las instrucciones máquina del *smart contract*, utilizando una pila o memoria de información [3].

La máquina virtual de *ethereum* es el motor que ejecuta los smart contracts. El funcionamiento de la EVM se recoge en un documento denominado '*yelollow paper*' [5].

Cada instrucción máquina tiene un coste de ejecución que se mide en gas y que tiene una equivalencia con el *ether*. El emisor de una transacción ha de enviar la cantidad de ETH necesario para costear la ejecución de la función indicada en la transacción.

El gas de *blockchain* funciona como el combustible para el motor de un coche, el motor es la máquina virtual de *ethereum* y funciona mientras haya combustible para consumir.

El emisor puede enviar una cantidad de gas superior a la necesaria para la ejecución, ya que el gas no consumido por la EVM se devuelve al emisor.

¿Qué ocurre si en la ejecución de un *smart contract* se termina la cantidad de gas?

Esta situación puede ocurrir en los siguientes casos:

- Si el código está mal programado y genera un bucle o si se realiza una mala gestión de las llamadas otros *smart contracts*.
- Si el emisor no proporciona una cantidad suficiente para completar la ejecución de la función.

Cuando la máquina virtual de *ethereum* ejecuta una función y se agota el gas disponible para la ejecución de la transacción, si la máquina virtual de *ethereum* no ha terminado de procesar la función determinada, se revierten los cambios y se termina la transacción. En este caso la transacción no se ejecuta, pero el minero se queda con la cantidad de ETH correspondientes al gas enviado.

Este mecanismo permite evitar ataques sobre los mineros, eliminando la opción de crear *smart contracts* que puedan incluir bucles infinitos para bloquear a los mineros.

Programar un *smart contract*

Los *smart contracts* [4] son programas que se escriben en un lenguaje de programación de alto nivel.

El lenguaje de programación de los *smart contracts* depende de la solución *blockchain*. *Ethereum* utiliza un lenguaje propio llamado *solidity*, pero las nuevas soluciones *blockchain* que cuentan con *smart contract*, utilizan uno o varios lenguajes de programación estándar como *java, go, javascript, python*.

En *blockchain* el código de un *smart contract* desplegado no puede ser modificado, por lo que si existe algún fallo o error en un *smart contract*, el error no puede arreglarse. Ante estas situaciones la solución alternativa pasa por desplegar un nuevo *smart contract* con el error corregido. Las consecuencias de tener que desplegar un nuevo *smart contract* son:

- El nuevo *smart contract* tendrá una nueva dirección, esto suele impactar a las aplicaciones o *smart contracts* que operaban con el contrato inteligente, requiriendo actualizar estos para que puedan operar con el nuevo *smart contract*.
- El nuevo *smart contract* no dispone de la información del antiguo. Esto puede ser un problema para algunas operativas, por lo que es necesario que en el constructor se carguen los datos provenientes del antiguo *smart contract*.

El código de los *smart contracts* suelen publicarse en plataformas como *Github* para que las personas que quieran interactuar con el *smart contract* conozcan la lógica de este. Este acto de transparencia permite que la comunidad pueda revisar el código e informar si existe algún error o fallo en el mismo.

Normalmente los *smart contracts* pasan una auditoria de código que permite evaluar posibles vulnerabilidades o errores en el código, antes de ser desplegados en *blockchain*. Los errores de codificación son la causa principal del mal funcionamiento de un *smart contract*.

Coste de un *smart contract*

Unidades de *ethereum*

Un ether se divide en las siguientes unidades

- *1 finney* = 10−3 ETH
- *1 szabo* = 10−6 ETH
- *1 shannon* =10−9 ETH
- *1 babbage* = 10−12 ETH
- *1 lovelace* = 10−15 ETH
- *1 wei* = 10−18 ETH

¿Cuánto cuesta desplegar y ejecutar un *smart contract*?

Smart contract de tamaño medio requiere 4 000 000 unidades de gas para su despliegue.

Suponga que el *gas price* que paga es siempre el mismo 1,4 ETH por *Gwei*, en este caso el coste [6] de desplegar el *smart contracts* en las siguientes fechas sería el siguiente:

- Diciembre 2017 -> $ 2,47
- Enero 2018 -> $ 5,53
- Diciembre 2018 -> $ 0,504

Al existir una vinculación entre el gas y el *ether*, el coste de desplegar un *smart contract* varía en función de la cotización de *ether*.

Almacenar datos en *blockchain* es la operación que más gas requiere, por lo que hay que elegir muy bien los datos que ha de guardar un *smart contract*.

Uso de los *smart contracts*

Lo *smart contracts* pueden ser utilizados para definir cualquier regla o lógica de negocio, pero existen algunos usos que son ampliamente utilizados como:

Juegos

Se puede crear un smart contract con las reglas del juego de piedra, papel o tijera y desplegarlo en *ethereum* para que solo puedan jugar dos direcciones predefinidas. Los dos jugadores tendrían que enviar las transacciones indicando la opción (piedra, papel o tijera) seleccionada para el juego. El *smart contract* al procesar ambas transacciones resuelve el ganador y lo apunta, llevando la cuenta de las veces que ha ganado cada uno de los jugadores.

Apuestas

Este es uno de los claros usos de un *smart contract*, donde se implementa la función de realizar apuestas por un resultado de un partido de futbol. Para que el *smart contract* pueda determinar el ganador necesita conocer el resultado del partido, que puede ser proporcionado por un servicio de otro *smart contract* de *ethereum*. Cuando se resuelve la apuesta el *smart contract* transfiere el premio al ganador.

Seguros

Es sencillo crear un *smart contract* donde un número de participantes depositan una cantidad de criptomonedas con el fin de cubrir a estos participantes ante ciertos siniestros. Como ejemplo, se puede plantear el seguro de un dispositivo móvil, de manera que, si se rompe la pantalla, la reparación de esta queda cubierta por el *smart contract*, que puede transferir la cantidad definida a otro *smart contract* correspondiente a una empresa que ofrece servicios de reparación de dispositivos móviles.

Control de dispositivos con *smart contracts*

Se puede programar un *smart contracts* que controle una puerta, un aparato de aire acondicionado o un coche. Un *smart contract* puede ser un gemelo digital de distintos aparatos. Estos aparatos pueden consultar el estado del *smart contract*, de manera que una puerta puede abrirse o cerrarse en función del estado almacenado en el *smart contract* de la puerta.

También se puede incorporar a los aparatos su conjunto de claves, para que puedan enviar información a su *smart contract* y, de esta manera, actualizar o incorporar información sobre el estado del aparato.

En general esta solución se puede utilizar con cualquier aparato que pueda conectarse a internet y que permita procesar la información obtenida del *smart contract*.

Blockchain proporciona un entorno seguro para el control de dispositivos, que puede incluso extenderse a un ecosistema donde los dispositivos se comuniquen mediante *smart contracts*, creando un entorno donde todas las transacciones van firmadas por las correspondientes claves de los aparatos.

Ejemplo

En el siguiente ejemplo de un *smart contract*, se declaran dos variables de entorno 'x' e 'y'; y se define una función que recibe un número como parámetro y suma el valor de 'x' al parámetro, para finalmente guardar el resultado en 'y'.

```
Pragma solidity ^0.7.2; // Indica la versión del compilador de la EVM

contract Ejemplo{

 uint public x;

 uint public y;

// Constructor que permite inicializar el valor de x

 constructor(uint _x) public{

x=_x;

}

// Función suma que realiza la suma del parámetro con x y lo guarda en y

function suma(uint _num) public returns (uint){

y=x+_num;

return y;

}

}
```

Resumen

La incorporación de programas ejecutables dentro de *blockchain* ha permitido que la tecnología pueda ofrecer una gran variedad de servicios más allá del entorno financiero.

Las características de los *smart contracts* hacen de esta funcionalidad un elemento clave para establecer acuerdos entre distintos participantes, confiando la ejecución de los acuerdos al código instalado en *blockchain*, sin la necesidad de incorporar mediadores o terceros que no están implicados en la operativa y que añaden costes a las operaciones.

Es importante entender el funcionamiento de los *smart contracts* y sus limitaciones antes de poder implementar una solución basada en esta tecnología. El código que rige las reglas ejecutadas por el *smart contract* debe ser programado por personas con experiencia y se recomienda auditar dicho código para detectar posibles vulnerabilidades antes de su uso en *blockchain*.

Los *smart contracts* necesitan de una transacción para poder ejecutarse y esta transacción ha de pagar el coste de la ejecución. En *ethereum* el coste está bien definido, ya que cada instrucción máquina se mide en gas, siendo esta la unidad que permite calcular el cote total de la ejecución de una transacción.

El uso de los *smart contracts* se ha extendido y actualmente empresas de juegos utilizan la tecnología blockchain para ofrecer nuevos servicios.

Otra de las tecnologías que se va a beneficiar de la funcionalidad proporcionada por los *smart contracts* es el llamado internet de las cosas, mediante la implementación de smart contracts que permiten recolectar información de los distintos dispositivos.

Bibliografía

[1] Bit2Me Academy. (2020) *El glosario de los smart contracts de nick Szabo*. [En línea]. Disponible en: https://academy.bit2me.com/glosario-smart-contract-nick-szabo/

[2] N. Szabo. (1995) *Glosario de los smart contracts. Satoshi Nakamoto Institute*. [En línea]. Disponible en: https://nakamotoinstitute.org/smart-contracts-glossary/

[3] Ryancreatescopy (2020) *Documentación sobre el desarrollo del ethereum*. [En línea]. Disponible en: https://ethereum.org/en/developers/docs/

[4] Bit2Me Academy. (2020). *¿Qué es la ethereum virtual machine (EVM)?* [En línea]. Disponible en: https://academy.bit2me.com/que-es-ethereum-virtual-machine-evm/

[5] G. Wood. (2014, Abril 10) *Ethereum: a secure decentralised generalised transaction ledger*. [En línea]. Disponible en: https://web.archive.org/web/20140410013339/http:/gavwood.com/paper.pdf

[6] Ethgasstation.info [En línea]. Disponible en: https://www.ethgasstation.info/