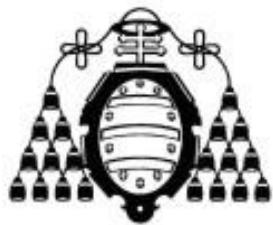


UNIVERSIDAD DE OVIEDO



ESCUELA DE INGENIERÍA INFORMÁTICA

TRABAJO FIN DE GRADO

“Análisis de tecnologías blockchain basadas en Hyperledger y Ethereum para su aplicación en entornos industriales”



Director: Vicente García Díaz

Autor: Pedro Fernández Álvarez

**Vº Bº del Director del
Proyecto**

Agradecimientos

Quiero agradecer a Vicente García Díaz su ayuda y apoyo desde el primer momento.

Resumen

Durante los últimos años se ha producido un incremento en la popularidad de las monedas digitales, cuyos fundamentos residen en una tecnología llamada blockchain, la cual surgió hace un poco más de una década y posee, entre otras cosas, características como transparencia, seguridad e inmutabilidad.

Hoy en día, cuando se habla de blockchain, normalmente se está haciendo referencia al sector económico, sin embargo, ante las propiedades tan interesantes que posee esta tecnología, su integración en otros entornos diferentes a los monetarios podría traer consigo numerosas ventajas. Por todo ello, el objetivo de este trabajo fin de grado es la investigación de esta tecnología de cara a ser aplicada a un entorno no financiero, en este caso, a los entornos relacionados con la industria.

Para comenzar, se explican los conceptos teóricos de blockchain. Tras la comprensión de los fundamentos, se procede a realizar el estado del arte de esta tecnología, haciendo también mención a su historia y tratando las diferentes líneas de investigación existentes. Una vez que se conocen las principales plataformas blockchain que permiten la realización de aplicaciones para el sector de la industria, se realiza un estudio comparativo, con la finalidad de analizarlas y compararlas entre sí. Posteriormente, en el momento en el que se sabe cuales son las plataformas más importantes y se conoce su funcionamiento, se hace uso de ellas de cara a construir un prototipo acerca de un caso de uso industrial. Concretamente, para su elaboración se empleará tanto Hyperledger Fabric como Ethereum, las cuales, tras la realización del estado del arte, se concluye que son, a día de hoy, las dos plataformas blockchain más importantes y asentadas que permiten realizar aplicaciones de propósito general. Finalmente, una vez que se conoce el funcionamiento de cada plataforma y se ha experimentado realmente con ellas mediante el desarrollo de un prototipo, se realiza una evaluación global de la manera en la que blockchain se puede integrar, hoy en día y con los avances más recientes, en el sector de la industria.

Palabras Clave

Industria, Blockchain, Transacción, Bloque, Consenso, Contrato Inteligente.

Abstract

In the last years, there has been an increase in the popularity of digital coins, whose fundaments lie in a technology called blockchain. This technology emerged a little bit more than a decade ago and has, among others, features such as transparency, security and immutability.

Nowadays, blockchain is normally associated with the economic sector. However, due to the interesting characteristics that this technology possesses, its integration in other fields apart from the financial one could bring numerous advantages. The aim of this project is the research of the blockchain technology in order to use it in the industrial sector, and not in the financial one.

To begin with, the theoretical aspects of blockchain are explained. After their correct understanding, the state of the art of this technology is done, mentioning also its history and all the existent research areas. Once the reader is familiar with the main blockchain platforms which are useful for the industrial sector, a comparative study is conducted in order to analyse and compare them with each other. Afterwards, and after knowing the most important platforms and how they work, they are used in order to create a prototype for a concrete industrial use case. In this elaboration process, the Hyperledger Fabric and Ethereum platforms will be employed, which are concluded to be, after the state of the art has been done, the most important ones and the ones that can be used to build general-purpose applications. Finally, after knowing how each platform works and having used them in the development of a prototype, a global evaluation is done to analyse the way in which the blockchain technology, with the most recent advances, can be applied to the industrial sector.

Keywords

Industry, Blockchain, Transaction, Block, Consensus, Smart Contract.

Índice General

Capítulo 1: Introducción.....	21
1.1 Motivación	21
1.2 Objetivos	21
1.3 Alcance	22
1.4 Estructura	22
1.5 Acerca del Título de este Trabajo	23
Capítulo 2: Aspectos Teóricos	25
2.1 Criptografía Asimétrica	25
2.2 Función Hash	27
2.3 Certificado Digital	27
2.4 Árbol de Merkle.....	29
2.5 Red P2P	30
2.6 Tecnología Blockchain	31
2.6.1 Transacciones	31
2.6.2 Bloques	32
2.6.3 Consenso.....	33
2.6.4 Contratos Inteligentes	33
2.7 Tipos de Redes Blockchain.....	34
2.7.1 Redes Públicas	34
2.7.2 Redes Privadas.....	34
2.7.3 Redes de Consorcios.....	34
Capítulo 3: Estado del Arte	35
3.1 Historia.....	35
3.1.1 Bitcoin	35
3.1.2 Criptomonedas Alternativas	37
3.1.3 Ethereum.....	38
3.1.4 Hyperledger	39
3.1.5 Corda	41
3.1.6 Quorum.....	41
3.1.7 Alastria.....	42
3.2 Investigación	42
Capítulo 4: Estudio Comparativo	47
4.1 Ethereum	47
4.1.1 Identidades	47
4.1.2 Estructura.....	48
4.1.3 Ledger	49
4.1.4 Ejecución de Transacciones.....	55

4.1.5	Consenso.....	59
4.2	Hyperledger Fabric	61
4.2.1	Identidades.....	61
4.2.2	Estructura.....	61
4.2.3	Ledger.....	66
4.2.4	Ejecución de Transacciones.....	67
4.2.5	Consenso.....	70
4.3	Resumen.....	71
Capítulo 5: Propuesta.....		73
5.1	Descripción del Caso de Uso	73
5.2	Características del Prototipo	74
5.2.1	Funcionalidad	75
5.2.2	Plataformas Empleadas.....	76
Capítulo 6: Análisis		77
6.1	Definición del Sistema	77
6.1.1	Determinación del Alcance del Sistema.....	77
6.2	Requisitos del Sistema	78
6.2.1	Requisitos Funcionales	79
6.2.2	Requisitos No Funcionales	81
6.2.3	Identificación de Actores del Sistema y Casos de Uso	81
6.3	Identificación de Subsistemas.....	84
6.4	Modelado Preliminar	84
6.4.1	Estado Actual.....	85
6.4.2	Contratos Inteligentes	87
6.5	Análisis de Casos de Uso y Escenarios.....	88
6.6	Análisis de la Interfaz de Usuario	94
Capítulo 7: Diseño del Sistema.....		97
7.1	Arquitectura del Sistema.....	97
7.1.1	Red de Consorcios	97
7.1.2	Red Pública.....	100
7.1.3	Visión Global.....	102
7.2	Gestión de Identidades	102
7.2.1	Hyperledger Fabric	102
7.2.2	Ethereum.....	104
7.3	Diseño del Estado Actual.....	105
7.4	Diseño de los Contratos Inteligentes.....	106
7.5	Diagramas de Estado.....	107
7.6	Diseño de la Interfaz de Usuario	108
Capítulo 8: Implementación del Sistema.....		111
8.1	Lenguajes de Programación.....	111
8.2	Software Utilizado	111

8.3	Creación del Sistema.....	114
8.3.1	Problemas Encontrados	114
8.3.2	Estructura.....	115
Capítulo 9: Desarrollo de Pruebas	119	
9.1	Composer Playground.....	119
9.2	Ganache.....	122
9.3	Ropsten y Etherscan.....	124
9.4	Navegadores Web	125
Capítulo 10: Manuales del Sistema.....	127	
10.1	Manual de Instalación	127
10.2	Manual de Ejecución.....	129
10.3	Manual de Usuario	130
Capítulo 11: Evaluación.....	135	
11.1	Almacenamiento de Datos	135
11.2	Permisos en una Red Pública	136
11.3	Mantenimiento	137
11.4	Rendimiento.....	139
11.5	Riesgos	141
11.6	Futuro	141
Capítulo 12: Planificación y Presupuesto	143	
12.1	Inicial	143
12.1.1	Planificación	143
12.1.2	Presupuesto.....	146
12.2	Real	148
12.2.1	Desarrollo Real del Proyecto	148
12.2.2	Presupuesto Resultante tras Finalizar el Proyecto	152
Capítulo 13: Conclusiones y Futuro Trabajo.....	155	
13.1	Conclusiones	155
13.2	Futuro Trabajo	157
Capítulo 14: Referencias Bibliográficas	159	
Capítulo 15: Apéndices	163	
15.1	Contenido Entregado	163

Índice de Figuras

Figura 2.1: Función de cifrado (C)	26
Figura 2.2: Función de descifrado (D).....	26
Figura 2.3: Función de firmado (F)	26
Figura 2.4: Función de verificación (V)	27
Figura 2.5: Función hash (H).....	27
Figura 2.6: Estructura de un certificado digital	28
Figura 2.7: Árbol de Merkle	29
Figura 2.8: Diferencia entre cliente-servidor y peer-to-peer	30
Figura 2.9: Función de transición de estados (T)	31
Figura 2.10: Ejemplo de transacción	31
Figura 2.11: Contenidos de un bloque	32
Figura 2.12: Cadena de bloques	33
Figura 3.1: Variación del valor de 1 bitcoin	37
Figura 4.1: Cuenta personal.....	51
Figura 4.2: Cuenta de contrato	51
Figura 4.3: Estado actual de una red Ethereum	52
Figura 4.4: Bloque en una red Ethereum	53
Figura 4.5: Árboles de Merkle del estado actual y de transacciones.....	53
Figura 4.6: Red Fabric	62
Figura 4.7: Nodo contribuyendo a dos canales.....	63
Figura 4.8: MSP de nodo	65
Figura 4.9: MSP de canal	66
Figura 4.10: Propuesta de transacción	68
Figura 4.11: Respuesta de un nodo ante una propuesta.....	68
Figura 4.12: Estructura de una transacción en Fabric	69
Figura 5.1: Cadena de producción y suministro	74
Figura 6.1: Casos de uso relativos al administrador	83
Figura 6.2: Casos de uso relativos a una identidad válida	83
Figura 6.3: Casos de uso relativos a cualquier individuo	83
Figura 6.4: Diagrama inicial de la interfaz de usuario	95
Figura 7.1: Arquitectura de la red de consorcios	98
Figura 7.2: Comunicación con la red Fabric	99
Figura 7.3: Host ejecutando 2 máquinas virtuales.....	100
Figura 7.4: Host ejecutando 2 contenedores.....	100
Figura 7.5: Comunicación con la red pública.....	101
Figura 7.6: Estructura global del prototipo.....	102
Figura 7.7: Modelo de datos de la red de consorcios	106
Figura 7.8: Modelo de datos de la red pública	106
Figura 7.9: Diseño del contrato inteligente relativo a Fabric	107
Figura 7.10: Diseño del contrato inteligente relativo a Ethereum	107

Figura 7.11: Diagrama de estados de un lote.....	107
Figura 7.12: Diagrama de estados de un documento	107
Figura 7.13: Diseño de la interfaz para interactuar con Fabric	109
Figura 7.14: Diseño de la interfaz para interactuar con Ethereum	110
Figura 9.1: Interfaz ofrecida por Composer Playground (1)	120
Figura 9.2: Interfaz ofrecida por Composer Playground (2)	120
Figura 9.3: Interfaz ofrecida por Ganache	123
Figura 9.4: Interfaz ofrecida por Etherscan (1)	125
Figura 9.5: Interfaz ofrecida por Etherscan (2)	125
Figura 10.1: Interfaz para interactuar con la red Fabric	131
Figura 10.2: Selección de participante en la red Fabric	132
Figura 10.3: Interfaz de enlace a la gestión documental	132
Figura 10.4: Interfaz para interactuar con la red Ethereum	133
Figura 10.5: Selección de cuenta mediante MetaMask	133
Figura 10.6: Mensajes relativos a la generación de tarjetas de participantes	134
Figura 10.7: Mensajes relativos a transacciones y eventos en la red Fabric	134
Figura 11.1: Código fuente publicado en Ropsten	139
Figura 12.1: Diagrama de Gantt y desglose de tareas inicial	145
Figura 12.2: Diagrama de Gantt sobre el desarrollo real.....	150
Figura 12.3: Tareas realizadas durante el aprendizaje de fundamentos	151
Figura 12.4: Tareas realizadas durante el análisis del estado del arte	151
Figura 12.5: Tareas realizadas durante el estudio comparativo.....	151
Figura 12.6: Tareas realizadas durante la elaboración del prototipo	151
Figura 12.7: Tareas realizadas durante la implementación del prototipo.....	152
Figura 12.8: Tareas realizadas durante la evaluación.....	152

Índice de Tablas

Tabla 4.1: Comparativa entre Ethereum y Hyperledger Fabric.....	71
Tabla 6.1: Requisitos funcionales.....	81
Tabla 6.2: Requisitos no funcionales.....	81
Tabla 6.3: Datos relativos a cada lote	86
Tabla 6.4: Datos relativos a cada prueba de existencia	87
Tabla 6.5: Análisis del contrato a desplegar en la red Fabric	87
Tabla 6.6: Análisis del contrato a desplegar en la red Ethereum	88
Tabla 6.7: Caso de uso de configuración del sistema	89
Tabla 6.8: Caso de uso de registro de documentos.....	90
Tabla 6.9: Caso de uso de creación de un lote.....	91
Tabla 6.10: Caso de uso de envío de un lote	91
Tabla 6.11: Caso de uso de entrega de un lote	92
Tabla 6.12: Caso de uso de utilización de un lote	93
Tabla 6.13: Caso de uso de lectura de datos en Fabric	93
Tabla 6.14: Caso de uso de comprobación de existencia de documentos	94
Tabla 9.1: Pruebas realizadas con Composer Playground.....	122
Tabla 9.2: Pruebas realizadas con Ganache.....	124
Tabla 11.1: Tiempos de confirmación en Ropsten	140
Tabla 12.1: Costes del personal del proyecto	146
Tabla 12.2: Costes directos.....	146
Tabla 12.3: Costes indirectos.....	147
Tabla 12.4: Presupuesto interno	148
Tabla 12.5: Presupuesto de cliente	148
Tabla 12.6: Presupuesto interno resultante tras finalizar el proyecto	152
Tabla 12.7: Presupuesto de cliente resultante tras finalizar el proyecto.....	153

Capítulo 1: Introducción

En este primer capítulo, a modo de introducción, se redactan la motivación, objetivos, alcance y estructura del presente trabajo fin de grado.

1.1 Motivación

Desde muy pequeño he sido aficionado a escuchar la radio. Programas informativos, científicos, deportivos, tertulias radiofónicas, y, de hecho, lo que estuviesen retransmitiendo en el momento de encenderla. Centrándome en estos últimos años, he comenzado a escuchar diversas menciones al concepto de monedas digitales, también llamadas criptomonedas, tratándose esta temática unas veces con más y otras veces con menos nivel de detalle. Normalmente, al hablar de este tipo de monedas, escuchaba la palabra blockchain, y se solía mencionar que era una tecnología reciente, innovadora y con potencial. Una peculiaridad importante era que cuando se hablaba de estos términos, casi siempre se trataban en programas, secciones o tertulias relacionadas con el sector económico.

Algo que llamaba cada vez más mi atención era que normalmente siempre aparecían los mismos conceptos de transparencia, privacidad, no gobernabilidad, inmutabilidad y seguridad asociados a blockchain. Yo escuchaba estos términos y me ilusionaba con las propiedades tan positivas que se comentaban acerca de esta tecnología, pero realmente no sabía el apartado técnico que había detrás de ella y desconocía la manera en la que se conseguían esas propiedades. Otro aspecto que me intrigaba era que, como he dicho anteriormente, estos conceptos casi siempre se mencionaban cuando se hablaba acerca del sector económico, pero, si realmente es cierto que esta tecnología posee estas propiedades, lo convencional es pensar que su aplicación a otros sectores diferentes de los financieros podría también ser beneficiosa.

Debido a todo lo mencionado anteriormente, se decide proponer y realizar este trabajo fin de grado de tipo investigación, con el propósito de estudiar la aplicación de la tecnología blockchain a un sector diferente del económico y de evaluar los resultados que se deriven de ello. Concretamente, en el presente trabajo se decide investigar la integración de esta tecnología en el sector industrial.

Previamente a la realización de este trabajo mi conocimiento acerca de blockchain era casi nulo, y se centraba casi todo en lo que había escuchado hablar de esta tecnología. Era algo nuevo para mí, lo cual me motivó e ilusionó mucho desde el comienzo, ya que me parecía un área realmente interesante de la que apenas había escuchado hablar. Además, el hecho de adentrarme en el mundo de la investigación era algo que me parecía también muy interesante y novedoso.

1.2 Objetivos

El objetivo global de este trabajo es el estudio de la tecnología blockchain de cara a ser aplicada en el sector de la industria.

Con este propósito en mente, se listan a continuación los subobjetivos marcados:

1. Llevar a cabo un análisis acerca del estado actual de la tecnología blockchain.
2. Realizar un estudio comparativo en el que se analicen y comparen las plataformas blockchain más importantes actualmente que permitan el desarrollo de aplicaciones industriales.
3. Desarrollar un prototipo acerca de un escenario o caso de uso industrial que haga uso de la tecnología blockchain. Para su elaboración se emplearán las plataformas Hyperledger Fabric y Ethereum.
4. Evaluar la adaptación de la tecnología blockchain al sector industrial tras el aprendizaje adquirido en el estudio comparativo y la experiencia obtenida en el desarrollo del prototipo.

1.3 Alcance

El alcance del presente trabajo se limita al estudio de la tecnología blockchain de cara a ser utilizada en sectores industriales, no se tiene como finalidad la investigación de esta tecnología en otros ámbitos como podrían ser los financieros o las criptomonedas.

En cuanto al prototipo creado, se desarrolla con dos objetivos, el primero es el de ofrecer una solución en la que se use la tecnología blockchain para un caso de uso industrial concreto, y el segundo objetivo es el de investigar lo que el estado actual de esta tecnología permite desarrollar a día de hoy y ver los resultados que ofrece. Este prototipo se realiza exclusivamente con estos dos objetivos, de carácter investigativo y académico, sin la existencia de fines comerciales ni lucrativos.

1.4 Estructura

Tras la presente introducción, se redactan los conceptos teóricos relativos a la tecnología blockchain, para en el tercer capítulo pasar a analizar su estado actual, tanto a nivel de plataformas como en materia de investigación, haciendo también alusión a sus orígenes.

En el capítulo 4, una vez que se tiene conocimiento de la situación actual, se seleccionan las plataformas blockchain más importantes a día de hoy que permiten la realización de aplicaciones para el ámbito industrial. En este capítulo se estudian y comparan dichas plataformas, con la intención de comprobar qué es lo que cada una de ellas ofrece.

Posteriormente, en el capítulo 5, se realiza la propuesta, en la cual se expone el caso de uso de la industria elegido para realizar el prototipo y se argumentan las características que este poseerá.

Una vez explicada y argumentada la propuesta, desde el capítulo 6 hasta el 10 se trata el desarrollo del prototipo, desde su análisis hasta su utilización, pasando por su diseño, implementación y pruebas.

Tras la realización de un estudio comparativo a nivel teórico y la experiencia práctica adquirida mediante la creación del prototipo, se lleva a cabo en el capítulo 11 una evaluación acerca de la integración de blockchain en el sector industrial a día de hoy.

En el capítulo 12 se incluyen los contenidos relativos tanto a la planificación como al presupuesto, y en el 13 se redactan las conclusiones globales y el futuro trabajo. Posteriormente, se listan las referencias bibliográficas utilizadas para la elaboración de este proyecto, y, por último, se incluyen los apéndices.

1.5 Acerca del Título de este Trabajo

El título de este trabajo fin de grado menciona a Ethereum y a Hyperledger. Originalmente, el título no hacía mención a ninguna plataforma concreta, pero, tras estudiar la situación actual y ver que Ethereum y Hyperledger eran las dos plataformas más importantes que permitían realizar aplicaciones de propósito general y que iban a ser las dos que se utilizarían para la realización del prototipo, decidimos modificar el título e incluir sus nombres en él, para que de esta manera fuese menos general y más descriptivo sobre los contenidos reales del trabajo.

Capítulo 2: Aspectos Teóricos

En el actual capítulo se explican los conceptos teóricos relacionados con este trabajo fin de grado.

2.1 Criptografía Asimétrica

La criptografía asimétrica [1] es un sistema criptográfico que permite conseguir confidencialidad y verificación tanto del origen como de la integridad en el intercambio de mensajes. En este sistema se usan pares de claves, estando cada par compuesto por una clave privada y una clave pública, las cuales están ligadas entre sí. Cada persona tiene un par de claves, donde la privada solo la sabe el propietario y debe mantenerla en secreto, y la pública se puede distribuir abiertamente a todo el mundo. El hecho de decir clave o llave tiene el mismo significado, ya que a este sistema también se le conoce como criptografía de clave pública, traducido del inglés *public key cryptography*, sin embargo, en este trabajo se utilizará el término clave.

Para la generación de las claves privadas se emplean procedimientos aleatorios y para la creación de las claves públicas se usan funciones unidireccionales, a las cuales se les proporciona como entrada una clave privada. En este tipo de funciones, dada una entrada, es fácil obtener una salida, pero dada una salida obtenida a partir de una entrada aleatoria, es extremadamente complicado calcular el valor de dicha entrada, lo cual implica que el obtener la clave privada correspondiente a una clave pública sea computacionalmente inviable y hace que este sistema sea seguro. De esta manera, siempre y cuando una persona mantenga en secreto su clave privada, el hecho de que todo el mundo pueda conocer su clave pública no supone un riesgo para que su clave privada se vea comprometida.

En criptografía asimétrica existen dos usos principales. El primero es el de cifrar un mensaje destinado a un receptor y que solo ese receptor sea el que pueda descifrarlo, y el segundo tiene que ver con las firmas digitales, donde un emisor firma un mensaje y el receptor de ese mensaje puede estar seguro de que ha sido realmente firmado por dicho emisor.

En lo relativo al caso del cifrado de mensajes, el emisor utiliza la clave pública del receptor para cifrar el mensaje, tras ello, dicho mensaje cifrado puede ser enviado a través de canales inseguros y comprometidos con la certeza de que ninguna persona que no sea el destinatario va a ser capaz de descifrarlo, ya que, para ello, se necesita la clave privada del receptor. De esta manera, se obtiene confidencialidad entre emisor y destinatario, ya que ningún individuo excepto el receptor va a poder descifrar el mensaje.

Para ejemplificar lo anterior y verlo de manera esquemática, supongamos que un emisor desea enviar un mensaje a un destinatario, el cual ha publicado su clave pública a todo el mundo, por tanto, el emisor la conoce. Este emisor usa una función de cifrado (C), para cifrar el mensaje. Esta función C, la cual se muestra en la Figura 2.1, posee dos entradas, un mensaje sin cifrar y la clave pública del receptor, y produce un mensaje cifrado como salida.

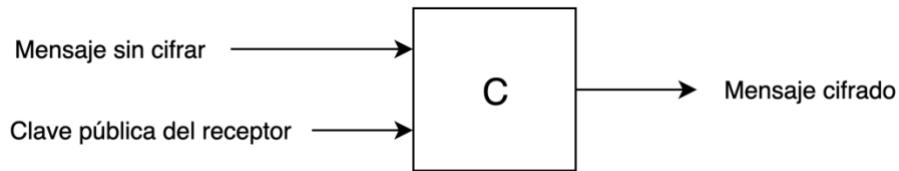


Figura 2.1: Función de cifrado (C)

Cuando el receptor obtiene el mensaje, usa la función de descifrado (D), mostrada en la Figura 2.2, para descifrarlo y poder leerlo. Esta función tiene también dos entradas, un mensaje cifrado y la clave privada del receptor, y como salida produce el mensaje original sin cifrar.

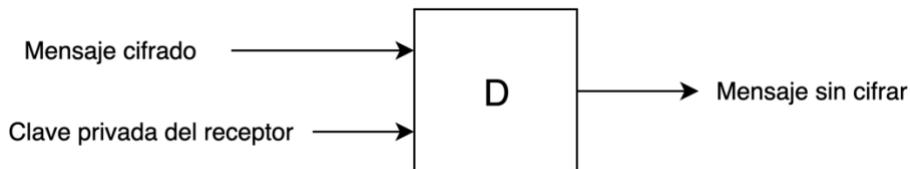


Figura 2.2: Función de descifrado (D)

En cuanto al caso de las firmas digitales, el emisor firma un mensaje con su clave privada y esa firma puede ser verificada por todo el mundo que tenga acceso a la clave pública del propio emisor. Si la verificación es correcta, significa que el emisor al que pertenece la clave pública firmó realmente ese mensaje con su clave privada, proporcionando autenticidad tanto del origen como de la integridad del mensaje. En caso de que un actor con malas intenciones cambie los contenidos del mensaje sin modificar la firma, la verificación fallará, ya que la firma está asociada tanto a la clave privada como al mensaje. En este caso de las firmas digitales, hay que tener en cuenta que el mensaje no se cifra, ya que el objetivo principal no es mantener el mensaje confidencial, sino probar que ese mensaje fue escrito por una determinada persona. Si se quiere que el mensaje viaje cifrado durante su envío, se debe utilizar un canal de transporte seguro.

En la Figura 2.3 se puede ver el esquema de una función de firmado (F), la cual recibe como entradas el mensaje y la clave privada del emisor, y produce una firma digital como salida.

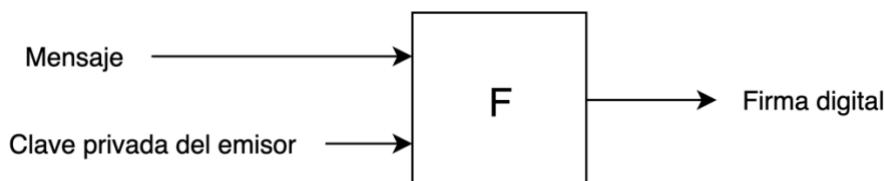


Figura 2.3: Función de firmado (F)

La firma obtenida en la salida de la función de firmado se asocia al mensaje y se envía al destinatario tanto el mensaje como la firma. Cuando le llega el mensaje al receptor, utiliza la función de verificación (V), mostrada en la Figura 2.4, para verificar que ese mensaje ha sido firmado por el emisor. Esta función recibe como entradas el mensaje, la firma y la clave pública del emisor, y como salida dice si la firma es válida o no.

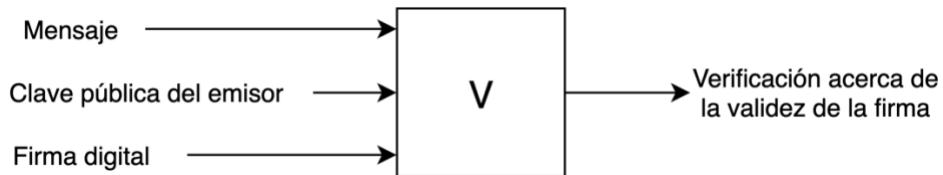


Figura 2.4: Función de verificación (V)

2.2 Función Hash

A diferencia de la criptografía asimétrica, que está pensada para que el mensaje cifrado se pueda descifrar, una función hash se utiliza para cifrar un mensaje que, una vez cifrado, ya no se pueda descifrar.

Una función hash recibe como entrada datos de cualquier longitud y tipo, y produce como salida una serie de caracteres alfanuméricos de una longitud concreta y fija, es decir, el tamaño de la salida es siempre el mismo, no depende del tamaño de la entrada. Dada una función hash determinada, siempre producirá la misma salida ante la misma entrada. A esta salida se le conoce simplemente con el nombre de hash.

Si una función hash es de buena calidad, creará salidas completamente distintas ante entradas casi idénticas. Esto permite que una salida sirva como identificador único de los datos de entrada, es decir, si la entrada a una función hash es un documento PDF, la salida es un identificador único de este documento, ya que, si el PDF se edita, aunque solamente se cambie una letra, la nueva salida de la función hash será completamente distinta a la salida obtenida antes de hacer la modificación. Todo lo mencionado anteriormente significa que si se está en posesión de un hash y se quiere saber cuales fueron los datos de entrada a una función hash que produjeron esa salida, la única posibilidad de averiguarlo es probar combinaciones aleatorias de entradas para ver si alguna coincide con ese hash concreto que se tiene en posesión, lo cual es computacionalmente muy costoso y temporalmente inviable.

En la Figura 2.5 se muestra la estructura de una función hash (H).

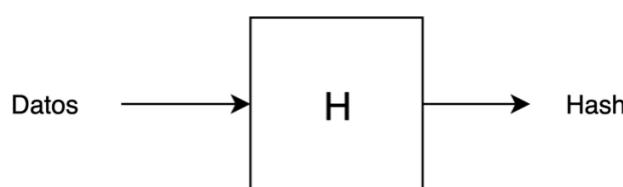


Figura 2.5: Función hash (H)

2.3 Certificado Digital

Un certificado digital es un archivo electrónico que contiene información acerca de su propietario y que sirve para comprobar su identidad. En él, siempre se incluye la clave pública del propietario, sus datos y una firma digital de una autoridad certificadora. En caso de que el certificado pertenezca a una persona, los datos que se incluyen pueden ser, entre otros, su nombre, apellidos, fecha de nacimiento u organismo en la que trabaja o estudia.

Una autoridad certificadora es una entidad que firma y emite certificados. Cada autoridad certificadora tiene una clave privada y una clave pública. La clave privada se usa para firmar los

certificados emitidos y la clave pública es usada para verificar posteriormente las firmas. A las autoridades certificadoras también se les conoce como CAs, ya que en inglés se traduce como *certificate authority*.

Las personas u organizaciones que van a verificar la identidad del propietario de un certificado necesitan saber que la autoridad certificadora que lo emitió es de confianza. Es decir, un certificado puede estar firmado por cualquier entidad, ya que, realmente, es firmar unos datos con una clave privada, pero lo importante es que esa firma provenga de una autoridad certificadora de confianza, en la que el verificador de la identidad confíe.

Para que una autoridad certificadora emita un certificado para una persona se siguen los siguientes pasos:

1. La persona proporciona los datos necesarios a la autoridad certificadora, entre los cuales se encuentra siempre la clave pública de la propia persona.
2. La autoridad certificadora verifica que los datos sean válidos.
3. En caso de que los datos proporcionados sean correctos, la entidad certificadora los firma con su clave privada y genera el certificado digital.

De esta manera, el verificador de la identidad puede usar la clave pública de la autoridad certificadora para verificar que la firma es realmente de dicha autoridad y que los datos de la persona son válidos.

En la Figura 2.6 se muestra la estructura de un certificado digital.

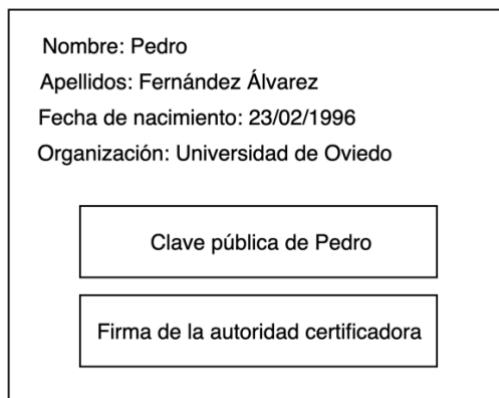


Figura 2.6: Estructura de un certificado digital

Una cuestión importante es la manera en la que una autoridad certificadora consigue que se confíe en ella. Esto es muy relativo, ya que existen autoridades certificadoras muy importantes y populares en las que generalmente todo el mundo confía, conocidas como entidades certificadoras raíz o CAs raíz, y existen otras que son más pequeñas, por ejemplo, una autoridad certificadora puede estar controlada por una empresa mediana, y dicha empresa y otras pequeñas empresas cercanas o filiales solo aceptan certificados emitidos por esta autoridad. En realidad, todo depende de las autoridades certificadoras en las que el verificador de la identidad confía, pero, en general, si un certificado proviene de una CA raíz, el verificador confiará en su validez.

En este último caso, realmente, un certificado no tiene que estar directamente firmado por una CA raíz, sino que puede estar firmado por una autoridad certificadora menos popular, conocida como CA intermedia, siempre y cuando esta CA intermedia cuente con la confianza de

una CA raíz, de esta manera, se crean las llamadas cadenas de confianza. Esto se consigue debido a que cada autoridad certificadora tiene también su propio certificado, en el que figuran los datos de la autoridad, su clave pública y una firma. En el caso de una CA intermedia en la que una CA raíz haya depositado confianza, el certificado de esta CA intermedia estará firmado por esa CA raíz. A su vez, dicha CA intermedia puede firmar el certificado de otra CA intermedia, creándose así las mencionadas cadenas de confianza, a través de las cuales se pueden ir viendo las firmas sucesivas en cadena hasta llegar a una CA raíz, cuyo certificado está firmado por ella misma.

Por último, una autoridad certificadora tiene la potestad de revocar un determinado certificado que ha emitido, lo cual significa que este certificado deja de ser válido. A modo de ejemplo, se puede revocar el certificado de una persona que se ha cambiado de organización, para que no exista un certificado válido que diga que dicha persona está trabajando en una organización que ya ha abandonado. Adicionalmente, los certificados poseen una fecha a partir de la cual dejan de ser válidos.

2.4 Árbol de Merkle

Un árbol de Merkle [2] es una estructura de datos en forma de árbol en la que cada nodo hoja es el hash de unos determinados datos y cada nodo interno contiene el hash de sus hijos, hasta llegar al nodo raíz, el cual contiene un único hash dependiente de todos los datos. Generalmente se conoce a los árboles de Merkle como árboles binarios, pero pueden ser no binarios.

En la Figura 2.7 se muestra un ejemplo concreto de un árbol de Merkle.

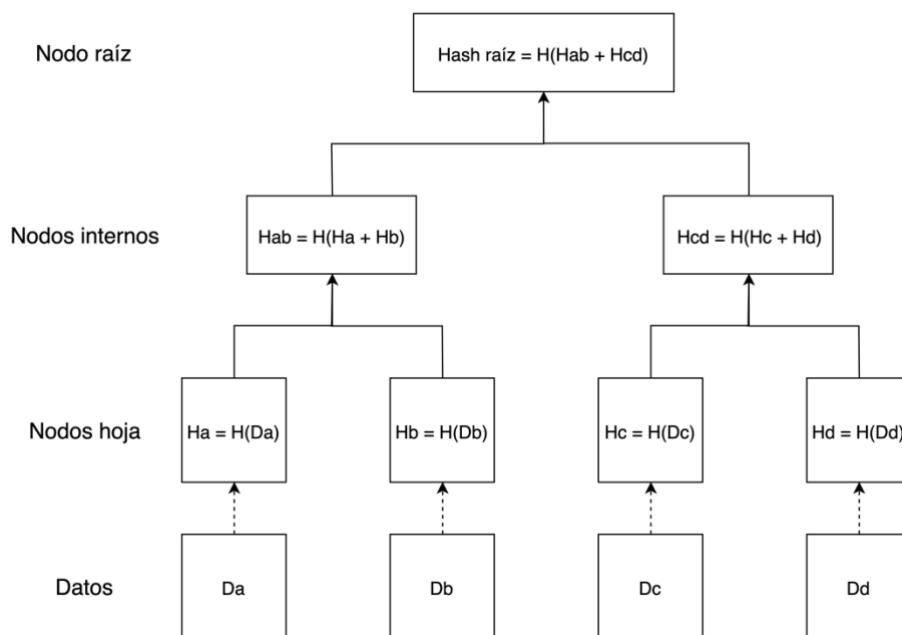


Figura 2.7: Árbol de Merkle

Este tipo de árboles se suelen usar a la hora de validar un dato sin necesidad de almacenar todos los datos. En este ejemplo hay cuatro datos: dato a (Da), dato b (Db), dato c (Dc) y dato d (Dd). Supongamos que una persona llamada Pedro quiere validar ante otra persona llamada Guillermo, el cual solamente posee el hash raíz del árbol, el valor de Db. En lugar de enviarle todos los datos para que Guillermo calcule el hash raíz y compruebe que es igual al suyo, Pedro

solamente necesita enviarle el propio Db y los hashes Ha y Hcd. Con esta información, Guillermo realiza el hash de Db y sigue calculando hashes hasta obtener el hash raíz, el cual, si coincide con el que él posee, Guillermo puede estar seguro de que Db es correcto y de que Pedro no le ha mentido.

El ejemplo anterior es un caso muy sencillo, pero cuando la cantidad de datos que se almacenan es muy superior, este método ofrece la posibilidad de verificar si un dato concreto es válido solamente obteniendo el dato, el hash raíz y una serie de hashes intermedios.

En caso de que el hash raíz fuese $H(H(Da) + H(Db) + H(Dc) + H(Dd))$, entonces, asumiendo que Guillermo tiene ese hash y ningún dato, Pedro tendría que enviar todos los datos a Guillermo para validar un solo dato, ya que Guillermo necesita todos los datos para obtener el hash raíz, lo cual es muy poco eficiente.

2.5 Red P2P

Una red P2P (*peer-to-peer* en inglés) o red de pares es una red en la que todos los equipos que forman parte de ella, llamados nodos, tienen los mismos derechos, es decir, se tratan y comportan como iguales. La popularidad de este tipo de redes comenzó a crecer a partir del lanzamiento en 1999 de la plataforma Napster [3], la cual, actualmente, ofrece servicios de música en *streaming*.

A diferencia de una arquitectura tradicional cliente-servidor, donde el cliente consume servicios y es el servidor quien los proporciona, cada nodo formando parte en una red P2P puede tanto consumir servicios como proporcionarlos, esto significa que, en este caso, cada nodo se comporta a la vez como cliente y como servidor.

Todos los nodos en una arquitectura *peer-to-peer* aportan a la red un porcentaje de sus propios recursos, como pueden ser la capacidad de procesamiento, el ancho de banda o el almacenamiento.

En el caso de que un usuario desee enviar un archivo que se encuentra en su ordenador a otro usuario, en una arquitectura cliente-servidor, ese archivo se envía primero al servidor, y este lo envía al usuario de destino. En cambio, en una red P2P todos los ordenadores pueden estar conectados con los demás, normalmente a través de internet, por tanto, el archivo se envía desde el ordenador del usuario al ordenador del destinatario de manera directa, sin la intervención de ningún servidor central.

En la Figura 2.8 se puede observar gráficamente la diferencia entre una arquitectura cliente-servidor y una arquitectura *peer-to-peer*.

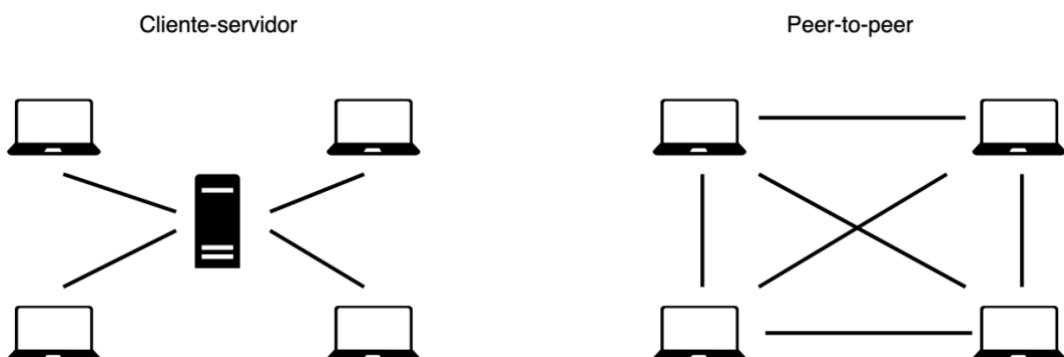


Figura 2.8: Diferencia entre cliente-servidor y peer-to-peer

2.6 Tecnología Blockchain

La tecnología blockchain, surgida por vez primera en [4], puede visualizarse de manera genérica como un sistema de transición de estados, en el cual, para pasar de un estado a otro, debe producirse un evento. Se puede definir una función de transición de estados (T), mostrada en la Figura 2.9, la cual recibe el estado actual y un evento, y produce un nuevo estado.

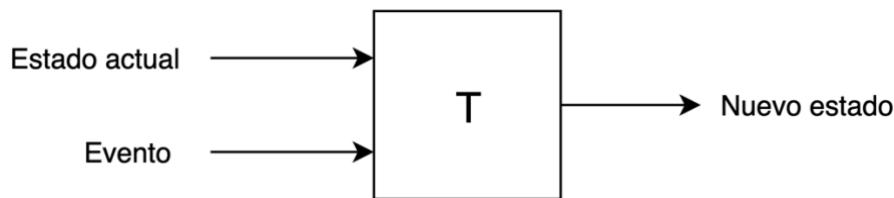


Figura 2.9: Función de transición de estados (T)

2.6.1 Transacciones

En blockchain, los eventos mencionados anteriormente se conocen como transacciones, siendo estas una parte vital cuando se habla de esta tecnología, ya que es el mecanismo que permite que haya cambios de estado.

A modo de ejemplo, imagínese que el estado refleja las canicas que posee cada integrante de un grupo de amigos. Este grupo de amigos tiene tres miembros (Pedro, David y Guillermo) y cada uno de ellos tiene en este momento un número concreto de canicas (Pedro posee 4, David tiene 5 y Guillermo 8). Tras una partida de canicas en la que David queda el primero y Pedro queda el último, Pedro debe dar una de sus canicas a David, por tanto, cuando esto ocurra, Pedro tendrá 3 canicas y David 6.

La transacción en este caso es “Pedro da una de sus canicas a David”. En la Figura 2.10 se representa de manera gráfica el estado previo y el posterior a la realización de esta transacción.

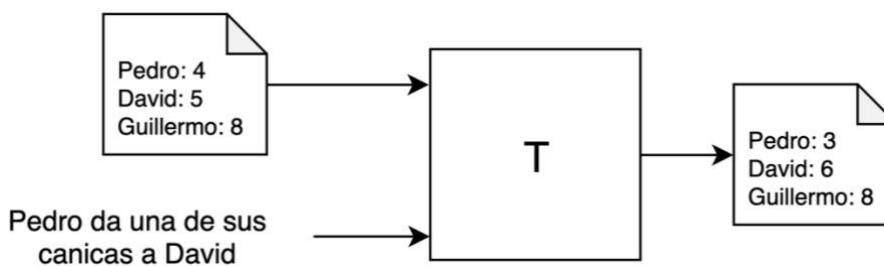


Figura 2.10: Ejemplo de transacción

En el caso de que una transacción sea inválida y no se pueda realizar, como sería la transacción “Guillermo da 100 canicas a Pedro”, ya que Guillermo solamente dispone de 8, lo que ocurre es que se produce un error y el estado no cambia.

Previamente a la realización de la primera transacción de todas, el estado es igual al estado vacío. A partir de ahí, se van produciendo transacciones del tipo: “Pedro y David se hacen amigos” para añadir miembros al grupo y del tipo “David compra 3 canicas” o “David da 2 canicas a Pedro” para que el número de canicas varíe y el estado vaya reflejando estos eventos o transacciones. Es importante mencionar que el estado actual es el resultado de aplicar todas las

transacciones partiendo de un estado vacío, por tanto, lo único que se requiere conocer para obtener el estado actual son todas las transacciones realizadas hasta el día de hoy.

2.6.2 Bloques

Las transacciones mencionadas en la sección anterior se almacenan ordenadamente en unidades llamadas bloques. A medida que estas transacciones se realizan, se van agrupando en dichos bloques, los cuales se enlazan o encadenan uno tras otro siguiendo un orden para formar una cadena, originando de esta manera el concepto de cadena de bloques, traducido del término en inglés *blockchain*. Cabe destacar que la palabra blockchain se suele usar para hacer referencia tanto a la tecnología blockchain en general como a la propia cadena de bloques.

Para que cada bloque esté enlazado a su predecesor, cada uno de ellos almacena información acerca de su bloque anterior, concretamente, cada bloque almacena el identificador o hash de su bloque previo. Además, cada bloque también almacena su propio hash.

La Figura 2.11 muestra de forma esquemática los contenidos principales de un bloque. En ella se aprecia que cada uno de ellos también guarda el número de bloque, el cual hace referencia a la posición que ocupa en la cadena. Al primer bloque, el número 0, se le conoce como el bloque génesis. Cada bloque almacena un número de transacciones limitado, por tanto, cuando un bloque alcanza el límite máximo de transacciones que puede almacenar se crea un bloque nuevo.

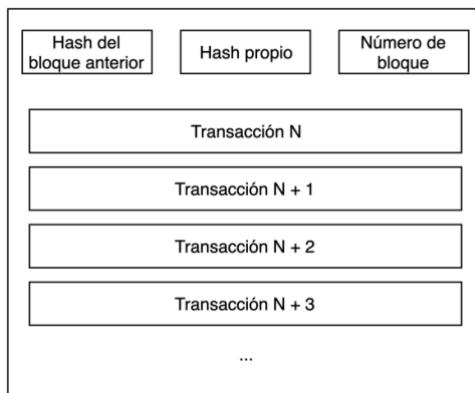


Figura 2.11: Contenidos de un bloque

El hash de cada bloque se calcula en base a la información que contiene, esto implica que, si se cambia algún contenido de un bloque, su hash también cambie, provocando que el campo del bloque sucesor en el que se almacena el hash del bloque anterior ya no coincida con el hash del bloque modificado, lo cual hace que la cadena ya no esté enlazada. Este hecho es lo que hace que la cadena de bloques sea inmutable, ya que, si se realiza un cambio, por pequeño que sea, en los contenidos de un bloque intermedio, la cadena dejará de estar unida.

En la cadena de bloques se almacenan todas las transacciones de manera ordenada, por tanto, ejecutando todas las transacciones de la cadena en orden se obtiene el estado actual. Sin embargo, aunque el estado actual esté relacionado con la cadena de bloques, son conceptos diferentes. Existe el término inglés *ledger*, el cual agrupa a la cadena de bloques y al estado actual a la vez, y, debido a su popular uso, en este trabajo también se utilizará dicho término.

Para finalizar esta subsección, se presenta en la Figura 2.12 una porción de una cadena de bloques, en la que, para simplificar el ejemplo, se almacenan dos transacciones en cada bloque.

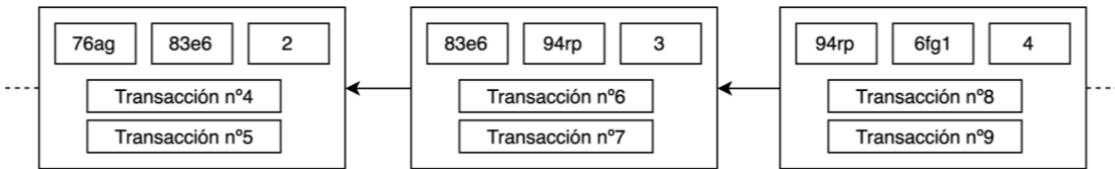


Figura 2.12: Cadena de bloques

2.6.3 Consenso

La cadena de bloques no está almacenada en un único lugar, sino que se encuentra replicada en diversas máquinas que están conectadas entre sí, formando una red. Este tipo de redes se conocen como redes blockchain. Todas las máquinas, conocidas como nodos, formando parte de una red blockchain deben tener la misma versión o copia de la cadena de bloques, de esta manera, no importa el nodo con el que me comunique, ya que todos me darán la misma respuesta. El proceso mediante el cual todos los nodos se comunican y operan entre sí para almacenar la misma copia de la cadena de bloques a medida que esta aumenta de tamaño se llama consenso, y existen diversas estrategias o algoritmos para conseguirlo.

El hecho de que la cadena esté replicada en muchos lugares significa que nos encontramos ante un sistema distribuido, y no centralizado, evitando de esta manera tener un único punto de fallo, ya que, si ocurre un problema en un determinado nodo, los datos no se verán comprometidos al estar replicados en otros nodos.

Un concepto a tener en cuenta es que la cadena de bloques, de manera lógica y abstracta, es única, pero de manera física, existen muchas copias de ella, cada una almacenada en un nodo.

2.6.4 Contratos Inteligentes

Los nodos, además de almacenar una copia actualizada de la cadena, pueden ejecutar programas llamados normalmente contratos inteligentes, traducidos del término en inglés *smart contract*. Un contrato inteligente es software diseñado para ejecutar transacciones. Estos contratos se despliegan en los nodos de una red blockchain, y son los propios nodos los que se encargan de su ejecución cuando reciben una petición para ello.

En estos contratos es donde está la lógica de las diferentes funciones de transición de estados. En el ejemplo relacionado con las canicas mostrado en la sección 2.6.1, se podría programar un contrato inteligente, el cual, entre otras, tendría una función para ejecutar transacciones en las que una persona da canicas a otra. Esta función se ejecutaría cuando ocurriese una transacción del tipo “Pedro da una de sus canicas a David” y estaría parametrizada con la persona que da las canicas, la que las recibe, y el número de canicas, y lo que haría sería decrementar el número de canicas del donante, e incrementar las de la persona que las recibe, actualizando de esta manera el estado actual. En dicho contrato también podría haber funciones para añadir amigos al grupo o para que un miembro compre un determinado número de canicas.

Previamente a la inserción de una transacción en un bloque se ejecuta una función de transición de estados implementada dentro de un contrato inteligente. Los nodos reciben transacciones y ejecutan las funciones de transición de estados correspondientes antes de introducirlas en un bloque.

2.7 Tipos de Redes Blockchain

Existen diferentes tipos de redes blockchain según su ámbito o alcance: públicas, privadas y de consorcios.

2.7.1 Redes Públicas

Una red blockchain pública es aquella en la que cualquier persona puede unir un nodo para mantener en él una copia actualizada de la cadena de bloques y para ejecutar transacciones que llegan a la red con el objetivo de generar nuevos bloques. En este tipo de redes todo el mundo puede enviar transacciones a la red para que sean procesadas o ejecutadas por los nodos. El acceso a los contenidos de la cadena de bloques es completamente público, por tanto, cualquier persona tiene acceso a dichos contenidos.

Una red pública no está controlada por una entidad ni un conjunto de entidades, sino que su gestión es completamente descentralizada, ya que todos los nodos se comunican entre sí para contribuir al mantenimiento de una única versión de la cadena a medida que esta crece.

2.7.2 Redes Privadas

En el caso de una red privada, todos los nodos son controlados por una misma compañía o entidad, por tanto, aunque es cierto que al existir varios nodos la cadena está almacenada en todos ellos, realmente, el control de estos nodos está centralizado en una sola empresa, es decir, el almacenamiento de los datos no está centralizado pero el control de la red sí.

Generalmente, este tipo de redes poseen un sistema de permisos tanto para unir un nuevo nodo como para reflejar quién puede realizar transacciones o acceder a los datos de la cadena de bloques.

2.7.3 Redes de Consorcios

Una red de consorcios está compuesta por una serie de nodos controlados por diferentes empresas. A diferencia de las redes privadas, donde todos los nodos son controlados por la misma empresa, en las redes de consorcios participan diferentes compañías y cada una de ellas aporta una cantidad determinada de nodos a la red. Por tanto, el control de una red de consorcios no está centralizado en una empresa, sino que son varias compañías las que se encargan de su control y gestión.

En este caso, al igual que en las redes privadas, lo normal es que exista un sistema de permisos para gestionar la adición de nuevos nodos, el acceso a los datos almacenados en la cadena o las personas o identidades que tienen permitido realizar transacciones.

Capítulo 3: Estado del Arte

En este capítulo se hace referencia a la historia y a la situación actual de la tecnología blockchain. Para comenzar, se describen las plataformas y proyectos más relevantes que existen actualmente, haciendo referencia a sus orígenes e introduciéndolos de manera cronológica. Tras ello, se estudia el estado actual en el ámbito de la investigación.

3.1 Historia

Los orígenes de la tecnología blockchain se remontan al año 2008. Desde entonces, su popularidad ha ido creciendo poco a poco, y son numerosas las plataformas que han ido aparecido durante la última década de cara a explorar y hacer uso de esta tecnología.

3.1.1 Bitcoin

En el mes de octubre de 2008 se publica en internet el documento titulado *Bitcoin: A Peer-to-Peer Electronic Cash System* [4], elaborado por Satoshi Nakamoto. Aunque actualmente no está del todo claro, Satoshi Nakamoto se cree que es un seudónimo, y no se conoce la persona o grupo de personas que se han encargado de confeccionar dicho documento.

En esta publicación se proponía la creación de una moneda digital segura llamada Bitcoin [5], con la cual se podrían realizar transacciones directamente entre dos partes sin que exista una tercera parte involucrada, como podría ser un banco. Esto implica que una persona podría entregar dinero a otra persona de manera directa sin que un banco tenga que actuar de intermediario, es decir, sería como si una persona entregase monedas físicas a otra, pero en este caso serían monedas digitales, conocidas como bitcoins.

Hasta ese momento, por ejemplo, si una persona llamada Pedro quería hacer una transferencia digital de 50 € desde su cuenta bancaria a la cuenta bancaria de otra persona llamada Lucía, era el banco el encargado de efectuar esta transacción, retirando 50 € de la cuenta de Pedro y añadiéndolos a la de Lucía. En el caso de Bitcoin, se eliminaría la necesidad de que un banco tenga que actuar de intermediario en estas transferencias digitales de dinero, y hace posible que Pedro le pueda enviar dinero a Lucía de manera digital y sin ningún tipo de intermediarios.

Para que esto fuese posible, se propone la idea de crear una cadena de bloques, también llamada blockchain, que permita guardar todas las transacciones y que no se puedan modificar una vez confirmadas. En esta concatenación de bloques, cada bloque está ligado al anterior de manera que no se puede romper ese enlace. Es dentro de cada bloque donde se guardan las transacciones monetarias que se van produciendo en tiempo real. Para enlazar estos bloques, se propone que cada uno de ellos guarde información acerca del anterior, concretamente, cada bloque guarda el hash de su bloque antecesor, de manera que, si los contenidos del bloque anterior se modifican, la unión se rompe, haciendo que esta cadena de bloques sea inmutable.

Estos bloques no están almacenados en un solo lugar, sino que se almacenan de manera distribuida, es decir, existen copias de estos bloques en distintas máquinas, conocidas como nodos. Estas máquinas están sincronizadas unas con otras formando una red, siendo esta

sincronización necesaria para que en cada máquina se mantenga la misma copia de la cadena de bloques. Hasta este momento, cada banco tenía en su propia base de datos los datos de las transacciones realizadas, por lo tanto, todos estos datos se encontraban centralizados. El hecho de que la cadena de bloques esté replicada en muchas máquinas pertenecientes a cualquier persona o entidad y repartidas por todo el mundo evita que los datos estén centralizados en posesión de un banco, encontrándose ahora distribuidos y haciendo posible que ya no haga falta depositar confianza en una tercera parte para realizar y verificar cada transacción.

Para la creación de los bloques se propone la idea de la prueba de trabajo, la cual consiste en que, para crear un bloque, se debe resolver un problema con la peculiaridad de que resolverlo sea computacionalmente costoso, pero una vez resuelto, comprobar que se ha hecho correctamente sea inmediato. Cualquier máquina que esté formando parte de la red Bitcoin puede intentar crear bloques, esto provoca que haya competencia entre ellas para conseguir crear el próximo bloque que forme parte de la cadena. Esta competencia se produce porque la creación de un bloque proporciona incentivos al creador, al cual se le anima a crear bloques válidos y a no cometer acciones con malas intenciones, ya que, si lo hace, su bloque no será aceptado en la cadena, y habrá perdido tiempo y empleado electricidad intentando resolver el problema para no conseguir ningún beneficio.

Las transacciones que se almacenan en la cadena son anónimas y seguras, lo cual se consigue gracias a la criptografía asimétrica. Para que Pedro pueda enviar 50 bitcoins a Lucía, Lucía tiene que enviar su clave pública a Pedro, tras ello, Pedro crea una transacción en la que indica la cantidad de bitcoins que quiere enviar y la clave pública de Lucía, una vez creada la transacción, Pedro la firma con su clave privada y distribuye su clave pública para que la firma pueda ser verificada, por último, Pedro envía la transacción a la red para que sea incluida en un bloque.

Todo el mundo tiene acceso a la cadena de bloques, por tanto, todos pueden ver todas las transacciones que se han realizado, lo cual otorga al sistema un grado de transparencia muy elevado. La privacidad se consigue haciendo que las claves públicas sean anónimas, es decir, cualquiera puede ver que alguien que posee una determinada clave pública le está enviando dinero a alguien que es poseedor de otra determinada clave pública, pero no se sabe quién es la persona real que posee cada clave pública. Una persona solamente debe saber la clave pública de otra persona si le va a enviar bitcoins, pero para más privacidad, se propone que se generen un par de claves para cada transacción que se realice, por tanto, en el caso del ejemplo anterior, la clave pública que Lucía le envía a Pedro es única para realizar esa transacción de 50 bitcoins, pero Lucía tendrá una clave pública distinta en cada futuro pago que reciba. Si una persona tuviese solamente una clave pública para todas sus transacciones, aunque la mayoría de la gente no supiese quién es el poseedor de esa clave, se podrían rastrear todas sus transacciones y podría ser posible averiguar quién su poseedor, pero al generarse un par de claves para cada transacción, la dificultad para rastrear las transacciones que involucren a una cierta persona es mucho más alta.

Los aspectos teóricos de la publicación de Bitcoin resumidos anteriormente se pusieron en práctica a comienzos del año 2009, momento en el cual se crean los primeros bitcoins y se pone en marcha la red Bitcoin, de la cual cualquier máquina de cualquier persona podía formar parte y poseer la capacidad de generar bloques.

Anteriormente se ha hecho referencia al concepto de prueba de trabajo, este mecanismo es el que comenzó empleando Bitcoin y sigue empleando en la actualidad para que exista consenso entre todos los nodos de la red. La cadena de bloques válida será siempre la que tenga una mayor longitud, ya que es la cadena en la que más ha trabajado toda la red en conjunto, por tanto, ante cualquier duda sobre cual es la cadena válida, se escoge la que más bloques

contenga. El inconveniente de que para crear un bloque se tenga que resolver un problema computacionalmente complejo y de tener a muchos nodos intentando resolverlo simultáneamente implica que el gasto de electricidad es muy alto. Un estudio reciente [6] refleja que la red Bitcoin consume aproximadamente la misma electricidad que los países de Rumanía, Portugal o Singapur, y más del triple de la electricidad que consume Croacia.

3.1.2 Criptomonedas Alternativas

Bitcoin, cuyo símbolo es BTC, fue la primera moneda virtual que utilizaba una cadena de bloques para almacenar las transacciones. A este tipo de monedas se les conoce como criptomonedas, ya que la criptografía es un pilar básico para su correcto funcionamiento y seguridad.

El código del protocolo de la red Bitcoin es *open source*, por tanto, tras su publicación, comenzaron a aparecer otras criptomonedas diferentes partiendo de la base del código de Bitcoin e intentando mejorar algunas de sus funcionalidades. Algunas de las que se crearon tras el lanzamiento de Bitcoin fueron Litecoin (LTC) [7], en el año 2011, o Peercoin (PPC) [8], en 2012.

Los fundamentos de todas las criptomonedas eran comunes, la creación de una base de datos pública, distribuida e immutable, donde se guardasen todas las transacciones monetarias de manera anónima, transparente y segura, y donde la gestión de dichas transacciones no es tarea de ninguna autoridad central, sino que es la propia red la que se encarga de gestionarlas y confirmarlas.

Cada sistema de criptomonedas posee su propia red de nodos y su propia cadena de bloques, y la manera mediante la cual se crean y ponen en circulación nuevas monedas es determinada por el propio protocolo de cada criptomoneda. En lo relativo al valor de estas monedas, no existe ninguna entidad central, organismo o gobierno que se encargue de regularlo, por tanto, el valor que posee cada criptomoneda sufre muchas variaciones a lo largo del tiempo.

Durante los últimos años han seguido apareciendo multitud de nuevas criptomonedas, cada una con sus peculiaridades y avances, pero hoy en día, y a pesar de que fue la primera en aparecer, la criptomoneda más popular sigue siendo Bitcoin.

En la Figura 3.1, obtenida del sitio web Coindesk [9], se muestra la evolución en el valor de 1 bitcoin a lo largo del tiempo.

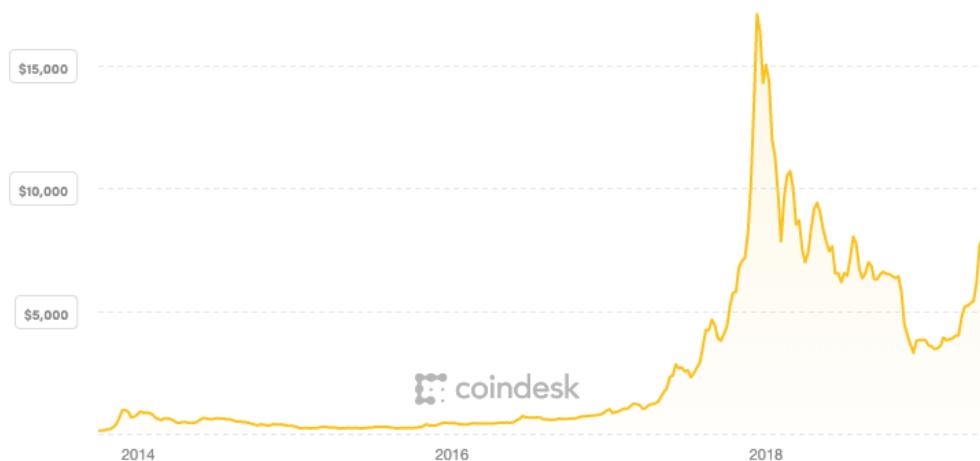


Figura 3.1: Variación del valor de 1 bitcoin

Desde la puesta en marcha de Bitcoin en 2009 el valor de 1 bitcoin sufrió variaciones, pero siempre situándose en cifras bajas hasta principios de 2014, siendo \$0,30 a comienzos de 2011, \$5,27 a comienzos de 2012 y \$13,30 a comienzos de 2013. Es a comienzos del año 2014 cuando su valor pasa a situarse en \$770, y a partir de ese momento se puede observar en la gráfica la evolución del valor hasta el día de hoy. El valor máximo que ha alcanzado 1 bitcoin hasta la fecha es de \$19.783,06 el día 17 de diciembre de 2017. Sin embargo, cinco días después, su valor bajó hasta \$13.800, y continuó decayendo hasta situarse en \$6.200 el 5 de febrero de 2018, lo cual refleja los cambios bruscos en el valor de las criptomonedas que se pueden producir debido a la falta de actividad regulatoria.

En el momento de escribir este texto, Bitcoin es la criptomoneda que más valor tiene, con un valor de \$8.709, con mucha diferencia sobre todas las demás, ya que ninguna del resto supera los \$1.000 de valor.

3.1.3 Ethereum

En el año 2013, Vitalik Buterin publica a la edad de 19 años el documento titulado *A Next-Generation Smart Contract & Decentralized Application Platform* [10], cuya versión más actual se encuentra en [11]. En este documento se menciona a Bitcoin, pero haciendo hincapié en que solo almacena transacciones monetarias y en que está solamente pensada para ello. Tras ello, se hace referencia a la tecnología que se utiliza para almacenar dichas transacciones, y se propone la idea de que no solamente sean transacciones de tipo monetario, sino que puedan almacenarse todo tipo de transacciones. Para ello, se propone la creación de una plataforma blockchain llamada Ethereum [12] en la que se pueda ejecutar software destinado a tratar con cualquier caso de uso, no solamente casos de uso monetarios. El objetivo, por tanto, es la creación de una plataforma blockchain de propósito general.

La ejecución de software en Ethereum se realiza a través de contratos inteligentes que cualquier persona puede desplegar a la red y que están escritos en un lenguaje Turing completo conocido como *Ethereum Virtual Machine Code* (EVM Code). Mediante la ejecución de estos contratos, se pueden realizar transacciones de cualquier tipo. El EVM Code es código binario resultado de la compilación de un contrato inteligente escrito en un lenguaje de alto nivel, este código binario está diseñado para ser ejecutado por la *Ethereum Virtual Machine* (EVM).

En Bitcoin ya existía la posibilidad de ejecutar contratos, pero estos contratos están muy limitados, ya que están escritos en un lenguaje de programación que no es Turing completo, y, por ejemplo, no permiten la posibilidad de ejecutar bucles. Si bien es cierto que el hecho de no contener bucles no impide realizar cualquier cosa que podría ser hecha con ellos, a través de la repetición de sentencias y del uso de sentencias condicionales, sí que contribuye al hecho de que realizar un programa que en principio es sencillo pueda convertirse en una tarea compleja. Adicionalmente, los contratos en Bitcoin, al igual que Bitcoin en sí, solamente están pensados para casos de uso monetarios.

Con esta idea de una cadena de bloques que pueda contener transacciones de propósito general se abrió un nuevo mundo de posibilidades para la exploración de lo que podría aportar esta tecnología en otros casos de uso distintos a los monetarios.

Un año después, en 2014, Gavin Wood publica el documento *Ethereum: A secure decentralized generalized distributed ledger* [13], en el que se describe formalmente el protocolo de Ethereum, o, dicho de otra manera, se describe de manera formal el funcionamiento de Ethereum. Actualmente, este documento está mantenido por Nick Savers y cuenta con contribuciones de gente residente en diferentes lugares del mundo. La versión

actualizada de este documento se puede encontrar en [14]. También en este año, 2014, comienza una campaña de financiación para el proyecto de Ethereum, y en 2015 el proyecto se convierte en una realidad, poniendo a disposición de todo el mundo una plataforma en la que se pueden realizar transacciones de tipo genérico de forma anónima y transparente.

Aunque Ethereum está pensado como un proyecto de propósito general, también cuenta con una criptomonedas integrada conocida como Ether (ETH), cuyo valor actual es de unos \$240. En junio de 2016, un atacante descubrió una vulnerabilidad en un contrato desplegado en la red de Ethereum y consiguió robar Ether por valor de unos \$50.000.000, lo cual hizo que la cadena de bloques de Ethereum sufriese una bifurcación, por un lado se quedó la rama en la que se sigue reflejando el robo del atacante, conocida como Ethereum Classic y por otro lado, este robo se canceló y la cadena actual en la que no se refleja este ataque es conocida simplemente como Ethereum. Este hecho trajo consigo una gran polémica, ya que la filosofía de blockchain reside en que las transacciones realizadas sean inmutables.

Ethereum ha sido concebido para su uso público y global. No existe ningún tipo de permiso integrado en su protocolo para poder unirse a una red, desplegar contratos inteligentes en ella, realizar transacciones o crear bloques, cualquier persona puede realizar todas estas tareas. Todo el mundo puede también leer el código de los contratos inteligentes y ver todas las transacciones que se han hecho en la red.

Ethereum fue la primera plataforma en crear una estructura blockchain públicamente accesible y de propósito general, y hoy en día es la más utilizada y la más importante en el ámbito de las redes blockchain públicas. Existen proyectos con la misma idea que Ethereum, y que intentan mejorar sus características, los dos más prometedores actualmente son EOSIO [15] y Cardano [16]. La primera versión estable de EOSIO fue lanzada el 1 de junio de 2018, y Cardano sigue en fase de desarrollo, lo cual quiere decir que ambos son proyectos muy recientes y cuyo grado de madurez es muy bajo todavía. Como se ha mencionado anteriormente, en el ámbito actual de las redes blockchain públicas, Ethereum es el proyecto más estable y relevante a día de hoy, siendo la Ethereum Foundation [17] la encargada de su desarrollo, mantenimiento e investigación.

Actualmente, Ethereum emplea el mismo mecanismo que la red Bitcoin para la generación de bloques y el consenso entre los nodos, la prueba de trabajo. Por lo tanto, existe también en el caso de Ethereum el problema del gasto energético para el mantenimiento de la red. En los planes de Ethereum para su siguiente versión, la 2.0, está el cambiar este método y pasar a otro llamado prueba de participación, el cual es mucho más eficiente energéticamente. Pero la siguiente versión de Ethereum todavía está en construcción e investigación, como se indica en la nota de prensa lanzada el 29 de marzo de este año [18].

3.1.4 Hyperledger

En el mes de diciembre del año 2015, The Linux Foundation [19] lanza el proyecto Hyperledger [20], con la intención de crear una plataforma para impulsar la tecnología blockchain en lo referente a la comunicación entre diferentes empresas. Hasta ese momento, la filosofía y tendencia de los proyectos blockchain estaba enfocada a las redes públicas, pero Hyperledger decide explorar el desarrollo de herramientas que permitan la creación de redes en las que no todo el mundo pueda participar libremente, sino que existan una serie de restricciones o permisos para controlar quién puede leer datos o realizar transacciones.

Hyperledger cuenta actualmente con seis proyectos *open source*: Fabric [21], Sawtooth [22], Indy [23], Iroha [24], Burrow [25] y Grid [26]. Tanto Burrow como Grid están en fase de

desarrollo y todavía no cuentan con versiones en producción. Los cuatro restantes son muy recientes, pero en cambio sí cuentan con versiones en producción.

A continuación, se mencionan las principales características de cada uno de los seis proyectos:

- **Fabric:** Desarrollado inicialmente gracias a la contribución de IBM y Digital Asset, Fabric es un proyecto que permite la creación de redes blockchain de consorcios y de propósito general, las cuales poseen un sistema de permisos asociado. Su desarrollo no es fijo ni cerrado, sino que es modular, ofreciendo la posibilidad de poder elegir entre diversas opciones para un mismo propósito, por ejemplo, se puede elegir el tipo de base de datos que se va a emplear para guardar las transacciones y los bloques. La principal y exclusiva característica de este proyecto es el uso de canales, los cuales sirven para alojar una cadena de bloques propia. Es decir, en una red de Fabric puede haber varios canales, y en cada uno de ellos una cadena diferente. Esta es una característica única de Fabric y muy interesante, ya que varias empresas pueden formar parte de la misma red, pero se puede configurar a qué canales pueden acceder, pudiendo, por tanto, mantener datos privados entre empresas dentro de una misma red.
- **Sawtooth:** Al igual que Fabric, Sawtooth es una herramienta para crear redes de consorcios de propósito general, presentando en este caso también una arquitectura modular, pero a diferencia de Fabric, ofrece además la posibilidad de crear redes en las que no exista ningún tipo de permiso y que el acceso a estas sea completamente libre.
- **Indy:** Indy es un proyecto orientado a la gestión de identidades en blockchain y a hacer posible que esas identidades sean interoperables entre diferentes organismos o empresas.
- **Iroha:** Este proyecto está orientado a la construcción de redes de propósito general que implementen un sistema de control de acceso. Ha sido desarrollado por las compañías Soramitsu, Hitachi NTT Data y Colu.
- **Burrow:** Con el apoyo de Intel y de Monax, Burrow es un proyecto cuyo objetivo es la creación de una herramienta que permita añadir un sistema de permisos a los contratos inteligentes escritos en un lenguaje que pueda ser compilado a EVM Code. Para ello, el propósito es crear una plataforma blockchain que posea un intérprete de EVM Code, y dotarla de un sistema de control de acceso.
- **Grid:** Hyperledger Grid, a diferencia del resto, no es un proyecto dedicado a construir una plataforma blockchain que de soporte al mantenimiento y gestión de una cadena de bloques. Su objetivo es la construcción de un framework que se pueda integrar con una red blockchain ya existente. Concretamente, ese framework está pensado para casos de uso en los que existen cadenas de suministro de productos.

De los proyectos descritos anteriormente, Fabric es en la actualidad el proyecto más importante y relevante de todos. Su mayor virtud es el uso de canales, los cuales son una funcionalidad muy interesante y que dota a este proyecto de muchas ventajas y posibilidades.

Además de ser el más relevante, Fabric también fue el primer proyecto de Hyperledger en pasar a fase de producción, su primera versión estable (v1.0) fue puesta en producción el 11 de julio de 2017, siendo por tanto un producto todavía muy reciente, pero muy interesante, cuyo

desarrollo sigue siendo constante. Actualmente su versión estable más reciente es la versión 1.4, la cual se lanzó a comienzos de este año 2019.

En cuanto a la fecha de lanzamiento del resto de proyectos, la primera versión estable de Sawtooth fue lanzada a comienzos de 2018, la de Indy fue también lanzada en 2018, pero a finales de año. En lo relativo a la primera versión de producción de Iroha, ha sido liberada en mayo de este año 2019, y, como se ha mencionado con anterioridad, Burrow y Grid todavía no poseen una versión estable en producción.

3.1.5 Corda

En 2015, Barclays, Credit Suisse, Goldman Sachs, J.P. Morgan y RBS forman el consorcio llamado R3 [27], con el objetivo de investigar la aplicación de blockchain a la hora de realizar operaciones financieras entre empresas. Un año más tarde, R3 lanza Corda, una plataforma destinada a registrar transacciones financieras entre empresas de cualquier tipo, por ejemplo, médicas, de seguros, o de logística, permitiéndoles comerciar de manera transparente y privada. Hoy en día, este consorcio ha crecido, teniendo actualmente más de 300 miembros.

El lanzamiento de Corda es comprensible desde el punto de vista de las entidades financieras, ya que normalmente entre los intereses de este tipo de organismos no están el que todas sus transacciones sean públicamente accesibles, o que todo el mundo pueda ver el software que se usa para llevar a cabo dichas transacciones, ya que en ese código puede haber procedimientos o políticas que cada entidad quiere mantener privadas, como datos relativos a tipos de intereses o comisiones.

Corda solamente contempla y está pensada para casos de uso financieros, y no es una plataforma de propósito general como lo son Ethereum o Hyperledger Fabric.

3.1.6 Quorum

Además de participar en la creación de Corda, la entidad financiera J.P. Morgan puso en marcha el proyecto llamado Quorum [28], el cual está basado en Ethereum, pero orientado en este caso a las empresas y a la creación de redes de consorcios.

Quorum ofrece una versión de Ethereum con permisos, en la que solamente entidades conocidas puedan formar parte de una red, además, añade privacidad al protocolo de Ethereum, ofreciendo la posibilidad de que los detalles de las transacciones realizadas sean privados.

Inicialmente, Quorum se creó para que las empresas financieras comenzasen a adoptar la tecnología blockchain, se fijó en una tecnología con mucho potencial como era Ethereum, y la adaptó del ámbito público al ámbito empresarial, pero su ámbito no se limita a los casos de uso financieros, sino que se puede extrapolar su uso para casos de uso genéricos.

La primera versión estable de Quorum fue lanzada en noviembre del año 2016, y, actualmente, está intentando hacerse un hueco dentro del entorno empresarial, aunque su progresión y crecimiento han sido menores que en el caso de Hyperledger Fabric.

Si bien es cierto que las redes públicas pueden ser muy útiles en diversos casos de uso empresariales, hay casos en los que se requiere un cierto grado de privacidad que en las redes públicas es más difícil de conseguir.

Una ventaja también de Quorum es que no está separada de Ethereum, sino que incorpora todos los avances y mejoras que se hacen a lo largo del tiempo en el protocolo de Ethereum.

3.1.7 Alastria

En lo relativo al ámbito español, Alastria [29] es un proyecto nacido a mediados de 2017 con el objetivo de construir una red blockchain a nivel nacional haciendo uso de la plataforma Quorum. Esta red está concebida para que puedan estar todo tipo de empresas, interactuar entre ellas y tener algunos de sus procesos integrados en ella, programándolos mediante contratos inteligentes.

Alastria todavía está en fase de construcción y experimentación, y se están considerando y ponderando los beneficios que puede tener este proyecto para las empresas. Se pretende que la administración pública también forme parte de esta red, y también que los contenidos y transacciones que se encuentren en ella tengan valor legal.

En principio, para ser miembro de esta red y poder participar en ella, habrá que pagar una cantidad determinada de dinero. Si todo avanza según lo previsto, se espera que para 2020 se pueda comenzar a usar esta red nacional.

3.2 Investigación

Hoy en día, la tecnología blockchain está más inmersa en el ámbito de la investigación que en el ámbito del desarrollo, debido en gran parte a que su estado de madurez es todavía bajo. Las áreas de investigación de esta tecnología son múltiples y muy variadas.

Debido a la variedad de plataformas blockchain, existen investigaciones centradas en la comparativa de estas plataformas. En [30] se realiza una comparativa entre Ethereum, Fabric y Corda, y en [31] también se realiza una comparativa entre estas tres tecnologías, y además se comparan los diferentes proyectos de Hyperledger. En [32] se compara Ethereum con la topología de los proyectos de Hyperledger, pero en este caso se incluye también a Bitcoin y adicionalmente se trata la relación de las redes blockchain con la red Tor. En este último caso se hace referencia a que la red Tor está pensada para que se envíen peticiones de manera privada y no para almacenar datos, como es el caso de una red blockchain.

Otro de los ámbitos en los que se centran numerosas investigaciones es en lo relativo a los diferentes sistemas de consenso que existen para que todas las máquinas mantengan la misma copia de la cadena de bloques y permanezcan siempre sincronizadas. Existen trabajos como [6], [33] o [34], en los que se analizan y comparan en base a diferentes criterios los algoritmos de consenso existentes y también otros, que aunque no existan implementaciones hechas, son pruebas de concepto. Un punto en común que comparten todos estos trabajos es que la prueba de trabajo es actualmente el algoritmo más utilizado y más popular, ya que su funcionamiento produce buenos resultados para mantener el consenso, pero su principal desventaja, como se ha mencionado ya en este documento, es la enorme cantidad de energía que se consume para tal fin, por tanto, la investigación en la búsqueda de nuevos métodos de consenso más eficientes se considera un área muy importante para el futuro de la tecnología blockchain.

Tras el lanzamiento de Ethereum, la primera plataforma blockchain de propósito general, se abrió un nuevo mundo de posibilidades para comenzar a descubrir los sectores a los que esta tecnología podría ser aplicada, además del financiero. Por ello, hoy en día existen muchos trabajos en los que se investiga y analiza el potencial de la tecnología y los diferentes casos de uso en los que se podría emplear. En [35] se investigan numerosos casos de uso en los que la tecnología blockchain podría emplearse. Centrando más el foco en la industria, existen publicaciones como [36] o [37] en las que se analiza la integración de blockchain en diferentes industrias.

Normalmente, al hablar de la integración de blockchain en entornos industriales siempre aparece el concepto de Internet of Things (IoT), o, traducido al español, Internet de las Cosas, que hace referencia a los dispositivos que tienen la capacidad de poder conectarse a una red para emitir eventos o recibir notificaciones. Últimamente, el precio de los sensores y de este tipo de dispositivos ha decrecido significativamente, lo que ha hecho que se popularicen cada vez más y se comience a investigar su potencial y utilidad. En [38] se investiga la combinación entre IoT y la tecnología blockchain, señalando que estos dispositivos son los puntos de contacto entre el mundo físico y el digital. En ella, se mencionan diversos casos de uso y se hace mención al ahorro de tiempo y dinero que esta integración podría suponer.

A continuación, se nombran y describen brevemente los casos de uso principales en los que se está investigando y experimentando su integración con la tecnología blockchain. Primero se tratarán los relativos y enfocados a entornos industriales, ya que este trabajo se centra en este tipo de casos de uso:

- **Gestión y configuración de dispositivos IoT:** Cada día existen más dispositivos IoT conectados a internet, una arquitectura tradicional cliente servidor podría ser una buena idea para gestionar algunos de ellos, pero cuando la cantidad de estos dispositivos es muy grande, esta arquitectura puede presentar limitaciones y ser un posible cuello de botella, haciendo que los dispositivos no se sincronicen correctamente. Una arquitectura descentralizada como blockchain podría ayudar a resolver este problema de cuello de botella y permitir una mejor gestión de todos estos pequeños dispositivos conectados. En [39] se habla de este caso de uso más profundamente y se construye un prototipo.
- **Sistemas logísticos o cadenas de suministro:** En la actualidad, los sistemas logísticos suelen estar centralizados en las propias empresas, lo cual provoca que el grado de transparencia relativa a la información sobre la trazabilidad de los productos sea muy bajo. Esto es debido a que todos estos datos son controlados por las propias compañías, pudiendo modificarlos y alterarlos a su manera. Debido a la falta de transparencia e integridad de los sistemas actuales, es posible privar a los clientes de la verdad, haciéndoles creer que las condiciones de su envío han sido correctas cuando no lo han sido, o que su producto es original cuando realmente es una falsificación. Con la transparencia e integridad que ofrece la tecnología blockchain, la investigación de su aplicación a este caso de uso se considera muy útil de cara a mejorar los sistemas de gestión logística actuales.
- **Plantas de producción, ensamblaje, envasado o tratado de productos:** La instalación de dispositivos IoT en estas plantas para monitorizar los procesos y el almacenamiento de los datos que emiten estos dispositivos en la cadena de bloques proporcionaría un alto grado de transparencia a la hora de hacer inspecciones o auditorías por parte de personal interno de las empresas o personal externo, ya que los datos se guardan de manera inmutable y transparente. Por tanto, ante una incidencia o una inspección, se estarían siempre auditando datos que no se han modificado ni falsificado. Adicionalmente, a la hora de vender un producto a un cliente, se le puede asegurar la calidad teniendo los datos de los procesos que se han seguido para crear el producto y de las pruebas que ha pasado antes de ser vendido guardados en la blockchain.

Una vez tratados este tipo de casos de uso, se procede a continuación a describir otro tipo de sectores o casos de uso que se podrían beneficiar de las propiedades de la tecnología blockchain:

- **Registros médicos:** Actualmente, los datos médicos de los pacientes son gestionados por las entidades médicas (hospitales o clínicas privadas, por ejemplo) y almacenados centralizadamente en ellas. Debido a esto, se dan en ocasiones situaciones en las que se roban estos datos o se accede de manera ilegal a ellos para falsificarlos. Además, existen limitaciones legales, temporales y de compatibilidad a la hora de compartir los registros médicos almacenados con otras entidades médicas, como se menciona en [40]. Proteger la red interna de estas entidades y los datos puede ser una solución para dificultar los ataques que proceden del exterior, pero un administrador interno podría seguir modificando y alterando los registros. Por ello, una solución que se propone es que los propios pacientes sean los dueños de sus datos. En [41] se propone la creación de un prototipo con el objetivo de almacenar en la blockchain los permisos que existen para acceder a determinados datos médicos, donde solamente el dueño de sus datos puede modificar esos permisos. De esta manera, son los propios pacientes los que pueden conceder acceso a sus datos médicos a quienes ellos quieran. En este caso, se propone que los registros médicos estén almacenados localmente en diferentes bases de datos, y los únicos datos que se guarden en la blockchain sean los relativos a los permisos. En ambos estudios también se propone la idea de que al ser los pacientes los dueños de sus datos sean ellos los que decidan si los desean compartir para que se puedan usar en investigaciones médicas, de esta manera, toda la sociedad se vería beneficiada.
- **Música:** Con el auge de internet en las últimas décadas, la industria de la música ha sufrido un cambio radical, teniendo que transformarse al mundo digital y sufriendo por ello enormes pérdidas económicas debido a la piratería. Una posible idea para mejorar este sector sería hacer uso de blockchain para almacenar información sobre los derechos musicales. Los contratos inteligentes serían los encargados de decidir si alguien que solicita reproducir una canción realmente tiene derechos para reproducirla.
- **Notaría:** Las agencias de notaría actúan como intermediarios para verificar y certificar la autenticidad de las partes envueltas. La tecnología blockchain podría eliminar la necesidad y los costes derivados de acudir a una notaría para realizar estas certificaciones, gracias al uso de la criptografía para validar la identidad de las partes, y también a la inmutabilidad e integridad que proporciona sobre lo que las diferentes partes hayan acordado.
- **Sistemas de votación electrónica:** El voto online es uno de los servicios más atacados por cibercriminales, con el objetivo de alterar los datos de unas determinadas elecciones. Los sistemas de hoy en día, al no estar, generalmente, distribuidos, poseen un único punto de fallo, por tanto, un pequeño error en el sistema puede exponerlo al exterior, provocando que los resultados de las votaciones puedan ser falsificados. Debido también a la centralización de estos sistemas, son propensos a recibir ataques de denegación de servicio. Una solución sería el uso de la tecnología blockchain para registrar en ella todo el proceso de votación, para hacerlo más transparente, seguro e inmutable, pero siempre protegiendo la privacidad de lo que los participantes han decidido votar. En [42] se

menciona esta problemática descrita de manera breve anteriormente y se implementa un prototipo en Ethereum.

- **Organizaciones autónomas:** Mediante el despliegue de contratos inmutables en la blockchain, los cuales son llamados y ejecutados cuando determinados eventos suceden, se pueden realizar tareas y registrar acciones y datos de manera automática. Adicionalmente, debido a la existencia de monedas digitales, estos contratos pueden hacer pagos, haciendo posible automatizar y agilizar procesos que actualmente requieren de más tiempo, documentación, firmas y personal para su gestión.
- **Prueba de existencia de documentos:** La prueba de que un documento existe es muy importante, por ejemplo, en procesos judiciales. Hoy en día los documentos importantes en papel se suelen almacenar en solamente un lugar, pudiendo ser robados o destruidos. En cuanto a los documentos digitales, también se suelen guardar en un solo lugar, pudiendo ser modificados si se accede a ellos. Por ello, una idea es almacenar una prueba de que un documento existe, pero sin la necesidad de almacenar el documento y exponerlo a todo el mundo. Para ello, se guarda un identificador (hash) del documento en la cadena de bloques, y si ese documento no se modifica, seguirá teniendo el mismo hash, en cambio, si se modifica, su hash cambia y se sabe que el documento no es el original, ya que su actual hash no coincide con el que se almacenó en la blockchain, pudiendo de esta manera demostrar que un cierto documento existió en una fecha concreta del pasado.

En ocasiones, el término industrial es un tanto ambiguo, ya que, por ejemplo, se puede hablar de la industria musical o la industria médica, pero en este trabajo, cuando se habla de industria o de entornos industriales, se centra en la fabricación y tratado de cualquier tipo de producto y de su transporte de un lugar geográfico a otro.

En lo relativo a este trabajo fin de grado, el caso de uso que se usará para la construcción del prototipo será tanto el de la logística como el de las plantas de producción, ensamblaje, envasado o tratamiento de productos. Concretamente, se tratará el caso de uso de la fabricación de mermelada, desde la importación de las materias primas, en este caso las frutas, hasta el envío de los tarros llenos de mermelada desde la fábrica hacia el establecimiento donde van a ser vendidos. Por tanto, con lo anterior en mente, se han buscado investigaciones y trabajos en los que se incluyan casos de uso similares.

En [43] se habla de la experiencia de haber construido una red blockchain privada distribuida geográficamente de cara a almacenar datos acerca de los eventos que ocurren en una cadena de suministros y de los planes de involucrar a diferentes empresas para ampliar dicha red y convertirla en una red de consorcios. Algunos de los retos que se mencionan son el riesgo que en ocasiones no quieren asumir las empresas tradicionales a la hora de adoptar una nueva tecnología, y los acuerdos que se tienen que producir entre las empresas para acordar los datos que se van a guardar, siendo por tanto la construcción de una red de consorcios un proceso lento.

En cuanto a la integración de las plantas de producción, en [44] se propone un prototipo en el que se guarde en la blockchain información desde el cultivo de las frutas y hortalizas, incluso de los proveedores de los pesticidas y herbicidas, hasta que estas se venden al consumidor final, pasando entre medias por una planta industrial para, por ejemplo, aplastar las aceitunas de cara a hacer aceite, o introducir en cajas los productos. En esta investigación se construye una red

privada tanto con Ethereum como con Hyperledger Sawtooth, mencionando que el estado de madurez de Ethereum es superior al de Sawtooth.

En cuanto a desarrollos, existen diversas empresas de reciente creación que están comenzando a ofrecer servicios de trazabilidad empleando la tecnología blockchain. Una de las más asentadas y populares es Provenance [45], cuyas tarifas dependen del tamaño de la compañía que contrate estos servicios.

Un denominador común en las investigaciones leídas suele ser que todavía se necesita mucha investigación en la tecnología blockchain, tanto para mejorar su rendimiento como para descubrir el potencial que posee, ya que es una tecnología muy reciente. En este trabajo fin de grado lo que pretendo es analizar y comparar las principales tecnologías blockchain con todos los avances que han sufrido hasta el día de hoy. Comenzar primero realizando un estudio teórico, y tras el aprendizaje de cada tecnología, pasar a la parte práctica y trabajar en la creación de un prototipo usando la tecnología real implementada por cada plataforma hasta este momento. Finalmente, tras un estudio teórico y una componente práctica, estaré en disposición de hacer una evaluación acerca del estado real de la tecnología blockchain hoy en día, centrándome siempre en su aplicación a entornos industriales.

Capítulo 4: Estudio Comparativo

Tras analizar la situación actual de la tecnología blockchain en el capítulo anterior, la conclusión es que actualmente las plataformas o tecnologías blockchain más importantes, populares y maduras que permiten crear redes de propósito general son Ethereum y Hyperledger Fabric, por tanto, serán las dos que se van a incluir en este estudio comparativo.

En lo relativo a la bibliografía consultada, además de los recursos a los que he hecho referencia hasta ahora, he utilizado para aprender y poder realizar este estudio comparativo los siguientes recursos: [46] y [47] en lo relativo a Ethereum y [48], [49] y [50] en lo relacionado con Hyperledger Fabric.

Los conceptos se introducen de manera gradual y ordenada a lo largo de todo el capítulo y también dentro del estudio de cada plataforma. Cada estudio está dividido en cinco secciones principales, dentro de las cuales puede haber subsecciones dependientes de las características de la plataforma. Las cinco secciones generales mencionadas anteriormente son las siguientes:

- **Identidades:** En esta primera sección se trata el método que usa cada plataforma para poder identificar al creador de cada transacción.
- **Estructura:** En esta sección se habla acerca de la arquitectura de las redes y de las posibles topologías que se pueden crear.
- **Ledger:** En este apartado se tratan los contenidos, estructura y almacenamiento tanto del estado actual como de los bloques.
- **Ejecución de transacciones:** Una vez que se ha hablado de cómo se identifica el origen de las transacciones, de la arquitectura de las redes y de la estructura de la *ledger*, se estudia el flujo que siguen las transacciones, desde que se crean hasta que se confirman.
- **Consenso:** Por último, se analiza el sistema que implementa cada plataforma para que todos los nodos de la red mantengan la misma versión de la cadena de bloques.

4.1 Ethereum

Para comenzar se realiza un estudio acerca de Ethereum, la plataforma blockchain para la creación de redes públicas de propósito general más importante a día de hoy.

4.1.1 Identidades

Las transacciones que se realizan siempre son originadas o creadas por alguien, por tanto, para identificar a su creador se necesitan identidades. Una identidad puede pertenecer a, entre otros, un objeto, una persona o una empresa. En Ethereum, estas identidades se conocen como direcciones, por tanto, para realizar transacciones en Ethereum se debe estar en posesión de una dirección. Cada una de ellas está compuesta por 20 bytes, y, para su generación, se siguen los siguientes pasos:

1. Se elige una clave privada de 32 bytes empleando para ello procesos aleatorios. El propietario de una clave privada firmará con ella las transacciones que realice en el futuro.
2. A partir de la clave privada se genera su correspondiente clave pública, la cual tiene una longitud de 64 bytes. Para este proceso de generación de claves se emplea el algoritmo llamado ECDSA [51] (*Elliptic Curve Digital Signature Algorithm*).
3. Una vez que se obtiene la clave pública, se emplea la función hash llamada Keccak-256 [52] para obtener un hash de dicha clave pública. Este hash tiene un tamaño de 32 bytes, del cual se seleccionan los últimos 20 bytes, que son los que componen la dirección. Por lo tanto, un ejemplo de dirección sería 0x21a5d25f3f949b357993Ee43e296c02105f731B1. La generación de una dirección a partir de una clave privada es un proceso determinista, es decir, dada una clave privada, siempre se generará la misma dirección.

En Ethereum todo el mundo puede crear una dirección y comenzar a realizar transacciones, pero, debido al hecho de que una dirección se genera a partir de un proceso aleatorio, existe la posibilidad de que se creen dos direcciones iguales, lo cual se conoce como colisión. Si una colisión llega a producirse, significa que las dos personas que poseen la misma dirección pueden hacer transacciones en nombre de la otra, y dichas transacciones serían validadas de manera correcta. Aunque es cierto que es posible que se produzcan colisiones, la probabilidad de que este hecho ocurra es ínfima, ya que, al estar cada dirección compuesta por 160 bits, existen 2^{160} direcciones posibles. Para hacerse una idea de lo grande que es este número, se estima que viven unos 7.500 millones de personas actualmente en el planeta Tierra, en el supuesto caso de que cada una de ellas tuviese 1000 direcciones, la probabilidad de que al crearse una nueva dirección se produjese una colisión sería de $5,13 \times 10^{-36}$.

4.1.2 Estructura

Una red Ethereum es una red P2P a la cual cualquier persona puede unir un nodo. Ethereum no tiene ningún sistema de permisos integrado que impida añadir nuevos nodos, estando, por tanto, pensado para la creación de redes públicas en las que no existen restricciones de acceso y en las que todo el mundo puede contribuir a su mantenimiento y gestión. Lo anteriormente mencionado no impide que no se puedan crear tanto redes privadas como redes de consorcios con Ethereum, pero, de hacerse, es conveniente que dichas redes posean un sistema de permisos o de gestión para impedir que cualquiera pueda añadir su propio nodo y sincronizar la cadena de bloques. Este sistema de permisos tendría que ser implementado por un tercero, ya que Ethereum, como se ha mencionado previamente, no lo ofrece.

Centrándose en las redes públicas, pueden existir muchas diferentes, teniendo cada una de ellas su propia cadena de bloques. Sin embargo, la más popular e importante es la red en la cual se mantiene la cadena principal de Ethereum, conocida como la red principal de Ethereum.

El propietario de una dirección determinada puede realizar transacciones con dicha dirección en diferentes redes, ya que cada red tiene su propia cadena de bloques. Las transacciones que se hacen en una red no afectan ni tienen ninguna repercusión sobre las realizadas en otras redes, siendo cada red independiente de las demás.

Al tratarse de redes P2P, cuando un nodo determinado consigue generar un bloque, lo envía a los demás nodos que forman parte de la red, los cuales intentan verificar que el bloque que han

recibido es un bloque válido, y, si dicha verificación es correcta, lo añaden a su propia copia de la cadena, la cual, de esta manera, se mantiene sincronizada entre todos los nodos.

Para poder unir un nodo a una red se hace uso de un cliente Ethereum, el cual es software que permite a una máquina conectarse a una determinada red, descargar la cadena de bloques, verificarla, y comenzar a contribuir a la red ejecutando transacciones para intentar crear bloques y hacer así que la cadena crezca. El protocolo de Ethereum [14] define el funcionamiento de una red, el comportamiento de los nodos y las reglas generales que se deben seguir para formar parte de una red Ethereum. A partir de esta especificación se han creado diferentes implementaciones de clientes Ethereum, los más populares a día de hoy son Go Ethereum, más conocido como Geth [53], el cual está implementado en Go, y Parity [54], implementado en el lenguaje de programación Rust.

4.1.3 Ledger

La *ledger*, como se ha mencionado anteriormente en este trabajo, es un concepto que hace referencia al estado actual y a la cadena de bloques. En esta sección se comienza describiendo el estado actual en Ethereum, para posteriormente analizar el almacenamiento de la cadena y la estructura interna de cada bloque.

4.1.3.1 Estado Actual

El estado actual en Ethereum está compuesto por un conjunto de cuentas, teniendo cada una de ellas su propia dirección. Considérese por ejemplo una cuenta bancaria, la cual, antes de realizar una transacción monetaria que la involucre, tendrá un determinado estado, y tras realizar la transacción tendrá otro estado en el que se refleje la nueva cantidad de dinero que queda en la cuenta. Cada cuenta de Ethereum sigue el mismo principio anterior, es decir, tiene su propio estado, y al realizarse transacciones que la involucren, dicho estado cambia. Por tanto, el estado actual de Ethereum está compuesto por el estado en el que está cada cuenta.

Existen dos tipos de cuentas, las cuentas personales (*externally owned accounts* en inglés) y las cuentas de contratos (*contract accounts* en inglés), las cuales se describen a continuación:

- **Cuentas personales:** Una cuenta personal tiene un único propietario, el cual tiene en posesión la clave privada con la que se pueden hacer transacciones desde esa cuenta, es decir, una cuenta personal está controlada por el poseedor de dicha clave privada.
- **Cuenta de contrato:** Este tipo de cuentas no están controladas por el propietario de una clave privada, sino que están controladas por un contrato inteligente. En cada una de estas cuentas reside el código de un contrato, y el estado de una cuenta cambia cuando el contrato que reside en ella es ejecutado. El lenguaje de programación más popular para la implementación de contratos inteligentes en Ethereum es Solidity [55], pero existen alternativas como Vyper [56] y otros que aún están en fase de desarrollo como Flint [57]. Todos estos lenguajes son compilados a EVM Code y el código de los contratos inteligentes desplegados en una red lo puede ver todo el mundo que tenga acceso a ella.

Cada cuenta tiene su propio estado y su propia dirección de 20 bytes. El proceso que se sigue en la generación de direcciones para las cuentas personales se ha descrito en 4.1.1, en

cambio, el proceso para las cuentas de contrato es diferente. Para crear una cuenta de contrato, una persona debe implementar un contrato inteligente y desplegarlo en una red Ethereum. Para este despliegue, esa persona tiene que realizar una transacción de despliegue, y esa transacción se tiene que hacer desde la cuenta personal de dicha persona. La dirección de una cuenta de contrato depende de la dirección de la cuenta personal de la persona que lo ha desplegado y del número total de transacciones que dicha persona haya realizado previamente al despliegue.

Las cuentas de contrato no tienen potestad para iniciar transacciones por ellas mismas. Lo que sí es posible es que una cuenta de contrato cree una transacción como consecuencia de haber recibido una desde una cuenta personal o desde otra cuenta de contrato, pero las únicas cuentas que pueden generar transacciones por ellas mismas son las cuentas personales (las transacciones son generadas por los propietarios de las cuentas). Todas las cuentas, sean del tipo que sean, pueden comunicarse entre sí, es decir, una cuenta personal puede crear una transacción destinada a otra cuenta personal o a una cuenta de contrato, esta última, a su vez, tras recibir una transacción desde una cuenta personal o desde una cuenta de contrato diferente, puede generar una transacción con destino a otra cuenta de contrato o a una cuenta personal. Las transacciones generadas por una cuenta de contrato se conocen como transacciones internas, y, a diferencia de las transacciones generadas desde una cuenta personal, no se insertan en bloques, es decir, las transacciones que se almacenan en los bloques solamente son las generadas desde una cuenta personal.

Una transacción siempre tiene un origen y un destino, representados por direcciones de cuentas. Ethereum, como se ha mencionado previamente, tiene la criptomoneda Ether integrada, por tanto, si una persona quiere enviar Ether a otra, creará una transacción en la que el origen será la dirección del donante y el destino será la dirección del recibidor, y en dicha transacción se indicará la cantidad de Ether que se quiere transferir. El resultado tras la ejecución de esta transacción será que en el estado de la cuenta del donante habrá disminuido la cantidad de Ether, y en la cuenta del destinatario habrá aumentado. El anterior ha sido un ejemplo de una transacción entre dos cuentas personales, en cambio, si quiero ejecutar la función presente en un contrato inteligente, el cual se encuentra almacenado en una cuenta de contrato, debo realizar una transacción cuyo destino sea la dirección de dicha cuenta de contrato y el origen sea la dirección de mi cuenta, en esta transacción indicaré la función que quiero que se ejecute y los argumentos que quiero pasar a esa función. Cuando esta función sea ejecutada, el estado de la cuenta de contrato se modificará. Un caso podría ser que el contrato tuviese variables globales, cuyos valores se almacenan en el estado de la cuenta de contrato, si una de las funciones del contrato actualiza el valor de una de esas variables y yo creo una transacción con el objetivo de que se ejecute dicha función, estaré provocando un cambio en el estado de la cuenta de contrato.

En lo relacionado con la información guardada en el estado de cada cuenta, los campos más importantes en una cuenta personal son los siguientes:

- **Nonce:** En este atributo se guarda el número de transacciones que se han hecho desde la dirección de la cuenta.
- **Balance:** Esta propiedad refleja la cantidad de Ether que posee el propietario de la cuenta.

En la Figura 4.1 se representa un esquema de una cuenta personal con sus campos más relevantes.

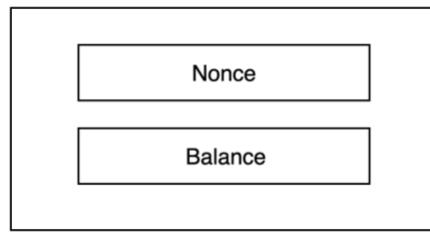


Figura 4.1: Cuenta personal

En lo relativo a las cuentas de contrato, los principales atributos que se guardan en ellas son los siguientes:

- **Balance:** Una cuenta de contrato también puede tener Ether, ya que, por ejemplo, puede enviar Ether a una cuenta personal. Esta propiedad indica la cantidad de Ether que posee la cuenta.
- **StorageRoot:** En cada cuenta de contrato se almacenan datos, por ejemplo, si un contrato se encarga de la gestión de una pastelería, los datos relativos a dicha pastelería serán almacenados en la cuenta, y cuando se ejecuten las funciones del contrato estos datos irán cambiando. En este ejemplo, esta propiedad almacenaría el hash raíz del árbol de Merkle asociado a los datos de la pastelería. Entre los datos almacenados en una cuenta de contrato, también se encuentra el código binario del propio contrato.
- **CodeHash:** Esta propiedad contiene el hash del contrato inteligente.

La Figura 4.2, en la cual H representa una función hash, muestra de manera esquemática una cuenta de contrato y la información más importante que se guarda en ella.

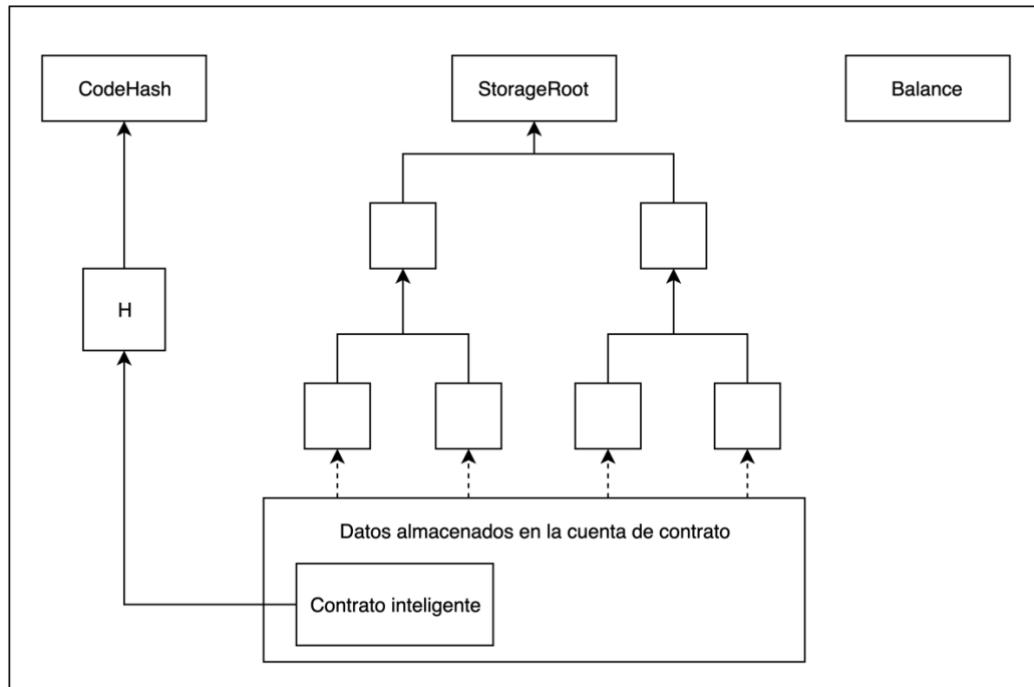


Figura 4.2: Cuenta de contrato

Por último, en la Figura 4.3 se observa una representación del estado actual de una red Ethereum, compuesto por numerosas cuentas de ambos tipos.

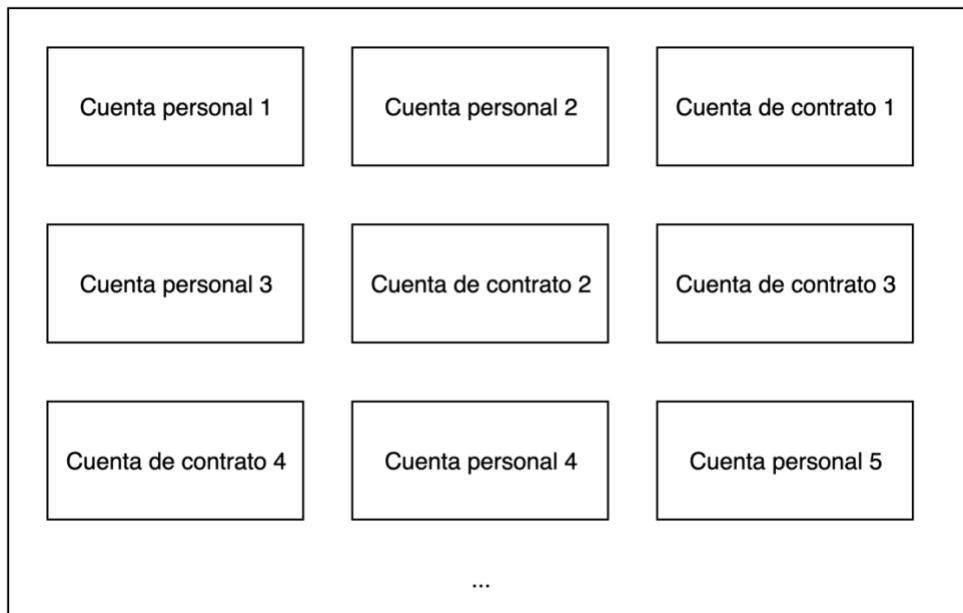


Figura 4.3: Estado actual de una red Ethereum

4.1.3.2 Cadena de Bloques y Almacenamiento

En Ethereum existen dos tipos de nodos, los llamados nodos completos (traducido del inglés *full nodes*) y los conocidos como nodos ligeros (*light nodes*).

4.1.3.2.1 Nodos Completos

Los nodos completos almacenan toda la cadena de bloques y son los únicos que pueden ejecutar transacciones. Adicionalmente a la cadena de bloques, los nodos completos también almacenan el estado actual, lo cual es útil de cara al rendimiento de las consultas, ya que, si se quiere consultar un valor a un nodo, lo mirará en el estado actual y lo devolverá, pero si no almacena el estado actual, debe hacer un recorrido por toda la cadena, desde el bloque génesis, ejecutando todas las transacciones relacionadas con ese valor para proporcionar un resultado. Si bien es cierto que esto último funcionaría correctamente, ya que el estado actual depende y es un reflejo de todas las transacciones realizadas, sería muy ineficiente, por tanto, la opción de almacenar el estado actual para mejorar el rendimiento de las consultas es sustancialmente mejor.

El estado actual es almacenado en cada nodo completo dentro de una base de datos, y, como se ha dicho previamente, cada nodo completo almacena una copia de la cadena de bloques. Las partes más importantes de cada uno de estos bloques en Ethereum son la cabecera y la lista de transacciones almacenadas en él, como se puede apreciar en la Figura 4.4.

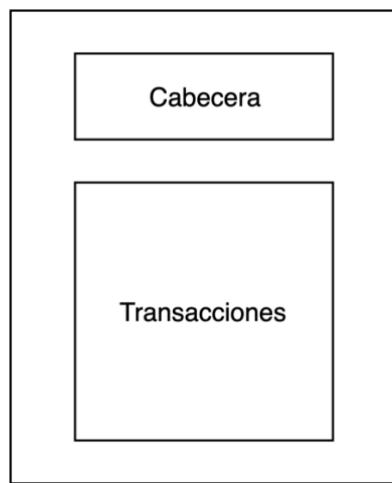


Figura 4.4: Bloque en una red Ethereum

Los árboles de Merkle juegan un papel muy importante en Ethereum, particularmente dos de ellos. Tras la creación de un bloque, la ejecución de todas sus transacciones da lugar a un nuevo estado actual. A partir de los datos de este nuevo estado actual se genera un árbol de Merkle, y también se genera otro árbol a partir de los datos relativos a las transacciones almacenadas en el bloque. Cada transacción posee sus propios datos, los cuales serán descritos en la sección 4.1.4.2. En la Figura 4.5 se muestra el árbol generado a partir del nuevo estado actual y el generado a partir de los datos de transacciones.

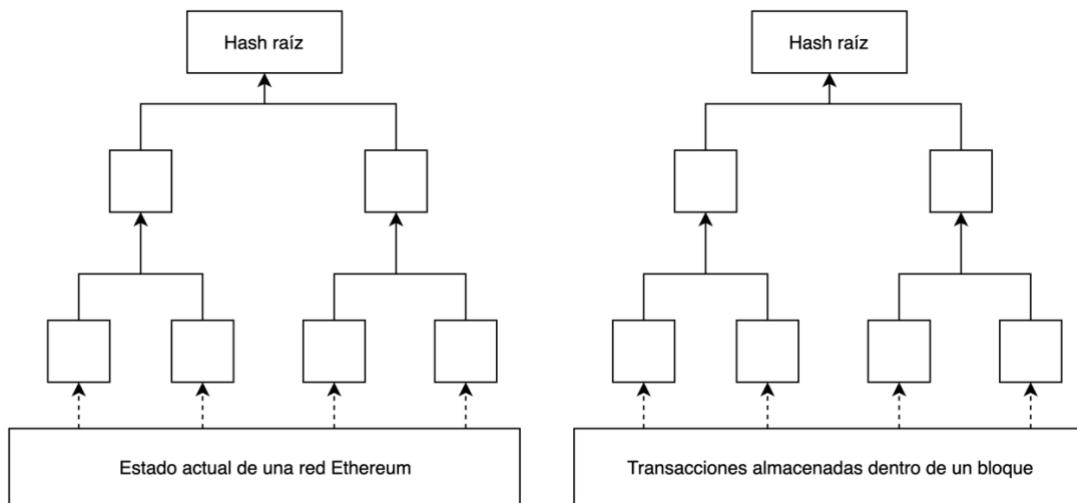


Figura 4.5: Árboles de Merkle del estado actual y de transacciones

Estos dos tipos de árboles también se almacenan en cada nodo completo. Cada vez que se genera un bloque se guardan dos nuevos árboles de Merkle, el relativo al nuevo estado actual y el relativo a las transacciones del bloque. Es decir, si la cadena en un momento dado está formada por 100 bloques, un nodo completo tendrá almacenados 100 árboles de Merkle relativos al estado actual (cada uno representando los datos que había en el estado actual tras la generación de cada bloque) y 100 árboles de Merkle relativos a transacciones. Las raíces de estos árboles se guardan en la cabecera de cada bloque, la cual es una de sus partes

fundamentales, y es lo que permite que existan los nodos ligeros. Las propiedades más importantes de la cabecera son las siguientes:

- **ParentHash:** Hash del nodo antecesor en la cadena de bloques. En Ethereum, el hash de un nodo es el hash de su cabecera.
- **Number:** Posición en la que se encuentra el bloque dentro de la cadena.
- **StateRoot:** Hash raíz del árbol de Merkle generado a partir del estado actual resultante tras haber ejecutado todas las transacciones almacenadas en el bloque.
- **TransactionsRoot:** Este atributo contiene el valor del hash raíz del árbol de Merkle resultado de partir de los datos de las transacciones contenidas en el bloque.

También existen más propiedades importantes en una cabecera de un bloque, como el *GasLimit*, *GasUsed* o las relativas a la prueba de trabajo, las cuales se tratarán en las próximas secciones.

En Ethereum, el hash de un nodo es el hash de su cabecera y cada nodo no necesita almacenar su propio hash. Para comprobar que un nodo es el sucesor de otro, se hace un hash de la cabecera del predecesor y se comprueba que dicho valor coincide con el valor almacenado en la propiedad ParentHash de la cabecera del sucesor.

4.1.3.2.2 Nodos Ligeros

La existencia de los árboles de Merkle mencionados anteriormente es lo que permite que se puedan crear nodos ligeros. Un nodo ligero sirve para almacenar en él y mantener actualizada una porción del estado actual sin tener que almacenar todo el estado actual completo, ya que este, al contener todas las cuentas, puede llegar a ocupar demasiado espacio. Recordemos que esto es exactamente lo que permiten los árboles de Merkle, validar un dato concreto sin necesidad de poseer todos los datos, por ello, cualquier nodo puede validar una porción del estado actual sin tener que poseer todos los datos del estado actual ni toda la cadena de bloques guardada. Para ello, un nodo ligero se conecta a un nodo completo o a un conjunto de ellos y se descarga solamente las cabeceras de todos los bloques de la cadena, las cuales ocupan muy poco en comparación con toda la cadena de bloques. Tras descargar las cabeceras, valida que sean correctas, y, una vez validadas, se conecta a un nodo completo para descargar los datos que quiera del estado actual y los hashes del árbol de Merkle más reciente relativo al estado actual que sean necesarios para validar que los datos que se descarga son correctos. Una vez descargados los datos y los hashes, se calcula el hash raíz del árbol a partir de ellos, y se comprueba que es el mismo que el valor almacenado en la propiedad *StateRoot* de la última cabecera descargada. En caso de que estos valores coincidan, los datos que se han descargado son los más recientes y se pueden considerar como válidos. De esta manera, en un nodo ligero se pueden almacenar los datos del estado actual que se deseen y tener seguridad de que son válidos, además, estos datos se pueden mantener actualizados, ya que, cuando se genere otro bloque, se puede repetir el mismo proceso descrito anteriormente para tener en el nodo ligero la última versión de los datos.

En un nodo ligero también se pueden almacenar y validar las transacciones que se deseen. Para validar una transacción determinada se descargan sus datos y los hashes necesarios del árbol de Merkle relativo a las transacciones del bloque donde se encuentre dicha transacción, y se sigue el procedimiento descrito previamente, pero el hash calculado se compara con el *TransactionsRoot* de la cabecera del bloque en el que se encuentre la transacción.

4.1.3.2.3 Liberación de Espacio

En un nodo ligero, una vez que se han descargado todas las cabeceras de todos los bloques y se han validado que son correctas, no es necesario seguir almacenándolas todas, por tanto, se pueden borrar las que no se vayan a utilizar para liberar espacio, ya que, con el paso del tiempo, en una red pública con una cadena muy larga, aunque las cabeceras sean de poco tamaño, si son muchas pueden ocupar una cantidad de espacio sustancial. Es decisión de la persona que tiene el nodo ligero el borrar o no las cabeceras que no necesite, y de borrarlas, decidir cuáles borrar, ya que quizás alguna la necesite para verificar determinadas transacciones antiguas.

Retornando a los nodos completos, un problema presente para ellos puede ser el del espacio de almacenamiento, ya que la cadena de bloques siempre está aumentando a medida que se realizan nuevas transacciones. Para resolver este problema, una solución posible es que, a medida que crezca la cadena, se eliminen del nodo datos antiguos como árboles de Merkle pertenecientes a antiguos bloques, incluso podrían borrarse bloques antiguos enteros. En este último caso, sería necesario que en la red hubiese un tipo de nodos conocidos como nodos de archivo (del inglés *archive nodes*), en los cuales se almacenan todos los datos de todos los bloques de la cadena, para que, en caso de necesitar obtener datos antiguos, se recurra a este tipo de nodos. Realmente, los nodos completos sí que pueden eliminar bloques antiguos, porque para crear un nuevo bloque solamente necesitan saber el estado actual completo, ya que en el nuevo bloque que van a generar necesitan insertar en su cabecera el hash que identifica al nuevo estado actual, pero no es para ello condición necesaria el tener almacenada toda la cadena de bloques.

4.1.4 Ejecución de Transacciones

En esta sección se comienza haciendo mención al concepto de gas, lo cual permite que los nodos tengan incentivos por crear bloques, tras ello, se analiza la estructura interna de las transacciones, y por último, se estudia el flujo de una transacción, desde que se realiza hasta que se confirma.

4.1.4.1 Gas

Los nodos ejecutan transacciones con el objetivo de crear bloques debido a que reciben incentivos por ello. Si estos incentivos no existiesen, los nodos no tendrían interés en emplear electricidad para contribuir al mantenimiento y crecimiento de la cadena de bloques.

En Ethereum, todo cálculo llevado a cabo debido a la ejecución de transacciones tiene una tarifa asociada, ya que conlleva un gasto energético. Esta tarifa debe ser pagada por los creadores de las transacciones, es decir, realizar transacciones en Ethereum conlleva un coste. Para pagar estos costes se utiliza la moneda integrada en Ethereum, el Ether (ETH), sin embargo, el Ether es una unidad monetaria muy alta, por ello, existen unidades menores. La más pequeña es el wei ($1 \text{ ETH} = 10^{18} \text{ wei}$), y la más común para medir costes es una unidad intermedia, el Gwei ($1 \text{ Gwei} = 10^9 \text{ wei}$).

En el lenguaje EVM Code, cada instrucción tiene una cantidad asociada de gas, dicha cantidad depende de lo exigente que sea la instrucción. Las instrucciones que más electricidad consumen tienen asociada una cantidad de gas mayor que las que consumen menos electricidad.

Cuando una transacción llega a la red, todos los nodos pueden intentar ejecutarla para incluirla en un bloque, por tanto, si un atacante enviase una transacción a la red que impidiese la

ejecución de un bucle infinito, muchos nodos podrían intentar ejecutarla, haciendo que la red se ralentizase. Para que estas acciones malintencionadas no puedan llevarse a cabo gratuitamente existen las tarifas, y, adicionalmente, también existe un límite de gas por transacción.

Al crear una transacción, su creador debe indicar el límite de gas que está dispuesto a que se gaste para ejecutar su transacción y el precio que va a pagar por cada unidad de gas empleada durante su ejecución. Por ejemplo, al generar una transacción, su creador puede decir que la máxima cantidad de gas que se puede emplear para su ejecución es de 100 unidades, y que cada una de esas unidades la va a pagar a un precio de 5 Gwei. Por tanto, el precio máximo que el creador de la transacción está dispuesto a pagar por su ejecución es de 100 unidades de gas \times 5 Gwei, es decir, 500 Gwei. Si tras ejecutar la transacción se han consumido menos de 100 unidades de gas, se devuelve el dinero restante a la cuenta del creador de la transacción. Por ejemplo, si finalmente la ejecución de la transacción consumió 70 unidades de gas, se devuelve a su creador el precio que estaba dispuesto a pagar por ellas, 30 unidades de gas \times 5 Gwei, es decir, 150 Gwei, lo cual significa que realmente el creador pagó 350 Gwei por la ejecución de la transacción. En cambio, si durante la ejecución llega un momento en el que se necesitan más unidades de gas para ejecutar todas las instrucciones requeridas por la transacción, el nodo la marca como inválida y se queda los 500 Gwei, por tanto, el creador pierde todo el dinero que pagó por la transacción y la transacción es inválida. Las transacciones inválidas son también almacenadas en la cadena.

La moneda ETH solamente tiene su valor real monetario en la red principal de Ethereum, es decir, para que una transacción sea ejecutada en la red principal se debe abonar ETH real. En cambio, en otra red Ethereum pública o en una red Ethereum privada, el valor del Ether puede ser diferente, pero no es Ether con el mismo valor monetario que el de la red principal. Cualquier persona puede crear su propia red privada de Ethereum y tener Ether en ella, pero ese Ether no tiene el valor monetario del Ether que se encuentra en la red principal.

Cada nodo completo en una red tiene una dirección asociada, la cual pertenece al dueño del nodo. Los beneficios de la ejecución de transacciones se transfieren a la cuenta correspondiente a la dirección asociada con el nodo que las ejecuta, siempre y cuando este nodo genere un bloque válido que contenga dichas transacciones y ese bloque sea añadido a la cadena. Además de las tarifas que pagan los creadores por la ejecución de transacciones, también existe un incentivo adicional para los nodos por el hecho de crear un bloque válido y añadirlo a la cadena, de esta manera, se generan y se ponen en circulación nuevas criptomonedas. Todos estos incentivos provocan que haya competencia entre todos los nodos para generar el próximo bloque de la cadena.

Como se ha mencionado previamente, cada nodo completo guarda en su propia base de datos una copia del estado actual, el cual, en una red pública, puede ocupar mucho tamaño. Ethereum, además de incentivar a los nodos para que creen bloques, incentiva a los usuarios de la red a liberar espacio del estado actual. Por ejemplo, si en el estado actual de una cuenta de contrato hay una lista con datos, y una transacción elimina ciertos elementos de ella, se devuelve al creador una parte del precio que pagó por ejecutar dicha transacción.

Por último, las tarifas solamente se pagan cuando se pretende ejecutar transacciones para modificar el estado actual, en cambio, leer datos del estado actual o de la cadena de bloques no tiene coste, ya que se leen de manera local en un determinado nodo y no se involucra a toda la red como en el caso de la ejecución de transacciones.

4.1.4.2 Estructura Interna de una Transacción

Una transacción se genera desde una cuenta personal, y consiste en un conjunto de datos firmados con la clave privada del propietario de la cuenta. En Ethereum existen dos tipos de transacciones:

- **Message call:** Este tipo de transacciones son las más comunes y sirven tanto para transferir Ether entre dos cuentas como para llamar a funciones de contratos inteligentes. Los datos más importantes que contienen este tipo de transacciones son los siguientes:
 - **Nonce:** Número de transacciones que el creador de esta transacción ha realizado.
 - **GasPrice:** Cantidad de Ether que el creador está dispuesto a pagar por unidad de gas para que su transacción sea ejecutada.
 - **GasLimit:** Cantidad máxima de unidades de gas que se pueden utilizar para la ejecución de la transacción.
 - **Value:** En caso de que se esté enviando Ether a otra cuenta, en este campo se indica la cantidad que se envía.
 - **Data:** En caso de que se esté llamando a la función de un contrato inteligente, se indica en esta propiedad el nombre de la función y los argumentos.
- **Contract creation:** Estas transacciones sirven para crear cuentas de contrato. Los datos más importantes en una transacción de este tipo son los siguientes:
 - **Init:** Este campo sirve para indicar el código del contrato inteligente que se va a almacenar en la cuenta de contrato. Al desplegar un contrato en una red se le pueden pasar una serie de parámetros iniciales, los cuales también se indican en este campo.
 - **GasPrice y GasLimit:** Cuando se inicia una cuenta de contrato, se ejecutan una serie de instrucciones para iniciar el estado actual de la cuenta, por tanto, estos campos tienen la misma función que en el caso de las transacciones de tipo *message call*.

Adicionalmente, todas las transacciones, sean del tipo que sean, tienen un origen y un destino, representados ambos por direcciones. Los datos que contiene cada transacción están firmados con la clave privada del creador, por tanto, para la verificación posterior de la firma se usa la clave pública asociada a la cuenta del creador. En Ethereum, todas las transacciones son públicamente accesibles, por tanto, todo el mundo puede ver los datos que se incluyen en todas las transacciones realizadas.

4.1.4.3 Flujo de una Transacción

Una transacción puede ser iniciada desde una aplicación por, entre otros, una persona, un dispositivo, o incluso de manera automática. Esta transacción es un conjunto de datos firmados

por su creador que en un momento dado llegarán a un nodo de la red. Para que esto pase, el creador debe establecer conexión con un nodo y enviarle la transacción.

Cuando una transacción llega a un nodo, este la envía a otros nodos, los cuales, a su vez, repiten el mismo proceso, con el objetivo de que la transacción se propague por la red y llegue al máximo número de nodos posible.

A medida que la transacción va llegando a múltiples nodos, estos pueden escogerla para proceder a su ejecución. Cada nodo escoge, de entre todas las transacciones que le han llegado y no ha ejecutado, la que quiera. Normalmente, la elección de la próxima transacción a ejecutar por parte de un nodo depende de la recompensa que le proporcione, es decir, lo común es que un nodo escoja la transacción cuyo resultado de multiplicar el *GasPrice* y el *GasLimit* sea máximo, de hecho, cada nodo las puede ordenar por ese criterio, y cada vez que quiera ejecutar una transacción escoger la primera de la lista. Esto depende del tipo de cliente Ethereum, anteriormente se mencionaron dos clientes, Geth y Parity, y en ambos de ellos, para elegir la próxima transacción a ejecutar, lo más importante a tener en cuenta es la recompensa que aporta su ejecución. Sin embargo, el protocolo Ethereum deja esta decisión en manos de los clientes, por tanto, cada cliente Ethereum lo implementará a su manera.

Antes de ejecutar una transacción, se comprueba que su firma sea válida y también se comprueba que el valor *Nonce* de la transacción sea el mismo que el valor *Nonce* de la cuenta correspondiente al origen de la transacción. Esta última comprobación es muy importante, ya que los nodos no tienen por qué ejecutar las transacciones en el orden que se realizan, pueden hacerlo, pero no suele pasar, como se ha mencionado previamente. Por tanto, en un momento dado, un nodo puede tener dos transacciones (T1 y T2) cuyo origen es la misma dirección, y la que se realizó en segundo lugar (T2) es la que más recompensa ofrece, por tanto, el nodo intentará ejecutarla, pero verá en el estado actual que el *Nonce* de la cuenta del creador de la transacción no es igual que el *Nonce* de T2, por tanto, no la ejecutará. Deberá ejecutar primero T1, ya que su *Nonce* coincide con el de la cuenta de su creador. Tras ejecutar T1, el nodo modifica el estado actual de la cuenta, incrementando en uno su *Nonce*, y ahora es cuando puede ejecutar T2. De esta manera, se evita que se ejecute primero una transacción que se realizó después de otra desde la misma dirección.

Debido a que los nodos suelen elegir las transacciones que les proporcionarán una mayor recompensa, el creador de la transacción debe encontrar un equilibrio entre ofrecer una recompensa muy alta para que su transacción se ejecute rápido u ofrecer una baja recompensa y que su transacción tarde más tiempo en ser incluida en un bloque. Es decisión del creador el precio a pagar por cada unidad de gas consumida durante la ejecución de la transacción.

A los nodos que ejecutan transacciones para intentar crear bloques se les conoce como nodos mineros, o mineros simplemente. Para que un nodo sea minero debe ser un nodo completo, pero no todos los nodos completos deben ser mineros, puede haber nodos completos que decidan solamente recibir la versión de la cadena más actualizada y mantener una copia del estado actual completo, pero sin ejecutar transacciones.

El número máximo de transacciones que un minero puede insertar en un bloque depende de la propiedad *GasLimit* de su cabecera. Esta propiedad se regula a nivel de toda la red y dicta la cantidad de gas máxima que no debe ser superada entre la ejecución de todas las transacciones incluidas en el bloque. En la cabecera de cada bloque también existe la propiedad llamada *GasUsed*, por tanto, para que un bloque se considere como válido, su propiedad *GasUsed* debe ser menor o igual que *GasLimit*.

Una vez que un minero ejecuta una serie de transacciones y actualiza el estado actual, para generar un bloque válido, debe resolver un problema computacionalmente costoso, cuyos fundamentos se describen en el apartado 4.1.5.1. Una vez resuelto ese problema de manera

correcta, ha creado un bloque. Tras ello, distribuye el bloque a una serie de nodos de la red, los cuales, al igual que ocurre con las transacciones, propagan este bloque por toda la red. Cada nodo que recibe un nuevo bloque lo intenta validar. Los dos pasos más importantes a la hora de validar un bloque son los siguientes:

1. Verificar que el problema computacionalmente costoso está bien resuelto.
2. Partiendo del estado actual del nodo, ejecutar todas las transacciones en orden, verificando sus firmas previamente, y una vez ejecutadas todas comprobar que el hash raíz del árbol de Merkle del estado actual resultante sea igual a la propiedad *StorageRoot* de la cabecera del bloque que se está validando.

En caso de que el proceso de validación sea correcto, el nodo que lo validó añade dicho bloque a su versión de la cadena.

4.1.5 Consenso

El método o algoritmo de consenso que emplea Ethereum para mantener la misma versión de la cadena de bloques en todos los nodos de la red se conoce como prueba de trabajo (traducido del inglés *proof of work*). El algoritmo de prueba de trabajo concreto que usa Ethereum se llama Ethash, el cual se describe formalmente en el apéndice J en [14]. Sin embargo, existen muchos otros algoritmos concretos de prueba de trabajo diferentes, los cuales comparten siempre la misma finalidad general, por ello, en esta sección no se describirá Ethash, sino los fundamentos de todo algoritmo genérico de prueba de trabajo, así como aspectos relacionados con su seguridad y rendimiento.

4.1.5.1 Prueba de Trabajo

Su nombre lo describe correctamente, prueba de trabajo, es decir, demostrar que una cantidad de trabajo ha sido realizado. Este algoritmo propone la resolución de un problema matemático computacionalmente costoso para poder crear un bloque válido, y que, a su vez, validar que el problema se ha resuelto correctamente sea inmediato.

Un algoritmo de prueba de trabajo, en esencia, persigue la búsqueda de un valor determinado que dependa de todos los contenidos del bloque y que satisfaga una serie de restricciones, haciendo que la obtención de ese valor sea costosa computacionalmente.

Un ejemplo de algoritmo de prueba de trabajo podría ser el intento de encontrar un valor (llamado solución), el cual se inserte en la cabecera del bloque y provoque que al realizar el hash de la cabecera se obtenga un valor que empiece por siete ceros. Para encontrar la solución, lo único que se puede hacer es probar combinaciones de posibles soluciones hasta que haya una en la que al realizar el hash de la cabecera el resultado comience por siete ceros. En este caso, la restricción es que el hash debe empezar por un número determinado de ceros, por tanto, si se encuentra una solución que satisfaga dicha restricción, se demuestra que se ha trabajado para encontrarla.

En el caso del ejemplo anterior, para validar que la prueba de trabajo de un bloque es correcta solamente se necesitaría hacer un hash de su cabecera (ya que entre sus datos viaja la solución) y comprobar que cumple con la restricción de comenzar por siete ceros.

Esta es la filosofía de un algoritmo de prueba de trabajo. Un factor importante a tener en cuenta es que las restricciones, conocidas también como la dificultad, se pueden regular, para

que sea más o menos difícil encontrar una solución. En el ejemplo anterior, si la restricción fuese que el hash comenzase por dos ceros, sería más fácil encontrar la solución. Este principio se utiliza en las redes de Ethereum para controlar a qué ritmo se generan los bloques. Si se desea que se creen más lentamente, se incrementa la dificultad, para que sea más complicado para los mineros resolver el problema, y si se quiere que los bloques se creen más rápido, se reduce la dificultad. Actualmente, en la red principal de Ethereum se genera un nuevo bloque aproximadamente cada 14 segundos.

Como se ha mencionado, la única posibilidad de encontrar una solución que satisfaga las restricciones es probar diferentes combinaciones continuamente, por lo tanto, este método favorece a los mineros que tengan una capacidad de cómputo mayor.

4.1.5.2 Seguridad

Todos los bloques mantienen la versión de la cadena más larga, es decir, la versión en la que la red en su conjunto ha empleado más trabajo para construir, lo cual proporciona un alto nivel de seguridad. Si un atacante quiere alterar una transacción que él mismo hizo en el pasado, tendría que recalcular una prueba de trabajo válida para el bloque alterado y para todos los siguientes hasta que la nueva cadena construida por el atacante sea más larga que la original, lo cual es inviable cuando toda la red en su conjunto tiene mucho más poder computacional que el atacante. Sin embargo, sí sería posible si el atacante tuviese el 51% del poder de cómputo de la red, ya que, poco a poco, su cadena iría aproximándose en tamaño a la original y llegaría un momento en el que la sobrepasase en longitud, como consecuencia de ello, todos los nodos aceptarían ahora la cadena del atacante por ser esta la más larga.

Otro aspecto importante es que un atacante no podría modificar transacciones pasadas de otras personas porque no podría generar una firma válida al no tener sus claves privadas, solo podría modificar transacciones que él realizó en un momento del pasado.

Debido a lo mencionado con anterioridad, realmente, no se puede tener la certeza absoluta de que una transacción está confirmada en ningún momento, pero sí se puede tener un nivel de certeza muy alto, ya que, cuantos más bloques se encadenen tras una transacción, más difícil será que un atacante pueda rehacer la cadena desde un momento anterior.

4.1.5.3 Rendimiento

La prueba de trabajo está enfocada a proporcionar seguridad a la cadena de bloques haciendo que sea muy costoso alterar una transacción antigua, pero para ello sacrifica el rendimiento, el cual no es bueno ni a nivel de tiempo ni a nivel de eficiencia energética.

Tener a todos los mineros de una red simultáneamente intentando resolver un problema complejo consume una cantidad de energía muy elevada. Además, el hecho de tener que resolver un problema costoso antes de poder crear bloques válidos provoca que las transacciones tarden tiempo en ser incluidas en bloques tras ser enviadas a la red. A día de hoy, la red principal de Ethereum procesa unas 10 transacciones por segundo.

4.2 Hyperledger Fabric

En la presente sección se describirán y estudiaron las características de Hyperledger Fabric, también conocido simplemente como Fabric, el cual es actualmente el proyecto más popular y consolidado de todos los proyectos Hyperledger.

4.2.1 Identidades

A diferencia de Ethereum, Fabric no utiliza direcciones para la gestión de identidades, sino que utiliza certificados digitales. Cada compañía que forma parte de una red Fabric tiene su propia o propias autoridades certificadoras (CAs) en las que confía, las cuales emiten certificados válidos con los que personas o dispositivos de las diferentes empresas se identifican en la red y consiguen permiso para acceder e interactuar con ella. Sin embargo, un punto en común con Ethereum sigue siendo que cada participante en la red tiene su propia clave privada, la cual debe mantener en secreto.

Con este sistema de gestión de identidades, es cada empresa la que controla los certificados que se emiten, por tanto, cada compañía tiene un grado de control muy alto sobre quién puede participar en una determinada red. De esta manera, si un empleado pasa a formar parte de la empresa, se le crea su propio certificado, y si un trabajador deja de formar parte de la compañía se le puede revocar el certificado para que deje de tener validez.

4.2.2 Estructura

Una red Fabric está formada por un conjunto de nodos, entre los cuales existe un tipo especial, conocidos como nodos *orderer*. En este trabajo, a los nodos *orderer* se les llamará *orderers*, y a los demás nodos se les llamará simplemente nodos.

Fabric está diseñado para la construcción de redes de consorcios, aunque también es posible crear redes privadas, en las cuales solamente habría una empresa y todos los nodos estarían controlados por ella. Sin embargo, esta sección se centra en la creación de redes de consorcios, ya que el propósito principal de Fabric son este tipo de redes, en las cuales participan varias organizaciones, cada una de las cuales contribuye a la formación y crecimiento de la red aportando sus propios nodos.

4.2.2.1 Canales

La característica principal de Fabric respecto a otras plataformas blockchain es el uso de canales, lo cual permite que en una red existan diferentes cadenas de bloques, a diferencia de lo que ocurre en una red Ethereum, donde solo existe una. En cada canal de una red Fabric existe una cadena de bloques diferente, y dentro de una red puede haber varios canales. Debido al hecho de que cada canal tiene su propia cadena, cada canal también tiene su propio estado actual.

El acceso a un canal está limitado, es decir, no por el hecho de formar parte de la red Fabric se posee acceso a un determinado canal. Cada canal puede ser accedido por todas las empresas que formen la red o solamente por un subconjunto de ellas, lo cual permite que se puedan mantener datos privados entre compañías en una misma red, ya que, como se ha dicho

anteriormente, aunque una empresa se encuentre en la red, si no tiene acceso a un determinado canal, no podrá acceder a la información que contiene.

Un canal se podría definir como una capa lógica situada encima de una capa física, siendo esta capa física un conjunto de nodos. Para explicar este concepto se considerará un ejemplo. En la Figura 4.6 se muestra un diagrama de una red Fabric en la cual existen dos canales, el canal 1 (*Can1*) y el canal 2 (*Can2*). La red está compuesta por seis nodos (*Nod1*, *Nod2*, *Nod3*, *Nod4*, *Nod5* y *Nod6*) y un *orderer*, cuya función se abordará en el apartado 4.2.4 y no se incluye en el diagrama por razones de simplicidad. Tres son las organizaciones que forman parte de la red, la organización 1 (*Org1*), la organización 2 (*Org2*) y la organización 3 (*Org3*).

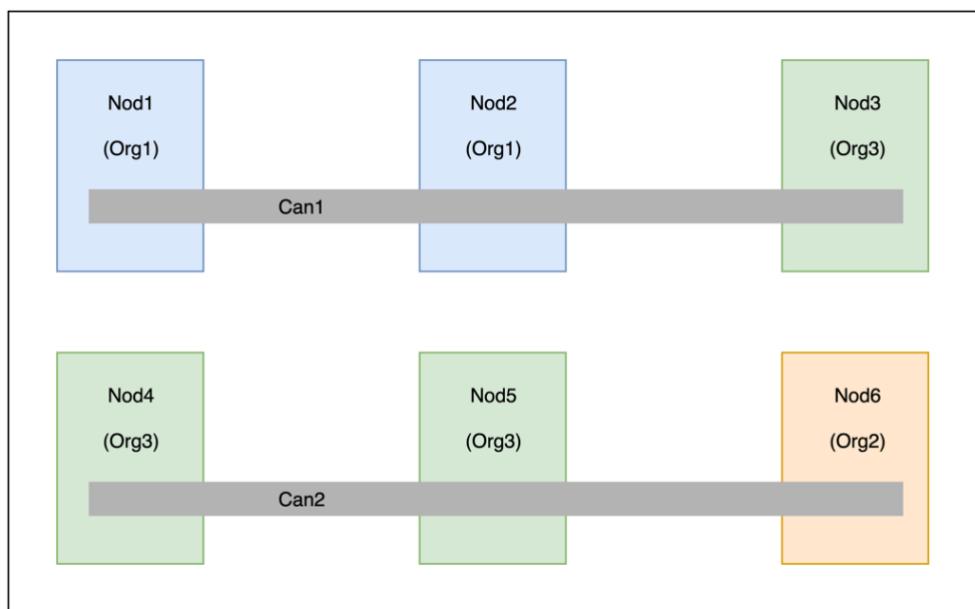


Figura 4.6: Red Fabric

Can1 está compuesto por tres nodos (*Nod1*, *Nod2* y *Nod3*). *Nod1* y *Nod2* son propiedad de *Org1* y *Nod3* es propiedad de *Org3*. A su vez, *Can2* está soportado por otros tres nodos (*Nod4*, *Nod5* y *Nod6*), donde *Org3* es la dueña de los nodos *Nod4* y *Nod5*, y *Org2* es la propietaria de *Nod6*. Con esta disposición de la red, solamente los miembros de *Org1* y *Org3* pueden acceder a la información y hacer transacciones en *Can1*. En cuanto al *Can2*, solamente miembros de *Org2* y *Org3* pueden acceder al canal y realizar transacciones en él. Tanto para leer el estado actual del canal como para realizar transacciones en un canal se debe tener un certificado válido que muestre la pertenencia a una de las organizaciones que componen el canal.

Tanto en *Nod1*, *Nod2* y *Nod3* existe la misma copia de una cadena de bloques, a su vez, en *Nod4*, *Nod5* y *Nod6* existe otra cadena de bloques diferente, es decir, en cada canal se mantiene una cadena de bloques distinta e independiente. Por tanto, los nodos que forman un determinado canal mantienen una misma versión de la cadena, la cual, junto con el correspondiente estado actual, son almacenados en cada nodo que forme parte del canal.

Cada organización que forma parte de la red puede contribuir con los nodos que quiera, en este caso, *Org1* contribuye con dos nodos, *Org2* con uno, y *Org3* con tres. Para formar parte de un canal, cada organización aporta al menos un nodo al canal, de cara a poder consultar en él los datos acerca de la cadena o del estado actual. Lo común es que cada organización consulte en sus propios nodos la información que deseé, y no que para ello acuda a nodos de otras compañías.

En el caso del ejemplo anterior, aunque *Org1* y *Org2* estén en la misma red, realmente, no se comunican entre sí, ya que no hay ningún canal en el que estén las dos, lo cual implica que ninguna de ellas puede ver los datos de la otra. De esta manera, a pesar de pertenecer las dos a la misma red, se consigue privacidad.

El estado actual de una red pública en Ethereum puede contener muchas cosas que a una persona en particular no le interesen, por tanto, para no tener que descargarse todo el estado actual para acceder a sus datos, esta persona hace uso de un nodo ligero. En cambio, en Fabric, solamente las organizaciones que pertenezcan a un canal pueden hacer transacciones en él, por ello, generalmente, ni la cadena de bloques ni el estado actual van a llegar a tener un tamaño tan grande como en una red pública. Debido a esta razón, en Fabric no se consideran los nodos ligeros como imprescindibles, pueden existir, ya que en Fabric también se hace uso de árboles de Merkle, siguiendo el mismo principio en este caso que Ethereum, pero no son comunes, ya que un nodo ligero es para guardar solamente la información que le interesa a su propietario. En Fabric, normalmente, si un nodo forma parte de un canal, el propietario de dicho nodo deseará tener toda la información de ese canal, por tanto, lo común es que se almacene en el nodo la cadena de bloques completa y el estado actual entero.

La topología mostrada en el ejemplo anterior es sencilla, sin embargo, un nodo también puede dar soporte a varios canales. En el caso tratado anteriormente, *Nod5* también podría tener una copia de la cadena de bloques y del estado actual de *Can1*, al cual seguirían perteneciendo *Org1* y *Org3*. En este caso, representado en la Figura 4.7, *Nod5* guardaría información tanto de *Can1* como de *Can2*.

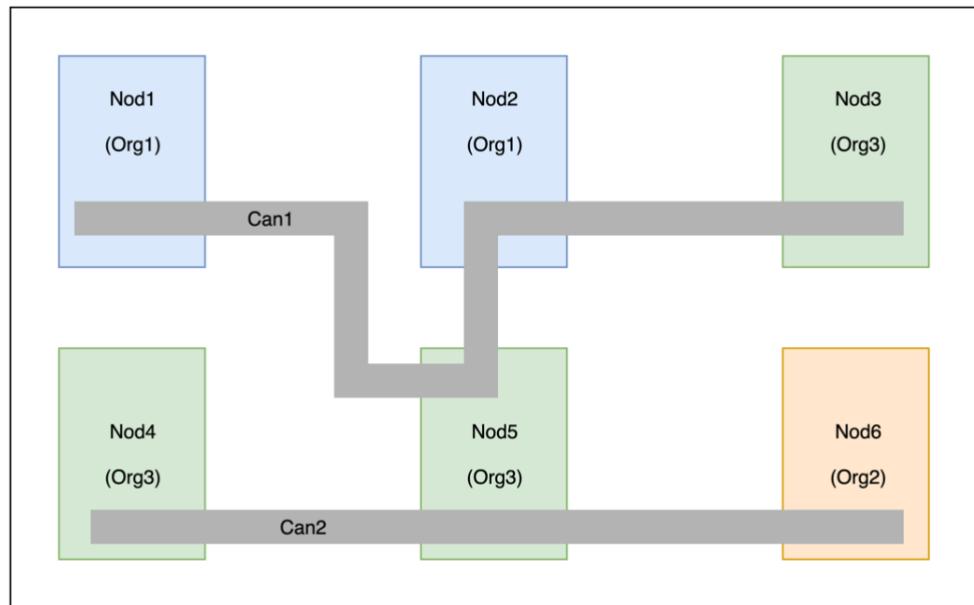


Figura 4.7: Nodo contribuyendo a dos canales

4.2.2.2 Contratos Inteligentes

En Fabric, los contratos inteligentes están desplegados a nivel de canal, esto quiere decir que el hecho de ejecutar una transacción en un contrato inteligente actualiza el estado actual del canal, del cual, al igual que la idea mencionada en Ethereum, existen varias copias físicas en cada nodo del canal, pero lógicamente es uno, ya que todas las copias son iguales. En un canal puede haber desplegados varios contratos inteligentes diferentes. Una curiosidad acerca de la

nomenclatura es que en la documentación de Fabric se emplea, además del término *smart contract*, la palabra *chaincode* para referirse a un contrato inteligente.

De cara a desplegar un contrato inteligente en un determinado canal se siguen los siguientes pasos:

1. Instalar el contrato inteligente en, al menos, un nodo del canal. Los nodos en los cuales se instale un contrato son importantes de cara a la ejecución de las transacciones, como se verá en el apartado 4.2.4.
2. Desplegar el contrato en el canal. Este proceso se conoce como instanciar el contrato en el canal. Una vez instanciado, se pueden enviar transacciones a la red para que las funciones del contrato sean ejecutadas y se actualice el estado actual del canal como consecuencia.

A día de hoy, Fabric soporta Go, JavaScript y Java como lenguajes de programación para contratos inteligentes.

4.2.2.3 Control de Acceso

De cara a controlar el acceso a una red Fabric, existen los llamados sistemas de control de acceso o sistemas de permisos, a los que Fabric denomina *Membership Service Providers*, o MSPs. Un MSP sirve para conceder o denegar permisos de acceso o de gestión de una red tras comprobar que a quién se van a conceder estos permisos ha proporcionado una firma válida y de confianza. En una red existen diferentes tipos de MSPs según su ámbito, siendo los más relevantes los MSPs de nodo y MSPs de canal.

Cada nodo tiene su propio MSP, cuya estructura se representa en la Figura 4.8 y sus contenidos más importantes se describen a continuación:

- **Administradores:** Un MSP de un nodo indica quién o quiénes son sus administradores. En el caso mencionado en la sección 4.2.2.2 relativo a la instalación de un contrato inteligente en un nodo, no todo el mundo puede hacerlo, solamente los administradores de dicho nodo. Cuando un administrador quiere, entre otras cosas, instalar un contrato en un nodo, debe realizar una petición de instalación y firmarla con su clave privada. Los certificados de los administradores se guardan en el MSP del nodo para comprobar si dicha firma es válida y proviene realmente de un administrador.
- **Identidad del nodo:** Cada nodo en una red Fabric posee su propio certificado y su propia clave privada, lo cual es de vital importancia debido al proceso de ejecución de transacciones en Fabric, el cual será analizado en el apartado 4.2.4.

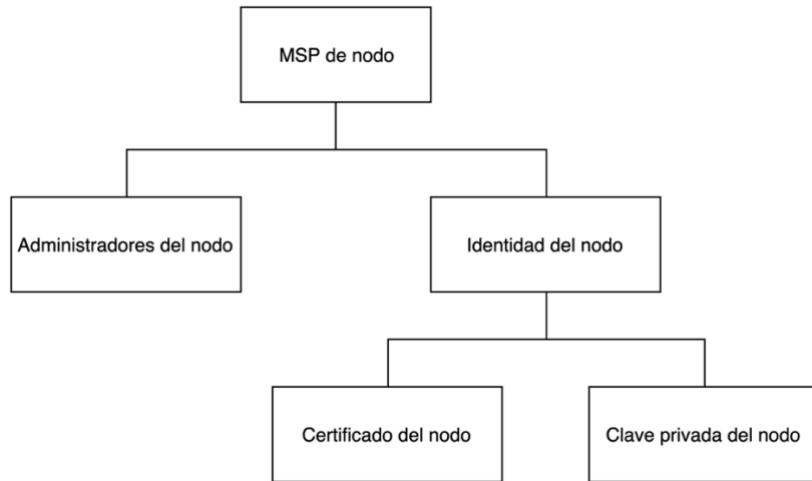


Figura 4.8: MSP de nodo

En lo relativo a un MSP de canal, su función es la de controlar quién tiene acceso a un determinado canal. La idea en este caso es similar a la de la cadena de bloques o al estado actual, lógicamente uno, pero cada nodo que forma parte del canal tiene una copia de dicho MSP. Cada canal tiene su propio MSP, cuyos contenidos principales, mostrados en la Figura 4.9, son los siguientes:

- **CAs de confianza:** Cada empresa, como se ha dicho anteriormente, confía en una o varias CAs, por tanto, un MSP de canal guarda los certificados de las CAs en las que las empresas que pertenecen al canal confían. Para conectarse a un canal de cara a, por ejemplo, leer datos de su estado actual, una persona debe hacerlo a través de un nodo concreto. Para ello, esta persona tiene que firmar una petición de conexión y presentar su certificado, en el cual se menciona la empresa a la que pertenece. Para saber si el certificado es válido, el nodo comprueba que haya sido emitido por una CA en la que la empresa a la que pertenece la persona confía. Si la firma es válida y el certificado ha sido emitido por una CA de confianza se concede acceso a la persona al canal, de lo contrario, se le deniega.
- **Administradores:** Se persigue aquí la misma idea que en el caso de los MSPs de nodo, pero en este caso son los administradores del canal, los cuales pueden ser distintos de los administradores de cada nodo individual. Retornando al caso en el que se instancia un contrato inteligente en un canal, mencionado en el apartado 4.2.2.2, tampoco lo puede realizar cualquier persona, solamente administradores del canal. Para la realización de otras operaciones como podrían ser la agregación de una nueva organización o de un nuevo nodo al canal también se necesita que sean llevadas a cabo por un administrador del canal.
- **Certificados revocados:** En cada MSP se guardan los certificados emitidos para miembros de empresas pertenecientes al canal que han sido revocados. Antes de conceder acceso, aunque la firma sea válida y el certificado provenga de una CA en la que se confía, se debe mirar si el certificado ha sido revocado o no.

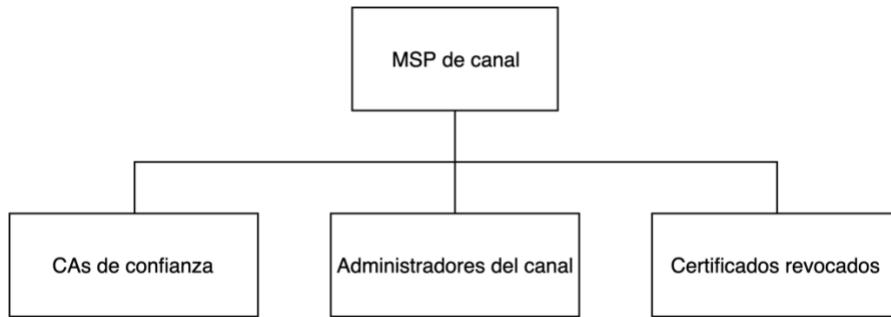


Figura 4.9: MSP de canal

Para finalizar con esta subsección, además de los mencionados anteriormente, existe también un MSP a nivel global de red, en el que una de las cosas que se definen es quién puede crear canales en la red. Una vez que un canal se crea, son los administradores de ese canal los que se encargan de gestionarlo.

4.2.3 Ledger

El concepto de ledger en Fabric es idéntico al de Ethereum, una cadena de bloques y su correspondiente estado actual. En el caso de Fabric, como se ha mencionado previamente, existe una cadena de bloques por cada canal, a diferencia de Ethereum, donde en una red solamente existe una cadena.

Una diferencia sustancial entre las dos plataformas es que en Ethereum, el código de los contratos inteligentes se almacena en el estado actual, en cambio, en Fabric no sucede esto, sino que el código solamente se almacena en los nodos donde se encuentre instalado, sin formar parte del estado actual, el cual solamente contiene los datos con los que los contratos operan.

Los contenidos principales de un bloque en Fabric también son, al igual que en Ethereum, sus transacciones y su cabecera. En cuanto a la cabecera, los campos principales que se guardan en ella son el número de bloque, el hash del bloque antecesor y un hash que identifique a todas las transacciones contenidas en el bloque. Sin embargo, a diferencia de Ethereum, no es necesario guardar datos relativos al concepto de gas ni a la prueba de trabajo, debido a que esos conceptos no existen en Fabric.

Una peculiaridad acerca de la base de datos en la que se almacena el estado actual es que es modular, es decir, pueden usarse diferentes tipos de bases de datos. Por defecto, se hace uso de LevelDB [58], desarrollada por Google, pero pueden utilizarse otras como CouchDB [59], la cual es un proyecto de Apache.

4.2.3.1 Datos Privados

Una funcionalidad importante y novedosa que ofrece Fabric es el hecho de poder tener datos privados dentro de un mismo canal. Imagíñese un canal (*Can3*) en el que están las introducidas anteriormente *Org1*, *Org2* y *Org3*, Fabric ofrece la posibilidad de que, por ejemplo, *Org1* y *Org3* puedan ver unos determinados datos en *Can3* que *Org2* no puede ver. Esto provoca que en un canal no tenga por qué haber transparencia total, ya que la transparencia completa se consigue cuando todas las organizaciones pertenecientes a un canal tienen acceso a todos los datos, sin embargo, proporciona una funcionalidad interesante para casos de uso en los que determinadas empresas no quieran que otras tengan acceso a ciertos datos.

Para conseguir mantener este tipo de datos privados dentro de un mismo canal, existen también bases de datos en los nodos diferentes a las del estado actual, con el propósito de guardar en ellas los datos privados. En el caso de *Can3*, los nodos de *Org1* y *Org3* tendrían una base de datos adicional en la que se guardasen los datos privados entre estas dos organizaciones, provocando que *Org2* no tenga acceso a dichos datos.

Las transacciones relativas a los datos privados se guardan en la cadena de bloques general del canal, pero los datos que viajan en esas transacciones no son los datos reales, sino un hash de dichos datos, de esta manera, en el caso de *Can3*, *Org2* podría ver que *Org1* y *Org3* están realizando transacciones, pero no podría ver los datos que viajan en ellas. Los datos reales se enviarían desde los nodos de *Org1* a los de *Org3* o viceversa de manera directa.

Para que una organización pueda enviar datos privados a otra directamente, necesita saber qué nodos tiene dicha otra organización en el canal. Para esto, al menos un nodo en un canal tiene que desempeñar el rol de nodo de comunicación. El hecho de que un nodo tenga este rol solamente significa que tiene información acerca de la localización de los demás nodos del canal, por tanto, cuando se quieren transmitir datos privados entre diferentes organizaciones, se consulta y se hace uso de la información guardada en los nodos que desempeñan este rol para saber la localización de los nodos a los que se van a enviar dichos datos. Como mínimo, debe existir un nodo en el canal que desempeñe este rol, sin embargo, lo común es que cada organización tenga al menos uno de sus nodos desempeñándolo.

4.2.4 Ejecución de Transacciones

La ejecución de transacciones en Fabric está compuesta por tres fases: proposición, ordenación y validación.

4.2.4.1 Fase de Proposición

Las transacciones destinadas a un canal se realizan desde aplicaciones, para cuyo desarrollo se emplea el SDK (*Software Development Kit*) de Fabric. Este SDK proporciona una serie de librerías e interfaces para interactuar con una red Fabric, por ejemplo, para realizar firmas o para establecer comunicación con los nodos.

Cada contrato inteligente desplegado en un canal posee una política de aprobación, en la cual se indican las organizaciones que tienen que aprobar una transacción antes de que pase a formar parte de un bloque de la cadena. Esta política se indica en el momento de instanciar el contrato en el canal. Solamente los nodos que tienen instalado un contrato inteligente pueden aprobar transacciones, ya que para aprobarlas necesitan ejecutar el contrato.

Para realizar una transacción en un determinado canal, en primer lugar, una persona o dispositivo, a través de una aplicación, envía una propuesta de transacción a dicho canal. A la hora de enviar esta propuesta, se deben indicar los nodos que la van a ejecutar, ya que, para que una propuesta de transacción sea finalmente aprobada, debe ser aprobada por todas las organizaciones que se encuentren en la política de aprobación del contrato. Lógicamente, se puede considerar que se envía una sola propuesta al canal, acompañada de una lista de nodos que la tienen que procesar, sin embargo, físicamente, se envía una propuesta a cada nodo que la va a ejecutar. En la Figura 4.10 se representa una propuesta de transacción, cuyos principales datos son los siguientes:

- **Canal:** En este campo se indica el canal al que se envía la propuesta.

- **Contrato inteligente:** Contrato instanciado en el canal que se quiere utilizar.
- **Función:** En esta propiedad se indica la función del contrato inteligente que se quiere ejecutar.
- **Argumentos:** Valores que se pasan como parámetro a la función.
- **Firma:** La propuesta se firma con la clave privada de la persona o dispositivo que realiza la propuesta.
- **Certificado:** El certificado del realizador de la propuesta, para que cuando llegue al canal, el MSP de dicho canal compruebe que el certificado ha sido emitido por una CA de confianza.
- **Nodo aprobador:** Nodo que va a ejecutar la propuesta.

Canal	Contrato inteligente	Función	Argumentos
Firma	Certificado	Nodo aprobador	

Figura 4.10: Propuesta de transacción

Cuando una propuesta llega a un nodo aprobador, se comprueba mediante el certificado y la firma que el realizador tiene acceso al canal, ya que cada nodo tiene una copia del MSP de canal, y, en caso afirmativo, el nodo ejecuta la propuesta.

Para que una organización apruebe una propuesta, debe ejecutar en uno de sus nodos la función del contrato inteligente que se indica en la propuesta, como consecuencia de esta ejecución, se produce una modificación en el estado actual, la cual es provisional, y únicamente será definitiva si la propuesta es finalmente aprobada por todas las organizaciones indicadas en la política de aprobación del contrato.

Tras enviar una proposición de transacción a un canal, la aplicación se mantiene a la espera hasta obtener las respuestas de todos los nodos que están ejecutando la propuesta. La estructura de una respuesta se muestra en la Figura 4.11, y sus contenidos se describen a continuación:

- **Cambios en el estado actual:** Cambios que se han producido en el estado actual como consecuencia de la ejecución del contrato inteligente. En estos cambios se reciben los datos del estado actual que han cambiado debido a la ejecución del contrato, tanto sus valores antes de la ejecución como sus valores posteriores a ella.
- **Firma del nodo:** Cada respuesta está firmada con la clave privada del nodo que ha ejecutado la propuesta
- **Certificado del nodo:** La respuesta se acompaña del certificado del nodo que ha ejecutado la propuesta, de cara a poder ser validada en la fase de validación.

Cambios en el estado actual	Firma del nodo	Certificado del nodo

Figura 4.11: Respuesta de un nodo ante una propuesta

Esta primera fase concluye cuando la aplicación ha recibido todas las respuestas de los nodos a los que ha solicitado la ejecución del contrato.

Una situación que se puede dar es que los cambios del estado actual reflejados en cada respuesta recibida no sean los mismos, lo cual puede ser debido a que los nodos que han ejecutado la propuesta no tuvieran en ese momento la misma copia de la cadena de bloques, debido a que todavía se estuviesen sincronizando. En este caso, la aplicación puede volver a solicitar que los nodos vuelvan a ejecutar la propuesta.

Para que una propuesta sea finalmente confirmada, los cambios en el estado actual de todas las respuestas recibidas deben de ser los mismos, si ocurre esto, la aplicación construye una transacción, cuya estructura se representa en la Figura 4.12, en la que se incluyen los datos de la propuesta y todas las respuestas recibidas por parte de los nodos.

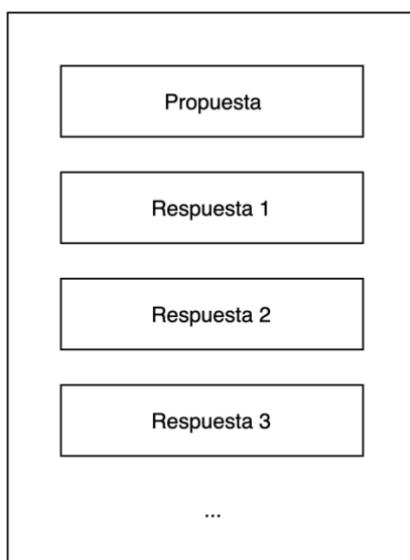


Figura 4.12: Estructura de una transacción en Fabric

4.2.4.2 Fase de Ordenación

En esta segunda fase es donde aparecen los nodos de tipo *orderer*. En una red Fabric existe un componente llamado servicio de ordenación, el cual puede estar compuesto por uno o más *orderers*.

Tras formar una transacción, la aplicación la envía al servicio de ordenación, el cual se encuentra continuamente recibiendo transacciones desde diferentes aplicaciones, y, a medida que las recibe, las va insertando en bloques. El número de transacciones que se insertan en un bloque depende de la configuración del canal.

En cuanto al orden en el que las transacciones son insertadas en bloques, no tiene que ser necesariamente ni el mismo en el que se realizaron ni el mismo en el que llegan al servicio de ordenación, lo importante en este caso es que se insertan en un orden determinado dentro del bloque. Dicho orden es de vital importancia de cara a la posterior fase de validación.

Un aspecto a tener en cuenta es que los *orderers* no leen el contenido de las transacciones, solo se encargan de insertarlas en bloques, y, cada vez que se crea uno de ellos, se envía a todos los nodos del canal correspondiente, pudiendo un servicio de ordenación dar soporte a más de un canal.

4.2.4.3 Fase de Validación

A los nodos les llegan bloques procedentes del servicio de ordenación, pero el resultado de ejecutar las transacciones que contienen todavía es provisional, y, para que pase a ser definitivo, los bloques deben ser validados, para lo cual se deben validar todas y cada una de las transacciones que se encuentran en ellos.

El proceso de validación de cada transacción comprende los siguientes pasos:

1. En primer lugar, se comprueba que los cambios en el estado actual de cada respuesta contenida en la transacción sean los mismos, en caso de que haya discrepancias en este paso, la transacción no se valida, por tanto, aunque una aplicación envíe una transacción al servicio de ordenación a pesar de haber recibido respuestas distintas de los nodos, esta no será validada.
2. Tras ello, se comprueba que haya sido aprobada por todas las organizaciones incluidas en la política de aprobación del contrato. Para ello, se comprueba que todas las respuestas contenidas en la transacción posean firmas válidas y que los nodos que las firmaron pertenezcan a las organizaciones adecuadas.
3. Si se cumplen los dos casos anteriores, el nodo comprueba que en su estado actual real, los datos a los que afecta esta transacción que se está validando no han sido modificados por otra transacción que se ha validado previamente. Si esta comprobación es correcta, se actualiza el estado actual del nodo y la transacción se marca como válida, en caso contrario, la transacción se mantiene en el bloque pero se marca como inválida, sin producir cambios en el estado actual.

Cuando se valida un bloque y se actualiza el estado actual de manera definitiva, se puede notificar a la aplicación desde la cual se realizó la transacción, diciéndole si ha sido validada o invalidada, para que la aplicación, por ejemplo, muestre un mensaje al usuario.

Es importante resaltar que en esta última fase no se ejecutan contratos inteligentes para validar las transacciones, sino que se validan según los cambios en el estado actual que estas indican, por tanto, los contratos no tienen que estar instalados en todos los nodos, únicamente en los que vayan a aprobar transacciones.

Por último, no es necesario que todos los nodos de un canal estén conectados al servicio de ordenación, con que algunos de ellos lo estén es suficiente, ya que estos recibirán los bloques del servicio de ordenación y los propagarán al resto de nodos del canal. En el caso de que a un nodo no le llegue un bloque o de que note su falta debido a un problema de conexión, puede solicitar al servicio de ordenación que se lo vuelva a enviar o puede conectarse a otro nodo para que se lo haga llegar.

4.2.5 Consenso

El proceso de ejecución de transacciones en Fabric es lo que provoca que haya consenso entre todos los nodos de cada canal, ya que, para que una transacción pase a formar parte de la cadena de bloques de manera correcta, debe haber sido aprobada por las organizaciones adecuadas y posteriormente validada en todos los nodos, los cuales siguen las mismas reglas de validación.

En lo relacionado al servicio de ordenación, existen diferentes topologías para su implementación. Una de ellas se conoce como *Solo*, la cual se suele usar en desarrollo, en la que el servicio de ordenación solamente está compuesto por un *orderer*. Para usos en producción se

suele emplear una topología maestro-esclavo, haciendo uso de Raft o Kafka, en la que las transacciones lleguen al *orderer* maestro y este las distribuya a los *orderers* esclavos para que sean ellos los que se encarguen de insertarlas en bloques. Esta topología se recomienda para producción debido a que, si un líder sufre un problema, uno de los esclavos se convierte en el nuevo líder y el servicio de ordenación puede seguir funcionando correctamente, en cambio, si solamente hay un *orderer* y sufre un problema, el servicio de ordenación deja de funcionar y se para la generación de bloques.

Para terminar este estudio de Hyperledger Fabric, hacer mención a su rendimiento, el cual depende en cierta medida de lo rápidas que sean las conexiones entre las aplicaciones y la red y entre los componentes de una misma red. Otro factor a tener en cuenta en el rendimiento es el tamaño del servicio de ordenación en comparación con el volumen de transacciones que se producen, ya que si en una red se producen muchas transacciones y el servicio de ordenación cuenta con pocos *orderers*, se encontrará muy congestionado y será un cuello de botella debido a que no será capaz de insertar las transacciones en bloques al mismo ritmo que estas van llegando.

4.3 Resumen

En la Tabla 4.1 se muestra, a modo de resumen de este capítulo, una comparativa entre las principales características de las dos plataformas analizadas.

	Ethereum	Hyperledger Fabric
Orientación de diseño	Redes públicas	Redes de consorcios
Gestión de identidades	Direcciones	Certificados digitales
Sistema de permisos integrado	No	Sí
Método de consenso	Prueba de trabajo	Política de aprobación y validación
Criptomoneda integrada	Ether	No
Posibilidad de datos privados integrada	No	Sí
Lenguaje de programación para contratos inteligentes	Solidity	Go, JavaScript, Java
Diferentes cadenas de bloques en una misma red	No	Sí

Tabla 4.1: Comparativa entre Ethereum y Hyperledger Fabric

Capítulo 5: Propuesta

En el presente capítulo, tras obtener el conocimiento de lo que ofrece cada plataforma mediante el análisis llevado a cabo en el capítulo anterior, se describe el caso de uso industrial concreto que se va a emplear a la hora de realizar el prototipo, para posteriormente mencionar sus características principales.

5.1 Descripción del Caso de Uso

La propuesta para la integración de la tecnología blockchain en el sector de la industria es la realización de un prototipo funcional acerca de la producción y el transporte de mermelada. En este caso, se trata la elaboración de este alimento en grandes cantidades, y no de manera casera o artesanal.

La materia prima principal para la producción de mermelada es la fruta, la cual hay que transportarla desde el lugar de cultivo hasta la planta industrial o fábrica donde la confitura vaya a ser elaborada.

En el prototipo propuesto se pretende recoger el proceso que comienza con el envasado de las frutas y finaliza con la llegada a las tiendas de los tarros de mermelada, donde los consumidores finales los pueden adquirir. En la Figura 5.1 se representa de manera gráfica este proceso completo, el cual es descrito a continuación:

1. Para comenzar, los proveedores cultivan y recogen la fruta, para posteriormente envasarla en lotes de diferentes tamaños. Este proceso de envasado se realiza en las plantas de envasado de los propios proveedores.
2. Una vez que las frutas se encuentran envasadas, se deben transportar hasta la planta o plantas industriales de la empresa productora de mermelada, de lo cual se encarga una compañía de transportes. En el caso de este prototipo, esta compañía es independiente, es decir, ni los proveedores ni la propia productora de mermelada se encargan del transporte.
3. Posteriormente a la recepción de los lotes, estos se desembalan para extraer la fruta y poder comenzar a tratarla. Tras diferentes procesos, entre los cuales se encuentran la trituración y cocción de la fruta, y la agregación de la cantidad de azúcar adecuada, se obtiene el producto final, el cual se deposita en tarros de diferentes capacidades. Dichos tarros son etiquetados con la información necesaria acerca de los ingredientes, fechas de elaboración y de caducidad o procedencia de la fruta con la que se ha realizado el alimento.
4. Los tarros llenos de mermelada se agrupan también en lotes, de cara a ser distribuidos y enviados a diferentes tiendas y supermercados. De nuevo, en el caso concreto de este prototipo, la función de transporte la realiza una empresa independiente de la empresa productora de mermelada y de los propios establecimientos de venta.
5. Una vez que los lotes de tarros llegan a los puntos de venta, o bien se guardan en el almacén para su posterior venta o se ponen a disposición del consumidor directamente.

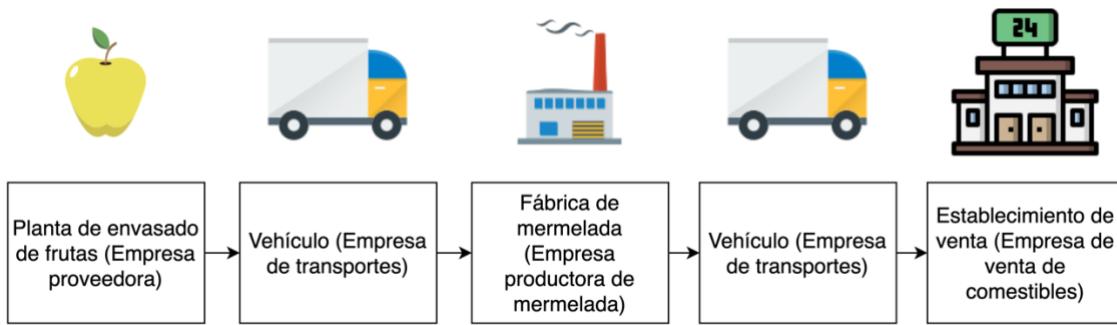


Figura 5.1: Cadena de producción y suministro

Cada proveedor puede poseer una o varias plantas dedicadas al envasado. En caso de contar con superficies de cultivos muy amplias y repartidas geográficamente quizás necesite tener varias plantas, en cambio, si el proveedor es una empresa pequeña o familiar y su zona de cultivo está concentrada en una región, puede que solamente posea una planta de envasado. En este tipo de plantas no solamente se pueden envasar contenidos, ya que, dependiendo de la compañía y de los acuerdos con la empresa destinataria, quizás se necesiten lavar o pelar los frutos antes de ser enviados. Además, previamente a su envasado, la fruta debe ser seleccionada para evitar en la medida de lo posible enviar frutos en mal estado.

En lo relativo a la empresa de producción de mermelada, en este prototipo solamente se considerará una empresa, pero podrían ser más. De hecho, cada una de ellas podría no ser una empresa que se dedicase exclusivamente a la elaboración de confituras, sino también de otros alimentos o bebidas como zumos. Por tanto, estas empresas podrían tener diferentes subsecciones dedicadas a cada tipo de consumible que produzcan. Además, estas empresas podrían ser de diversos tamaños y tener varias plantas de producción. En el caso de una multinacional podría contar con diferentes fábricas distribuidas por diversos países, sin embargo, una empresa nacional o local podría tener una sola fábrica en la que se elaboren todos los alimentos.

Los conceptos mencionados previamente también se aplican a los establecimientos que reciben los tarros de mermelada para venderlos. Es decir, una gran empresa como una cadena de supermercados nacional o internacional posee diferentes establecimientos en diversas ciudades, en cambio, un negocio llevado por una persona o familia quizás solamente cuente con una pequeña tienda para la venta de cara al público.

Para finalizar esta subsección, en cuanto a las empresas de transporte, al igual que en los casos anteriores, también pueden ser de mayor o menor tamaño, contando con más o menos vehículos, sin embargo, como se ha mencionado previamente, son independientes de los proveedores y de los destinatarios, es decir, son otras compañías las que contratan sus servicios para transportar una determinada cantidad de productos desde un origen hasta un destino. Por último, la empresa que se encarga del transporte de las frutas puede ser la misma o distinta de la que transporta la mermelada.

5.2 Características del Prototipo

En esta sección se mencionan las funcionalidades que presentará el prototipo y se deciden y argumentan las plataformas y el tipo de redes blockchain que se van a utilizar para su construcción.

5.2.1 Funcionalidad

En el prototipo a elaborar se pretende recoger y almacenar los datos relativos a la cadena de producción y suministro descrita anteriormente. De este modo, se tendrían almacenados estos datos de manera inmutable, distribuida y segura, proporcionando transparencia y confianza tanto entre las empresas colaboradoras como a los consumidores finales.

Poseer los datos relativos a la trazabilidad almacenados de este modo permite la comprobación del origen de los alimentos de forma verídica, reduciendo de esta manera las falsificaciones o engaños. Además, en el caso de que un conjunto de alimentos tenga algún tipo de sustancia nociva o haya ocurrido algún problema durante su cosecha o elaboración, se dispondría en todo momento de los datos relativos a su producción y transporte en la cadena de bloques, sabiendo por tanto cuales de ellos se deben dejar de utilizar o de vender y cuales son aptos para el consumo.

En un entorno real, los datos recogidos procederían de transacciones realizadas desde dispositivos IoT colocados en plantas industriales o en vehículos de transporte. Por ejemplo, a la hora de envasar una serie de frutas en un lote, un dispositivo colocado en la planta emitiría una transacción en la que indicaría el momento y lugar en el que se ha producido el lote, además de diversos datos acerca del propio lote. En el caso de este prototipo, se simularán los dispositivos IoT por software, pero las transacciones recogidas y los datos almacenados son equivalentes a los de un entorno real, en el cual, los administradores u otras personas de cada empresa también podrían tener un cuadro de mandos conectado a una o varias redes blockchain de cara a recibir eventos cuando determinadas transacciones se producen, actualizándose su interfaz ante la recepción de dichos eventos. De esta manera, personal de cada empresa tendría acceso visual en tiempo real a las transacciones que se van realizando y a los datos almacenados en el estado actual.

Adicionalmente al almacenamiento de datos, se pretende también que esta aplicación pueda almacenar pruebas de existencia de documentos, ya que, durante los procesos de transporte de mercancías o de fabricación de productos o comestibles se produce mucha documentación, tanto digital como en papel. Estos documentos en ocasiones se guardan de forma centralizada en almacenes o en repositorios digitales, y si ocurre algún problema en el lugar físico (como una inundación o un incendio) o en el repositorio digital, se pueden dañar o perder. Además, una persona con malas intenciones podría modificar estos documentos o sustituirlos por otros distintos. Por tanto, se considera útil que este prototipo ofrezca la posibilidad de guardar en la cadena de bloques pruebas de existencia de documentos, para que sea posible detectar y comprobar si han sido modificados desde su creación.

Por lo tanto, el prototipo a construir estará orientado a ofrecer las siguientes funcionalidades principales:

- Simular transacciones producidas por dispositivos IoT.
- Ofrecer un cuadro de mandos al personal de las empresas en el que se refleje el estado actual y las transacciones que se producen en tiempo real.
- Proporcionar la posibilidad de registrar pruebas de existencia de documentos generados en un momento determinado del pasado.

5.2.2 Plataformas Empleadas

En los procesos descritos anteriormente se aprecia que son varias las empresas involucradas (proveedores de fruta, productora de mermelada, compañías de transporte y establecimientos de venta). Por ello, una red de consorcios parece una posible buena solución para implementar un prototipo con estas características, ya que este tipo de redes ofrecen privacidad entre empresas, sin necesidad de que los datos sean públicamente accesibles, y, a su vez, proporcionan un alto grado de transparencia y confianza entre ellas, ya que todas tienen acceso a los mismos datos y ninguna puede modificar o falsificar datos pasados. Por tanto, para el almacenamiento de los datos se diseñará y construirá una red de consorcios de la que formen parte estas empresas que colaboran entre sí.

De cara a registrar pruebas de existencia, se decide optar en este prototipo por una red pública, ya que el caso es distinto al anterior, en el cual, todo el mundo que tenga acceso a los datos puede leerlos, por tanto, si se almacenan en la red pública, cualquier persona podría hacerlo. Pero en el caso de las pruebas de existencia, lo que se guarda en la red pública son identificadores de documentos, de los cuales, aunque puedan ser públicamente accesibles, nadie podrá identificar su significado o el documento original al que representan. Este tipo de identificadores ocuparán muy poco espacio (256 bits cada uno), lo cual, como se verá a continuación, supone una ventaja en cuanto al almacenamiento y las tarifas a pagar.

Por tanto, el prototipo a realizar estará conectado tanto a una red de consorcios como a una red pública. Para la construcción de la red de consorcios se utilizará Hyperledger Fabric y para la red pública se hará uso de Ethereum, ya que son las dos plataformas más populares y maduras a día de hoy. A través de cada una de ellas se implementará la funcionalidad concreta que se cree más adecuada tras su estudio teórico, guardando más datos y de manera no pública con Fabric, y almacenando muy pocos datos accesibles por todo el mundo pero sin significado aparente para ellos gracias a Ethereum, en la cual, almacenar muchos datos no es una buena idea, ya que las instrucciones de almacenamiento, llamadas SSTORE, consumen mucho gas. Por lo tanto, si se decidiesen almacenar datos relativos a la trazabilidad en Ethereum, además de que el rendimiento a día de hoy es bajo (10 transacciones por segundo), el precio de ejecutar instrucciones de almacenamiento sería muy alto. Para ponerlo en contexto, en general, el gas que consume una instrucción de SSTORE (la cual guarda una palabra de 256 bits en disco) es de 20.000 unidades, en comparación con las instrucciones ADD o AND, las cuales consumen 3 unidades de gas cada una.

Capítulo 6: Análisis

En el capítulo actual se elabora la documentación relativa al análisis del prototipo, a partir de la cual se realizará posteriormente su diseño.

6.1 Definición del Sistema

La definición general del sistema se ha realizado en el capítulo anterior, sin embargo, en él no se ha tratado el alcance ni los límites concretos del prototipo, a lo cual se hace mención en la siguiente subsección.

6.1.1 Determinación del Alcance del Sistema

El prototipo que se va a realizar tendrá un ámbito nacional, concretamente, estará centrado en el noroeste de la Península Ibérica. Las empresas que formen parte del sistema pueden ser de mayor o menor tamaño, multinacionales o pequeñas empresas, pero o bien poseen plantas u operan en el noroeste de España.

Se decide comenzar a desarrollar el prototipo en una superficie geográfica pequeña en lugar de a nivel completamente nacional, continental o global debido a que se considera que, de cara a la investigación de la tecnología, es más conveniente empezar dando un paso pequeño, evaluar la experiencia, y, si ha ido bien, dar pasos adicionales en el futuro, quizás extendiéndolo hacia zonas geográficas más amplias.

Por ello, este prototipo permite recoger una hipotética experiencia en la que diferentes empresas deciden colaborar entre ellas para mejorar sus procesos y relaciones, haciendo uso de la tecnología blockchain para ello. Sin embargo, para empezar a experimentar con esta nueva tecnología, deciden comenzar a pequeña escala, con solamente algunas de sus plantas, establecimientos o vehículos. Una vez que evalúen la experiencia, quizás decidan extender el prototipo a una aplicación real e integrar más plantas, medios de transporte (aéreos o acuáticos) o procesos (nuevos sabores de mermelada o nuevos productos), o, por el contrario, puede que la experiencia no haya sido buena y decidan intentar mejorar sus procesos de otra manera. Sin embargo, para saberlo, se considera necesaria la elaboración de un primer prototipo funcional y la evaluación de los resultados derivados de su construcción y funcionamiento.

El prototipo estará orientado a la elaboración de mermelada de dos sabores concretos (manzana y fresa), y para ello, se considerará la colaboración entre 7 organizaciones diferentes, las cuales se listan y tratan a continuación:

- **Empresa proveedora de manzanas:** Empresa que se encarga del cultivo, recolección y envasado de manzanas. En el supuesto de este sistema, esta empresa tendrá tierras cultivadas en los alrededores de Lugo (Galicia) y de Luarca (Asturias), y en cada una de estas localidades poseerá una planta de envasado de manzanas.

- **Empresa proveedora de fresas:** Similar a la anterior, pero en este caso se cultivan y envasan fresas. En este prototipo, esta empresa tendrá también dos plantas de envasado, una situada en Tineo (Asturias) y otra en Ponferrada (Castilla y León).
- **Empresa productora de mermelada:** Esta compañía será la encargada de elaborar mermelada tanto de manzana como de fresa. Para ello, compra la materia prima a las dos empresas proveedoras mencionadas previamente. En el presente prototipo, se supone que esta empresa solamente posee una planta de producción o fábrica de mermelada, la cual está situada en Oviedo (Asturias).
- **Cadena de supermercados:** Esta compañía representa a una cadena de supermercados que posee varios establecimientos de venta en diferentes poblaciones. En esta aplicación, esta empresa participará con un supermercado situado en Ribadesella (Asturias) y otro en Santander (Cantabria).
- **Empresa de transporte de manzanas:** Empresa dedicada al transporte de manzanas desde las plantas de envasado de los proveedores de manzanas a la fábrica de mermelada.
- **Empresa de transporte de fresas:** Similar a la empresa anterior, pero relativo al transporte de las fresas.
- **Empresa de transporte de mermelada:** Compañía que transporta los lotes de tarros de mermelada desde la fábrica de mermelada hasta los diferentes supermercados que participen en este prototipo.

Para transportar cada producto (manzanas, fresas y mermelada) se ha decidido emplear una empresa diferente, las cuales no se dedican exclusivamente a estos envíos, sino que son empresas generales que se contratan cuando se necesitan, pero que también participan en el prototipo, y lo hacen con 2 vehículos cada una (un camión y una furgoneta). Como se ha mencionado, la cantidad de vehículos, plantas industriales o establecimientos con los que cada empresa participa en el prototipo no es grande, con el objetivo de centrarse en la tecnología y no dificultar la realización y gestión del prototipo debido a la inclusión de muchos participantes. También, a modo de recordatorio, en el caso por ejemplo del proveedor de manzanas, el cual participa en la aplicación con dos plantas de envasado, podría tener más plantas repartidas por España o Europa, o podría tener solamente esas dos, lo importante es que participa en la aplicación con dos plantas. Este concepto se aplica también a las demás empresas, lo importante no es su tamaño real, sino solamente su presencia en el prototipo.

Por último, la nomenclatura definida anteriormente es genérica, debido a que se ha decidido no utilizar ningún nombre comercial de empresas reales en este trabajo fin de grado, ya que es una investigación que no depende de ni está condicionada por ninguna empresa.

6.2 Requisitos del Sistema

En esta sección se identifican los requisitos del sistema, los actores que van a interactuar con él, y los casos de uso.

6.2.1 Requisitos Funcionales

En este apartado se muestra el resultado de la recolección de los requisitos funcionales, los cuales se incluyen en Tabla 6.1. Estos requisitos han sido refinados tras realizar el modelado 6.4 y analizar los casos de uso 6.5.

Id	Nombre	Descripción
RF.1	Seleccionar identidad	El sistema debe permitir elegir la identidad con cuya clave privada se firmarán las transacciones realizadas.
RF.2	Creación de lotes	El sistema debe permitir realizar transacciones para crear lotes.
RF.2.1	Contenido de lotes	Los lotes creados podrán contener una única opción de las mostradas a continuación: <ul style="list-style-type: none"> • Frutas de un único tipo. Siendo los tipos de frutas contemplados los siguientes: <ul style="list-style-type: none"> ○ Manzanas ○ Fresas • Tarros de mermelada
RF.2.2	Propiedades de lotes	Cada lote deberá disponer de las siguientes propiedades: <ul style="list-style-type: none"> • Identificador • Creador • Destinatario • Estado • Contenido • Masa • Fecha de creación • Fecha de envío • Fecha de entrega • Fecha de utilización
RF.2.3	Estado de los lotes	Un lote podrá permanecer en uno de los siguientes estados: <ul style="list-style-type: none"> • Almacenado • Enviado • Recibido • Utilizado
RF.3	Envío de lotes	El sistema debe permitir realizar transacciones para reflejar el envío de lotes.
RF.4	Entrega de lotes	El sistema debe permitir realizar transacciones para reflejar la entrega de lotes.
RF.5	Uso de lotes	El sistema debe permitir realizar transacciones para

		reflejar el uso de los contenidos de los lotes. La fábrica de mermelada usa las frutas para elaborar mermelada y los establecimientos de venta usan los tarros para venderlos.
RF.6	Registrar prueba de existencia de documentos	<p>El sistema debe permitir registrar pruebas de existencia de documentos relativos a la entrega de mercancía. Cada prueba de existencia constará de:</p> <ul style="list-style-type: none"> • Un identificador o hash del documento • Fecha en la que se registró el documento • Identidad que ha registrado el documento
RF.6.1	Comprobación del registro de documentos	El sistema debe permitir comprobar si un determinado documento ha sido registrado en algún momento del pasado.
RF.7	Cuadro de mandos	El sistema debe proporcionar un cuadro de mandos en el que se reflejen los cambios en el estado actual en tiempo real.
RF.7.1	Actualizar cuadro de mandos	El cuadro de mandos solamente se actualizará mediante la recepción de eventos.
RF.8	Privacidad	El sistema debe impedir que las empresas proveedoras de manzanas tengan acceso a los datos relativos a los lotes de fresas y viceversa.
RF.9	Permisos	El sistema debe proporcionar un sistema de permisos para gestionar quién posee acceso a los datos y que impida a ciertas identidades realizar determinadas transacciones.
RF.9.1	Permisos de creación de lotes	El sistema solamente debe permitir la creación de lotes de fruta a los proveedores de fruta y la creación de lotes de mermelada a los productores de mermelada.
RF.9.2	Permisos de envío de lotes	El sistema únicamente debe permitir realizar transacciones de envío de lotes a identidades que pertenezcan a empresas de transporte.
RF.9.3	Permisos de entrega de lotes	El sistema solamente debe permitir la realización de transacciones de entrega a identidades que pertenezcan a empresas de transporte.
RF.9.4	Permisos de utilización de lotes	El sistema únicamente debe permitir a identidades que pertenezcan a destinatarios de lotes realizar transacciones de utilización de lotes.
RF.9.4.1	Otros destinatarios no pueden usar lotes	Un destinatario no puede utilizar lotes dirigidos a otros destinatarios diferentes.
RF.9.5	Permisos de lectura	Los datos relativos a la cadena de producción y suministro pueden ser leídos por todas aquellas

		identidades que tengan acceso a ellos.
RF.9.6	Permisos de registro de documentos	El sistema solamente debe permitir a administradores de las empresas de transporte registrar documentos de entrega de mercancía.
RF.9.7	Permisos para comprobar la existencia de un documento	El sistema debe permitir a cualquier persona comprobar si un determinado documento de entrega ha sido registrado y ha existido en el pasado.
RF.10	Interfaz de usuario	El sistema debe proporcionar una interfaz de usuario mediante la cual se puedan realizar transacciones de registro de documentos y se puedan simular transacciones de dispositivos IoT.

Tabla 6.1: Requisitos funcionales

6.2.2 Requisitos No Funcionales

En la Tabla 6.2 se muestran los requisitos no funcionales.

Id	Nombre	Descripción
RNF.1	Tecnología blockchain	El sistema debe hacer uso de la tecnología blockchain para el almacenamiento y la gestión de datos.
RNF.1.1	Tecnología Fabric	Los datos relativos a la cadena de producción y suministro deben gestionarse a través de Hyperledger Fabric.
RNF.1.2	Tecnología Ethereum	Los datos relativos a las pruebas de existencia de documentos deben gestionarse mediante la plataforma Ethereum.
RNF.2	Firma digital	Previamente a la ejecución de una transacción, esta debe haber sido firmada con la clave privada del realizador.
RNF.2.1	Claves privadas	Cada participante o identidad que forme parte del sistema debe tener su propia clave privada.

Tabla 6.2: Requisitos no funcionales

6.2.3 Identificación de Actores del Sistema y Casos de Uso

A continuación se identifican tanto los actores que pueden interactuar con el sistema como sus funciones dentro de él:

- **Administradores de las empresas participantes:** Estos actores poseen permisos para configurar el sistema y son los únicos que tienen permitido registrar documentos.
- **Identidades válidas emitidas por CAs:** Personas o dispositivos en posesión de identidades válidas emitidas por ciertas CAs tendrán permiso de lectura sobre determinados datos relativos a la parte del sistema realizada con Fabric. Además, dependiendo del tipo de identidad, tendrán permiso para realizar unas transacciones u otras. No todo dispositivo tiene necesariamente que poseer una identidad, sino que una identidad puede usarse para representar a un conjunto de dispositivos, que es, como se verá más adelante, lo que se hará en este prototipo. Otra posibilidad es que el dispositivo pertenezca a una persona (por ejemplo, al conductor de un vehículo), y las transacciones realizadas desde ese dispositivo se firmen con la clave privada de dicha persona. En un entorno real, estos dispositivos existirían físicamente, sin embargo, en este prototipo se simulará su comportamiento, lo cual no impide que se consideren actores del sistema. Dichos dispositivos se encuentran situados en plantas industriales, zonas de carga de lotes, vehículos o en zonas de descarga de lotes, y cuando determinadas condiciones se cumplen, emiten una transacción o un evento que en el futuro se transforme en una transacción. En la sección 5.2.1 se ha mencionado un ejemplo acerca de un posible funcionamiento durante la generación de un lote. De cara a, por ejemplo, enviar un lote, este puede estar identificado con una etiqueta especial o un código, y al introducirlo en un vehículo, un lector de estas etiquetas o códigos colocado en el propio vehículo lo identificaría y provocaría que se emitiera una transacción de envío de un lote. Existen diversas posibilidades a la hora de realizar una implementación real con dispositivos IoT, pero como norma general, en todas ellas los lotes deberían estar identificados de forma inequívoca y deberían existir dispositivos colocados en diferentes lugares capaces de identificarlos y así provocar que se produzcan las transacciones adecuadas. Cada identidad emitida por una CA tiene un certificado y una clave privada asociados, si dicho certificado está caducado o revocado, la identidad se considera inválida. En este prototipo, la idea es que cada planta industrial, vehículo y establecimiento tendrá su propia identidad, perteneciendo cada identidad a una empresa. Por ejemplo, cada uno de los 2 establecimientos de venta tendrá su propia identidad, y esas dos identidades pertenecerán a una misma empresa (la cadena de supermercados). Cada una de estas identidades, como se ha indicado previamente, representará a un conjunto de dispositivos, por ejemplo, en el caso de la identidad correspondiente a la planta de envasado que tiene la empresa proveedora de manzanas en Lugo, representará a todos los dispositivos IoT que se encuentren situados en esta planta, por tanto, cada transacción que se haga desde esa planta (originada por el dispositivo que sea) estará firmada por la misma clave privada (la relativa al certificado de la planta industrial). Este concepto relativo a la gestión de identidades en este prototipo se explicará más a fondo en la sección relativa al diseño (7.2), pero se considera necesario y conveniente mencionarlo de forma resumida aquí para poder comprender mejor el análisis.
- **Cualquier individuo:** A la hora de leer datos relativos a las pruebas de existencia en la parte del sistema realizada con Ethereum, cualquier persona podría hacerlo, ya que son públicamente accesibles.

Una vez identificados los actores y la manera en la que pueden interactuar con el sistema, se muestran a continuación los diagramas de casos de uso. El análisis de los casos de uso identificados en esta sección se realizará posteriormente en el apartado 6.5.

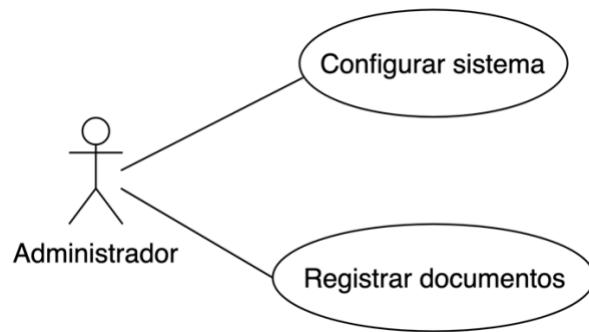


Figura 6.1: Casos de uso relativos al administrador

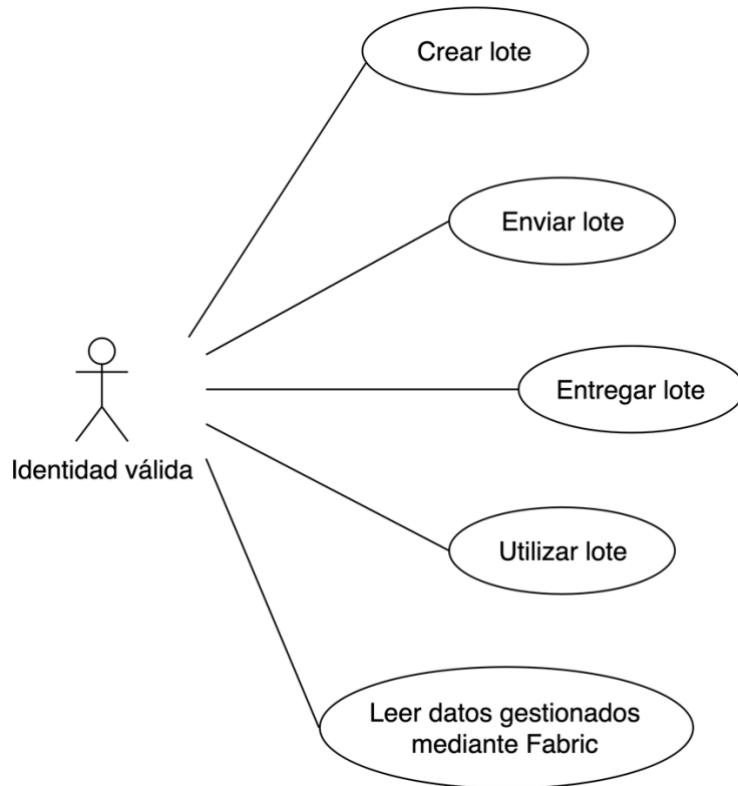


Figura 6.2: Casos de uso relativos a una identidad válida

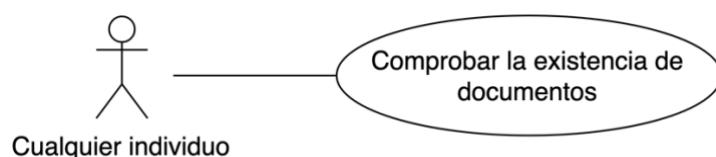


Figura 6.3: Casos de uso relativos a cualquier individuo

6.3 Identificación de Subsistemas

El sistema que se va a construir posee, como se ha mencionado, dos partes un tanto diferenciadas, cada una de ellas haciendo uso de una plataforma diferente y centrándose en un propósito también distinto. Sin embargo, ambas partes se complementan y aprovechan las ventajas que ofrece cada plataforma de cara a que el prototipo posea todas las funcionalidades requeridas. A continuación se mencionan los dos principales módulos o subsistemas que componen el sistema completo:

- **Subsistema relativo a los datos de la cadena de producción y suministro (Subsistema Fabric):** Este subsistema estará desarrollado mediante Hyperledger Fabric, y en él se gestionarán los datos relativos a la creación, envío, entrega y uso de lotes, pero no los relativos a ningún documento. Por lo tanto, cuando se haga referencia a los datos de la cadena de producción y suministro o a los datos de lotes se está haciendo referencia a este subsistema. En él se gestiona el almacenamiento de los datos concretos de cada lote, como su fecha de envío, su masa o su estado.
- **Subsistema relativo a la gestión de documentos (Subsistema Ethereum):** Ethereum será la plataforma que se utilice para el desarrollo de este subsistema, mediante el cual se gestionan los datos relativos a la existencia de documentos, pero en él no se trabaja con datos relativos a lotes ni a la cadena de producción y suministro explícitamente (como masas, estados, fechas de envío o creadores de los lotes), sino solamente con identificadores de documentos. Los documentos reales (los cuales sí contienen datos acerca de la cadena de producción) se guardarían siempre fuera de la cadena de bloques (quizá en una sala si son documentos en papel o en una base de datos o repositorio si son digitales), pero dicho lugar no depende de este subsistema. Al hacer mención a la gestión de documentos o a pruebas de existencia se estará haciendo referencia a este subsistema. Por último, a lo largo de una cadena de producción y suministro real se generan numerosos documentos, sin embargo, en este subsistema se experimentará con un tipo de ellos, los relativos a la entrega de lotes.

6.4 Modelado Preliminar

En este apartado se elaborará el modelo preliminar de los datos que se almacenarán en el estado actual de cada plataforma y de los contratos inteligentes que se desplegarán.

En blockchain no existe el concepto de clase explícitamente, sino que existen contratos inteligentes que poseen funciones, las cuales operan sobre unos datos que se almacenan en el estado actual. El concepto de clase no es muy diferente al de un contrato inteligente, ya que ambos tienen una serie de operaciones parametrizables que actúan sobre unos datos. Sin embargo, la diferencia principal se encuentra en el estado. En el paradigma orientado a objetos, una clase se instancia tantas veces como se deseé para crear los objetos necesarios, y es cada uno de dichos objetos los que tienen su propio estado en cada momento. En cambio, en blockchain existe un estado asociado a cada contrato, el cual se deriva de todas las transacciones que se han ejecutado sobre dicho contrato. El estado actual de una red está compuesto por el estado de cada contrato inteligente que se encuentra desplegado en ella. Esta idea se podría intentar ver equivalente a que en el paradigma orientado a objetos, el estado de una clase fuese

el estado en el que se encuentran todos los objetos instanciados a partir de esa clase, lo cual no es realmente así, ya que una clase no tiene estado, solamente los objetos.

Por ello, se decide en este trabajo no realizar la documentación a nivel de clases, y realizarla a nivel de estado actual y de contratos inteligentes, ya que, debido a las características de la tecnología blockchain, se considera que es la manera que mejor se ajusta. Aunque, si el lenguaje de programación en el que se programan los contratos inteligentes permite la creación de objetos, el estado actual podría contener datos relativos a objetos serializados.

6.4.1 Estado Actual

Previamente en el presente documento (apartado 5.2.2) se han mencionado las razones por las cuales almacenar muchos datos en Ethereum no es conveniente.

Aunque en Fabric no existe el concepto de gas y no hay que pagar altas tarifas por almacenar datos, tampoco es común guardar muchos datos en el estado actual de una red. Esto es debido a que en una red Fabric suelen participar varias empresas, y son ellas las que se encargan de mantener las copias del estado actual. Por tanto, si se guardan muchos datos, estas bases de datos crecerán cada vez más a medida que se produzcan nuevas transacciones y su mantenimiento será cada vez más caro para estas compañías, ya que tendrían que aumentar el espacio de almacenamiento de sus nodos cada poco tiempo mediante la adquisición de nuevo hardware. Además, normalmente, al producirse operaciones entre empresas suelen aparecer conflictos de intereses. En este caso, podrían darse a la hora de decidir los datos que se almacenan, ya que una compañía podría querer almacenar datos de interés para ella en el estado actual, pero otra no tiene interés en ver ni mantener esos datos, ya que ocuparían espacio adicional en sus nodos, lo cual le supondría un coste económico que la empresa considera innecesario.

Debido a estas razones, en este prototipo se pretende que los datos almacenados sean los imprescindibles para conseguir trazabilidad hasta el origen. La intención es guardar la menor cantidad de datos posible, pero sin que ello impida obtener la trazabilidad relativa a cada lote. Por tanto, en este caso es importante encontrar el equilibrio entre poseer la información necesaria acerca de un lote y que esta ocupe lo menos posible.

Para ello, se muestran y describen en la Tabla 6.3 los datos que se guardarían acerca de cada lote.

Campo propuesto	Propósito
Id	Identificar de forma única cada lote.
Creador	Indicar quién ha creado el lote.
Destinatario	Indicar a quién va dirigido el lote.
Estado	Indicar el estado en el que se encuentra el lote en cada momento. Existirán 4 posibles estados: Almacenado, enviado, recibido y utilizado.
Contenido	Describir el contenido de un lote.
Masa	Indicar la masa, medida en kilogramos (kg) que posee el lote.

Fecha de creación	Indicar la fecha en la que el lote fue creado.
Fecha de envío	Indicar la fecha en la que se envió el lote.
Fecha de entrega	Reflejar la fecha en la que el lote fue entregado a su destinatario correspondiente.
Fecha de utilización	Indicar la fecha en la que el destinatario utilizó los contenidos del lote.

Tabla 6.3: Datos relativos a cada lote

Estos datos se cree que son suficientes para navegar hacia atrás en la cadena de producción y suministro. Para exemplificar esta navegabilidad hacia atrás, si se desea saber el origen de la fruta usada para elaborar la mermelada contenida en unos tarros puestos a la venta en una determinada fecha se seguirían estos pasos:

1. Dado un lote lleno de tarros de mermelada, se necesitaría observar su fecha de creación y su creador.
2. Con esta información, se deberían de buscar los lotes de frutas cuyo destinatario sea el mismo que dicho creador y que hayan sido utilizados en dicha fecha para elaborar mermelada, y, una vez encontrados, se miraría el campo creador de estos lotes de frutas para saber el proveedor del cual provienen los ingredientes con los que se ha elaborado la mermelada.

En el caso mencionado anteriormente se supone que los tarros de mermelada se rellenan el mismo día que se elabora la mermelada. Adicionalmente, para saber la empresa de transporte que ha hecho los envíos y entregas, se podría consultar el histórico de transacciones de la cadena relativa a los lotes correspondientes, para ver quién ha firmado las transacciones de envío y entrega. De esta manera, se sabe quién ha hecho qué a lo largo de toda la cadena de producción y suministro, de forma verídica y transparente, ya que los datos registrados son inmutables y no se almacenan de forma centralizada.

En cuanto a los datos guardados en Ethereum, la intención es también almacenar, para cada prueba de existencia, la menor cantidad de datos posible, pero al mismo tiempo que sea posible comprobar que un documento dado existió en un momento determinado del pasado. Con este objetivo, se proponen en la Tabla 6.4 el conjunto de datos a almacenar para cada prueba de existencia.

Campo propuesto	Propósito
Hash del documento	Identificar el documento de forma única tras hacer un hash de sus contenidos.
Fecha de registro	Indicar la fecha en la que se registró el documento, para que a la hora de comprobar su existencia se sepa cuando fue registrado.
Registrador	Persona que se encargó de registrar el documento.

Tabla 6.4: Datos relativos a cada prueba de existencia

6.4.2 Contratos Inteligentes

De cara a desarrollar este prototipo se elaborarán 2 contratos inteligentes, uno de ellos se desplegará en una red Fabric de consorcios y otro en una red Ethereum pública. Ambos contratos poseerán funciones a las que haya que invocar para realizar transacciones y producir como consecuencia modificaciones en el estado actual.

En el caso del contrato desplegado en la red Fabric, deberá tratar con lotes, y contener todas las operaciones requeridas a lo largo de la cadena de producción y suministro. Con esta intención, se propone inicialmente el siguiente modelo de contrato, descrito en la Tabla 6.5.

Operación propuesta	Descripción
Crear lote	Función que permite realizar una transacción de creación de un lote para añadir un nuevo lote al estado actual.
Enviar lote	Función para realizar una transacción de envío y modificar el lote correspondiente en el estado actual.
Entregar lote	Función para provocar una transacción de entrega y modificar el lote almacenado en el estado actual.
Utilizar lote	Función a la cual invocar para producir una transacción de utilización de un lote, a través de la cual se actualiza el lote.
Obtener lotes	Función de lectura del estado actual que permite recuperar un conjunto de lotes.
Obtener lote	Función que busca y lee del estado actual un lote.

Tabla 6.5: Análisis del contrato a desplegar en la red Fabric

En lo relativo al contrato que se desplegará en Ethereum, deberá ofrecer la posibilidad de registrar pruebas de existencia de documentos, y también de poder comprobar si un documento está registrado. Para ello, se proponen las operaciones mostradas y descritas en la Tabla 6.6.

Operación propuesta	Descripción
Registrar documento	Función que permite registrar un documento y añadir una prueba de existencia al estado actual.
Comprobar existencia	Función que se encarga de leer del estado actual para comprobar si un determinado documento ha sido registrado en algún momento del pasado, para de esta manera confirmar que existió.

Tabla 6.6: Análisis del contrato a desplegar en la red Ethereum

6.5 Análisis de Casos de Uso y Escenarios

A continuación se procede al análisis de cada caso de uso identificado en la sección 6.2.3.

Configurar sistema	
Precondiciones	Cada administrador debe poseer una identidad válida de administración. En el caso de Hyperledger Fabric, estas identidades serán emitidas por una entidad certificadora, y, en lo relativo a Ethereum, se generará una dirección de administración para cada administrador.
Poscondiciones	La configuración del sistema se actualizará para reflejar los cambios producidos.
Actores	Los administradores del sistema serán los únicos que puedan configurarlo.
Descripción	<p>Los administradores podrán configurar tanto la parte del sistema realizada con Hyperledger Fabric como la realizada con Ethereum.</p> <p>En lo relativo a Fabric, tendrán permiso para configurar y actualizar la red añadiendo los siguientes elementos:</p> <ul style="list-style-type: none"> • Participantes con identidades válidas • Canales • Organizaciones • Nodos <p>Adicionalmente, podrán instalar, actualizar e instanciar contratos inteligentes.</p> <p>En el caso de Ethereum, serán los encargados del despliegue del contrato inteligente en la red pública.</p>
Escenarios secundarios	<ul style="list-style-type: none"> • Escenario alternativo 1: El elemento que se está intentando añadir a la red Fabric ya existe. <ul style="list-style-type: none"> ○ Reintentar el proceso cambiando la nomenclatura del nuevo elemento.

Excepciones	<ul style="list-style-type: none"> • Límite de gas insuficiente: La transacción de despliegue del contrato en Ethereum no ha podido completarse debido a que requiere más gas del límite indicado para su ejecución. <ul style="list-style-type: none"> ○ Notificar del error al administrador, el cual deberá aumentar el límite de gas permitido para que la transacción pueda ejecutarse completamente. • Fondos insuficientes: La transacción de despliegue del contrato en Ethereum no ha podido enviarse debido a que no se dispone de suficiente Ether para pagar la tarifa que supone su ejecución. <ul style="list-style-type: none"> ○ Notificar al administrador de que los fondos de la cuenta son insuficientes, por lo tanto, deberá adquirir suficiente Ether para que la dirección de administración desde la cual se realice el despliegue pueda costear la transacción.
--------------------	---

Tabla 6.7: Caso de uso de configuración del sistema

Registrar documentos	
Precondiciones	Para registrar un documento en una red pública de Ethereum, el administrador deberá tener creada una cuenta personal de administración y estar en posesión de su clave privada. También deberá conocer la dirección de la cuenta en la que se encuentra desplegado el contrato y el nombre y argumentos de la función implementada en dicho contrato que permite registrar un documento.
Poscondiciones	Se añade la prueba de existencia del documento al estado correspondiente al contrato desplegado en la red. En el estado de este contrato estarán almacenadas todas las pruebas de existencia registradas.
Actores	Únicamente los administradores del sistema podrán registrar documentos.
Descripción	<p>Para registrar un determinado documento digital se deben seguir estos pasos:</p> <ol style="list-style-type: none"> 1. Realizar un hash de sus contenidos. 2. Enviar una transacción a la red para invocar a la función situada en el contrato inteligente que se encarga de registrar un documento. <p>La transacción enviada debe estar acompañada del hash del documento a registrar, la fecha en la que se registra y la persona encargada del registro.</p>
Escenarios secundarios	<ul style="list-style-type: none"> • Escenario alternativo 1: El documento a registrar está

	<p>en papel.</p> <ul style="list-style-type: none"> ○ El documento se deberá digitalizar mediante algún procedimiento como el escaneado. Una vez digitalizado, se procederá a calcular su hash.
Excepciones	<ul style="list-style-type: none"> ● Límite de gas insuficiente: El límite de gas es menor que el gas necesario para ejecutar la función de registro del documento. <ul style="list-style-type: none"> ○ Notificar al administrador para que pueda aumentar el límite de gas permitido. ● Fondos insuficientes: La cuenta desde la que se realiza el registro no dispone de suficiente Ether para pagar la tarifa. <ul style="list-style-type: none"> ○ Notificar al administrador de tal hecho, el cual deberá insertar más Ether en la cuenta para que se pueda registrar el documento desde ella.

Tabla 6.8: Caso de uso de registro de documentos

Crear lote	
Precondiciones	El creador del lote deberá estar en posesión de un certificado válido emitido por una CA de confianza. Además, solamente certificados pertenecientes a proveedores de fruta o a productores de mermelada tendrán permiso para crear lotes.
Poscondiciones	Se añade un lote al estado actual del contrato desplegado en la red Fabric al que se ha invocado. En una red Fabric puede haber varios canales, y en cada canal puede haber desplegados uno o varios contratos inteligentes. De hecho, el mismo contrato puede estar desplegado en diferentes canales. El estado del lote será “almacenado”, ya que tras crear los lotes, se almacenan en un determinado lugar antes de ser enviados a su destino.
Actores	Identidades válidas pertenecientes a proveedores o a productores de mermelada podrán crear lotes en la red Fabric.
Descripción	El creador invocará la función de creación de un lote presente en un determinado contrato desplegado en la red Fabric. Al realizar esta invocación, se enviarán los datos necesarios para dar valor a los campos del lote que se creará.
Escenarios secundarios	<ul style="list-style-type: none"> ● Escenario alternativo 1: Ya existe un lote con el mismo identificador que el que se pretende crear. <ul style="list-style-type: none"> ○ El nuevo lote no se crea y los campos del lote existente no sufren modificaciones.
Excepciones	<ul style="list-style-type: none"> ● No se poseen permisos de creación de lotes: La

	<p>creación del lote no es realizada por una identidad correspondiente a un proveedor ni a un productor de mermelada.</p> <ul style="list-style-type: none"> ○ La transacción no se ejecuta y por lo tanto el lote no se crea.
--	---

Tabla 6.9: Caso de uso de creación de un lote

Enviar lote	
Precondiciones	La transacción de envío deberá producirse por alguien que posea un certificado válido emitido por una CA de confianza. Solamente certificados pertenecientes a empresas de transporte tendrán permiso para enviar lotes. Además, para que un lote se pueda enviar, este debe estar en estado “almacenado”.
Poscondiciones	El estado del lote deberá cambiar a “enviado”, y se deberá dar valor al campo relativo a la fecha de envío.
Actores	Identidades válidas pertenecientes a empresas de transporte podrán enviar lotes en la red Fabric.
Descripción	Se deberá llamar a la función de envío presente en un determinado contrato desplegado en la red Fabric. Al realizar esta invocación, se enviará la fecha en la que se envía el lote.
Escenarios secundarios	<ul style="list-style-type: none"> • Escenario alternativo 1: El lote que se pretende enviar no existe. <ul style="list-style-type: none"> ○ No se envía ningún lote y el estado del contrato permanece intacto.
Excepciones	<ul style="list-style-type: none"> • No se poseen permisos de envío: El envío del lote no es realizado por una identidad correspondiente a una empresa de transportes. <ul style="list-style-type: none"> ○ La transacción no se ejecuta y el lote no se envía, permaneciendo sus campos inalterados.

Tabla 6.10: Caso de uso de envío de un lote

Entregar lote	
Precondiciones	Una transacción de entrega debe realizarse por alguien que posea un certificado válido emitido por una CA de confianza. Únicamente certificados pertenecientes a compañías de transporte podrán entregar lotes. Adicionalmente, para que un lote pueda ser entregado, debe estar en estado “enviado”.
Poscondiciones	El estado del lote deberá cambiar a “recibido”, y se deberá dar valor al campo relativo a la fecha de entrega.

Actores	Identidades válidas pertenecientes a empresas de transporte podrán realizar transacciones de entrega.
Descripción	Se deberá invocar la función de entrega presente en un determinado contrato desplegado en la red Fabric. En esta llamada se incluirá la fecha de entrega del lote.
Escenarios secundarios	<ul style="list-style-type: none"> • Escenario alternativo 1: El lote que se desea entregar no existe. <ul style="list-style-type: none"> ○ No se entrega el lote y el estado del contrato no se modifica.
Excepciones	<ul style="list-style-type: none"> • No se poseen permisos de entrega: El procedimiento de entrega no es hecho por una identidad perteneciente a un transportista. <ul style="list-style-type: none"> ○ La transacción no es realizada y el lote no se entrega.

Tabla 6.11: Caso de uso de entrega de un lote

Utilizar lote	
Precondiciones	<p>Las transacciones de utilización deben realizarse por alguien que posea un certificado válido emitido por una CA de confianza. Únicamente certificados pertenecientes a compañías receptoras de lotes podrán utilizarlos. Estas compañías son:</p> <ul style="list-style-type: none"> • Productor de mermelada: Utiliza lotes que contienen frutas para elaborar mermelada con ellas. • Cadena de supermercados: Utiliza lotes que contienen tarros de mermelada para ponerlos a la venta. <p>Una identidad perteneciente a una empresa receptora solamente podrá utilizar lotes si estos están dirigidos explícitamente a dicha identidad. Por último, para que un lote pueda ser utilizado, debe estar en estado “recibido”.</p>
Poscondiciones	El estado del lote deberá cambiar a “utilizado”, y se deberá dar valor al campo referente a la fecha de utilización.
Actores	Identidades válidas pertenecientes a empresas receptoras de lotes podrán usarlos.
Descripción	Se debe invocar la función de utilización implementada en un determinado contrato instanciado en la red Fabric. En dicha invocación se incluirá la fecha de uso del lote.
Escenarios secundarios	<ul style="list-style-type: none"> • Escenario alternativo 1: El lote que se quiere utilizar no existe. <ul style="list-style-type: none"> ○ No se usa el lote y el estado del contrato no

	sufre cambios.
Excepciones	<ul style="list-style-type: none"> • No se poseen permisos de utilización: El uso del lote no es hecho por la identidad de la compañía receptor a la que va dirigido. <ul style="list-style-type: none"> ○ La transacción no es realizada y el lote permanece sin usarse.

Tabla 6.12: Caso de uso de utilización de un lote

Leer datos gestionados mediante Fabric	
Precondiciones	El lector de los datos deberá estar en posesión de un certificado emitido por una CA en la que se confía. Para leer datos se debe establecer una conexión a uno de los canales existentes en la red Fabric, por tanto, el lector deberá poder conectarse al canal del cual desea leer datos y debe tener permisos de lectura sobre dicho canal.
Poscondiciones	Los datos almacenados en el canal que desea leer el lector son retornados.
Actores	Una identidad válida con permisos de acceso y lectura sobre un canal puede leer sus contenidos.
Descripción	Un lector se conecta a un canal presentando su certificado. Si el MSP del canal determina que este lector posee los permisos necesarios le dejará acceder. Al poder existir varios canales en una red, quizás el lector pueda acceder a algunos de ellos y a otros no.
Escenarios secundarios	-
Excepciones	<ul style="list-style-type: none"> • No se poseen permisos de acceso: El lector no tiene permisos de acceso al canal. <ul style="list-style-type: none"> ○ El MSP del canal al que intenta conectarse no le permitirá acceder y por tanto no podrá leer los datos. • No se poseen permisos de lectura: El lector sí tiene permisos de acceso al canal pero no tiene permisos de lectura sobre los datos que quiere ver. <ul style="list-style-type: none"> ○ En este caso, el MSP del canal sí le permite al lector acceder y leer otros datos del canal, pero no puede ver los datos que él desea.

Tabla 6.13: Caso de uso de lectura de datos en Fabric

Comprobar la existencia de documentos	
Precondiciones	Para comprobar la existencia de un documento en la red

	Ethereum, se debe saber la dirección de la cuenta de contrato en la que está desplegado el contrato inteligente relativo al registro de documentos. Adicionalmente, también se debe conocer el nombre y los parámetros que acepta la función implementada en el contrato que permite la comprobación de la existencia de documentos. No es necesario disponer de una cuenta o dirección para leer del estado actual de una red Ethereum pública, pero sí para realizar transacciones en ella.
Poscondiciones	El usuario debe ser notificado acerca de si el documento fue registrado o no.
Actores	Todo el mundo puede comprobar si un determinado documento ha sido registrado.
Descripción	Para comprobar la existencia de un determinado documento, se debe realizar su hash, y llamar a la función de comprobación de existencia implementada en el contrato inteligente, enviando como argumento el hash calculado. Tras comprobar la existencia, se notificará al comprobador acerca de si dicho documento fue o no registrado.
Escenarios secundarios	-
Excepciones	<ul style="list-style-type: none"> • Fallo de conexión: Para leer contenidos del estado actual de una red Ethereum, si no se dispone de un propio nodo completo o ligero, se debe establecer conexión con uno, y dicha conexión puede fallar. <ul style="list-style-type: none"> ○ Intentar conectarse a otro nodo diferente de la red.

Tabla 6.14: Caso de uso de comprobación de existencia de documentos

6.6 Análisis de la Interfaz de Usuario

Al comienzo de este trabajo fin de grado existía la duda de cómo ofrecer la posibilidad de interactuar con las redes blockchain, para ello, se consideraron las siguientes alternativas:

- **Línea de comandos:** A través de una terminal y de comandos, se podrían realizar transacciones y leer datos almacenados en una red blockchain.
 - **Ventajas:** La interfaz sería muy simple y el tiempo para desarrollarla sería corto.
 - **Inconvenientes:** La interacción con la blockchain quizás sería un tanto tediosa, ya que la interfaz no sería visual y las ideas subyacentes podrían permanecer un tanto abstractas y menos claras.
- **Interfaz de usuario:** Mediante una interfaz de usuario se podría visualizar el estado actual de una red blockchain y realizar transacciones pulsando en diferentes lugares de la pantalla.
 - **Ventajas:** La interacción con una red sería más visual y cómoda, lo cual podría favorecer la comprensión de la tecnología que permite el

almacenamiento de los datos y haría más sencillo el proceso de interacción con una red.

- **Inconvenientes:** El desarrollo de una interfaz de usuario requeriría de una mayor inversión de tiempo y trabajo que una interfaz de línea de comandos.

Finalmente, se ha decidido realizar una interfaz de usuario, debido a que permite que la interacción con la red se realice de una manera menos tediosa, más visual y más cómoda. La intención es que esta interfaz sea simple y sencilla de utilizar, sin embargo, no se pretende que tenga un gran peso en este trabajo fin de grado, ya que solamente se realiza con el propósito de facilitar la interacción con las redes blockchain, las cuales son uno de los principales focos del presente proyecto. El diseño de la interfaz de usuario se llevará a cabo en el apartado 7.6, sin embargo, antes del mismo, se ha hecho un pequeño diagrama inicial acerca de los elementos principales que estarán presentes en ella, el cual se representa en la Figura 6.4. A través de esta interfaz, se podrían realizar transacciones relacionadas con lotes y con documentos, permitiendo elegir la identidad que realiza dichas transacciones.

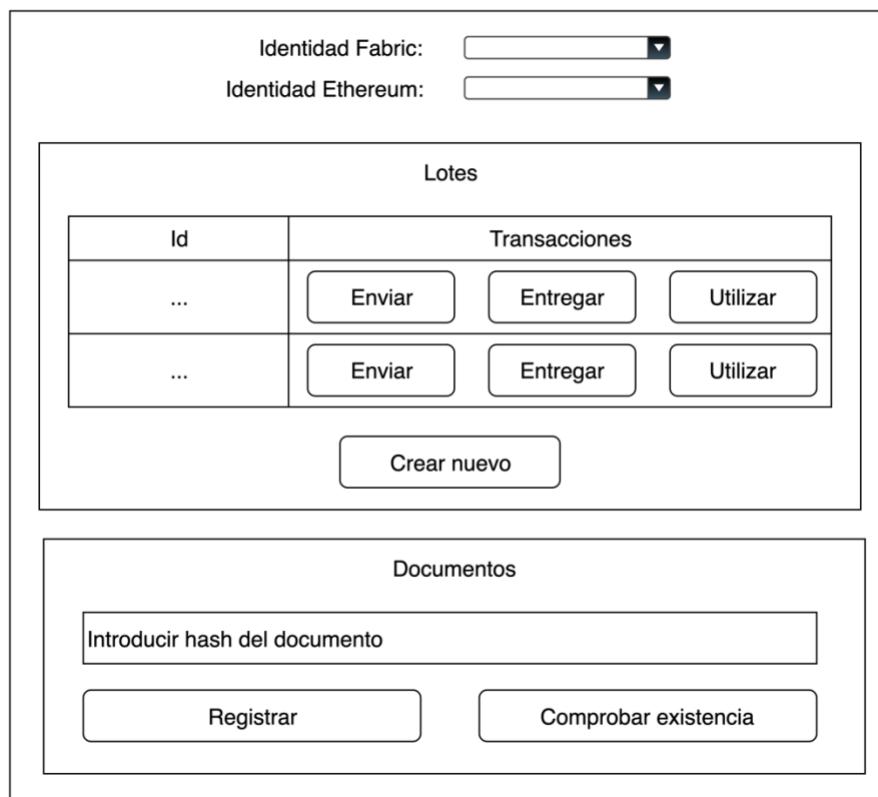


Figura 6.4: Diagrama inicial de la interfaz de usuario

Capítulo 7: Diseño del Sistema

Partiendo del análisis realizado en el capítulo previo, se documenta en el presente capítulo el diseño del sistema.

7.1 Arquitectura del Sistema

El sistema a realizar estará conectado a dos redes blockchain, una de consorcios (elaborada con Hyperledger Fabric) y otra pública (utilizando Ethereum como plataforma). Al comienzo de este apartado se diseñará la arquitectura de la red de consorcios y la manera en la cual se interactuará con ella. Tras ello, se mencionarán los aspectos relativos a la comunicación con la red pública, y, para concluir, se mostrará la arquitectura completa del sistema conectado a ambas redes.

7.1.1 Red de Consorcios

La red de consorcios (red Fabric) que se va a construir poseerá las siguientes características:

- **Organizaciones:** En la red participarán 7 organizaciones, las cuales han sido mencionadas y descritas en el apartado 6.1.1.
- **Nodos:** Cada una de las organizaciones participantes aportará un nodo a la red, por lo tanto, la red constará de 7 nodos.
- **Canales:** Existirán 3 canales en la red, los cuales se mencionan y describen a continuación:
 - **Canal relativo a las manzanas (CanalManzanas):** En este canal se reflejará el proceso relativo al envasado y transporte de manzanas. Cada canal, como se ha visto en el estudio teórico de Hyperledger Fabric, posee su propio estado actual, el cual es independiente de los demás canales. Por lo tanto, el estado actual de este canal almacenará los lotes que contienen manzanas. En este canal participarán 3 empresas (el proveedor de manzanas, el transportista de manzanas y el productor de mermelada).
 - **Canal relativo a las fresas (CanalFresas):** Este canal tiene un propósito similar al anterior, pero en este caso se reflejará en él el proceso de envasado y transporte de fresas hasta la fábrica de mermelada. De nuevo, 3 compañías formaran parte de este canal (el proveedor de fresas, el transportista de fresas y el productor de mermelada).
 - **Canal relativo a la mermelada (CanalMermelada):** Este canal reflejará el proceso de producción de tarros de mermelada y su transporte hasta los diferentes supermercados. La compañía productora de mermelada, el transportista de mermelada y la cadena de supermercados serán las empresas que formen parte de este canal.
- **Servicio de ordenación:** El servicio de ordenación, al tratarse de un prototipo, se ha decidido que solamente conste de un nodo *orderer*.

- **Autoridades certificadoras:** Cada empresa que forma parte de la red tendrá su propia autoridad certificadora en la que confía.

El hecho de que cada alimento se gestione en un canal distinto permite que todas las organizaciones puedan colaborar para formar una red, pero que los datos relativos a cada tipo de alimento estén separados de manera independiente y que el acceso a dichos datos se pueda gestionar de forma adecuada, permitiéndolo o denegándolo a las empresas que se consideren oportunas. Por ejemplo, en el caso del proveedor de fresas y del transportista de las fresas, no tendrán acceso al *CanalManzanas*, y por tanto, no podrán conocer información acerca de la compra de manzanas por parte del productor de mermelada.

En la Figura 7.1 se muestra la arquitectura de la red de consorcios que se ha diseñado, en la cual se aprecia como el nodo perteneciente al productor de mermelada guarda una copia del estado actual y de la cadena de bloques relativos a cada canal (ya que esta compañía está presente en los 3 canales), mientras los demás nodos solamente guardan la información relativa a un canal.

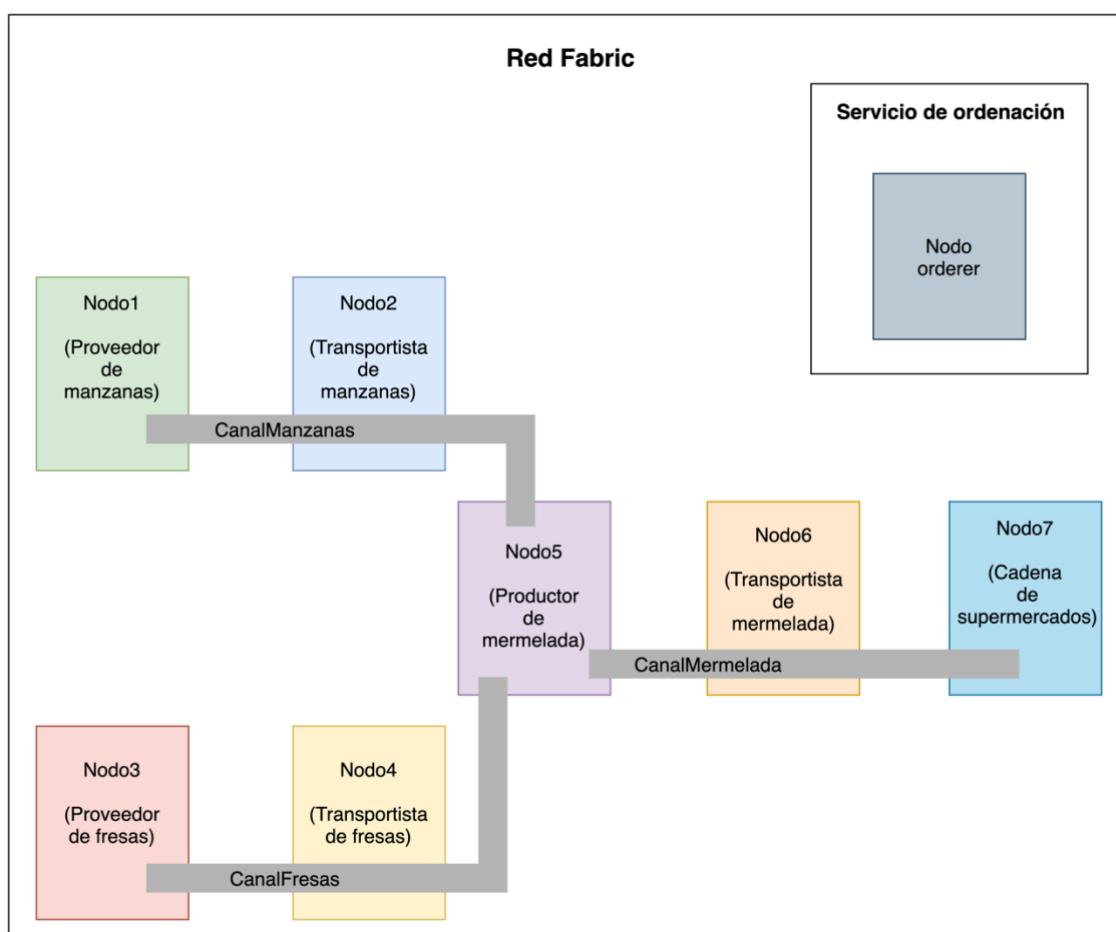


Figura 7.1: Arquitectura de la red de consorcios

En cada canal se encontrará instanciado o desplegado un contrato inteligente (analizado en la Tabla 6.5). Dicho contrato estará instalado en todos los nodos de cada canal, con el objetivo de que todos ellos puedan ejecutar transacciones, ya que, si un nodo no tiene instalado un contrato inteligente, solamente puede mantener una copia de la cadena de bloques y del estado actual, pero no ejecutar transacciones. Por lo tanto, en la red habrá desplegados tres contratos

inteligentes idénticos, cada uno de ellos en un canal diferente, como consecuencia de esto, el estado relativo a cada contrato estará aislado en un canal distinto. El contrato desplegado en el *CanalManzanas* trabaja con lotes de manzanas, el desplegado en el *CanalFresas* con lotes de fresas, y el desplegado en el *CanalMermelada* con lotes llenos de tarros de mermelada.

En este prototipo, la comunicación con la red Fabric se hará mediante un cliente y un servidor, los cuales se describen a continuación:

- **Cliente:** El cliente actuará tanto de cuadro de mandos como de simulador de transacciones. Dará la posibilidad de conectarse a uno de los tres canales para ofrecer al usuario una interfaz en la que se muestre el estado actual del canal al que se está accediendo, y permitirá simular transacciones de creación, envío, entrega y uso de lotes invocando al contrato inteligente desplegado en dicho canal. El cliente ofrecerá la posibilidad de conectarse a los tres canales, pero solo a uno de ellos de manera simultánea.
- **Servidor:** Este servidor actúa de intermediario entre el cliente y la red Fabric. El cliente se comunica con una interfaz ofrecida por el servidor para la realización de transacciones, las cuales se firman en el servidor y se envían a la red Fabric. Para leer datos de la red, el cliente también realiza una petición de lectura al servidor, el cual obtiene los datos de la red y se los envía al cliente para que el usuario pueda visualizarlos. De cara a la recepción de eventos, es el servidor el que capta los eventos emitidos por la red Fabric cuando se confirman determinadas transacciones, y es él el que se encarga de redirigirlos al cliente, para que la interfaz de usuario del cuadro de mandos pueda actualizarse en consecuencia. Aunque normalmente se mencionará un único cliente, podrá haber varios, cada uno de ellos comunicándose con el mismo servidor, el cual acepta peticiones de los diferentes clientes y se encarga de redirigir los datos a los clientes adecuados.

La Figura 7.2 representa las relaciones entre el cliente, el servidor y la red Fabric, donde tanto la comunicación entre el cliente y el servidor y entre el servidor y la red es bidireccional.

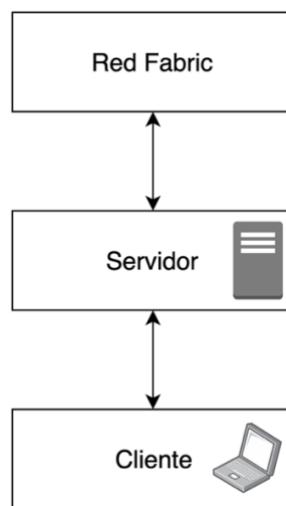


Figura 7.2: Comunicación con la red Fabric

Cada cliente, el servidor y cada elemento que forma parte de la red Fabric podría encontrarse en una máquina física distinta y situada en cualquier lugar del mundo, las cuales se

comunicarían entre sí para que todo funcionase adecuadamente. Sin embargo, en este prototipo, todos estos elementos se encontrarán en una misma máquina, empleando para ello un servicio de contenedores. Concretamente, cada elemento de la red Fabric se ejecutará en su propio contenedor, pero no el servidor ni el cliente.

Un contenedor permite la ejecución de una serie de servicios o procesos de forma aislada, a su vez, cada contenedor puede comunicarse con otros contenedores diferentes o con otros procesos software que no se ejecuten en ningún contenedor. De esta manera, se puede simular el tener varias máquinas físicas dentro de una sola, lo cual se asemeja al concepto de máquina virtual, aunque no es exactamente lo mismo, ya que cada máquina virtual posee su propio sistema operativo, en cambio, un contenedor hace uso del sistema operativo perteneciente a la máquina física en la que se está ejecutando, conocida como host. Por lo tanto, al no necesitar de su propio sistema operativo, un contenedor es más ligero que una máquina virtual.

Sin pretender entrar en profundos detalles en este trabajo acerca de esta temática, se muestra en la Figura 7.3 la estructura principal de una máquina física que contiene varias máquinas virtuales, y en la Figura 7.4 la de otra máquina física que contiene varios contenedores.

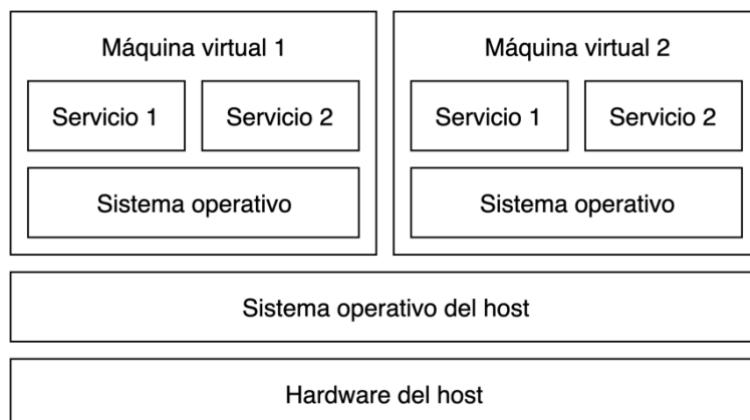


Figura 7.3: Host ejecutando 2 máquinas virtuales

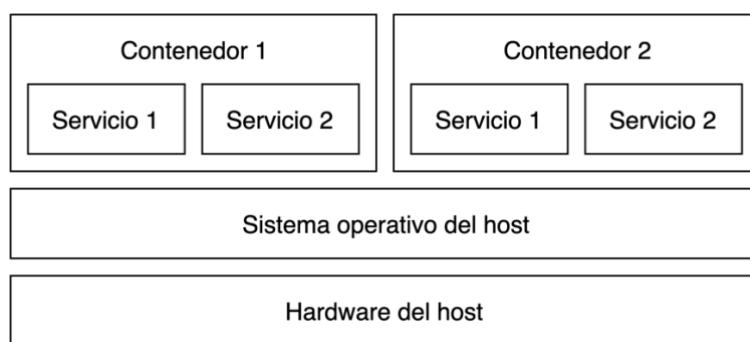


Figura 7.4: Host ejecutando 2 contenedores

7.1.2 Red Pública

De cara a interactuar con una red pública existen dos opciones principales:

- **Contribuir a la red:** Esta opción consiste en crear un nodo completo propio y sincronizarlo con los demás nodos de la red, haciendo que esta posea otro nodo más. Al enviar transacciones a la red, se enviarían a este nodo completo para que las propagase al resto de la red, y para leer datos del estado actual, también se leerían en este nodo.
- **Conectarse a un nodo existente en la red:** En lugar de poseer un nodo completo para enviarle transacciones y leer datos, es posible conectarse a un nodo existente en la red sin la necesidad de tener que crear uno nuevo. De esta manera, tanto para leer datos como para enviar transacciones, se hace uso de un nodo existente.

Poseer un nodo completo que ejecute transacciones tiene la ventaja de que, si consigue generar un bloque válido, se obtiene una recompensa, en cambio, la desventaja es que se debe adquirir o poner a disposición hardware para almacenar toda la cadena de bloques y el estado actual de la red.

De cara a elaborar este prototipo, se optará por la opción de conectarse a un nodo existente, del cual se leerán los datos del estado actual y al cual se enviarán las transacciones, las cuales, a su vez, serán propagadas por este nodo hacia otros nodos de la red para que sean ejecutadas. Para conectarse a un nodo existente se hará uso de un proveedor de conexión, un servicio que ofrece conexiones a distintos nodos presentes en una red pública.

Una de las muchas cuentas que existan en la red será la cuenta de contrato relativa al contrato inteligente analizado en la Tabla 6.6, por lo tanto, tanto el código del contrato como su estado se encontrará presente en todos los nodos completos de la red.

La comunicación con la red pública se hará directamente desde un cliente, en el cual se firmarán las transacciones antes de enviarse a la red. Este cliente ofrecerá una interfaz de usuario para registrar y comprobar la existencia de documentos. En la Figura 7.5 se muestra el esquema de comunicación del prototipo con la red pública, en la cual se aprecia como el cliente está conectado a uno de los muchos nodos presentes en la red, el cual, a su vez, está conectado con otros nodos.

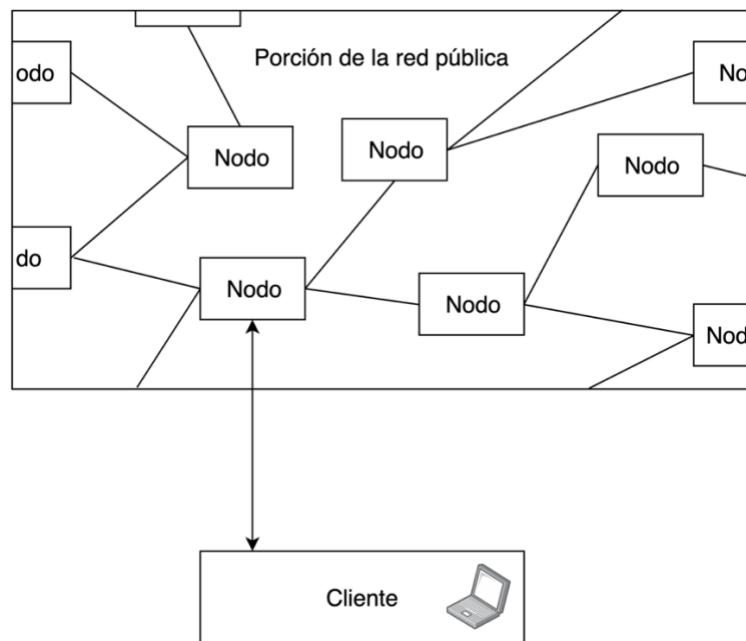


Figura 7.5: Comunicación con la red pública

7.1.3 Visión Global

Una vez explicada la arquitectura y la comunicación con cada red, se resume en esta sección la estructura completa del prototipo, en el cual existe un cliente que podrá conectarse a ambas redes. Se conectará a la red de consorcios para trabajar con los datos relativos a los lotes de frutas y mermelada, y se conectará a la red pública para lo relativo a la gestión documental.

En la Figura 7.6 se muestra la arquitectura global del prototipo, en la que se observa cómo un solo cliente puede conectarse a diferentes redes, dependiendo de las acciones que desee llevar a cabo.

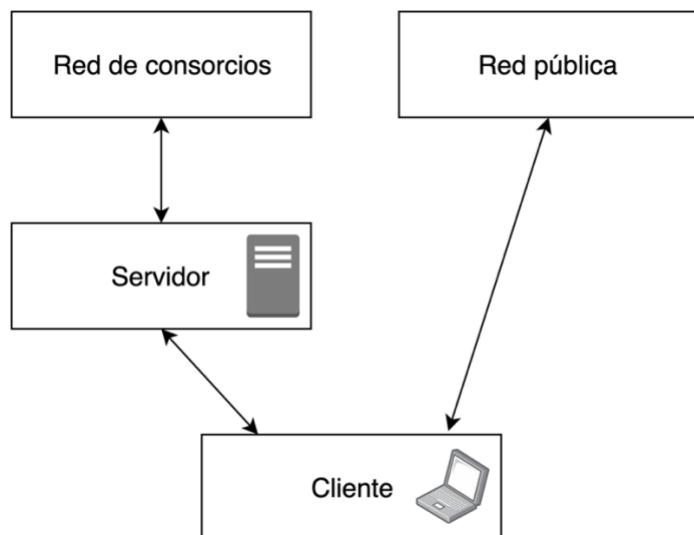


Figura 7.6: Estructura global del prototipo

7.2 Gestión de Identidades

En el presente apartado se tratará el diseño relativo a la gestión de identidades en ambas plataformas.

7.2.1 Hyperledger Fabric

Tras lo introducido en el apartado 6.2.3, se describirá a continuación en más detalle el proceso de gestión de identidades en Fabric.

Para el desarrollo de la parte relativa a la red de consorcios se empleará el framework llamado Hyperledger Composer [60], a pesar de que todavía se encuentra en fase de desarrollo y no existen versiones estables. Este framework, a día de hoy, solamente permite la realización de aplicaciones para interactuar con redes construidas con Hyperledger Fabric.

La manera en la que se gestionan las identidades en Composer es un tanto particular, y para ello se hace uso de *cards*, o, traducido al castellano, tarjetas. Una tarjeta es un archivo digital cuyos contenidos se describen a continuación:

- **Certificado:** Certificado emitido por una CA.

- **Clave privada:** Clave privada correspondiente a la clave pública contenida en el certificado.
- **Información de conexión:** Información relativa a la conexión con la red Fabric. Una particularidad es que cada tarjeta solo puede servir para conectarse a un canal, es decir, no se puede establecer conexión con dos canales diferentes a partir de una misma tarjeta. En este campo se incluye el canal concreto, e información de dicho canal, como los nodos u organizaciones que forman parte de él.

En términos generales, una identidad puede pertenecer o referirse a una persona, dispositivo, empresa o un conjunto de estos. Cada identidad posee su propia tarjeta, en la que se refleja, como se ha descrito anteriormente, el certificado y la clave privada de la identidad.

La nomenclatura usada por Composer es un tanto ambigua, diferente y peculiar, ya que en lugar de usar el término contrato inteligente, usa el término *business network* (BN), sin embargo, en este documento se usará el término contrato inteligente, ya que es equivalente. Adicionalmente en lugar de usar el término instanciar, usa el término *start* para desplegar un contrato en un canal. Para finalizar con los aspectos relativos a la nomenclatura, en el caso de las identidades que pueden interactuar con los contratos inteligentes, son conocidas como participantes.

De cara a gestionar las identidades en Fabric, se generarán las siguientes tarjetas:

- **Tarjetas de administración de hardware:** Las personas encargadas de administrar los nodos tendrán sus propias tarjetas. Estas personas serán los administradores que tengan permiso para instalar contratos en los nodos y para instanciar contratos en un canal.
- **Tarjetas de administración de contratos inteligentes:** Cada contrato inteligente que se encuentra desplegado en un canal posee también una serie de administradores, los cuales se encargan de añadir participantes al canal y de crearles tarjetas para que puedan interactuar con el contrato.
- **Tarjetas para los participantes de la red:** Cada participante en la red tiene una tarjeta. En esta red, cabe pensar que existen 13 participantes (5 plantas industriales, 6 vehículos y 2 supermercados). Pero uno de ellos, la planta industrial productora de mermelada se puede conectar a los tres canales de la red, por ello, debido a la restricción de Composer de que cada tarjeta solamente puede conectarse a un canal, esta planta tendrá que tener 3 tarjetas, lo cual implica que tendrán que generarse 3 certificados distintos para lo que daba la impresión de que era un mismo participante. Esto es una restricción de Composer a día de hoy que puede que en el futuro no exista. Quizá lo más normal sería que la fábrica de producción de mermelada tuviese un solo certificado y que con él se pudiese conectar a cualquier canal, lo cual, en futuras versiones de Composer puede que sea posible. Por lo tanto, tras esta serie de aclaraciones, la conclusión es que técnicamente existen 15 participantes en la red, por lo tanto, existirán las siguientes 15 tarjetas relativas a participantes:
 - **Vehículos:** 6 tarjetas, una para cada vehículo.
 - **Supermercados:** 2 tarjetas, una para cada supermercado.
 - **Plantas industriales:** 7 tarjetas, una para cada planta de envasado de fruta y 3 para la planta productora de mermelada (una tarjeta para conectarse a cada canal). En el caso de la productora de mermelada, es completamente

normal pensar o decir que es un participante en la red o que posee una identidad, de hecho, parece que se entiende mejor desde ese enfoque. Sin embargo, al trabajar con Composer, técnicamente, se obliga a tratarla como 3 participantes o identidades, aunque se puede seguir viendo de manera abstracta como un participante que se puede conectar a 3 canales distintos. Es cuestión de cómo el lector entienda mejor el concepto. En mi caso, prefiero verlo de manera abstracta, pero comprendiendo las peculiaridades subyacentes de Composer que han sido explicadas previamente.

En el prototipo, todas las tarjetas de los participantes estarán almacenadas en el servidor, que será desde donde se firmen las transacciones. En un entorno real, cada planta industrial, vehículo o establecimiento tendría su propio servidor, de cuya configuración se encargaría un administrador. Cada servidor contaría con acceso a la tarjeta perteneciente al vehículo, planta o establecimiento correspondiente.

Por ejemplo, en el caso de la planta de envasado de manzanas de Lugo, habría un servidor, el cual tendría acceso a la tarjeta de la planta. El lugar en el que se almacena la tarjeta o la forma de acceder a ella lo configuraría y gestionaría el administrador del servidor. Esta tarjeta actuaría en representación de todos los dispositivos IoT presentes en la planta, los cuales, cuando se cumpliesen ciertas circunstancias como el cierre de un lote, enviarían eventos al servidor, que, ante la recepción de dichos eventos, produciría transacciones, las firmaría con la clave privada presente en la tarjeta y las enviaría a la red para su ejecución y confirmación. A través de este servidor solamente se podría establecer conexión con el canal relativo a las manzanas. En este entorno real, la planta también podría tener más de un servidor, incluso uno podría estar dedicado a recibir eventos y transmitirlos hacia el cliente donde se ofrece la interfaz del cuadro de mandos, y otros podrían estar dedicados a recibir mensajes de dispositivos IoT y enviar transacciones a la red. Con este diseño, en un entorno real, las claves privadas serían gestionadas por los administradores encargados de configurar y administrar cada servidor.

Sin embargo, el prototipo solamente contará con un servidor, a través del cual se podrá establecer conexión con cualquiera de los tres canales, tan solo seleccionando la tarjeta adecuada.

7.2.2 Ethereum

En lo relativo a Ethereum, la gestión de identidades es más sencilla que en Fabric, ya que, a la hora de registrar documentos de entrega de lotes, solamente administradores de las empresas de transporte podrán hacerlo. Al existir 3 empresas de transporte, cada una de ellas tendrá un administrador para la parte de la red pública, y cada uno de ellos tendrá su propia identidad. Cada identidad en Ethereum se corresponde con una cuenta o dirección, y cada cuenta posee su propia clave privada, por lo tanto, cada administrador posee su propia clave privada.

Las transacciones relativas a la red pública se firman en el cliente y se envían directamente a la red. Es responsabilidad de cada administrador mantener en secreto su clave privada.

En un entorno real, cada administrador dispondría de su propio cliente desde donde se realizarían las transacciones, sin embargo, en el caso del prototipo, se permitirá hacer transacciones en nombre de los 3 administradores desde un mismo cliente.

7.3 Diseño del Estado Actual

El estado actual de cada canal en la red Fabric estará compuesto por un conjunto de lotes. En el *CanalManzanas* estos lotes contendrán manzanas, en el *CanalFresas* fresas y en el *CanalMermelada* tarros de mermelada.

Composer contiene un lenguaje integrado que permite modelar el dominio de una red, definiendo los elementos que formarán parte del estado actual y los tipos de participantes que interactuarán con la red. En lo relativo a estos últimos, se definen los siguientes tipos de participantes:

- **Creador:** Un participante de tipo creador podrá crear lotes.
- **Vehículo:** Los participantes de tipo vehículo tendrán permitido enviar y entregar lotes.
- **Destinatario:** Los destinatarios serán los que tengan la potestad para utilizar los contenidos de un lote.

Los participantes de cada tipo que podrán interactuar con cada uno de los canales serán los siguientes:

- **CanalManzanas:**
 - **Creadores (2):** Las dos plantas de envasado de manzanas.
 - **Vehículos (2):** Los dos vehículos correspondientes al transportista de manzanas.
 - **Destinatarios (1):** La planta de producción de mermelada es el destino de los lotes de manzanas.
- **CanalFresas:**
 - **Creadores (2):** Las dos plantas de envasado de fresas.
 - **Vehículos (2):** Los dos vehículos de la compañía encargada del transporte de fresas.
 - **Destinatarios (1):** La planta de producción de mermelada.
- **CanalMermelada:**
 - **Creadores (1):** La planta de producción de mermelada.
 - **Vehículos (2):** Los dos vehículos pertenecientes a la empresa de transporte de mermelada.
 - **Destinatarios (2):** Los dos establecimientos de venta de la cadena de supermercados.

El resultado tras el proceso de modelado se muestra en la Figura 7.7, en la cual se incluyen los 3 participantes descritos previamente y el esquema de todo lote que se almacene en el estado actual.

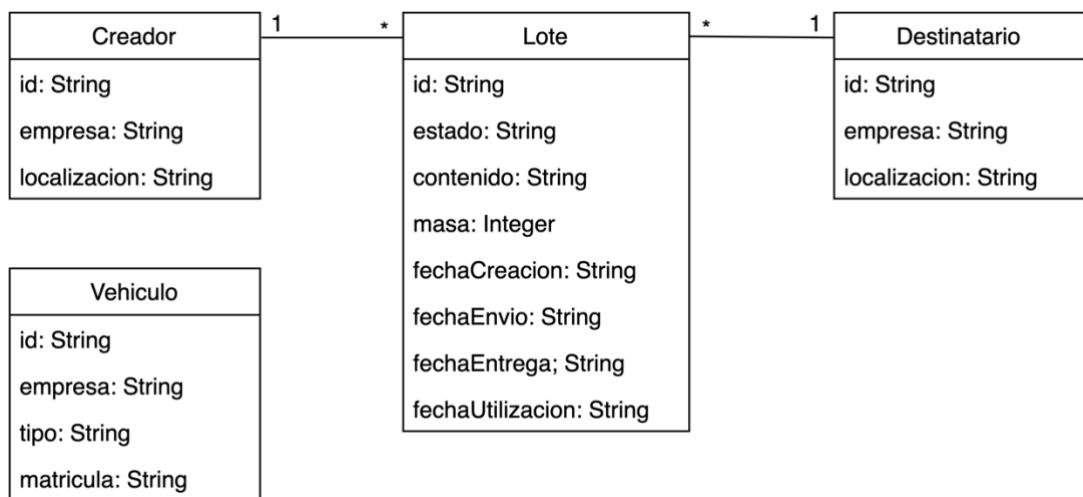


Figura 7.7: Modelo de datos de la red de consorcios

En relación con la red pública, el estado actual estará formado por un conjunto de pruebas de existencia, cuyo esquema se representa en la Figura 7.8.

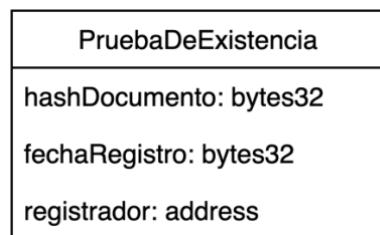


Figura 7.8: Modelo de datos de la red pública

Como se ha mencionado en la sección 6.4.1, se ha pretendido que los modelos de datos de cada red sean lo más sencillos posible.

7.4 Diseño de los Contratos Inteligentes

Tras la fase de análisis de los contratos, se incluyen en esta sección sus operaciones definitivas y los parámetros definidos para cada una de ellas. En lo relativo a las operaciones de lectura de lotes del contrato de Fabric, Composer ofrece la posibilidad de realizar dichas operaciones sin necesidad de incluirlas explícitamente en el contrato, por lo tanto, únicamente se incluirán las relativas a transacciones, mostradas en la Figura 7.9.

Operaciones del contrato inteligente Fabric
crearLote(creadorId: String, destinatarioId: String, contenido: String, masa: Integer, fechaCreacion: String)
enviarLote(lote: Lote, fechaEnvio: String)
entregarLote(lote: Lote, fechaEntrega: String)
utilizarLote(lote: Lote, fechaUtilizacion: String)

Figura 7.9: Diseño del contrato inteligente relativo a Fabric

En cuanto al contrato desplegado en Ethereum, se incluirán las operaciones representadas en la Figura 7.10.

Operaciones del contrato inteligente Ethereum
registrarDocumento(hashDocumento: bytes32, fechaRegistro: bytes32, registrador: address)
comprobarExistencia(hashDocumento: bytes32): bytes32

Figura 7.10: Diseño del contrato inteligente relativo a Ethereum

7.5 Diagramas de Estado

Debido a que la tecnología blockchain puede verse como una máquina de transición de estados, se incluyen en esta sección los diagramas de este tipo.

En la Figura 7.11 se muestra el diagrama representando los estados por los que pasa cada lote cuando se producen determinadas transacciones que le afectan.



Figura 7.11: Diagrama de estados de un lote

En relación con los documentos, estos no son almacenados en el estado actual de la red Ethereum, pero se puede considerar de manera abstracta que tras producirse una transacción de registro, pasan de no estar a sí estar registrados, por lo tanto, se puede modelar este proceso también con un diagrama de estados, el cual se incluye en la Figura 7.12.



Figura 7.12: Diagrama de estados de un documento

7.6 Diseño de la Interfaz de Usuario

Como se ha dicho en la sección 6.6, se elaborará una interfaz de usuario de cara a interactuar con las redes blockchain, la cual será una interfaz web. El cliente representado en la Figura 7.6 se puede conectar tanto a la red de consorcios como a la red pública, y ofrecerá una interfaz de usuario para interactuar con cada una de ellas. Por lo tanto, para este prototipo, se diseñarán las siguientes pantallas:

- **Pantalla de interacción con la red Fabric:** A través de esta pantalla se podrán leer los datos de los canales de la red Fabric y realizar transacciones de creación, envío, entrega y utilización de lotes.
- **Pantalla de interacción con la red Ethereum:** Mediante esta pantalla se podrán registrar documentos, enviando para ello transacciones a la red pública, y comprobar si determinados documentos se han registrado.

Estas dos pantallas tendrán un estilo similar, pero cada una de ellas estará pensada y diseñada para realizar diferentes acciones.

Las principales características y elementos de la interfaz para interactuar con la red Fabric, cuyo diseño se incluye en la Figura 7.13, se comentan a continuación:

- **Selección de participante:** Arriba a la derecha existe un ícono que al pulsarlo mostrará una lista con todos los participantes (15) en la red. Dependiendo del participante que se seleccione, la interfaz mostrará los datos relativos a un canal u otro, ya que cada participante posee una tarjeta, mediante la cual solamente puede conectarse a un canal. En la pantalla se mostrarán los datos del participante seleccionado en cada momento y el canal al que se está conectado.
- **Creación de lotes:** Mediante un formulario será posible crear lotes. Para ello, el participante seleccionado deberá ser de tipo creador, ya que los demás no tienen permitido crear lotes. El lote se añadirá al estado actual del canal correspondiente al participante seleccionado.
- **Visualización de lotes:** La interfaz ofrecerá 4 tablas en las que se vean los lotes que se encuentran en cada uno de los 4 estados. En la Figura 7.13, para que no ocupe mucho espacio, solamente se muestra la tabla relativa a los lotes en estado almacenado, las demás se encontrarían debajo suya. Al pulsar en el botón *Detalles* relativo a un lote, se mostrarán sus detalles en la pantalla, ya que en la tabla solamente se muestran los principales. De cara a simular transacciones, cada fila tendrá un botón que servirá para realizar una transacción sobre un lote concreto, estos botones se encontrarán activos o inactivos dependiendo del tipo de participante que se encuentre seleccionado. En la tabla de los lotes en estado almacenado, la transacción que se podrá realizar será la de envío. Estas tablas se actualizarán dinámicamente a medida que se reciban eventos emitidos por la red cuando se confirmen transacciones.

PrototipoTFG - Red de consorcios


Datos del participante seleccionado

Tipo:
Empresa:
Canal:

Creación de un lote

Contenido:

Masa (kg):

Destinatario:

Detalles del lote seleccionado

Identificador:
Creador:
Destinatario:
Estado:
Contenido:
Masa (kg):
Fecha de creación:
Fecha de envío:
Fecha de entrega:
Fecha de utilización:

Lotes en estado almacenado (X)

Identificador	Contenido	Masa (kg)	Información	Transacciones
...	<input type="button" value="Detalles"/>	<input type="button" value="Enviar"/>
...	<input type="button" value="Detalles"/>	<input type="button" value="Enviar"/>
...	<input type="button" value="Detalles"/>	<input type="button" value="Enviar"/>

...

Figura 7.13: Diseño de la interfaz para interactuar con Fabric

Tras el diseño de la interfaz para interactuar con la red de consorcios, se muestra el diseño relativo a la pantalla de interacción con la red pública en la Figura 7.14, cuyas características son las siguientes:

- **Selección de identidad:** Similar a la pantalla para interactuar con Fabric, se ofrece arriba a la derecha la posibilidad para elegir una de las tres direcciones o cuentas que pueden registrar documentos.

- **Registro de documentos:** De cara al registro documental, se ofrece un formulario en el que existirá la opción para elegir el documento que se desee registrar. Al pulsar dicho botón, se podrá elegir un documento que se encuentre en la máquina donde se ejecuta el cliente. Tras la elección, se mostrarán sus contenidos en la pantalla, de cara a que el usuario pueda ver que son adecuados. Al pulsar el botón *Registrar documento* se enviará una transacción de registro a la red pública firmada con la clave privada de la cuenta que se encuentre seleccionada.
- **Comprobación de existencia de documentos:** Similar al proceso descrito con anterioridad, el formulario para comprobar si un documento ha existido en el pasado permite seleccionar el documento adecuado, observar sus contenidos, y, al pulsar en el botón *Comprobar existencia*, se hará una lectura en el estado actual de la red para saber si ha sido registrado o no. Tanto si el documento fue registrado como si no lo fue, se notificará al usuario acerca de ello.

PrototipoTFG - Red pública

Registro de documentos

Selecciona un documento:

Contenido del documento seleccionado:
...

Comprobar existencia de documentos

Selecciona un documento:

Contenido del documento seleccionado:
...

Figura 7.14: Diseño de la interfaz para interactuar con Ethereum

Capítulo 8: Implementación del Sistema

En este capítulo se tratan los aspectos relativos a la implementación del sistema. Para ello, se hace referencia a los lenguajes de programación empleados, al software utilizado, y, por último, se mencionan diferentes aspectos acerca de la creación del prototipo.

8.1 Lenguajes de Programación

Los lenguajes de programación que se han empleado para desarrollar el prototipo han sido los siguientes:

- **JavaScript:** Para la elaboración del contrato inteligente en la red Fabric se ha hecho uso de JavaScript. Fabric soporta otros lenguajes adicionales como Go y Java, sin embargo, el framework Composer solamente soporta a día de hoy el lenguaje JavaScript, por ello, ha sido la única alternativa posible. La implementación del código que se ejecuta en el servidor y el que lo hace en el cliente también ha sido realizada en JavaScript. En este prototipo, el cliente es un navegador web, y JavaScript es el lenguaje empleado para programar la manera de interactuar con una interfaz web, manipular y editar sus elementos, definir el comportamiento ante eventos y programar la lógica de negocio existente en el lado del cliente. En cuanto al servidor, el entorno de ejecución que se usa es Node.js, gracias al cual es posible la ejecución de código JavaScript. Al igual que en el caso de la implementación del contrato, para el lado servidor también se ha hecho uso de Composer, de cara en este caso a interactuar con la red, y esta ha sido la razón por la cual se ha empleado JavaScript también en el lado servidor.
- **Solidity:** De cara a la implementación del contrato inteligente relativo a la red pública, se ha realizado en Solidity, ya que es el lenguaje más popular a día de hoy a la hora de desarrollar contratos en Ethereum. Solidity es un lenguaje de alto nivel empleado para la elaboración de contratos inteligentes, el cual posee comprobación estática de tipos y puede ser compilado a EVM Code. El grado de madurez de este lenguaje comparado con otros como Java o el propio JavaScript es bajo, ya que fue concebido al mismo tiempo que la plataforma Ethereum. Para este trabajo, se ha empleado la versión 0.4.25 de Solidity.

8.2 Software Utilizado

A continuación se lista y describe el software empleado para llevar a cabo este proyecto:

- **Hyperledger Composer [60]:** Como se ha mencionado en 7.2.1, para la parte del prototipo relativa a Fabric se ha empleado el framework Hyperledger Composer, el cual permite el desarrollo, instalación y despliegue de contratos inteligentes en redes Fabric, y la construcción de aplicaciones que interactúen con dichas redes. Este framework está diseñado explícitamente para la creación y comunicación con

redes construidas con la plataforma Hyperledger Fabric. A día de hoy se encuentra todavía en fase de desarrollo y no posee una versión estable, sin embargo, debido a todos los puntos positivos que ofrece, se ha decidido en este trabajo experimentar con él. Al comienzo del proyecto se ha seleccionado la versión 0.20.6 y siempre se ha trabajado con ella.

- **Docker [61]:** Docker es una plataforma que permite la creación y ejecución de contenedores. De cara a la ejecución de los elementos relativos a la red Fabric se ha hecho uso de esta plataforma.
- **CouchDB [59]:** CouchDB es un proyecto Apache dedicado a la construcción de una base de datos NoSQL, la cual hace uso del formato JSON para el almacenamiento de datos. Debido a que Hyperledger Fabric es una plataforma modular y permite usar diferentes bases de datos para almacenar la cadena de bloques y el estado actual, en este proyecto se ha decidido que cada nodo posea una base de datos CouchDB para tal fin.
- **Infura [62]:** Infura es el proveedor de conexión que se ha utilizado para conectarse a un nodo existente de una red pública de Ethereum. La red pública concreta en la que se ha desplegado el contrato inteligente se trata en el próximo capítulo. Aunque la red Ethereum pública más popular es la red principal de Ethereum, existen más redes públicas elaboradas con esta plataforma. En algunas de ellas, existen una serie de nodos completos que son propiedad de Infura, conocidos como nodos Infura. Infura es por tanto un servicio que provee conexiones a sus nodos, para hacer posible que todo el mundo pueda conectarse a una de las varias redes Ethereum existentes a través de uno de estos nodos, sin la necesidad de poseer uno propio.
- **MetaMask [63]:** MetaMask es una herramienta que se integra en el navegador web y permite gestionar las identidades relativas a Ethereum, encargándose de almacenar y gestionar las claves privadas. A través de MetaMask es posible conectarse a diferentes redes Ethereum de cara a leer su estado actual y enviar transacciones hacia ellas, para ello, esta herramienta hace uso de Infura. Debido a que el prototipo ofrece una interfaz web, se ha hecho uso de esta herramienta para gestionar las claves privadas y las cuentas Ethereum de los administradores de las diferentes empresas de transporte que tienen permiso para registrar documentos. En este prototipo se ha usado la versión 6.7.2.
- **Web3.js [64]:** Web3.js es una librería JavaScript que ofrece funciones para interactuar con redes Ethereum. Entre las muchas funciones de las que dispone, se encuentran la de firmar una transacción o la de leer del estado actual.
- **Truffle [65]:** Truffle es una herramienta que permite realizar procesos de despliegue de contratos inteligentes en redes Ethereum, ofreciendo la posibilidad de conectarse a diferentes redes para desplegar contratos en ellas. El despliegue de un contrato inteligente en una red implica la creación de una cuenta de contrato en ella. Previamente al despliegue, esta herramienta se encarga de la compilación del contrato. En este trabajo se ha usado la versión 5.0.24.
- **jQuery [66]:** jQuery es una librería JavaScript que facilita la gestión y la manipulación del DOM (*Document Object Model*) y gestiona las peculiaridades que cada navegador tiene en relación a su implementación. El DOM es la estructura de datos que usan los navegadores para representar en memoria una página web, por tanto, manipulando los elementos de esta estructura, se pueden editar los

contenidos de una página web de manera dinámica. Además, jQuery se usa también para gestionar los eventos producidos al interactuar con una interfaz web o para comunicarse con un servidor. Todo lo que se hace con jQuery se puede hacer sin él, pero su uso facilita en gran medida muchas operaciones. Esta librería se ha usado en el cliente de cara a realizar la interfaz de usuario.

- **Knockout.js [67]:** Knockout es un framework JavaScript que facilita la gestión de interfaces de usuario. Para ello, hace uso del patrón de diseño llamado *Model-View-ViewModel* (MVVM), el cual se describe a continuación a través de la propia descripción de sus tres elementos:
 - **Model (M):** Modelo traducido al castellano. En el modelo se definen los datos con los que la aplicación va a trabajar.
 - **ViewModel (VM):** Este componente se encarga de almacenar el subconjunto de los datos almacenados en el modelo que se muestran en la interfaz de usuario y operaciones que se pueden realizar sobre ellos. El *ViewModel* no tiene ninguna noción ni información sobre la interfaz en la que se mostrarán los datos, es decir, es independiente de ella.
 - **View (V):** Traducido al español, vista. La vista es el elemento en el que se muestran los datos presentes en el *ViewModel*. En una interfaz web, la vista es código HTML, en el cual, de cara a comunicarse con el *ViewModel*, se incluyen los llamados *bindings* (enlaces). Estos enlaces invocan a operaciones definidas en el *ViewModel* sobre los datos almacenados en él, y, al cambiar dichos datos, cambian los mostrados en la interfaz de manera acorde. De esta manera, los datos nunca se almacenan en la interfaz, sino que, los datos que se muestran en la pantalla son un reflejo de los almacenados en el *ViewModel*.
- **Bootstrap [68]:** Bootstrap es un framework empleado en el desarrollo de interfaces web, mediante el cual, entre otras cosas, se facilita que las interfaces se adapten a distintos tipos de pantallas. De cara a realizar la interfaz de usuario de este prototipo se ha partido de la plantilla citada en [69], la cual es gratuita y posee la licencia MIT.
- **Express [70]:** Para el servidor que se comunica con la red Fabric se ha hecho uso de Express. Express es un framework JavaScript que se ejecuta dentro de un entorno Node.js y permite exponer una API (*Application Programming Interface*) que acepte peticiones HTTP, haciendo posible que los clientes se comuniquen con el servidor y viceversa a través de este protocolo.
- **Webpack [71]:** Webpack es una librería JavaScript que se encarga de unir en un solo fichero el código dispersado en diferentes archivos que poseen dependencias entre ellos. Esta librería se ha usado para crear el archivo JavaScript relativo a la parte del cliente que se comunica con la red Ethereum, ya que, tras leer acerca de cómo interactuar con una red Ethereum, el uso de esta librería era la opción más utilizada y popular.
- **Sweetalert [72]:** Esta librería permite crear notificaciones con diferentes diseños. Ha sido empleada a la hora de mostrar notificaciones en la interfaz de usuario del prototipo.
- **Git [73]:** De cara a gestionar los cambios en el código durante el desarrollo del prototipo se ha empleado Git como sistema de control de versiones. Además, se ha usado GitHub [74] como sistema de alojamiento de repositorios remotos, de esta

manera, el repositorio se encuentra tanto en local como en remoto, y, si ocurre algún problema en la máquina local y se elimina el repositorio, siempre se encontrará alojado remotamente.

- **Visual Studio Code [75]:** Visual Studio Code es una herramienta de edición de código gratuita desarrollada por Microsoft, a la cual se le pueden instalar o añadir extensiones para ampliar sus funcionalidades. Esta herramienta se ha utilizado en este proyecto a la hora de desarrollar los dos contratos inteligentes, ya que existen extensiones para Solidity para el lenguaje de modelado integrado en Composer desarrolladas para ella, las cuales son muy útiles a la hora de detectar errores sintácticos en el código.
- **WebStorm [76]:** WebStorm es un entorno de desarrollo para aplicaciones web desarrollado por JetBrains, el cual se ha utilizado en este proyecto a la hora de desarrollar el código presente en el servidor y en el cliente.
- **Microsoft Project [77]:** Project es una herramienta desarrollada por Microsoft que permite la creación de planificaciones de proyectos, para cuyo fin se ha empleado en este trabajo.
- **Microsoft Word [78]:** Word es un procesador de textos desarrollado por Microsoft, el cual se ha usado durante este trabajo de cara a elaborar la presente documentación, empleando como base la plantilla proporcionada por la Escuela de Ingeniería Informática de la Universidad de Oviedo relativa a los trabajos fin de grado.
- **Microsoft Excel [79]:** Excel, elaborada por Microsoft, ha sido la herramienta de hojas de cálculo que se ha utilizado para la elaboración del presupuesto.
- **Draw.io [80]:** Para la elaboración de los diagramas presentes en esta documentación se ha hecho uso de draw.io, herramienta online gratuita que permite la creación de diferentes tipos de diagramas.

8.3 Creación del Sistema

Durante la creación del sistema han surgido diversos problemas, los cuales se mencionan a continuación, para tras ello describir la estructura de ficheros que componen el prototipo desarrollado.

8.3.1 Problemas Encontrados

De cara a describir los problemas encontrados, se trata cada uno de ellos en un apartado individual.

8.3.1.1 Herramientas en Fase de Desarrollo

Uno de los principales problemas a la hora de elaborar este prototipo ha sido el hecho de haber trabajado con Hyperledger Composer, debido a que no existe una versión estable todavía de este framework. La documentación acerca de cómo usarlo es muy básica y muy poco detallada, ofreciendo ejemplos muy sencillos, de modo que, al intentar realizar cosas más complejas de las descritas en sus tutoriales, he tenido muchas complicaciones. Existen métodos reflejados en la documentación que no se encuentran implementados en el framework, y además, entre

diferentes versiones existen numerosos cambios de nomenclatura y de funcionalidad, lo que hace que, al intentar resolver dudas, no solamente tuviese que tener la esperanza de encontrar información acerca de ellas, sino de encontrar dicha información relativa a la versión con la que yo estaba trabajando. La información presente a día de hoy acerca de este framework es muy escasa, lo que ha implicado que, para conseguir hacer cosas aparentemente sencillas tras entender los conceptos teóricos, trasladar ese conocimiento a la práctica fuese en ocasiones complicado, empleando mucho más tiempo del planificado para descubrir la manera de implementar ciertas funcionalidades como la creación de identidades o la gestión de eventos emitidos por la red, teniendo en la mayoría de ocasiones que descubrir cómo realizar estas cosas a base de prueba y error, gracias a los conocimientos teóricos que había adquirido y los prácticos que iba adquiriendo poco a poco.

Otro de los retos al trabajar con Composer ha sido la automatización de los procesos, como el de crear la red o el de la generación de tarjetas para los participantes, ya que, en la documentación de Composer, todos estos procesos son manuales e involucran la ejecución de muchos comandos, lo cual es tedioso y lento. La automatización de estos procesos se ha conseguido, haciendo que sean más sencillos y transparentes para el usuario del prototipo, pero para ello se ha tenido que emplear una cantidad de tiempo que en principio no estaba prevista.

De cara al trabajo con web3.js, la versión 1 contiene muchas más funcionalidades que versiones inferiores, debido a lo cual se ha decidido trabajar con ella, pero el inconveniente es que se encuentra en fase beta. Sin embargo, en este caso no he tenido tantos problemas a la hora de utilizar esta librería. Concretamente, he trabajado con la versión 1.0.0-beta.37.

8.3.1.2 Documentación

En lo relativo a todas las plataformas y herramientas blockchain en general, el problema más común con el que me encontrado es que existe más documentación teórica que práctica, lo que hace en ocasiones difícil llevar a la realidad los conceptos teóricos. Adicionalmente, la documentación práctica suele ser toda muy parecida y muy cerrada, usando siempre las mismas herramientas y los mismos pasos, de modo que, al intentar salirse un poco de lo común, los problemas se agravan. Por ello, en este proyecto se ha procurado trabajar siguiendo los procesos más populares y comunes dentro de cada plataforma.

Para concluir, actualmente, las herramientas y plataformas relacionadas con la tecnología blockchain se encuentran en una evolución constante y rápida, provocando que documentación relativamente reciente esté muchas veces obsoleta y resulte inservible. Esto es debido a que, en ocasiones, al producirse cambios de versión, existen problemas de compatibilidades, lo cual también ha contribuido a que la mayoría de tareas relativas a la fase de elaboración del prototipo acabasen llevando más tiempo del planificado.

8.3.2 Estructura

En esta sección se describe la estructura elaborada a la hora de implementar y de distribuir los diferentes ficheros que forman parte del prototipo. Para ello, se describen los contenidos más importantes de los 5 directorios principales encontrados en el directorio raíz del prototipo.

8.3.2.1 Directorio industrialBusinessNetwork

Como se ha mencionado en 7.2.1, Composer usa el término *business network* para referirse a un contrato inteligente. Esta carpeta contiene la implementación del contrato destinado a ser instanciado en cada canal de la red Fabric. De cara a su implementación, 3 ficheros se ven involucrados:

- **Fichero de modelado (.cto):** En este archivo se definen los tipos de participantes de la red (*participants*), los elementos que se van a guardar en el estado actual (*assets*), las funciones que tendrá el contrato inteligente (*transactions*), y los eventos que la red va a emitir cuando ciertas condiciones se cumplan (*events*).
- **Fichero de lógica (.js):** En este fichero se sitúa el código que se va a ejecutar al realizar cada una de las transacciones definidas en el fichero de modelado.
- **Fichero de permisos (.acl):** De cara a establecer qué participantes tienen permiso para realizar determinadas acciones se usa este fichero. El acceso a un canal es controlado por el MSP, pero, una vez que se consigue acceso, los permisos dentro del propio canal se controlan con este fichero. Es decir, una identidad perteneciente al proveedor de fresas no podría tener acceso al *CanalManzanas*, porque el MSP de dicho canal lo impediría. Sin embargo, una identidad perteneciente al proveedor de manzanas sí tendría acceso a dicho canal, pero, una vez dentro, no podría, por ejemplo, realizar transacciones de entrega de lotes en el canal, ya que este fichero .acl lo impediría.

Una vez que estos archivos se han definido, se unen los tres para formar un fichero con extensión .bna, el cual es realmente el contrato inteligente y es el fichero que se instala en los nodos de la red. Esta es la manera que tiene este framework para la elaboración de contratos.

En este directorio también se guarda la información criptográfica (claves privadas y certificados) de los administradores de los contratos inteligentes introducidos en 7.2.1, así como sus tarjetas.

Para finalizar, se guardan también en este lugar las políticas de aprobación de transacciones relativas a cada canal, en las cuales, en el caso de este prototipo, se ha establecido que, para que una transacción sea aprobada en un determinado canal, todas las organizaciones presentes en dicho canal deben aprobarla.

8.3.2.2 Directorio scripts

Este directorio contiene los siguientes scripts principales:

- **Creación de la red (crearRedFabric.sh):** Este script se encarga de crear una red desde cero, iniciando todos los contenedores, creando todos los canales, generando las identidades de administración, instalando los contratos inteligentes en los nodos adecuados e instanciando los contratos en cada canal. El script incluye los comentarios que se han considerado necesarios para su correcta comprensión, y, mediante su elaboración, se ha automatizado la creación de la red Fabric.
- **Creación de tarjetas de administradores de hardware:** Estos administradores han sido mencionados en 7.2.1. Para la creación de sus tarjetas (las cuales también son almacenadas en este directorio) se han elaborado también una serie de scripts.

Dentro de este directorio existe otra carpeta llamada *composer*. Los principales contenidos que se encuentran en su interior son los siguientes:

- **Información criptográfica de cada empresa:** Para cada empresa, la información criptográfica principal que se incluye es la siguiente:
 - **Entidad certificadora:** Clave privada y certificado de la CA en la cual la empresa confía.
 - **Nodos:** Clave privada y certificado del nodo perteneciente a cada empresa.
 - **Administradores de hardware:** Clave privada y certificado de cada administrador de hardware.
- **Fichero docker-compose.yml:** En este archivo se definen y configuran todos los contenedores que se comunicarán entre sí para formar la red Fabric.
- **Fichero crypto-config.yaml:** En este fichero se define la lista de organizaciones que formarán parte de la red y los nodos y administradores de hardware que tendrá cada una.

8.3.2.3 Directorio bin

De cara a generar la información criptográfica mencionada en 8.3.2.2 se hace uso de la herramienta *cryptogen*, la cual se incluye en este directorio. Para generar dicha información toma como entrada el fichero *crypto-config.yaml*.

8.3.2.4 Directorio servidor

En este directorio se encuentra el código relativo a la interacción con la red Fabric, tanto el que se ejecutará en el servidor como el que lo hará en el cliente. También es este directorio donde se encuentra el código mediante el cual se generan las tarjetas de cada participante en la red.

8.3.2.5 Directorio existenciaDocumentos

Por último, esta carpeta contiene el código relativo a la gestión documental realizada con Ethereum. Sus principales contenidos son los siguientes:

- **Contrato inteligente:** El archivo llamado *Existencia.sol* es el contrato inteligente elaborado en Solidity que se desplegará en la red Ethereum. Se han incluido comentarios a lo largo de este archivo de cara a explicar y aclarar diferentes detalles.
- **Código de cliente:** El código HTML, CSS y JavaScript que se ejecutará en el cliente y permitirá interactuar con la red pública.
- **Archivos de configuración de Truffle:** Truffle se ha utilizado para el proceso de despliegue del contrato en la red, y, para ello, se han tenido que configurar los ficheros de los que hace uso esta herramienta.

En una de las subcarpetas de este directorio, llamada *documentos*, se incluyen algunos de los documentos que se han registrado en la red Ropsten.

Capítulo 9: Desarrollo de Pruebas

En este capítulo se hace referencia a los procesos de prueba que han sido llevados a cabo sobre el prototipo elaborado.

9.1 Composer Playground

De cara a realizar las pruebas relativas a la interacción con la red Fabric se ha hecho uso de la herramienta llamada Composer Playground [81]. Esta herramienta ofrece una interfaz de usuario web a través de la cual es posible conectarse a una red Fabric existente y hacer pruebas relacionadas, entre otros, con la ejecución de transacciones o la gestión de permisos.

Una vez que se inicia, Composer Playground permite seleccionar cualquiera de las tarjetas disponibles y conectarse a una red con ella. Tras conectarse, ofrece la posibilidad de realizar cualquier tipo de transacción, hacer modificaciones en el archivo de permisos, realizar cambios y redeployar el contrato desde la propia interfaz o comprobar a qué datos tiene acceso una identidad determinada. La herramienta ha sido creada recientemente de cara a probar la comunicación con redes Fabric, y es cierto que su uso es cómodo y sencillo, ya que al realizar operaciones ofrece mensajes descriptivos acerca del resultado, tanto si la operación se completó como si hubo algún tipo de error.

Además, permite conectarse a una red mediante tarjetas de administradores de contratos inteligentes, para así poder crear nuevos participantes.

En la Figura 9.1 y en la Figura 9.2 se muestra la interfaz que ofrece Composer Playground de cara a probar el funcionamiento de una red Fabric. En la primera se aprecia cómo la herramienta permite editar los ficheros que componen el contrato inteligente dinámicamente y redeployar una nueva versión del contrato, y en la segunda se muestra la interfaz ofrecida para probar el funcionamiento de transacciones y el acceso a datos. Posteriormente, en la Tabla 9.1, se incluyen los procedimientos de prueba llevados a cabo con esta herramienta.

```

30 rule SolamenteVehiculosPuedenEnviar {
31   description: "Permitir a los vehiculos realizar transacciones de envio de lotes"
32   participant: "org.tfg.industrialnetwork.Vehiculo"
33   operation: CREATE
34   resource(r): "org.tfg.industrialnetwork.Enviar"
35   condition: (r.lote.estado === "ALMACENADO")
36   action: ALLOW
37 }
38
39 rule SolamenteVehiculosPuedenEntregar {
40   description: "Permitir a los vehiculos realizar transacciones de entrega de lotes"
41   participant: "org.tfg.industrialnetwork.Vehiculo"
42   operation: CREATE
43   resource(r): "org.tfg.industrialnetwork.Entregar"
44   condition: (r.lote.estado === "ENVIADO")
45   action: ALLOW
46 }
47
48 rule SolamenteDestinatariosPuedenUtilizar {
49   description: "Permitir a los destinatarios reales de los lotes que utilicen su contenido"
50   participant(p): "org.tfg.industrialnetwork.Destinatario"
51   operation: CREATE
52   resource(r): "org.tfg.industrialnetwork.Utilizar"
53   condition: (r.lote.destinatario.getIdentifier() === p.getIdentifier() && r.lote.estado === "RECIBIDO")
54   action: ALLOW
55 }

```

FILES

- About README.md, package.json
- Model File models/org.tfg.industrialnetwork.cto
- Script File lib/logic.js
- Access Control permissions.acl

Add a file... Export

UPDATE NETWORK

From: 0.0.1 To: 0.0.2-deploy.o

Deploy changes

Legal GitHub

badminproveedorfresas

Playground v0.20.6 Tutorial Docs Community

Figura 9.1: Interfaz ofrecida por Composer Playground (1)

Participant registry for org.tfg.industrialnetwork.Creador

+ Create New Participant

ID	Data

This registry is empty!

To create resources in this registry click create new at the top of this page

Submit Transaction

Legal GitHub

Playground v0.20.6 Tutorial Docs Community

Figura 9.2: Interfaz ofrecida por Composer Playground (2)

Crear lote		
Prueba realizada	Resultado esperado	Resultado obtenido
Un participante de tipo Creador realiza una transacción de creación de un lote.	La transacción se confirma, el lote se añade solamente al estado actual del canal reflejado en la tarjeta del	Prueba superada.

	participante y su estado es “almacenado”.	
Un participante de tipo distinto a Creador realiza una transacción de creación de un lote.	La transacción no se confirma y el lote no se añade al estado actual de ningún canal.	Prueba superada.
Enviar lote		
Prueba realizada	Resultado esperado	Resultado obtenido
Un participante de tipo Vehículo realiza una transacción de envío de un lote, cuyo estado es “almacenado”.	La transacción se confirma, el lote cambia a estado “enviado” y su fecha de envío refleja el momento en el que se produjo la transacción.	Prueba superada.
Un participante de tipo Vehículo realiza una transacción de envío de un lote, cuyo estado es diferente de “almacenado”.	La transacción no se confirma y el lote no sufre modificaciones.	Prueba superada.
Un participante de tipo distinto a Vehículo realiza una transacción de envío de un lote.	La transacción no se confirma y el lote no sufre modificaciones.	Prueba superada.
Entregar lote		
Prueba realizada	Resultado esperado	Resultado obtenido
Un participante de tipo Vehículo realiza una transacción de entrega de un lote, cuyo estado es “enviado”.	La transacción se confirma, el lote cambia a estado “recibido” y su fecha de entrega refleja el momento en el que se produjo la transacción.	Prueba superada.
Un participante de tipo Vehículo realiza una transacción de entrega de un lote, cuyo estado es diferente de “enviado”.	La transacción no se confirma y el lote no sufre modificaciones.	Prueba superada
Un participante de tipo distinto a Vehículo realiza una transacción de entrega de un lote.	La transacción no se confirma y el lote no sufre modificaciones.	Prueba superada.
Utilizar lote		
Prueba realizada	Resultado esperado	Resultado obtenido

Un participante de tipo Destinatario realiza una transacción de utilización de un lote, cuyo estado es “recibido” y cuyo destinatario es el mismo que el realizador de la transacción.	La transacción se confirma, el lote cambia a estado “utilizado” y su fecha de utilización refleja el momento en el que se produjo la transacción.	Prueba superada.
Un participante de tipo Destinatario realiza una transacción de utilización de un lote, cuyo estado es “recibido” y cuyo destinatario es diferente al realizador de la transacción.	La transacción no se confirma y el lote no sufre modificaciones.	Prueba superada.
Un participante de tipo Destinatario realiza una transacción de utilización de un lote, cuyo estado es diferente de “recibido”.	La transacción no se confirma y el lote no sufre modificaciones.	Prueba superada.
Un participante de tipo distinto a Destinatario realiza una transacción de utilización de un lote.	La transacción no se confirma y el lote no sufre modificaciones.	Prueba superada.
Leer datos		
Prueba realizada	Resultado esperado	Resultado obtenido
Un participante de tipo Creador, Vehículo o Destinatario accede al canal de la red Fabric reflejado en su tarjeta.	El participante solamente puede ver la información de los lotes que se encuentran en dicho canal, pero no la de los lotes que se encuentran en otros canales.	Prueba superada.

Tabla 9.1: Pruebas realizadas con Composer Playground

9.2 Ganache

En lo relativo a las pruebas realizadas en la parte del prototipo hecha con Ethereum, se ha empleado la herramienta Ganache [82]. Para ejecutar transacciones en una red Ethereum se necesita disponer de Ether, lo cual no es un inconveniente a la hora de realizar pruebas gracias a esta herramienta. Ganache se encarga de simular una red Ethereum en una sola máquina, crear una serie de cuentas y proporcionarles Ether que solamente tendrá valor en la red simulada, por tanto, de cara a hacer pruebas, se puede usar una de estas cuentas para realizar la transacción de despliegue del contrato inteligente en esta red y comenzar a realizar transacciones invocando a sus funciones. Ganache proporciona una interfaz de usuario en la que se pueden ver las transacciones y los bloques que se van generando. Para cada transacción, se puede observar si es

válida o inválida, y datos como el gas que ha consumido su ejecución. Además, al ser una red local, las transacciones se ejecutan muy rápido, haciendo que el proceso de las pruebas relativas a la funcionalidad de un contrato sea mucho más cómodo que si estas tuviesen que hacerse en una red pública, ya que el rendimiento en una de estas redes sería inferior, provocando que el tiempo necesario para realizar las pruebas fuese muy superior.

En la Figura 9.3 se muestra la interfaz de usuario ofrecida por Ganache y en la Tabla 9.2 se incluyen las pruebas concretas realizadas con la ayuda de esta herramienta.

ADDRESS	BALANCE	TX COUNT	INDEX	
0x5F67E5c5540a6242ba55ab838F28921dd2106db8	100.00 ETH	0	0	
0x2A41F343cd20b7fa8d6982f3AD0c0A446A67B039	100.00 ETH	0	1	
0x0490c8Aa89B8a1220FE472b04Bad8b037cCd4A88	100.00 ETH	0	2	
0xa04ccB55950F728125DD89AD0a18a13566545D47	100.00 ETH	0	3	
0x1Bd8C44A99aBD1C9124F02121aCF92B5F1120EA8	100.00 ETH	0	4	
0xC95A41581EEF783bA8DdAb6b040c6ff4bcc48aa7	100.00 ETH	0	5	
0x1fdFE1F04e54EdCe89d5BEf34469F653b92679BE	100.00 ETH	0	6	

Figura 9.3: Interfaz ofrecida por Ganache

Registrar documento		
Prueba realizada	Resultado esperado	Resultado obtenido
Se realiza una transacción de registro de un documento desde una de las tres cuentas de administración válidas.	La transacción se confirma, se añade el hash del documento y la fecha de registro al estado actual relativo a la cuenta de contrato.	Prueba superada.
Se realiza una transacción de registro de un documento desde una cuenta diferente a las tres cuentas de administración.	La transacción no se confirma y el estado actual de la cuenta de contrato no sufre cambios.	Prueba superada.
Comprobar existencia de un documento		

Prueba realizada	Resultado esperado	Resultado obtenido
Invocar la función de lectura del contrato inteligente de cara a comprobar la existencia de un documento que se ha registrado.	La respuesta recibida incluye una fecha válida en la que fue registrado el documento.	Prueba superada.
Invocar la función de lectura del contrato inteligente para comprobar la existencia de un documento que no ha sido registrado previamente.	La respuesta recibida incluye una fecha de registro inválida, ya que el documento no está registrado.	Prueba superada.

Tabla 9.2: Pruebas realizadas con Ganache

9.3 Ropsten y Etherscan

A lo largo de todo este documento se ha estado mencionando que parte del prototipo interactuaría con una red Ethereum pública, pero no se ha dicho hasta ahora en qué red concreta se iba a desplegar el contrato inteligente. Desplegarlo en la red principal de Ethereum implicaría un coste monetario, ya que para hacer operaciones en esta red se necesita estar en disposición de Ether verdadero, el cual tiene un valor monetario real, razón por la cual no se ha decidido desplegar el contrato en la red principal. Sin embargo, existen otras redes públicas Ethereum, algunas de ellas son las conocidas como redes de prueba, las cuales están creadas explícitamente para probar el funcionamiento de un contrato en una red pública real. Hay varias redes de prueba, cada una con sus peculiaridades, tamaño y método de consenso. Para interactuar con redes de prueba también es necesario tener Ether, pero en estas redes no posee valor monetario real y se puede adquirir de manera gratuita. En este prototipo, se ha decidido desplegar el contrato en la red de prueba conocida como Ropsten, debido a que su método de consenso es el mismo que el existente en la red principal, la prueba de trabajo. Desplegar el contrato en una de estas redes es un paso normal antes de hacerlo en la red principal, ya que se puede observar su comportamiento en condiciones muy similares a las que habría en la red principal.

Para comprobar las transacciones que se realizan en una red pública existen herramientas de búsqueda que se conectan a estas redes y ofrecen información acerca de ellas en tiempo real, mostrando tanto los bloques y transacciones que van generando como los que se han generado en el pasado. La más popular de estas herramientas es Etherscan [83], la cual ha sido utilizada en este trabajo para comprobar los resultados de la ejecución de transacciones en la red Ropsten, ya que, además de ofrecer los datos relativos a la red pública, Etherscan también es capaz de conectarse a las redes de prueba más importantes y ofrecer información acerca de todo su historial.

La dirección de la cuenta de contrato en la que se ha desplegado el contrato elaborado en este proyecto es la siguiente: 0x71FA211d7160BF2f1112A15932c05890438e358E.

En la Figura 9.4 y la Figura 9.5 se muestran los datos ofrecidos por Etherscan acerca del contrato desplegado en la red pública de Ropsten (se muestran dos figuras de cara a no mostrar publicidad). En la primera de ellas se observa arriba a la derecha que Etherscan está mostrando información acerca de la red Ropsten, y que esa información corresponde con la cuenta de

contrato cuya dirección se ha indicado anteriormente. En la segunda se muestran todas las transacciones que he realizado sobre el contrato hasta ahora. Algunas de ellas se han realizado desde direcciones que no correspondían a ninguno de los administradores de las empresas de transporte, las cuales aparecen marcadas como inválidas. Al seleccionar una determinada transacción, se pueden comprobar datos adicionales relativos a su ejecución.

The screenshot shows the Etherscan interface for the Ropsten Testnet Network. At the top, there is a search bar labeled "Search by Address / Txn Hash / Block / Token / Ens" with a magnifying glass icon. Below the search bar are navigation links: "All Filters", "Home", "Blockchain", "Tokens", "Misc", and "Ropsten". The main content area has two sections: "Contract Overview" and "More Info". In the "Contract Overview" section, it shows a balance of "0 Ether". In the "More Info" section, it shows "My Name Tag: Not Available" and "Contract Creator: 0xffff07c57f4f5d7426... at txn 0x748b60355e8253...". Below these sections, a table lists the "Latest 9 txns" for the contract, showing details like Txn Hash, Block, Age, From, To, Value, and Txn Fee. One transaction is highlighted in red, indicating it is invalid.

Figura 9.4: Interfaz ofrecida por Etherscan (1)

This screenshot shows the transaction history for the My Name Tag contract on the Etherscan interface. The "Transactions" tab is selected. The table displays nine transactions from the last 9 hours. The columns include Txn Hash, Block, Age, From, To, Value, and Txn Fee. One transaction is highlighted in red, indicating it is invalid. A "Download CSV Export" button is located at the bottom right of the table.

Figura 9.5: Interfaz ofrecida por Etherscan (2)

Los procedimientos de prueba llevados a cabo sobre el contrato desplegado en la red Ropsten han sido los mismos que los realizados sobre el contrato desplegado en la red creada en local con Ganache, los cuales han sido descritos en la Tabla 9.2.

9.4 Navegadores Web

Por último, la interfaz de usuario proporcionada por el cliente se ha probado en dos navegadores web: Google Chrome y Mozilla Firefox. Sin embargo, la extensión de MetaMask solamente se ha instalado en Google Chrome y únicamente se ha comprobado su correcto funcionamiento desde este navegador, por lo tanto, se recomienda que a la hora de utilizar este prototipo se use Google Chrome. Aunque, si la extensión de MetaMask tiene un funcionamiento similar en los

demás navegadores web, el prototipo no debería presentar inconvenientes para poder interactuar con la red de Ropsten desde ellos.

Capítulo 10: Manuales del Sistema

En el actual capítulo se incluyen los manuales relativos a la instalación, ejecución y uso del sistema.

10.1 Manual de Instalación

En este manual se hace mención a los procesos de instalación y configuración necesarios para poder ejecutar el sistema.

La preparación del entorno para ejecutar el prototipo es un poco laboriosa y lleva un tiempo. No obstante, se indican en esta sección los pasos a seguir para configurar un entorno desde cero. En mi caso, el ordenador que he usado para desarrollar y ejecutar el prototipo ha sido un MacBook Air con sistema operativo macOS Mojave, 8 GB de memoria RAM y un procesador Intel Core i5 de 1.6 GHz. Sin embargo, también he ejecutado y probado el prototipo en el sistema operativo Ubuntu, aunque menos exhaustivamente que en macOS.

Para comenzar este proceso de instalación, es necesario tener el código del prototipo en la máquina en la que se vaya a ejecutar. Como primera tarea, se debe modificar en el archivo *crearRedFabric.sh* el valor asignado a la variable llamada *DIRECTORY*, situada en la primera línea de la función contenida en dicho script. Dicha variable tiene que poseer la ruta absoluta de la carpeta en la que el archivo *crearRedFabric.sh* esté situado en la máquina que se vaya a ejecutar.

A día de hoy, es posible la instalación de Hyperledger Fabric en un sistema operativo Windows, pero el proceso es más tedioso que para macOS o distribuciones Linux. Sin embargo, Hyperledger Composer no está disponible para Windows, y en su documentación solamente se muestra la información para ser instalado en macOS y Ubuntu.

Los procesos de instalación de Hyperledger Composer son descritos en [84] y en [85], durante los cuales también se descargan todos los archivos relativos a Hyperledger Fabric.

Nota: En mi caso particular, he usado la versión de Node.js 8.9.4 a la hora de realizar toda operación relativa a Hyperledger Composer (tanto de instalación como de ejecución), ya que este framework no soporta versiones diferentes de 8.9.x. Sin embargo, para la realización de todas las operaciones relacionadas con Ethereum (de nuevo tanto de instalación como de ejecución) he empleado una versión de Node.js más reciente, la 11.3.0. Para cambiar de versión de Node.js se debe ejecutar el siguiente comando: *nvm use versionAdecuada*, por ejemplo, *nvm use 8.9.4*.

Por último en lo relativo a Fabric, es necesario instalar las dependencias, para ello se debe ejecutar *npm install* (usando la versión de Node.js 8.9.4) situándose en el directorio *servidor*.

En lo relacionado a Ethereum, se han seguido los siguientes pasos para configurar el entorno:

- **Instalar Truffle:** Usando el comando *npm install -g truffle@5.0.24*.
- **Instalar dependencias:** Con el comando *npm install* (usando la versión de Node.js 11.3.0) situándose en el directorio *existenciaDocumentos/truffle/app*, y, en caso de encontrarse en Ubuntu, también será necesario ejecutar dicho comando en el directorio *existenciaDocumentos/truffle*.

- **Compilar el contrato inteligente usando Truffle:** Dentro del directorio *existenciaDocumentos/truffle* se debe ejecutar el comando *truffle compile*. Tras ello, se generará un archivo con el nombre *Existencia.json* dentro de la carpeta *existenciaDocumentos/truffle/build/contracts*. En la zona final de dicho archivo se debe realizar una modificación. Concretamente, tras generarse, la sección de *networks* estará vacía (“*networks*”: {}). Con el objetivo de hacer referencia en dicha sección a la dirección de la cuenta donde se encuentra el contrato inteligente desplegado en la red Ropsten, se deben introducir entre dichas llaves los siguientes contenidos (la red Ropsten se identifica con el número 3):

```
"3": {  
    "events": {},  
    "links": {},  
    "address": "0x71FA211d7160BF2f1112A15932c05890438e358E",  
    "transactionHash":  
    "0x748b60355e82538e181d9e95a7ba728963b22ad47df4c4484894f51b25c655e2"  
}
```

- **Instalar MetaMask:** Instalar la extensión de MetaMask en el navegador Google Chrome. Tras ello, puede ser usada para realizar transacciones en la red Ropsten, sin embargo, previamente, debe configurarse para que esas transacciones se puedan hacer desde las cuentas de los administradores de las empresas de transporte. Para ello, MetaMask debe saber las claves privadas de estas cuentas, lo cual es posible gracias a las conocidas como frases mnemónicas. Una frase mnemónica es un conjunto de 12 palabras que se deben mantener en secreto, y que permiten generar siempre un mismo conjunto de claves privadas, por lo tanto, sabiendo estas 12 palabras, se pueden generar las claves privadas, y, consecuentemente, también las direcciones de un conjunto de cuentas Ethereum. MetaMask ofrece la posibilidad de insertar una frase mnemónica para generar unas claves y direcciones concretas. La frase mnemónica para la generación de las claves privadas y direcciones de las cuentas de administración se entrega en un fichero de texto como parte de este trabajo fin de grado. Una vez insertada dicha frase en MetaMask, aparecerá solamente una cuenta de administración, para que aparezcan las dos restantes se deben crear dos cuentas, lo cual puede hacerse de manera muy sencilla a través de la propia extensión. Adicionalmente, también será requerido desactivar el modo privado con el que MetaMask ha estado experimentando durante los últimos meses, lo cual se puede hacer desde la sección de ajustes de la extensión. Por último en lo relativo a MetaMask, antes de realizar transacciones en la red Ropsten, es necesario adquirir Ether, lo cual puede hacerse de manera gratuita a través también de la extensión, sin embargo, las cuentas de administración tendrán Ether insertado, ya que he tenido que hacerlo para poder realizar transacciones.

Nota: En el caso de Ubuntu, si no permite ejecutar los scripts, se les deben proporcionar permisos de ejecución, usando para ello el comando *chmod +x nombreArchivo* (Por ejemplo: *chmod +x crearRedFabric.sh*).

10.2 Manual de Ejecución

Con todo el software instalado y el entorno configurado, es posible proceder a la ejecución del prototipo.

De cara a poner en marcha el sistema completo, es necesario ejecutar tanto la parte relativa a la interacción con la red de consorcios como la relativa a la red pública. Su ejecución se trata por separado, y, para que el cliente pueda comunicarse con ambas redes a la vez, ambas partes deben encontrarse en ejecución.

Para poner en marcha la parte relativa a Hyperledger Fabric se deben seguir los siguientes pasos (usando siempre la versión 8.9.4 de Node.js):

- **Iniciar la red:** Situarse en el directorio *scripts* y ejecutar el comando `./crearRedFabric.sh` de cara a iniciar la red Fabric. La ejecución completa de este script lleva unos minutos, ya que son muchas las operaciones que realiza. La primera vez que se ejecuta necesita crear una serie de archivos, llamados imágenes, que son las plantillas para que Docker construya los contenedores. Cada una de estas imágenes ocupa una cantidad sustancial de espacio y se tardan en generar un tiempo. Por lo tanto, la primera vez que se ejecuta este script tarda ciertos minutos en finalizar. En ocasiones, no es muy común, pero tampoco es demasiado extraño que ocurran errores relativos a tiempos de espera a la hora de crear estas imágenes. Si esto sucede, se debe parar la ejecución del script y volver a ejecutarlo. Las imágenes solamente se generan la primera vez que se ejecuta. En sucesivas ejecuciones usará las imágenes creadas previamente, sin embargo, aunque no es frecuente, ya que solamente me ha pasado una vez de entre las muchas que lo he ejecutado, también en futuras ejecuciones pueden ocurrir errores de tiempos de espera. En este caso, el script deberá ser ejecutado de nuevo hasta que todos los procesos se ejecuten adecuadamente. El proceso de ejecución de este script se puede seguir mediante mensajes mostrados en la terminal.
- **Iniciar el servidor:** Situarse en el directorio *servidor* y ejecutar el comando `node app.js` para poner en marcha el servidor.
- **Abrir el cliente:** Desde un navegador web, ir a la dirección `localhost:3000`, en la cual el servidor acepta peticiones, para solicitar el código HTML, CSS y JavaScript del cliente y que el navegador muestre la interfaz de usuario perteneciente a la red de consorcios, desde la cual también existe la posibilidad de acceder a la interfaz relativa a la interacción con la red pública. En este prototipo se pueden tener diferentes clientes abiertos simultáneamente, pudiendo estar cada uno de ellos conectado a un canal diferente de la red. La primera vez que se ejecuta el código del cliente se procede a crear todas las tarjetas de los 15 participantes, proceso que se puede seguir a través de la línea de comandos, de modo que, hasta que no se creen las tarjetas de los participantes, no será posible acceder a la red con ellos. Este proceso solamente se realiza una vez y puede llevar aproximadamente 2 minutos. De nuevo, no es común y solamente me ha pasado en una ocasión, pero, a la hora de generar las tarjetas de los participantes, puede darse el caso de que algunas no se generen debido a un error de tiempo de espera. Si esto ocurre, lo más adecuado será volver a iniciar la red, el servidor, y abrir el cliente de nuevo.

De cara a la puesta en marcha de la parte relacionada con la red pública, solamente será necesario ejecutar el cliente, el cual debe solicitar el código necesario para comunicarse con la red Ethereum. Para ello, se deben realizar las siguientes acciones (empleando la versión 11.3.0 de Node.js):

- **Ejecutar Webpack:** Situarse en el directorio *existenciaDocumentos/truffle/app* y ejecutar el comando *npm run dev*, lo cual hará que Webpack componga los archivos adecuados que el cliente ejecutará.
- **Solicitar archivos:** Desde un navegador web, ir a la dirección *localhost:8080* para conseguir los archivos necesarios generados por Webpack. Una vez conseguidos estos archivos, se muestra en el navegador la interfaz web y se puede comenzar a interactuar con la red pública.

Nota: Dependiendo de los permisos de los que se dispongan en la máquina donde se ejecute el prototipo, será necesario ejecutar algunos de estos comandos precedidos de *sudo* si dicha máquina cuenta con Ubuntu como sistema operativo.

10.3 Manual de Usuario

Tras explicar cómo se ejecuta el sistema, se tratan en esta sección lo aspectos relativos a su uso.

A través de la interfaz elaborada para interactuar con la red de consorcios se pueden crear, enviar, entregar y utilizar lotes. Dependiendo del tipo de participante que se encuentre seleccionado, determinados botones se encontrarán inactivos, para indicar visualmente las transacciones que puede realizar dicho participante.

Al iniciarse esta interfaz no se muestra ningún lote, para ello, es necesario pulsar en los recuadros donde se muestra el número de lotes que se encuentran en cada estado, permitiendo así mostrar en pantalla solamente los que el usuario desee.

De cara a seleccionar el participante que va a realizar transacciones, se debe pulsar en el ícono situado arriba a la derecha, el cual despliega una lista (en la cual se puede hacer scroll) que contiene los 15 participantes en la red Fabric, y seleccionar posteriormente el que se deseé.

En la Figura 10.1 se incluye una visión global de la interfaz para interactuar con la red Fabric y en la Figura 10.2 se muestra la selección de participantes.

The screenshot displays the 'SimuladorTFG' application interface, which facilitates interaction with a Fabric network for supply chain management. The interface includes a sidebar with navigation links: 'Cuarto de mando' and 'Documentos'. The main content area is organized into several sections:

- Datos del participante seleccionado:** Shows participant details: Tipo: Creador, Empresa: ProveedorManzanas, Localización de la planta: Luarca, Canal: canalmanzanas. Below are four buttons: Lotes en estado almacenado: 2, Lotes en estado enviado: 2, Lotes en estado recibido: 1, and Lotes en estado utilizado: 1, each with a 'Dejar de mostrar' link.
- Creación de un lote:** A form for creating a new lot. It includes fields for Contenido (Content) and Masa (kg) (Weight). Below these are fields for Destinatario (Recipient) and a dropdown menu showing 'ProductorMermeladaFabricaOviedoCManz'. A large green button at the bottom right is labeled 'Crear lote' (Create lot).
- Detalles del lote seleccionado:** Displays detailed information for a selected lot. Key details include Id: ProveedorManzanasPlantaLuarca1562170321421, Creador: ProveedorManzanasPlantaLuarca, Destinatario: ProductorMermeladaFabricaOviedoCManz, Estado: UTILIZADO, Contenido: Manzanas Golden, Masa (kg): 14, Fecha de creación: 3/6/2019 18:12:1, Fecha de envío: 3/6/2019 18:13:4, Fecha de entrega: 3/6/2019 18:14:31, and Fecha de utilización: 3/6/2019 18:15:20.
- Lotes en estado almacenado:** A table listing stored lots. It has columns: Identificador (Identifier), Contenido (Content), Masa (kg) (Weight), Información (Information), and Transacciones (Transactions). Two entries are shown:

Identificador	Contenido	Masa (kg)	Información	Transacciones
ProveedorManzanasPlantaLuarca1562170350166	Manzanas Fuji	25	Detalles	Enviar
ProveedorManzanasPlantaLugo1562170252336	Manzanas Fuji	18	Detalles	Enviar
- Lotes en estado enviado:** A table listing shipped lots. It has columns: Identificador (Identifier), Contenido (Content), Masa (kg) (Weight), Información (Information), and Transacciones (Transactions). Two entries are shown:

Identificador	Contenido	Masa (kg)	Información	Transacciones
ProveedorManzanasPlantaLuarca1562170437487	Manzanas Gala	26	Detalles	Entregar
ProveedorManzanasPlantaLugo1562170209669	Manzanas Golden	15	Detalles	Entregar
- Lotes en estado recibido:** A table listing received lots. It has columns: Identificador (Identifier), Contenido (Content), Masa (kg) (Weight), Información (Information), and Transacciones (Transactions). One entry is shown:

Identificador	Contenido	Masa (kg)	Información	Transacciones
ProveedorManzanasPlantaLugo1562170284179	Manzanas Gala	20	Detalles	Utilizar
- Lotes en estado utilizado:** A table listing used lots. It has columns: Identificador (Identifier), Contenido (Content), Masa (kg) (Weight), Información (Information), and Transacciones (Transactions). One entry is shown:

Identificador	Contenido	Masa (kg)	Información	Transacciones
ProveedorManzanasPlantaLuarca1562170321421	Manzanas Golden	14	Detalles	-

At the bottom center of the interface, there is a copyright notice: © 2019 Pedro Fernández Álvarez.

Figura 10.1: Interfaz para interactuar con la red Fabric

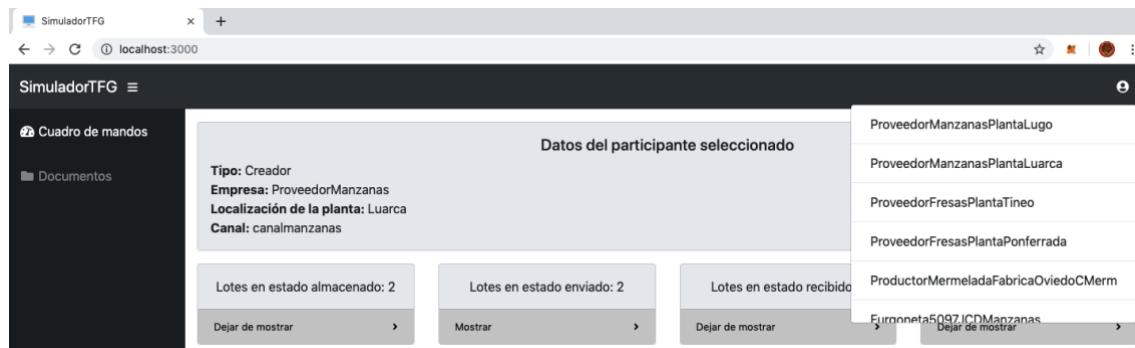


Figura 10.2: Selección de participante en la red Fabric

Si se pulsa en la parte relativa a los documentos en el menú de la izquierda, se accede a la pantalla mostrada en la Figura 10.3, en la cual se incluye un enlace a la pantalla de interacción con la red pública. Este enlace abre una nueva pestaña en *localhost:8080*, por lo tanto, para su correcto funcionamiento, se debe tener en ejecución la parte del prototipo relacionada con la red pública para poder acceder al código del cliente.



Figura 10.3: Interfaz de enlace a la gestión documental

Bien tras pulsar en el anterior enlace o bien tras ir a la dirección *localhost:8080* de manera directa, la interfaz mostrada será la incluida en la Figura 10.4, en la cual se muestran las direcciones de los administradores de cada empresa de transportes. Mediante esta interfaz es posible registrar un documento en formato de texto y comprobar la existencia de documentos registrados. Para ello, en cada caso, se debe seleccionar el documento adecuado almacenado en la máquina en la que se ejecute el prototipo y pulsar posteriormente el botón correspondiente.

Direcciones de los administradores

Empresa de transporte de manzanas: 0xFF07c57F4F5D74269124CaB2327f16e0096D84B9
 Empresa de transporte de fresas: 0x163Dcba9Bbc7DC1831402089f253f06005b74a8B
 Empresa de transporte de mermelada: 0x281AECaD6E2C4a52bFaCA7C559A666c21eDe41b0

Registro de documentos

Selecciona un documento:
 DocumentoEntregaFresas1.txt

Contenido del documento seleccionado:

Empresa de transporte: TransportistaFresas
 Origen: Planta de envasado de fresas situada en Ponferrada
 Destinatario: Planta de producción de mermelada situada en Oviedo
 Fecha de entrega: 24/06/2019
 Hora de entrega: 15:26
 Vehículo: Furgoneta de la marca AAAA y modelo BBBB
 Conductor del vehículo: Pedro Fernández Álvarez
 Fecha de fabricación del vehículo: Agosto de 2012
 Última inspección del vehículo: Octubre de 2018

Registrar documento

Comprobar existencia de documentos

Selecciona un documento:
 DocumentoEntregaManzanas2.txt

Contenido del documento seleccionado:

Hora de entrega: 17:35
 Vehículo: Furgoneta de la marca CCCC y modelo DDDD
 Conductor del vehículo: Pedro Fernández Álvarez
 Fecha de fabricación del vehículo: Julio de 2013
 Última inspección del vehículo: Enero de 2019
 Identificadores de los lotes entregados:
 - Lote8946378
 - Lote1526358
 - Lote9864035

Comprobar existencia

Figura 10.4: Interfaz para interactuar con la red Ethereum

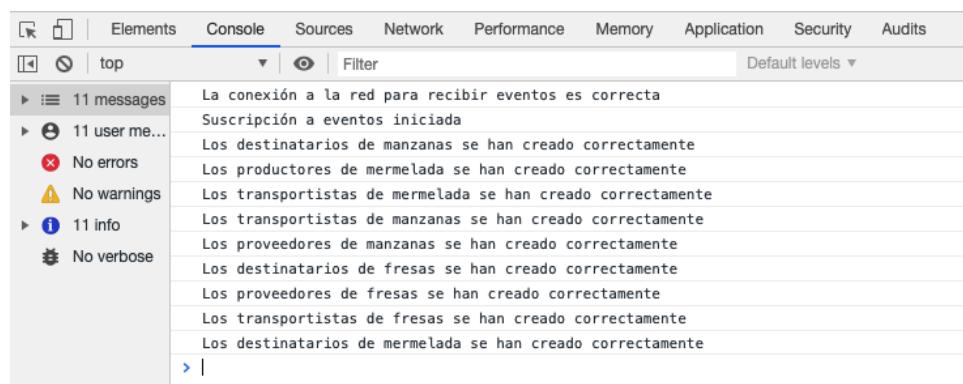
El sistema para seleccionar la cuenta adecuada con la cual registrar documentos es similar al caso de la interfaz para interactuar con la red de consorcios, pero ahora es la extensión de MetaMask la encargada de ofrecer esta funcionalidad. A través de esta extensión (tras haberla configurado con la frase mnemónica adecuada) se puede seleccionar la cuenta de administración con cuya clave privada se quieran firmar las transacciones de registro de documentos. En la Figura 10.5 se muestra la manera en la que MetaMask lo permite.



Figura 10.5: Selección de cuenta mediante MetaMask

Antes de realizar una transacción de registro de un documento, aparecerá una pantalla gestionada por MetaMask en la que se mostrará el Ether que será sustraído de la cuenta que la realiza, y en la que se deberá confirmar el envío de la transacción a la red pública.

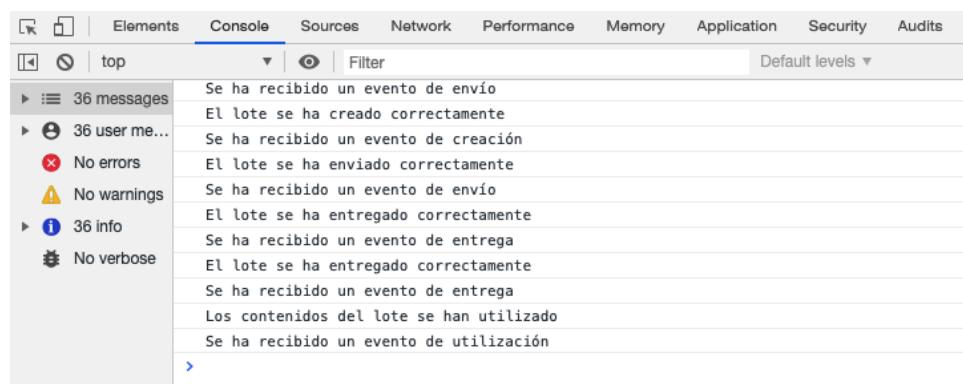
Por último, he decidido que algunos de los procesos se puedan seguir a través de la consola del navegador, esto es opcional para el usuario, y solamente verá estos mensajes si decide abrir la consola, de lo contrario, esta funcionalidad será transparente para él. Incluyo un ejemplo en la Figura 10.6, en la cual se muestran los mensajes a la hora de generar las tarjetas para los participantes y otro en la Figura 10.7, en la que se reflejan mensajes acerca de la realización de transacciones y de los eventos que se reciben de la red Fabric. En el caso de la interacción con Ethereum también he usado la consola en ocasiones para mostrar diferentes detalles. Todos los mensajes mostrados en la consola, relativos tanto a Fabric como a Ethereum, son siempre un reflejo de lo mostrado en la interfaz de usuario.



The screenshot shows the Chrome DevTools Console tab with the following log entries:

- 11 messages
 - La conexión a la red para recibir eventos es correcta
 - Suscripción a eventos iniciada
 - Los destinatarios de manzanas se han creado correctamente
 - Los productores de mermelada se han creado correctamente
 - Los transportistas de mermelada se han creado correctamente
 - Los transportistas de manzanas se han creado correctamente
 - Los proveedores de manzanas se han creado correctamente
 - Los destinatarios de fresas se han creado correctamente
 - Los proveedores de fresas se han creado correctamente
 - Los transportistas de fresas se han creado correctamente
 - Los destinatarios de mermelada se han creado correctamente
- 11 user me...
 - No errors
 - No warnings
- 11 info
 - No verbose

Figura 10.6: Mensajes relativos a la generación de tarjetas de participantes



The screenshot shows the Chrome DevTools Console tab with the following log entries:

- 36 messages
 - Se ha recibido un evento de envío
 - El lote se ha creado correctamente
 - Se ha recibido un evento de creación
 - El lote se ha enviado correctamente
 - Se ha recibido un evento de envío
 - El lote se ha entregado correctamente
 - Se ha recibido un evento de entrega
 - El lote se ha entregado correctamente
 - Se ha recibido un evento de entrega
 - Los contenidos del lote se han utilizado
 - Se ha recibido un evento de utilización
- 36 user me...
 - No errors
 - No warnings
- 36 info
 - No verbose

Figura 10.7: Mensajes relativos a transacciones y eventos en la red Fabric

Capítulo 11: Evaluación

Tras haber analizado la situación actual de la tecnología blockchain, haber realizado un estudio teórico acerca de las dos principales plataformas y haber experimentado con ellas mediante la creación de un prototipo, se procede en este capítulo a realizar una evaluación acerca de la integración actual de esta tecnología en el sector de la industria. Para ello, se tratará cada aspecto a evaluar en un apartado independiente.

11.1 Almacenamiento de Datos

A lo largo de este documento se han argumentado las razones por las cuales almacenar muchos datos en blockchain no es un procedimiento recomendado ni común. Por ello, es importante para una empresa seleccionar los datos de mayor relevancia de cara a ser guardados y mantenidos en una red blockchain.

En el caso de que una compañía quiera gestionar archivos o documentos mediante esta tecnología, lo más adecuado a día de hoy es guardar estos ficheros en un lugar concreto fuera de una red blockchain, y en ella guardar solamente los datos más importantes de estos archivos o los identificadores de su contenido, de cara a poder comprobar en un momento dado que sus contenidos no se han modificado y son los mismos que poseía cuando fue creado, como se ha hecho en el prototipo elaborado en este trabajo.

En cuanto a la visibilidad de los datos almacenados, una red de consorcios elaborada con Hyperledger Fabric proporciona privacidad entre empresas colaboradoras, ya que existen sistemas de control integrados mediante los cuales se controla quién puede realizar transacciones y quién puede acceder a los datos. Sin embargo, una red pública no permite esto de manera nativa, ya que en estas redes la transparencia es total entre los colaboradores, y debido a que todo el mundo puede ser miembro de una red pública, los datos pueden ser accesibles potencialmente por cualquier persona. Por lo tanto, si una empresa decide hacer uso de una red pública para gestionar o almacenar datos de algunos de sus procesos, deberá seleccionar minuciosamente qué datos almacenar.

Los datos que se almacenan en una red pública dependen en gran medida también de la política de la empresa y del grado de transparencia que quiera potenciar hacia sus clientes. Si una compañía desea ser completamente transparente acerca de determinados datos acerca de algunos de sus procesos, podrá gestionar estos datos en la red pública y permitir que todo el mundo que quiera pueda verlos. Sin embargo, si una empresa quiere aprovechar una red pública para almacenar determinados datos, pero no quiere que todo el mundo pueda acceder a ellos, también existen opciones, ya que estos datos podrían almacenarse de manera cifrada en la red, permitiendo que todo el mundo pueda acceder a ellos pero que no puedan descifrarlos. En este caso, una empresa A que quiera proporcionar acceso a otra empresa B a determinados datos, podría cifrarlos con la clave pública de la empresa B y tras ello almacenarlos en la red, donde todo el mundo podrá verlos pero solo la empresa B podrá descifrarlos.

Una plataforma no es mejor que otra, sino que el uso de una u otra depende del propósito e intención de las empresas. Si su intención es colaborar entre ellas de manera privada, la mejor solución es una red de consorcios, ya que el acceso a los datos está controlado. Como consecuencia de esto, se pueden guardar datos comprometidos que no deberían ser almacenados

de manera pública para que empresas rivales no tengan acceso a ellos. Pero, si la privacidad no es un factor fundamental y la empresa quiere proyectar transparencia hacia el público, el uso de una red pública puede ser una gran alternativa. Todo depende de las políticas de transparencia de las empresas y la privacidad que deseen poseer. Para cada caso concreto, una plataforma se ajustará mejor que la otra.

En lo relativo exclusivamente a una red de consorcios construida con Hyperledger Fabric, como se ha explicado en 4.2.3.1, existe la posibilidad de proporcionar privacidad dentro de un mismo canal. Por lo tanto, dependiendo de la sensibilidad de los datos que se almacenen y de con quién se quieran compartir, existen diversas opciones que proporcionan diferentes niveles de privacidad, desde una red Fabric haciendo uso de datos privados hasta una red pública global. El uso de datos privados en el prototipo construido en este trabajo no ha sido posible debido a que Hyperledger Composer a día de hoy no soporta esta funcionalidad.

11.2 Permisos en una Red Pública

Hyperledger Fabric, como se ha dicho anteriormente y a lo largo de este trabajo, ofrece un sistema de permisos nativo, del cual Ethereum carece. Sin embargo, si bien es cierto que el estado actual y la cadena de bloques pueden ser leídos por todo el mundo en una red pública, existe la posibilidad de limitar el número de identidades que pueden realizar transacciones relativas a un contrato en este tipo de redes.

Esto es precisamente lo que se ha hecho en este trabajo, construir un sistema de permisos sencillo y estático en el propio contrato para que solamente tres cuentas de administración relativas a cada una de las empresas de transporte pudiesen registrar documentos. Para ello, en el momento del despliegue, se han pasado como argumentos las tres direcciones de administración, las cuales se guardan en el estado actual de la cuenta de contrato. De este modo, al ejecutar la función relativa al registro de un documento, se comprueba si la dirección desde la cual se ha invocado la función pertenece a uno de estos administradores. De ser así, se registra el documento, de lo contrario, el documento no es registrado y la transacción se marca como inválida.

Esta manera de gestionar permisos descrita anteriormente e implementada en este trabajo es estática, es decir, una vez desplegado el contrato e indicadas las direcciones concretas de administración, ya no se pueden cambiar, lo cual presenta el inconveniente de que, si se quiere en el futuro que otra dirección pueda realizar transacciones de registro, no será posible. Como consecuencia de ello, en el caso de que la clave privada de una dirección de administración se haya visto comprometida o de que se quiera añadir otro administrador existirá un problema. Este inconveniente se podría resolver implementando un sistema de permisos dinámico en la red pública, el cual, a diferencia de uno estático, sí podría ser configurado tras el despliegue del contrato inteligente.

De cara a construir un sistema de permisos con estas características para el prototipo concreto desarrollado en este trabajo, podrían almacenarse en el estado actual una serie de listas de direcciones, tantas como empresas colaboradoras existan. Cada una de estas listas almacenaría las direcciones de cada empresa que tienen permiso para registrar documentos. Adicionalmente, el contrato tendría funciones para añadir o eliminar direcciones de estas listas. Existirían dos funciones por cada lista, una para añadir una dirección a la lista y otra con un fin similar, pero en lugar de para añadir, para eliminar una dirección. Cada una de estas funciones trataría con una sola lista y solamente podría ser ejecutada por los administradores que están en dicha lista, siendo ellos por tanto los únicos que puedan gestionar qué direcciones se encuentran

en su correspondiente lista. Por lo tanto, cada lista de direcciones sería controlada por las identidades de administración adecuadas, manteniendo de esta manera un control dinámico sobre quién puede registrar documentos en una red pública.

Los sistemas de permisos en una red pública como los anteriormente mencionados es necesario programarlos desde cero, lo cual presenta el inconveniente de que, cuanto más complejos sean, más difícil serán de implementar y de gestionar para una empresa. En el caso de Hyperledger Fabric, el sistema de permisos ya viene construido de manera nativa y solamente es necesario configurarlo como las empresas quieran, sin embargo, en Ethereum, antes de configurarlo con las identidades adecuadas, debe ser implementado.

11.3 Mantenimiento

El mantenimiento es un factor clave a tener en cuenta a la hora de que una compañía o una serie de ellas decidan adoptar esta tecnología.

De cara a hacer uso de una red pública, esta ya se encuentra constituida y funcionando, con todos los nodos en marcha y conectados. Por lo tanto, una empresa no necesita adquirir hardware ni realizar labores de configuración de redes para hacer uso de ella, siempre y cuando decida conectarse a un nodo existente, como es el caso de este prototipo. En caso contrario, es decir, en el supuesto de que una empresa decida contribuir a la red pública, deberá adquirir hardware y costear la electricidad empleada de cara a procesar transacciones, lo cual tiene la ventaja de que, si el nodo o nodos propietarios de la empresa consiguen generar bloques válidos, esta obtendría beneficios económicos como consecuencia. Sin embargo, una organización puede hacer uso de una red pública sin preocuparse en ningún momento por el mantenimiento del hardware.

La situación es distinta a la hora de tratar con redes de consorcios, las cuales son responsabilidad de las empresas que participan en ellas. De cara a la constitución y mantenimiento de estas redes, son las propias empresas las que se encargan de ello, bien sea por ellas mismas o subcontratando los servicios, pero, de un modo u otro, los costes monetarios relativos a la adquisición o alquiler del hardware y al mantenimiento de la red correrán siempre a cargo de las compañías que formen parte de ella.

Por lo tanto, en una red pública, las empresas tendrán que pagar por realizar transacciones, sin la necesidad explícita de costear hardware, al contrario que en una red de consorcios, donde la realización de transacciones no conlleva costes monetarios explícitos, pero sí la adquisición y mantenimiento del hardware que forma la red. La decisión sobre qué tipo de red utilizar en cada caso recae sobre las propias empresas, dependiendo en muchas ocasiones del presupuesto que tengan a su disposición.

Hasta ahora, en el presente apartado únicamente se han mencionado los aspectos relativos al hardware, pero, los referentes al software son igual de importantes. En una red elaborada con Hyperledger Fabric, cada contrato presente en ella puede ser modificado, pudiendo desplegar nuevas versiones para ampliar la funcionalidad o corregir errores identificados en el contrato. A diferencia de lo anterior, en una red Ethereum, una vez que se despliega un contrato, este no se puede modificar. El propósito de ello es que las empresas que vayan a interactuar con dicho contrato deben ponerse de acuerdo en cuanto a las funciones que tendrá y a la implementación de cada una de ellas antes de su despliegue, ya que, una vez desplegado, no existirá la manera de editar la funcionalidad que ofrece. Esto tiene la ventaja de que, una vez en la red, las empresas estarán seguras de que siempre se va a ejecutar de la misma manera y de que nadie puede modificar su comportamiento. Sin embargo, también existen inconvenientes derivados de

ello, ya que, si las empresas deciden cambiar la funcionalidad relativa a una función o si descubren un error en alguna de ellas, no podrán modificarlo y deberán desplegar un contrato completamente nuevo en otra dirección diferente, donde el estado actual de la nueva cuenta de contrato no tendrá los datos que permanecían en la cuenta de contrato original.

Adicionalmente, si se ha implementado de manera errónea uno de los sistemas de permisos mencionados en el anterior apartado, identidades que en principio no tendrían permitido ejecutar transacciones sobre ese contrato podrían aprovecharse de estos errores para ello. De cara a corregir este tipo de errores también sería necesario desplegar un nuevo contrato en otra dirección, lo cual provoca que, si la gestión de permisos en un contrato es compleja, implementar el sistema de permisos y realizar labores de mantenimiento sobre él pueda convertirse en una tarea difícil de llevar a cabo.

Para concluir con este apartado, el código binario de todos los contratos inteligentes presentes en una red Ethereum es público y todo el mundo puede verlo. Sin embargo, un usuario, al ver este tipo de código no sabrá las operaciones que se realizan. A partir de código binario se puede obtener código ensamblador, ya que hay una correspondencia directa entre ellos. En código ensamblador se indican operaciones de bajo nivel, pero a través de él, un usuario tampoco tendrá conocimiento de lo que hace el contrato al ver este código. Sin embargo, tras realizar un análisis del código ensamblador por una persona con los conocimientos necesarios se puede obtener información acerca de las operaciones que realiza, pero no se podrá obtener el código fuente exacto con los nombres reales de las variables o funciones. A día de hoy, Etherscan ofrece una herramienta, la cual se encuentra en fase experimental, para obtener una representación en código de alto nivel (no exacta, pero sí intentando que sea aproximada) a partir de código ensamblador.

De cara a conseguir transparencia real, es común publicar el código fuente real junto al código binario, de esta manera, todas las empresas o usuarios que van a interactuar con él podrán saber exactamente lo que hace y podrán fiarse de él. De lo contrario, tendrían que hacer ingeniería inversa para intentar conocer lo que realmente hace, lo cual sería un proceso tedioso para ellas y poco transparente. Una compañía, si no sabe realmente lo que hace un contrato, o no estará dispuesta o al menos mostrará cierta desconfianza a la hora de interactuar con él.

Debido a las razones mencionadas previamente, en este trabajo se ha publicado el código fuente. De esta manera, todas las empresas que interactúan con él están seguras de lo que hace y pueden confiar en él si así lo estiman oportuno. En la Figura 11.1 se muestra una imagen en la que se aprecia el código fuente del contrato elaborado en este trabajo desplegado en Ropsten, el cual todo el mundo puede ver a través de una plataforma como Etherscan.

```

23 // Función para comprobar si un documento existió en el pasado
24 // En un mapping de Solidity, todas las claves en principio existen
25 // Todas ellas poseen inicialmente un valor por defecto (0 para uint o false para bool)
26 // En caso de que no se retorne una fecha válida, significa que el documento no se registró
27 function comprobarExistencia(bytes32 _identificadorDocumento) view public returns (bytes32) {
28     return documentosRegistrados[_identificadorDocumento];
29 }
30
31 // Función para registrar el identificador de un documento
32 function registrarDocumento(bytes32 _identificadorDocumento, bytes32 fechaEnMilisegundos) public {
33     // msg.sender sirve para saber quién ha invocado a esta función
34     address invocador = msg.sender;
35
36     // En caso de que el invocador no sea un administrador válido, la transacción se cancela y no se realiza
37     // De esta manera, aunque todos los datos en Ethereum sean públicos, se puede limitar quién los puede editar
38     // En este caso, sólo unos administradores determinados pueden registrar documentos
39     require(
40         invocador == administradorTransporteDeManzanas || invocador == administradorTransporteDeFresos || invocador ==
41         administradorTransporteDeMermelada,
42         "No cuentas con permiso para registrar un documento"
43     );
44
45     // En caso de que la comprobación anterior sea válida, se registra el documento
46     documentosRegistrados[_identificadorDocumento] = fechaEnMilisegundos;
47

```

Figura 11.1: Código fuente publicado en Ropsten

11.4 Rendimiento

El rendimiento de una red Fabric depende de lo veloces que sean las conexiones entre los diferentes componentes de la red, pero, sobre todo, del servicio de ordenación. Si en una de estas redes la frecuencia de transacciones es muy alta y el servicio de ordenación está compuesto por pocos nodos *orderer*, el rendimiento de toda la red se verá reducido, ya que este servicio no podrá procesar las transacciones a medida que le van llegando, actuando de esta manera como cuello de botella y provocando que sea necesario un tiempo previo a la confirmación de las transacciones. En la parte del prototipo que se ha realizado en este trabajo con Fabric, las transacciones se confirman de manera casi instantánea, lo cual es comprensible, ya que todos los componentes de la red están en la misma máquina (por tanto las conexiones entre ellos son muy rápidas) y el servicio de ordenación (aunque está compuesto por un solo nodo) no supone ningún problema de cuello de botella, ya que la frecuencia de transacciones es baja.

Al tratar con una red pública Ethereum, la evaluación del rendimiento depende casi exclusivamente del dinero que se esté dispuesto a pagar por unidad de gas consumida. Es decir, como norma general, los nodos ejecutarán las transacciones que les proporcionen una mayor recompensa, por lo tanto, el tiempo que tarde una transacción en añadirse a un bloque válido depende en gran medida de lo que el realizador pague por ello. Esto implica que el rendimiento sea complicado de evaluar en una red Ethereum pública, ya que es un concepto muy relativo que depende de cuánto se esté dispuesto a pagar por la ejecución de una transacción. Las métricas actuales sobre la red principal de Ethereum indican, como se ha mencionado durante este documento, que el número de transacciones ejecutadas por segundo son unas 10.

Evaluar el rendimiento en la red Ropsten y extrapolar que en la red principal sería el mismo sería erróneo, ya que, aunque las dos redes funcionan de manera similar (empleando la prueba de trabajo como método de consenso), la red Ropsten es menor en número en nodos y recibe un

número de transacciones menor que la red principal. Sin embargo, se ha hecho una pequeña evaluación del rendimiento en Ropsten. Para ello, se han investigado los precios que se suelen pagar por unidad de gas consumida, y se ha visto que existe una gran variedad en cuanto a las cantidades que la gente paga, existiendo identidades que pagan un precio muy alto y otras que pagan precios muy bajos. Realizando una media aproximada de ellos, he considerado que 10 Gwei por unidad de gas se podría considerar un precio realista, y, finalmente, es la cantidad que he establecido. He decidido poner un precio ni muy alto ni muy bajo por cuestión de equilibrar rendimiento y costes (aunque los costes en Ropsten son gratuitos, ya que el Ether en dicha red carece de valor monetario). En la red principal, un precio más alto proporciona mayor rendimiento y un precio más bajo lo reduce, pero esto último evita que los costes de ejecución sean altos. El precio a pagar en la red principal lo debería decidir cada empresa al realizar cada transacción dependiendo del presupuesto del que disponga y de lo rápido que se desee que sea confirmada.

En la Tabla 11.1 se muestran los datos relativos a la ejecución de 8 transacciones en la red Ropsten. Cada transacción válida es relativa al registro de un documento y las inválidas son las que he enviado desde cuentas sin permisos intentando registrar también un documento. Para cada transacción se muestra el tiempo que ha tardado en confirmarse, y, por tanto, en añadirse a un bloque válido de la cadena mantenida en la red de Ropsten.

Transacción	Tiempo de confirmación
Transacción 1 (válida)	7 segundos
Transacción 2 (válida)	25 segundos
Transacción 3 (inválida)	1 minuto 25 segundos
Transacción 4 (inválida)	24 segundos
Transacción 5 (válida)	28 segundos
Transacción 6 (válida)	36 segundos
Transacción 7 (válida)	23 segundos
Transacción 8 (válida)	17 segundos

Tabla 11.1: Tiempos de confirmación en Ropsten

Es posible apreciar que la media de la mayoría de los tiempos ronda los 25 segundos, sin embargo, existen algunos que se alejan mucho de los valores comunes. Los tiempos obtenidos, debido a que el precio por unidad de gas en todas estas transacciones es el mismo, dependen además del resto de transacciones que hayan sido enviadas a la red desde otras direcciones en el momento de ejecutar cada una de las mostradas en la tabla. Si otros usuarios han enviado a la red transacciones pagando mucho por su ejecución, la transacción de registro tardará más en ejecutarse (transacción 3) y si la red tiene transacciones pendientes pero los precios por unidad de gas son bajos, la transacción de registro se ejecutará más rápido (transacción 1). Elaborar una tabla de este estilo permite obtener, dado un precio concreto, los tiempos medios aproximados de ejecución de transacciones, pero dada una transacción concreta, a priori, no se sabrá cuánto va a tardar en ejecutarse, ya que depende de las transacciones que estén enviando otros usuarios

a la red y de lo que estén pagando por ejecutarlas. Además, si el precio por unidad de gas fuese superior a 10 Gwei, los tiempos en esta tabla serían menores, y, si fuera inferior, serían superiores.

Para concluir este apartado, se hace mención al precio total a pagar por ejecutar cada transacción de registro, el cual depende también ínfimamente del administrador que hace la transacción (debido a la condición con operaciones OR al inicio de la función de registro). Sin embargo, estas mínimas diferencias no tienen una repercusión que se haga notar en el precio de la transacción. Cada transacción de registro tiene un coste de 0,00045 Ether, lo cual, con un valor actual del Ether de \$240, equivale a \$0,108. Para hacerse una idea, registrar 100 documentos en la red principal a día de hoy implicaría un coste de unos \$11. Dicho precio sería una decisión personal de cada empresa, la cual debe tomarse equilibrando costes y rendimiento.

11.5 Riesgos

A día de hoy, la tecnología blockchain todavía se encuentra en un estado de madurez temprano, lo cual provoca que su adopción por parte de las empresas suponga ciertos riesgos.

Sus aplicaciones iniciales estuvieron orientadas a casos de uso financieros, y no fue hasta la aparición de Ethereum cuando se comenzó a investigar su aplicación a otros entornos.

De cara a aumentar la integración de esta tecnología en entornos industriales se requiere tanto de inversión como de investigación. Hoy en día, mover procesos críticos de una empresa a un entorno blockchain sería una decisión arriesgada, debido que la tecnología es todavía muy reciente. Por lo tanto, la mejor opción sería que se intentase investigar lo que esta tecnología ofrece a través de prototipos que complementen y añadan valor a los procesos que se gestionan actualmente con otras tecnologías o en otros entornos de ejecución.

Tras el desarrollo y el análisis de diferentes prototipos, viendo las ventajas y las desventajas que aportan en cada caso concreto, sería decisión de cada compañía la adopción e integración de blockchain de cara a renovar y mejorar la gestión de sus procesos.

11.6 Futuro

Actualmente, los contratos inteligentes carecen de valor legal, pero, de cara al futuro, se menciona en muchas ocasiones el caso de que sí pasen a tener validez legal. Cuando las transacciones de un contrato involucran a varias organizaciones, son estas las que normalmente se encargan de elaborarlo y de aprobarlo, pudiendo crear un vínculo legal entre ellas a través de dicho contrato, el cual se puede desplegar en una red para que los contenidos que se almacenen en su estado actual pasen a tener valor legal real, ya que ambas empresas han colaborado y están de acuerdo con las operaciones llevadas a cabo en él.

Como norma general, en todos los sitios donde se puede leer información acerca de esta tecnología, es común encontrarse con las palabras transparencia, confianza, descentralización o inmutabilidad, las cuales no cabe duda de que invitan al optimismo. Sin embargo, y esto es muy importante, estos términos son vacíos si los datos que se guardan en las redes blockchain no son correctos. La tecnología ofrece inmutabilidad y transparencia, pero no sirve de nada si una compañía decide guardar datos falsos, tanto si lo hace a propósito como si es debido a errores no intencionados. Para intentar asegurar en la medida de lo posible que los datos que se envían a una red blockchain sean datos correctos, son necesarios procesos de auditoría en las empresas,

llevados a cabo periódicamente tanto por las propias compañías como por organizaciones externas e independientes.

Por último, en lo relativo exclusivamente a las redes públicas, será necesario investigar y desarrollar métodos de consenso más eficientes para proporcionar la seguridad que a día de hoy es tan costosa energéticamente de proveer. Este sistema de consenso basado en la competencia entre máquinas por resolver un problema complejo es completamente irrespetuoso con el medio ambiente si la energía usada para ello procede de fuentes no renovables, y, aunque proceda de fuentes renovables, es una manera de desaprovechar una energía que podría ser destinada a otros fines. Ethereum se encuentra investigando y haciendo planes relativos a los gastos energéticos, los cuales pasan por ofrecer un método de consenso llamado prueba de participación, el cual pretende eliminar la competencia que existe a día de hoy para generar un bloque. Para ello, la idea es que un nodo de la red es propuesto para generar el siguiente bloque de la cadena, de modo que, si lo hace bien, dicho nodo recibirá una recompensa por ello, pero, si intenta hacer acciones malintencionadas, será penalizado. De cara a generar el próximo bloque, un nuevo nodo diferente del que generó el anterior bloque será propuesto para ello. De esta manera el gasto energético necesario para mantener una red pública Ethereum se vería reducido de manera muy significativa. Sin embargo, el proceso de adopción de este método de consenso todavía puede llevar un tiempo, como se ha tratado en el apartado 3.1.3.

Capítulo 12: Planificación y Presupuesto

En este capítulo se incluyen los aspectos relativos a la planificación y presupuesto del proyecto. El desarrollo real del mismo finalmente no se ha correspondido con lo planificado al inicio, por ello, se tratará en un apartado lo planificado inicialmente y en otro el desarrollo real del proyecto.

12.1 Inicial

En esta sección se incluye la planificación y el presupuesto realizados al comienzo del proyecto.

12.1.1 Planificación

Antes de realizar la planificación del proyecto, se conocía de antemano que diciembre, enero y quizás principios de febrero serían meses en los que iba a resultar complicado dedicar horas al trabajo fin de grado. Diciembre iba a necesitarlo para realizar proyectos relacionados con otras asignaturas, y a comienzos de año sería cuando me encontraría realizando las prácticas en empresa, trabajando 8 horas diarias en una compañía que no se encontraba en la ciudad en la que yo residía durante el curso académico. Estas razones han llevado a que diciembre, enero y comienzos de febrero hayan sido reservados para aspectos ajenos al presente trabajo.

Sin embargo, se deseaba comenzar a realizar el proyecto antes del mes de diciembre, al menos de cara a tomar contacto con la tecnología blockchain, razón por la cual su comienzo se produjo en noviembre. En principio, la intención era presentar el trabajo en el mes de junio, para lo cual sería necesario tener terminado el proyecto en las primeras semanas de dicho mes. Por ello, se planificó la finalización del mismo para mediados de la segunda quincena de mayo, de cara a disponer de tiempo para revisar todo lo necesario antes de la entrega y a tener un margen para los problemas que pudiesen surgir durante el transcurso del proyecto.

La duración total del proyecto en horas se estimó alrededor de 300, ya que es la cifra indicada en el plan de estudios. En cuanto al horario, se estableció un trabajo medio diario de 3 horas durante 6 días a la semana, de cara a tener un día de descanso por semana. Combinando este número de horas y este horario resultó en que el tiempo durante el cual se trabajaría en el proyecto de manera explícita sería de unos 4 meses.

Para realizar la planificación, se decidió dividir el proyecto en diferentes fases, las cuales se describen a continuación en orden cronológico:

1. **Aprendizaje de fundamentos:** Esta es la fase inicial del proyecto, en la cual se tomará contacto con la tecnología blockchain, estudiando y analizando sus fundamentos teóricos.
2. **Ánalisis del estado del arte:** Etapa en la cual se estudiarán los orígenes de blockchain y su situación actual. En esta fase se analizarán investigaciones recientes y relevantes acerca de la tecnología y se estudiarán las principales características de las plataformas y proyectos blockchain que existen en la actualidad.

3. **Estudio comparativo de plataformas:** En esta fase se analizarán de manera profunda las plataformas blockchain más relevantes descubiertas durante la realización del estado del arte.
4. **Elaboración del prototipo:** Etapa en la cual se construirá un prototipo acerca de un caso de uso relativo a la industria, haciendo uso para ello de la tecnología blockchain.
5. **Evaluación:** Fase final del proyecto, en la cual se evaluarán los resultados de la investigación realizada.

Debido a que yo no poseía conocimientos acerca de la tecnología blockchain antes de realizar el presente trabajo, la definición de tareas para cada fase y su estimación fue un proceso complicado. Normalmente, cuanto más se conoce una tecnología o cuando se han hecho proyectos similares previamente, más sencillo resulta estimar los esfuerzos que conllevará un futuro proyecto de similares características. Sin embargo, tanto la propia tecnología como el mundo de la investigación eran cosas nuevas para mí. Por ello, se decidieron definir tareas generales, sin entrar en mucho nivel de detalle, realizando estimaciones aproximadas para intentar finalizar el proyecto con cierto margen sobre la fecha prevista de entrega, y que la duración total del mismo rondase las 300 horas. Finalmente, la planificación inicial era de unas 294 horas. En la Figura 12.1 se muestra el diagrama de Gantt y el desglose de tareas inicial del proyecto.

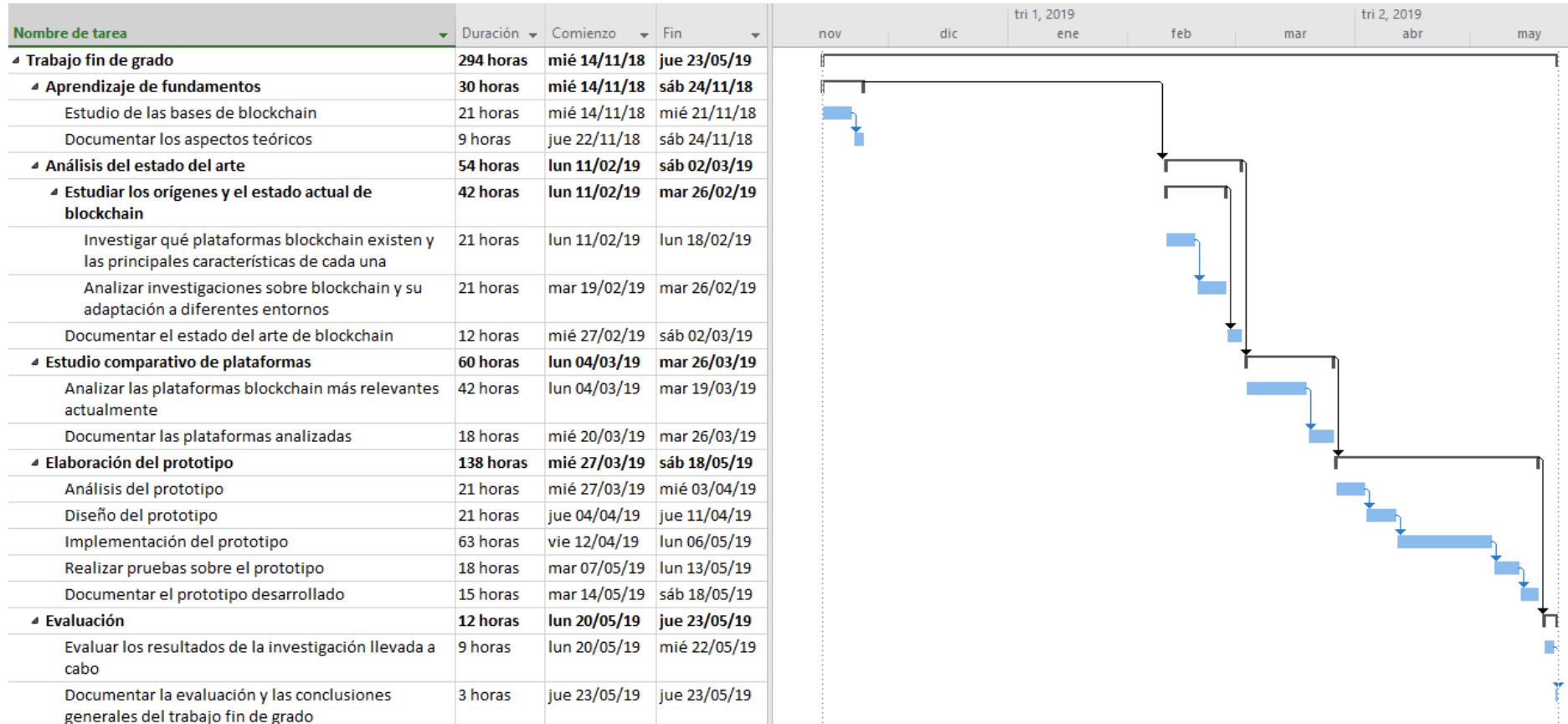


Figura 12.1: Diagrama de Gantt y desglose de tareas inicial

12.1.2 Presupuesto

En este caso concreto, la investigación realizada ha sido parte de un trabajo fin de grado, por lo tanto, el alumno no ha recibido dinero por llevarla a cabo. Sin embargo, la intención de este presupuesto no es la de reflejar los gastos reales del trabajo fin de grado, sino los de un posible proyecto similar que podría ser realizado por una determinada universidad o empresa.

A continuación se muestra el presupuesto interno llevado a cabo por el propio organismo encargado de la investigación, para posteriormente incluir el presupuesto que se mostraría a un potencial cliente que desee financiar el proyecto.

12.1.2.1 Presupuesto Interno

De cara a realizar este proyecto, se contará con un investigador, el cual se dedicará tanto a labores de investigación de la tecnología blockchain como a la construcción del prototipo. Para intentar aproximar los precios en la medida de lo posible a un proyecto real, se ha buscado información acerca de los salarios medios en España para investigadores con poca experiencia, y se ha concluido que un salario bruto de 10 € por hora es un precio realista a día de hoy. En la Tabla 12.1 se muestran los costes relativos al personal que se encarga de llevar a cabo la investigación.

Concepto	Horas	Precio/hora	Total
Investigador	294	10,00 €	2.940,00 €

Tabla 12.1: Costes del personal del proyecto

En la Tabla 12.2 se incluyen los costes directos del proyecto relativos a cada fase del mismo.

Concepto	Horas	Precio/hora	Total
Aprendizaje de fundamentos	30	10,00 €	300,00 €
Análisis del estado del arte	54	10,00 €	540,00 €
Estudio comparativo de plataformas	60	10,00 €	600,00 €
Elaboración del prototipo	138	10,00 €	1.380,00 €
Evaluación	12	10,00 €	120,00 €
Total:			2.940,00 €

Tabla 12.2: Costes directos

Una vez determinados los costes directos del proyecto, se procede a continuación a tratar los costes indirectos, los cuales se muestran en la Tabla 12.3.

Concepto	Meses	Precio/mes	Total
Licencia Microsoft Office	4	8,80 €	35,20 €
Licencia Microsoft Project	4	30,00 €	120,00 €
Licencia WebStorm	4	12,90 €	51,60 €
Licencia GitHub	4	7,00 €	28,00 €
Alquiler de oficina	4	200,00 €	800,00 €
Material de oficina	4	10,00 €	40,00 €
Ordenador	4	25,00 €	100,00 €
Conexión a internet	4	12,00 €	48,00 €
Electricidad	4	20,00 €	80,00 €
Agua	4	15,00 €	60,00 €
Transporte	4	60,00 €	240,00 €
Total:			1.602,80 €

Tabla 12.3: Costes indirectos

En el caso de este tipo de costes, se incluyen los importes correspondientes a los 4 meses que está previsto que dure el proyecto. En primer lugar, se incluyen las licencias del software necesario, y en segundo lugar, se han hecho una serie de supuestos acerca del funcionamiento interno de la organización. Para ello, se ha considerado que esta compañía posee oficinas alquiladas para sus trabajadores, los cuales necesitan herramientas para desempeñar sus labores, en este caso un ordenador y material de oficina (bolígrafos, lápices, folios o libretas, entre otros). El precio del ordenador no es correcto cargarlo completamente al proyecto, sino que la organización comprará un ordenador para cada uno de sus trabajadores cada cierto tiempo, los cuales se amortizarán durante todos los proyectos que se realicen con ellos. En este caso, se supone que dicho ordenador cuesta unos 900 €, y que se pretende amortizar durante los proyectos que realice el investigador durante 3 años, de ello el precio mostrado de 25 €. También son consideradas las tarifas relativas a la conexión a internet, electricidad y agua, cuyos precios se han calculado pensando en que la compañía tiene más oficinas alquiladas en un mismo edificio, y los precios mostrados serían la parte relativa a los gastos de la oficina donde trabajará el investigador. Por ejemplo, en el caso de la electricidad, la parte fija que existe en este tipo de facturas se pagaría para una serie de oficinas. En otro caso como el de la conexión a internet, se contrata para proveer de conectividad a una serie de oficinas, y no se contrata una conexión individual para cada oficina, lo cual sería sustancialmente más caro. En lo relativo al transporte del investigador hasta su lugar de trabajo, se han considerado 60 € mensuales, bien para gastos de combustible de su automóvil o para el uso de transporte público, pero esta tarifa dependería del lugar concreto donde el trabajador residiese y la localidad donde se encontrase su puesto de trabajo.

Tras hacer mención tanto a los costes directos como a los indirectos, se muestra en la Tabla 12.4 el presupuesto interno elaborado por la organización encargada de realizar la investigación. Para ello, se ha considerado que esta organización desea obtener un 20% de beneficios.

Costes directos	2.940,00 €
Costes indirectos	1.602,80 €
Subtotal	4.542,80 €
Beneficios (20%)	908,56 €
Total	5.451,36 €

Tabla 12.4: Presupuesto interno

12.1.2.2 Presupuesto de Cliente

En la Tabla 12.5 se incluye el presupuesto que se le mostraría al cliente que vaya a financiar el proyecto. Para ello, se han decidido repartir las cinco fases listadas en el apartado 12.1.1 en solamente dos: fase de construcción (la cual incluye la fase de elaboración del prototipo) y la fase de investigación (en la cual se incluyen las demás fases).

En estos casos, al cliente no se le muestran los beneficios que la organización pretende obtener ni los costes indirectos, sino que estos se prorratean entre las diferentes fases del proyecto mostradas al cliente. En este presupuesto se incluye el IVA, de cara a mostrar la factura real que el cliente tendría que pagar por financiar el proyecto.

Fase de investigación	2.892,56 €
Fase de construcción del prototipo	2.558,80 €
Subtotal	5.451,36 €
IVA (21%)	1.144,79 €
Total	6.596,15 €

Tabla 12.5: Presupuesto de cliente

12.2 Real

En esta sección se hace referencia tanto al desarrollo real del proyecto como al presupuesto resultante tras finalizar el mismo.

12.2.1 Desarrollo Real del Proyecto

Finalmente, el tiempo empleado para elaborar este proyecto ha sido superior al planificado, resultando en un tiempo explícito de trabajo en el proyecto de 5 meses (segunda mitad de noviembre y segunda mitad de febrero harían un mes, sumado a marzo, abril, mayo y junio) y en unas 419 horas empleadas. Este retraso sobre el tiempo planificado ha sido debido principalmente a los siguientes factores:

- **Subestimación de las tareas de documentación:** Inicialmente, he pensado que redactar la documentación me llevaría una cantidad menor de tiempo, sin embargo, al proceder a las labores de redacción me daba cuenta de que me solían llevar un tiempo sustancialmente superior al planificado, a lo cual es necesario sumar las

correcciones y modificaciones tras las revisiones periódicas por parte de mi director.

- **Falta de conocimientos previos sobre la tecnología:** Antes de comenzar a realizar el trabajo no contaba con conocimientos acerca de esta tecnología y desconocía cual era su grado de complejidad. Al final, he necesitado más tiempo para estudiarla y comprenderla adecuadamente del que había planificado.
- **Problemas a la hora de elaborar el prototipo:** Hasta la fase de elaboración del prototipo he intentado seguir la planificación de manera aproximada, pero durante esta fase he visto que no iba a ser posible, debido a todos los inconvenientes que me he encontrado al pasar de la parte teórica a la parte práctica, los cuales se han descrito en el apartado 8.3.1. En el repositorio remoto en el que he alojado el código y los archivos del proyecto se puede observar que el primer *commit* fue realizado el 13 de junio, tras comenzar a principios de ese mismo mes a implementar el contrato inteligente encargado de la gestión de lotes y a construir parte de la red de consorcios. Este primer *commit* ha sido hecho tan tarde respecto a la fecha planificada debido a que previamente he estudiado cómo hacer las cosas prácticas con cada plataforma y herramienta, realizando pequeños proyectos con cada una de ellas, lo cual me ha llevado alrededor de un mes. Pero, gracias a la elaboración de estos pequeños proyectos, he sido capaz de implementar el prototipo relativo al caso de uso industrial en aproximadamente un mes también, ya que tenía código previamente hecho que funcionaba y entendía, lo cual me sirvió de base para construir el prototipo industrial y tenerlo acabado para finales del mes de junio. La fase total de implementación me llevó dos meses, uno de formación práctica y otro para construir el prototipo, tiempo muy superior al planificado, y llegó un momento en el que noté que presentar el trabajo en junio no iba ser viable, por lo tanto, el proyecto se tuvo que retrasar hasta la convocatoria de julio.

En la Figura 12.2 se incluye el diagrama de Gantt en el que se refleja el desarrollo real del proyecto. Tras él, se muestra el desglose de tareas realizadas realmente durante todo el proyecto para cada fase. En la etapa de elaboración del prototipo se incluyen dos figuras, ya que el grado de congestión al intentar incluirlo todo en una era demasiado elevado. Debido a los avances constantes en la tecnología blockchain, algunas de las secciones que ya habían sido documentadas se han ido actualizando durante el posterior desarrollo del trabajo, conforme nuevas noticias o lanzamientos se iban produciendo.

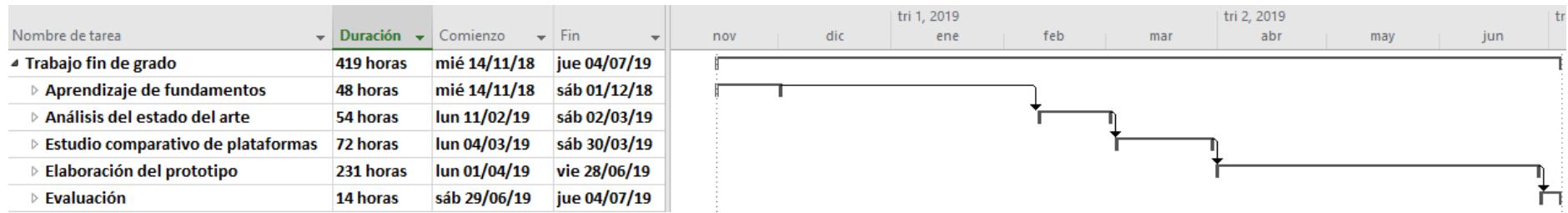


Figura 12.2: Diagrama de Gantt sobre el desarrollo real

» Aprendizaje de fundamentos	48 horas	mié 14/11/18	sáb 01/12/18
Estudio de los fundamentos teóricos de blockchain	27 horas	mié 14/11/18	vie 23/11/18
Estudio de los tipos de redes blockchain	9 horas	sáb 24/11/18	mar 27/11/18
Documentar los aspectos teóricos	12 horas	mié 28/11/18	sáb 01/12/18

Figura 12.3: Tareas realizadas durante el aprendizaje de fundamentos

» Análisis del estado del arte	54 horas	lun 11/02/19	sáb 02/03/19
» Estudiar los orígenes y el estado actual de blockchain	45 horas	lun 11/02/19	mié 27/02/19
Investigar qué plataformas blockchain existen y las principales características de cada una	20 horas	lun 11/02/19	lun 18/02/19
Analizar investigaciones sobre blockchain y su adaptación a diferentes entornos	25 horas	lun 18/02/19	mié 27/02/19
Documentar el estado del arte de blockchain	9 horas	jue 28/02/19	sáb 02/03/19

Figura 12.4: Tareas realizadas durante el análisis del estado del arte

» Estudio comparativo de plataformas	72 horas	lun 04/03/19	sáb 30/03/19
Análisis de la plataforma Hyperledger Fabric	25 horas	lun 04/03/19	mié 13/03/19
Analizar la plataforma Ethereum	25 horas	mié 13/03/19	vie 22/03/19
Documentar las plataformas analizadas	22 horas	vie 22/03/19	sáb 30/03/19

Figura 12.5: Tareas realizadas durante el estudio comparativo

» Elaboración del prototipo			
» Análisis del prototipo	30 horas	lun 01/04/19	jue 11/04/19
Idear y definir el caso de uso industrial	10 horas	lun 01/04/19	jue 04/04/19
Analizar qué plataforma usar para gestionar cada funcionalidad	3 horas	jue 04/04/19	vie 05/04/19
Definir los requisitos del sistema	3 horas	vie 05/04/19	sáb 06/04/19
Identificar y analizar los casos de uso	3 horas	sáb 06/04/19	lun 08/04/19
Realizar el modelado preliminar del estado actual y de los contratos inteligentes	5 horas	lun 08/04/19	mar 09/04/19
Documentar el análisis del prototipo	6 horas	mié 10/04/19	jue 11/04/19
» Diseño del prototipo	51 horas	vie 12/04/19	mié 01/05/19
Estudio teórico acerca de Hyperledger Composer	6 horas	vie 12/04/19	sáb 13/04/19
» Arquitectura	15 horas	lun 15/04/19	vie 19/04/19
Diseñar la arquitectura relativa a la red de consorcios	10 horas	lun 15/04/19	jue 18/04/19
Diseñar la arquitectura relativa a la red pública	5 horas	jue 18/04/19	vie 19/04/19
» Gestión de identidades	9 horas	sáb 20/04/19	mar 23/04/19
Diseñar la gestión de identidades en la red de consorcios	6 horas	sáb 20/04/19	lun 22/04/19
Diseñar la gestión de identidades en la red pública	3 horas	mar 23/04/19	mar 23/04/19
Diseño de los contratos inteligentes y de los modelos de datos en el estado actual	6 horas	mié 24/04/19	jue 25/04/19
Diseño de la interfaz de usuario	6 horas	vie 26/04/19	sáb 27/04/19
Documentar el diseño del prototipo	9 horas	lun 29/04/19	mié 01/05/19
» Implementación del prototipo	145 horas	jue 02/05/19	jue 27/06/19
Elaborar los manuales del sistema	5 horas	jue 27/06/19	vie 28/06/19

Figura 12.6: Tareas realizadas durante la elaboración del prototipo

Implementación del prototipo	145 horas	jue 02/05/19	jue 27/06/19
Configuración del entorno para elaborar la red de consorcios	3 horas	jue 02/05/19	jue 02/05/19
Estudio práctico sobre la construcción de redes con Hyperledger Fabric	21 horas	vie 03/05/19	vie 10/05/19
Estudio práctico de Hyperledger Composer	18 horas	sáb 11/05/19	vie 17/05/19
Automatización de comandos relativos a Hyperledger Composer	10 horas	sáb 18/05/19	mié 22/05/19
Implementar la interacción de Hyperledger Composer con una red Fabric desde un servidor	11 horas	mié 22/05/19	sáb 25/05/19
Aprender a interactuar con redes Ethereum y familiarización con Solidity	21 horas	lun 27/05/19	lun 03/06/19
Implementación de la gestión de lotes	33 horas	mar 04/06/19	sáb 15/06/19
Implementación del contrato inteligente encargado de la gestión de lotes	6 horas	mar 04/06/19	mié 05/06/19
Construcción de la red de consorcios	6 horas	jue 06/06/19	vie 07/06/19
Realizar pruebas unitarias sobre el funcionamiento del contrato inteligente con Composer Playground	3 horas	sáb 08/06/19	sáb 08/06/19
Realizar pruebas sobre la gestión de permisos de los administradores con Composer Playground	3 horas	lun 10/06/19	lun 10/06/19
Implementación del servidor	6 horas	mar 11/06/19	mié 12/06/19
Implementación del cliente para interactuar con la red de consorcios	6 horas	jue 13/06/19	vie 14/06/19
Realizar pruebas sobre la gestión de permisos de los participantes con Composer Playground	3 horas	sáb 15/06/19	sáb 15/06/19
Implementación de la gestión de documentos	23 horas	lun 17/06/19	mar 25/06/19
Implementación del contrato inteligente encargado de la gestión de documentos	5 horas	lun 17/06/19	mar 18/06/19
Implementación del cliente para interactuar con la red pública	7 horas	mar 18/06/19	jue 20/06/19
Realizar pruebas sobre el contrato inteligente en local con Ganache	6 horas	vie 21/06/19	sáb 22/06/19
Despliegue del contrato inteligente en la red Ropsten	2 horas	lun 24/06/19	lun 24/06/19
Realizar pruebas sobre el contrato inteligente en la red Ropsten	3 horas	lun 24/06/19	mar 25/06/19
Documentar los aspectos acerca de la implementación y la estructura del prototipo	5 horas	mar 25/06/19	jue 27/06/19

Figura 12.7: Tareas realizadas durante la implementación del prototipo

Evaluación	14 horas	sáb 29/06/19	jue 04/07/19
Evaluar los resultados de la investigación llevada a cabo	10 horas	sáb 29/06/19	mié 03/07/19
Documentar la evaluación y las conclusiones generales del trabajo fin de grado	4 horas	mié 03/07/19	jue 04/07/19

Figura 12.8: Tareas realizadas durante la evaluación

12.2.2 Presupuesto Resultante tras Finalizar el Proyecto

Debido a que el desarrollo real del proyecto ha sido diferente al planificado inicialmente, el presupuesto resultante también ha sufrido modificaciones. El modo de elaborar este presupuesto resultante es similar al proceso mostrado en 12.1.2, pero deben cambiarse los tiempos empleados, por lo tanto, en los costes directos deben tenerse en cuenta 419 horas y en los costes indirectos 5 meses. En la Tabla 12.6 se muestra el presupuesto interno resultante y en la Tabla 12.7 el presupuesto de cliente resultante, representando este último el coste real que habría tenido que pagar un potencial cliente.

Costes directos	4.190,00 €
Costes indirectos	2.003,50 €
Subtotal	6.193,50 €
Beneficios (20%)	1.238,70 €
Total	7.432,20 €

Tabla 12.6: Presupuesto interno resultante tras finalizar el proyecto

Fase de investigación	3.334,73 €
Fase de construcción del prototipo	4.097,47 €
Subtotal	7.432,20 €
IVA (21%)	1.560,76 €
Total	8.992,96 €

Tabla 12.7: Presupuesto de cliente resultante tras finalizar el proyecto

Capítulo 13: Conclusiones y Futuro Trabajo

En este capítulo se incluyen las conclusiones sobre el proyecto elaborado y se hace referencia al trabajo que puede ser realizado de cara a al futuro.

13.1 Conclusiones

El objetivo general de este trabajo fin de grado era la investigación de la tecnología blockchain de cara a ser aplicada en entornos industriales. Para ello, debido a que inicialmente no se poseían conocimientos sobre ella, la primera tarea ha sido estudiar sus bases teóricas, para posteriormente analizar su estado del arte, tanto a nivel de plataformas existentes como a nivel de investigación. Tras ello, se determinó que, a día de hoy, las plataformas más relevantes e importantes de cara a construir aplicaciones blockchain de propósito general, y que por lo tanto podrían ser utilizadas para realizar aplicaciones destinadas a la industria, son Hyperledger Fabric y Ethereum. Con la intención de conocer más detalles sobre ellas y comprobar lo que sus peculiaridades podrían ofrecer al sector industrial, se llevó a cabo un análisis de cada una.

De todo este estudio teórico, la conclusión global es que, actualmente, esta tecnología está en una etapa en la que los proyectos de investigación superan en cantidad a los proyectos de desarrollo de aplicaciones reales. Este hecho resulta comprensible, ya que el grado de madurez de blockchain es todavía bajo, debido a su reciente aparición.

El único caso que se puede diferenciar del resto es el caso de los entornos financieros, en los que a día de hoy sí que existen plataformas y proyectos dedicados a ellos, cuyo grado de madurez no se puede afirmar que sea alto, pero poco a poco están creciendo. La razón por la cual esto sucede es que el origen de esta tecnología se encuentra en los entornos monetarios, y no fue tras varios años cuando se decidió intentar extrapolar su uso a otros sectores, siendo por tanto el sector económico en el cual más tiempo se ha estado trabajando.

Las propiedades que blockchain aporta son realmente positivas y prometedoras, de ello que tantos proyectos de investigación estén siendo llevados a cabo durante estos últimos años para estudiar su adaptación a diferentes entornos, a lo cual se ha pretendido contribuir y aportar un granito de arena también gracias a la elaboración de este trabajo.

Sin embargo, tras su realización se han visto las principales razones por las cuales a día de hoy no existen muchos ni grandes proyectos consagrados que empleen esta tecnología fuera de los sectores económicos. El principal motivo es que las plataformas que permiten la realización de estos proyectos son de muy reciente creación y, en ocasiones, su funcionamiento, documentación y rendimiento todavía distan de ser adecuados.

Ethereum pone a disposición las propiedades de esta tecnología a todo el mundo, estando orientada principalmente a las redes públicas. Sin embargo, las empresas industriales, en la mayoría de ocasiones, quizás no deseen que sus datos estén disponibles para que cualquiera, incluyendo empresas competidoras, los puedan obtener. Aunque, si este fuese el caso y la empresa quisiese potenciar un grado de transparencia hacia sus clientes muy elevado, Ethereum también cuenta con la desventaja del rendimiento y de las tarifas que costaría tener datos gestionados en la red principal. No obstante, para cantidades reducidas de información, se

considera una opción a día de hoy viable, lo cual se ha argumentado durante este trabajo y se ha ejemplificado en el prototipo construido.

Hyperledger Fabric es una opción que proporciona control de acceso de manera nativa y privacidad, propiedades de la cuales las empresas industriales podrían hacer un buen uso, ya que les permiten establecer con quién colaboran y qué datos ponen a disposición de las demás. Este tipo de empresas suelen interactuar unas con otras, lo cual se ha intentado plasmar en el prototipo. Las bases y arquitectura propuestas por Fabric son prometedoras, pero se necesita un poco más de tiempo para que esta plataforma siga evolucionando y madurando de cara a que las compañías la consideren como una opción realmente viable para sustituir el modo de gestionar sus procesos.

El hecho de que se esté desarrollando un framework específico como Hyperledger Composer de manera explícita para Hyperledger Fabric dice mucho. En primer lugar, evidencia que Fabric es la plataforma más importante dentro de todos los proyectos Hyperledger, y en segundo lugar, intenta facilitar la utilización de Fabric a quienes deseen usarla. A día de hoy, el lenguaje más popular de cara a realizar aplicaciones para Fabric sin emplear Composer es Go. Últimamente también se está trabajando en JavaScript, pero su integración todavía no es del todo correcta. Java también es mencionado como lenguaje, pero no existe apenas documentación sobre su uso. Composer aparece en este escenario para hacer más cómodo el proceso de realizar aplicaciones para Fabric, pero su principal inconveniente es que no existe todavía una versión estable, provocando que las empresas se muestren reticentes a usarlo, lo cual resulta comprensible, debido a los riesgos que ello supone.

Sin embargo, en este trabajo se ha decidido operar con dicho framework, de cara a ver las posibilidades que ofrece e investigar su potencial. Se ha pensado que su desarrollo tiene que tener un propósito, es decir, algo actualmente no tiene que estar funcionando adecuadamente, y por ello se ha decidido crear este framework para facilitar determinados procesos. En un proyecto de desarrollo, la elección de Hyperledger Composer podría ser cuestionable, pero en un proyecto de investigación es argumentable. Se conoce de antemano que su utilización implica casi con toda certeza que en un momento u otro existirá algún tipo de inconveniente, pero estos posibles problemas no deben servir de excusa a la hora de investigar, ya que ese es el objetivo de la investigación en general, resolver problemas no resueltos o intentar mejorar determinados procedimientos.

Debido a las razones principales resumidas anteriormente, los aspectos argumentados a lo largo de todo el documento y la evaluación realizada en el capítulo 11, resulta entendible que actualmente las empresas muestren cierto rechazo a la hora de adoptar la tecnología blockchain para sustituir la manera de gestionar sus procesos, ya que no existen resultados contundentes que avalen dichas decisiones. Sin embargo, analizando los aspectos teóricos sí que se imaginan los potenciales beneficios que la tecnología puede proporcionar. Pero, para que puedan ser llevados realmente a la práctica, se deben producir mejoras en las actuales plataformas blockchain, tanto a nivel de funcionalidad, comodidad de uso, rendimiento y documentación, para lo cual es necesario tiempo, trabajo e investigación.

Por último, a nivel personal, antes de comenzar el proyecto me encontraba realmente motivado por adentrarme en el mundo de la investigación, aprender acerca de una tecnología completamente nueva para mí e intentar contribuir a su avance. Dicha motivación no decreció en ningún momento a lo largo de todo el proyecto, y, tras haber finalizado, estoy muy contento de todo el proceso llevado a cabo. Es cierto que ha habido días difíciles, aquellos en los que me costaba dormir porque existían aspectos de la tecnología que no lograba comprender adecuadamente o debido a que el código no funcionaba correctamente. Sin embargo, como se suele decir, las cosas que realmente merecen la pena no suelen ser fáciles. De hecho, no

pretendo engañar a nadie y hacerme pasar por un experto en esta tecnología, ni mucho menos soy un experto en ella, solamente alguien que sigue intentando comprenderla cada día mejor. La realización de este trabajo me ha servido para tomar contacto con ella. He escuchado a muchas personas mencionar la idea de que cuanto más se estudia sobre un tema, menos se sabe acerca de él, lo cual puede resultar comprensible, ya que más dudas suelen aparecer. El hecho de desconocer determinados aspectos no es negativo si existe la motivación por intentar aprender acerca de ellos de cara a resolver las dudas que uno posea. La duda, junto a la predisposición por resolverla, son, en mi humilde opinión, dos de los ingredientes principales para el avance en cualquier disciplina.

13.2 Futuro Trabajo

A continuación, se mencionan algunas posibilidades para extender o continuar este trabajo de cara al futuro:

- **Interacción con la red principal de Ethereum:** En este proyecto se ha usado una red pública de pruebas para desplegar el contrato, cuyo funcionamiento es similar al de la red principal, pero no es tan grande, tampoco procesa tantas transacciones, ni es necesario emplear Ether con valor real en ella. Una posible ampliación sería interactuar con la red principal de Ethereum, de cara a dejar de registrar los documentos en la red de Ropsten, y pasar a hacerlo en la principal.
- **Integrar dispositivos IoT reales:** Los dispositivos IoT en este trabajo han sido simulados, y, mediante la construcción de una interfaz de usuario, se han permitido realizar las transacciones deseadas en cualquier momento. Sin embargo, en un entorno real, dichas transacciones se harían cuando determinados eventos ocurriesen en las fábricas o vehículos. El resultado en la red sería el mismo, la llegada y ejecución de una transacción, pero el origen de la misma provendría de dispositivos IoT reales, lo cual podría también ser una ampliación para extender este prototipo.
- **Distribuir la red Fabric:** Ahora mismo, todos los componentes que forman parte de la red Fabric se encuentran ejecutándose en la misma máquina. En un entorno real, estos componentes estarían distribuidos en diferentes máquinas, las cuales se comunicarían entre sí. Esta es otra de las posibilidades de cara a mejorar y hacer crecer el prototipo elaborado.
- **Involucrar a empresas reales:** A partir de esta investigación, se podría intentar hablar con alguna empresa real del sector de la industria para preguntarles si estarían dispuestas a participar en un proyecto de colaboración con otras compañías. El objetivo de ello sería experimentar la integración de esta tecnología en sus procesos, y de esta manera observar los resultados obtenidos en un entorno real.

Capítulo 14: Referencias Bibliográficas

- [1] W. Diffie and M. E. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.
- [2] R. C. Merkle, "Protocols for Public Key Cryptosystems," *IEEE Symposium on Security and Privacy*, pp. 122–134, 1980.
- [3] *Napster*. [Online]. Available: <https://es.napster.com/> (Last accessed: 29-Nov-2019).
- [4] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008.
- [5] *Bitcoin*. [Online]. Available: <https://bitcoin.org/en/> (Last accessed: 12-Feb-2019).
- [6] L. M. Bach, B. Mihaljevic, and M. Zagar, "Comparative analysis of blockchain consensus algorithms," *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2018 - Proceedings*, pp. 1545–1550, 2018.
- [7] *Litecoin*. [Online]. Available: <https://litecoin.org/> (Last accessed: 12-Feb-2019).
- [8] *Peercoin*. [Online]. Available: <https://peercoin.net/> (Last accessed: 12-Feb-2019).
- [9] *CoinDesk*. [Online]. Available: <https://www.coindesk.com> (Last accessed: 13-Feb-2019).
- [10] V. Buterin, "A Next-Generation Smart Contract & Decentralized Application Platform," *Ethereum White Paper*, 2013.
- [11] *Última versión del Ethereum White Paper*. [Online]. Available: <https://github.com/ethereum/wiki/wiki/White-Paper> (Last accessed: 28-Mar-2019).
- [12] *Ethereum*. [Online]. Available: <https://www.ethereum.org/> (Last accessed: 02-Jun-2019).
- [13] G. Wood, "Ethereum: A secure decentralized generalized distributed ledger," *Ethereum Yellow Paper*, 2014.
- [14] *Última versión del Ethereum Yellow Paper*. [Online]. Available: <https://ethereum.github.io/yellowpaper/paper.pdf> (Last accessed: 29-May-2019).
- [15] *EOSIO*. [Online]. Available: <https://eos.io/> (Last accessed: 14-Feb-2019).
- [16] *Cardano*. [Online]. Available: <https://www.cardano.org/en/home/> (Last accessed: 14-Feb-2019).
- [17] *Ethereum Foundation*. [Online]. Available: <https://ethereum.foundation/> (Last accessed: 14-Feb-2019).
- [18] *Nota de prensa sobre Ethereum 2.0*, 2019. [Online]. Available: <https://notes.ethereum.org/c/Sk8Zs--CQ/https%3A%2F%2Fbenjaminion.xyz%2Fnewineth2%2F20190329.html> (Last accessed: 01-Jun-2019).
- [19] *The Linux Foundation*. [Online]. Available: <https://www.linuxfoundation.org/> (Last accessed: 15-Feb-2019).
- [20] *Hyperledger*. [Online]. Available: <https://www.hyperledger.org/> (Last accessed: 01-Jun-2019).
- [21] *Hyperledger Fabric*. [Online]. Available: <https://www.hyperledger.org/projects/fabric> (Last accessed: 10-Mar-2019).
- [22] *Hyperledger Sawtooth*. [Online]. Available: <https://www.hyperledger.org/projects/sawtooth> (Last accessed: 15-Feb-2019).
- [23] *Hyperledger Indy*. [Online]. Available: <https://www.hyperledger.org/projects/hyperledger-indy> (Last accessed: 15-Feb-2019).
- [24] *Hyperledger Iroha*. [Online]. Available: <https://www.hyperledger.org/projects/iroha> (Last accessed: 01-Jun-2019).
- [25] *Hyperledger Burrow*. [Online]. Available: <https://www.hyperledger.org/projects/hyperledger-burrow> (Last accessed: 01-Jun-2019).
- [26] *Hyperledger Grid*. [Online]. Available: <https://www.hyperledger.org/projects/grid> (Last accessed: 01-Jun-2019).
- [27] *R3*. [Online]. Available: <https://www.r3.com/> (Last accessed: 17-Feb-2019).

- [28] *Quorum*. [Online]. Available: <https://www.goquorum.com/> (Last accessed: 18-Feb-2019).
- [29] *Alastria*. [Online]. Available: <https://alastria.io/> (Last accessed: 18-Feb-2019).
- [30] M. Valenta and P. Sandner, "Comparison of Ethereum, Hyperledger Fabric and Corda," *Frankfurt School Blockchain Center*, no. June, p. 8, 2017.
- [31] C. Saraf and S. Sabadra, "Blockchain platforms: A compendium," *2018 IEEE International Conference on Innovative Research and Development, ICIRD 2018*, no. May, pp. 1–6, 2018.
- [32] J. Moubarak, E. Filiol, and M. Chamoun, "Comparative analysis of blockchain technologies and TOR network: Two faces of the same reality?," *2017 1st Cyber Security in Networking Conference, CSNet 2017*, vol. 2017-Janua, pp. 1–9, 2017.
- [33] N. Chaudhry and M. M. Yousaf, "Consensus Algorithms in Blockchain: Comparative Analysis, Challenges and Opportunities," *ICOSST 2018 - 2018 International Conference on Open Source Systems and Technologies, Proceedings*, pp. 54–63, 2018.
- [34] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends," *Proceedings - 2017 IEEE 6th International Congress on Big Data, BigData Congress 2017*, pp. 557–564, 2017.
- [35] Q. K. Nguyen and Q. V. Dang, "Blockchain Technology for the Advancement of the Future," *Proceedings 2018 4th International Conference on Green Technology and Sustainable Development, GTSD 2018*, pp. 483–486, 2018.
- [36] J. Al-Jaroodi and N. Mohamed, "Blockchain in Industries: A Survey," *IEEE Access*, vol. 7, pp. 36500–36515, 2019.
- [37] D. Miller, "Blockchain and the internet of things in the industrial sector," *IT Professional*, vol. 20, no. 3, pp. 15–18, 2018.
- [38] K. Christidis and M. Devetsikiotis, "Blockchains and Smart Contracts for the Internet of Things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016.
- [39] S. Huh, S. Cho, and S. Kim, "Managing IoT devices using blockchain platform," *International Conference on Advanced Communication Technology, ICACT*, pp. 464–467, 2017.
- [40] M. Mettler, "Blockchain technology in healthcare: The revolution starts here," *2016 IEEE 18th International Conference on e-Health Networking, Applications and Services, Healthcom 2016*, pp. 1–3, 2016.
- [41] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman, "MedRec: Using blockchain for medical data access and permission management," *Proceedings - 2016 2nd International Conference on Open and Big Data, OBD 2016*, pp. 25–30, 2016.
- [42] A. K. Koç, E. Yavuz, U. C. Çabuk, and G. Dalkılıç, "Towards secure e-voting using ethereum blockchain," *6th International Symposium on Digital Forensic and Security, ISDFS 2018 - Proceeding*, vol. 2018-Janua, pp. 1–6, 2018.
- [43] Q. Lu and X. Xu, "Adaptable Blockchain-Based Systems: A Case Study for Product Traceability," *IEEE Software*, vol. 34, no. 6, pp. 21–27, 2017.
- [44] M. P. Caro, M. S. Ali, M. Vecchio, and R. Giaffreda, "Blockchain-based traceability in Agri-Food supply chain management: A practical implementation," *2018 IoT Vertical and Topical Summit on Agriculture - Tuscany, IOT Tuscany 2018*, pp. 1–4, 2018.
- [45] *Provenance*. [Online]. Available: <https://www.provenance.org/> (Last accessed: 27-Feb-2019).
- [46] *Documentación de Ethereum*. [Online]. Available: <https://www.ethereum.org/learn/> (Last accessed: 29-May-2019).
- [47] *Documentación de desarrollo para Ethereum*. [Online]. Available: <https://www.ethereum.org/developers/> (Last accessed: 03-Jun-2019).
- [48] *Documentación de Hyperledger Fabric*. [Online]. Available: <https://hyperledger-fabric.readthedocs.io/en/release-1.4/> (Last accessed: 07-Jun-2019).
- [49] E. Androulaki *et al.*, "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains," *EuroSys'18*, 2018.
- [50] C. Cachin, "Architecture of the Hyperledger Blockchain Fabric *," *IBM Research*, 2016.
- [51] D. Johnson and A. Menezes, "The Elliptic Curve Digital Signature Algorithm

- (ECDSA)," *University of Waterloo*, 1999.
- [52] *Keccak*. [Online]. Available: <https://keccak.team/keccak.html> (Last accessed: 15-Mar-2019).
- [53] *Geth*. [Online]. Available: <https://geth.ethereum.org/> (Last accessed: 17-Mar-2019).
- [54] *Parity*. [Online]. Available: <https://www.parity.io/ethereum/> (Last accessed: 17-Mar-2019).
- [55] *Documentación de Solidity*. [Online]. Available: <https://solidity.readthedocs.io/en/latest/> (Last accessed: 18-Jun-2019).
- [56] *Vyper*. [Online]. Available: <https://vyper.readthedocs.io/en/latest/> (Last accessed: 19-Mar-2019).
- [57] *Flint*. [Online]. Available: <https://github.com/flintlang/flint> (Last accessed: 19-Mar-2019).
- [58] *LevelDB*. [Online]. Available: <https://github.com/google/leveldb> (Last accessed: 09-Mar-2019).
- [59] *CouchDB*. [Online]. Available: <http://couchdb.apache.org/> (Last accessed: 06-Jun-2019).
- [60] *Documentación de Hyperledger Composer*. [Online]. Available: <https://hyperledger.github.io/composer/latest/introduction/introduction.html> (Last accessed: 12-Jun-2019).
- [61] *Documentación de Docker*. [Online]. Available: <https://docs.docker.com/> (Last accessed: 07-Jun-2019).
- [62] *Documentación de Infura*. [Online]. Available: <https://infura.io/docs> (Last accessed: 24-Jun-2019).
- [63] *MetaMask*. [Online]. Available: <https://metamask.io/> (Last accessed: 20-Jun-2019).
- [64] *Documentación de web3.js*. [Online]. Available: <https://web3js.readthedocs.io/en/1.0/> (Last accessed: 20-Jun-2019).
- [65] *Documentación de Truffle*. [Online]. Available: <https://truffleframework.com/docs/truffle/overview> (Last accessed: 24-Jun-2019).
- [66] *jQuery*. [Online]. Available: <https://jquery.com/> (Last accessed: 19-Jun-2019).
- [67] *Documentación de Knockout.js*. [Online]. Available: <https://knockoutjs.com/documentation/introduction.html> (Last accessed: 14-Jun-2019).
- [68] *Bootstrap*. [Online]. Available: <https://getbootstrap.com/> (Last accessed: 26-Apr-2019).
- [69] *Plantilla SB Admin*. [Online]. Available: <https://startbootstrap.com/templates/sb-admin/> (Last accessed: 26-Apr-2019).
- [70] *Express*. [Online]. Available: <https://expressjs.com/> (Last accessed: 12-Jun-2019).
- [71] *Webpack*. [Online]. Available: <https://webpack.js.org/> (Last accessed: 20-Jun-2019).
- [72] *SweetAlert*. [Online]. Available: <https://sweetalert.js.org/> (Last accessed: 14-Jun-2019).
- [73] *Git*. [Online]. Available: <https://git-scm.com/> (Last accessed: 26-Jun-2019).
- [74] *GitHub*. [Online]. Available: <https://github.com/> (Last accessed: 09-Jul-2019).
- [75] *Visual Studio Code*. [Online]. Available: <https://code.visualstudio.com/> (Last accessed: 26-Jun-2019).
- [76] *WebStorm*. [Online]. Available: <https://www.jetbrains.com/webstorm/> (Last accessed: 26-Jun-2019).
- [77] *Microsoft Project*. [Online]. Available: <https://products.office.com/es-es/project/project-and-portfolio-management-software> (Last accessed: 26-Jun-2019).
- [78] *Microsoft Word*. [Online]. Available: <https://products.office.com/es-es/word> (Last accessed: 26-Jun-2019).
- [79] *Microsoft Excel*. [Online]. Available: <https://products.office.com/es-es/excel> (Last accessed: 26-Jun-2019).
- [80] *Draw.io*. [Online]. Available: <https://www.draw.io/> (Last accessed: 25-Jun-2019).
- [81] *Composer Playground*. [Online]. Available: <https://composer-playground.mybluemix.net> (Last accessed: 15-Jun-2019).
- [82] *Documentación de Ganache*. [Online]. Available: <https://truffleframework.com/docs/ganache/overview> (Last accessed: 22-Jun-2019).
- [83] *Etherscan*. [Online]. Available: <https://etherscan.io/> (Last accessed: 09-Jul-2019).
- [84] *Instalación de requisitos para Hyperledger Composer*. [Online]. Available:

- [85] *Instalación de Hyperledger Composer.* [Online]. Available: <https://hyperledger.github.io/composer/latest/installing/development-tools.html> (Last accessed: 27-Jun-2019).

Capítulo 15: Apéndices

En esta sección de apéndices se describirá el contenido que se entrega como parte de el trabajo fin de grado realizado.

15.1 Contenido Entregado

El presente documento es entregado. Adicionalmente, se hace entrega de un archivo ZIP, el cual cuenta con los siguientes contenidos:

- **Código y ficheros del prototipo:** *PrototipoRedDeConsorcios-master.zip* es el fichero en el cual se encuentra el código y archivos que componen el prototipo elaborado. Este fichero es el mismo que resultaría al descargar el archivo ZIP relativo al repositorio de GitHub que he utilizado para alojar el código de manera remota y controlar sus diferentes versiones, el cual se puede encontrar en esta dirección: <https://github.com/pedrofdez26/PrototipoRedDeConsorcios>.
- **Archivos de texto:** Se incluyen los siguientes ficheros de texto:
 - **FraseMnemonica.txt:** En este archivo se incluye la frase mnemónica (el conjunto de 12 palabras) de cara a generar las identidades adecuadas de administración para interactuar con la red pública.
 - **LogsCreacionRedFabric.txt:** Los contenidos de este fichero reflejan los mensajes mostrados en la terminal durante la ejecución del archivo *crearRedFabric.sh*.
 - **LogsDespliegueRopsten.txt:** En este caso, se incluyen los mensajes mostrados por Truffle en el momento en el que el contrato inteligente de gestión de documentos fue desplegado en la red Ropsten.

El repositorio tiene el nombre de *PrototipoRedDeConsorcios* debido a que en su interior se encuentra toda la información para construir la propia red de consorcios, sin embargo, la red pública con la que se interactúa ya está construida. Su nomenclatura es una cuestión que quizás pueda generar confusión, porque se pueda pensar que no tiene código relativo a la red pública. Quizás haberlo nombrado *PrototipoTFG* simplemente hubiese sido una idea que no produciría ningún tipo de duda sobre si el repositorio se refiere solamente al código que interactúa con la red de consorcios o con las dos.