Instituto de Computación - Facultad de Ingeniería

Universidad de la república

Montevideo, Uruguay

Interoperabilidad entre plataformas de blockchain

Proyecto de grado

Autores: Bruno Bradach, Juan Nogueira

Tutores: Guzmán Llambías, Raul Ruggia

Noviembre, 2021

Resumen

Blockchain es una tecnología que permite la construcción de sistemas de información distribuidos que proporcionan descentralización, transparencia e inmutabilidad de la información almacenada. La transparencia viene dada por el hecho de que todos los nodos participantes del sistema distribuido contienen una copia de la información, del código fuente y realizan la ejecución de los mismos. La inmutabilidad refiere al hecho de que la información ya almacenada no puede ser modificada ni borrada y, la descentralización, es debido a que la información almacenada es consensuada entre los nodos.

Esta tecnología ha permitido la creación de criptomonedas, siendo Bitcoin la primera en 2008. Ese ha sido su uso más común aunque, en los últimos años, se ha visto que su potencial puede ser utilizado para otros casos de uso; por ejemplo, sistemas que realizan la trazabilidad de algún producto.

Para facilitar la construcción de estos sistemas distribuidos, han surgido numerosas plataformas de blockchain. Estas plataformas proporcionan a los desarrolladores una abstracción de alto nivel, evitando a los mismos tener que preocuparse por tareas de bajo nivel como la comunicación entre los nodos o cómo la información es guardada.

La problemática con estas plataformas es que todas han sido desarrolladas de forma independiente, sin seguir ningún estándar y con diferencias en su funcionamiento e implementación. Esto hace que sistemas construidos sobre plataformas diferentes no puedan interoperar.

Para atacar esta problemática, se han propuesto muchas soluciones en los últimos años. Algunas de ellas solo existen a nivel teórico mientras que otras también cuentan con implementación de referencia.

En este proyecto se realiza un relevamiento de dichas soluciones y se propone una solución propia, la cual se basa en el uso de gateways que actúan de intermediarios entre las distintas plataformas y hacen posible la comunicación entre los sistemas construidos sobre las mismas. Con el fin de demostrar la factibilidad técnica de la solución, se realizó una implementación a modo de prueba de concepto y se la aplicó a dos sistemas construidos sobre las plataformas Hyperledger Fabric y Corda.

1. Introducción

Blockchain es la tecnología que permitió la creación de criptomonedas, siendo Bitcoin la primera en 2008. Esta fue la primera aplicación de esta tecnología y la razón por la cual la misma es conocida. Con el correr del tiempo, se descubrió que esta tecnología también se podía aplicar en otros contextos [1], donde las virtudes que proporciona blockchain son muy útiles. Blockchain permite la construcción de sistemas descentralizados, lo cual evita la necesidad de confiar en un actor central y aumenta la transparencia. Esta es la principal virtud proporcionada por esta tecnología.

Una vez que se descubrió la posibilidad de aplicar la tecnología blockchain en otros escenarios además de las criptomonedas, comenzaron a surgir las plataformas de blockchain. Estas permiten la construcción de sistemas descentralizados sobre blockchain y permite a los desarrolladores enfocarse en la lógica de dichos sistemas, sin tener que preocuparse por tareas de bajo nivel como el almacenamiento de la información o la comunicación entre los nodos del sistema descentralizado [2].

Los desarrollos de estas plataformas se realizaron de forma independiente, por lo cual difieren notoriamente en sus funcionalidades e implementación, lo cual imposibilita la interoperabilidad entre sistemas construidos sobre plataformas distintas.

La interoperabilidad entre las distintas plataformas es deseada y necesaria, ya que, la misma posibilita que sistemas construidos con propósitos distintos puedan conectarse y complementarse, y así aumentar su utilidad (tal como sucede hoy en día con sistemas construidos sobre otras tecnologías). Esto, a su vez, ayudaría a la adopción de la tecnología blockchain.

A raíz de lo anterior, en los últimos años comenzaron a surgir diversas propuestas que intentan resolver dicha interoperabilidad, de las cuales algunas cuentan con implementaciones de referencia y otras solo existen a nivel conceptual. Dichas soluciones difieren notoriamente entre sí, no existiendo consenso acerca de cómo atacar dicha problemática. Debido a lo anterior, esta es una área que está siendo explorada y la misma no se considera resuelta.

1.1. Objetivos

El principal objetivo de este proyecto es encontrar una forma genérica de interoperar sistemas descentralizados construidos sobre distintas plataformas de blockchain. Para esto se realizará un relevamiento de las soluciones existentes y se estudiará la factibilidad de utilizar alguna de ellas. En caso que ninguna de las soluciones existentes sea de utilidad, se propondrá una solución propia y se experimentará con la misma.

La solución resultante será aplicada a un escenario ficticio, el cual consiste en implementar dos sistemas descentralizados sobre plataformas diferentes e interoperar los mismos.

Los objetivos específicos del proyecto son los siguientes.

1. Relevar, en forma selectiva, experiencias y propuestas de mecanismos de interoperabilidad.
2. Experimentar con alguna de las propuestas relevadas o proponer una solución propia.
3. Implementar en dos plataformas un escenario de uso de blockchain que requiera intercambio de datos, donde ambas plataformas interoperan utilizando la solución del objetivo 2.

1.2. Aportes del proyecto

Los aportes que realiza este proyecto son los siguientes:

- Profundización y organización de información sobre la tecnología blockchain.
- Relevamiento de soluciones existentes que intentan resolver la problemática de la interoperabilidad, organizando el conocimiento sobre las distintas estrategias utilizadas.
- Diseño de una especificación agnóstica a la tecnología para alcanzar una interoperabilidad básica entre plataformas de blockchain.
- Implementación de referencia de la especificación.
- Conocimiento de las plataformas Hyperledger Fabric y Corda.
- Aplicación de dicha solución a un escenario ficticio a modo de demostración.

1.3. Organización del documento

El contenido de este documento está organizado en capítulos, los cuales se describen a continuación.

En el capítulo 2 se presentan los conceptos y definiciones que son necesarios para comprender el resto del documento.

En el capítulo 3 se describen los requerimientos del proyecto.

En el capítulo 4 se describe el diseño de una solución propia para la interoperabilidad.

En el capítulo 5 se presenta la implementación de referencia de la solución propuesta en el capítulo anterior.

En el capítulo 6 se realiza una evaluación de la solución anterior, mostrando sus puntos fuertes, débiles y realizando una comparativa con otras soluciones.

En el capítulo 7 se presenta la gestión del proyecto.

En el capítulo 8 se menciona el trabajo a futuro, donde se mencionan posibles mejoras a tener en cuenta en la solución. A su vez, en este capítulo se presentan las conclusiones del

proyecto, resumiendo el trabajo realizado y describiendo si se alcanzaron o no los objetivos planteados al inicio del proyecto.

2. Marco conceptual

En este capítulo se presenta el contexto y los conocimientos necesarios para entender la problemática planteada y abordada en el proyecto. El contenido se estructura de la siguiente manera:

- En la sección 2.1 se explica que es blockchain, cómo funciona y en qué escenarios puede ser utilizado.
- En la sección 2.2 se describen las plataformas de blockchain con las cuales se trabajó en este proyecto.
- En la sección 2.3 se presenta la interoperabilidad en blockchain, mencionando los desafíos que la misma implica y las soluciones relevadas.

2.1. Blockchain

La tecnología blockchain surge en 2008 con la publicación de Satoshi Nakamoto *Bitcoin: un sistema de dinero en efectivo electrónico peer-to-peer*. En esta publicación, blockchain surge como la tecnología que da soporte a los requisitos de esta nueva moneda electrónica, tales como la no existencia de una entidad de confianza y la necesidad de prevención de fraudes [3]. Sin embargo, el concepto fue evolucionando habiendo múltiples definiciones.

Para el diccionario de Cambridge, blockchain se define como *"un sistema utilizado para llevar un registro digital de todas las ocasiones en que una criptomoneda fue comprada o vendida, y que crece constantemente a medida que se agregan más bloques"* [4].

En la bibliografía también se encuentra como definición que *"blockchain es una tecnología digital emergente que combina criptografía, manejo de datos, redes y mecanismos de incentivo para realizar el chequeo, ejecución y almacenamiento de transacciones entre participantes"* [1].

Mientras tanto, en un artículo, PricewaterhouseCoopers define blockchain como *"un libro mayor (ledger) descentralizado de todas las transacciones que ocurren en una red peer-to-peer"*. Y agrega: *"Usando esta tecnología, los participantes pueden confirmar transacciones sin la necesidad de una autoridad central regulatoria. Puede tener aplicación en transferencia de fondos, liquidación de operaciones, votaciones, entre otras"* [5].

El concepto de *ledger* es ampliamente utilizado en el contexto de blockchain y será mencionado a lo largo de este documento. En el desarrollo del documento se hará referencia al *ledger* como el lugar donde se registran todas las transacciones que ocurren en la red.

Luego de procesar varias definiciones, resumimos el concepto de blockchain como una base de datos distribuida donde la información se almacena en bloques, que son enlazados formando una cadena, y es replicada en los miembros de la red. Esta base de datos distribuida usualmente es también descentralizada, pudiendo variar el grado de descentralización dependiendo de la implementación. Algunas implementaciones

consideran que la red es menos descentralizada cuando algunos miembros de la red poseen mayores privilegios que otros. Cabe destacar que estos conceptos generales pueden sufrir variaciones dependiendo de cada implementación cómo se va a ir presentando a lo largo de este capítulo.

A continuación se describirán las principales características de blockchain, formas de almacenamiento, red de nodos, mecanismos de consenso, smart contracts, casos de uso, entre otros.

2.1.1. Características de blockchain

A continuación se listan las principales características de la tecnología, siendo éstas pilares del posicionamiento que logró desde su aparición [1]:

- **Descentralización:** Hace uso de una red *peer-to-peer* que permite prescindir de una entidad centralizada que medie en todas las transacciones. De esta forma las tareas y responsabilidades se distribuyen entre los distintos nodos participantes de la red.

- **Trazabilidad:** Permite analizar el historial de transacciones ya que éstas se registran con un timestamp y son agregadas de forma secuencial.

- **Transparencia:** Los miembros de la red pueden almacenar una copia de la cadena de bloques, teniendo acceso a la información almacenada. De esta forma todos los miembros tienen las mismas posibilidades de ver lo que se almacena en el ledger.

- **Inmutabilidad:** Las transacciones ya confirmadas en el ledger no pueden ser modificadas, únicamente se admite agregar nuevas transacciones. El mecanismo criptográfico utilizado para enlazar los bloques hace que una modificación en una transacción sea fácilmente detectable por la red.

2.1.2. Almacenamiento de la información

En blockchain, como su nombre lo dice, la información se almacena en una cadena de bloques (lista de bloques enlazados) y lo que contienen esos bloques son las operaciones (transacciones) que se han realizado en la blockchain por parte de los usuarios de la misma [6]. Por ejemplo, en una blockchain en la cual se implementa una criptomoneda, cuando un usuario A realiza una transferencia de X cantidad de criptomonedas al usuario B, dicha transferencia (transacción) se almacenará en un bloque.

Cada bloque de la cadena posee un hash que depende de la lista de transacciones que contiene y del hash del bloque anterior, el cual se almacena para realizar el enlazamiento entre bloques. Esto hace que, si se modifica la información almacenada en un bloque, su hash cambie, y a su vez el de todos los bloques posteriores. De este hecho es que se desprende la característica de inmutabilidad de blockchain.

En la imagen 1, se ilustra el concepto de cadena de bloques.

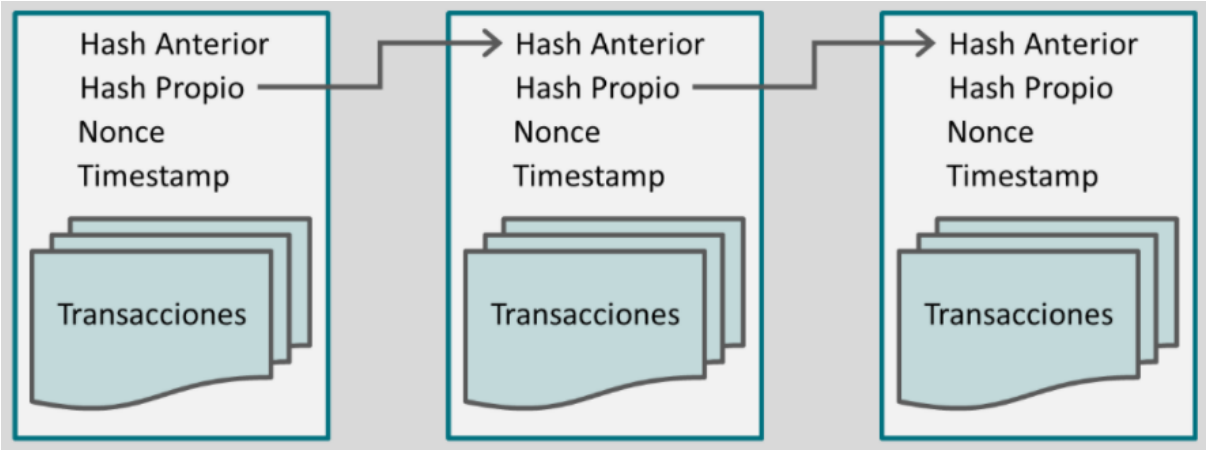

Imagen 1. Concepto de cadena de bloques [6].

2.1.3. Nodos

Una blockchain está compuesta por una red P2P de nodos que interactúan entre sí para transmitir y almacenar la información. Dependiendo de la blockchain, existen distintos tipos de nodos que cumplen distintas funciones. Por ejemplo, en Bitcoin existen nodos "completos" y nodos "ligeros". Los nodos completos son aquellos nodos que contienen una copia entera de la cadena de bloques, mientras que los ligeros dependen de otros nodos para tener acceso a toda la información [7]. Otro ejemplo es en blockchains que utilizan Proof of Work como algoritmo de consenso. En dichas blockchains, existen nodos "mineros" que se encargan de generar los bloques de la cadena.

2.1.4. Clasificación

Las blockchains se pueden clasificar según diferentes características [8]. A continuación se describen dos de las principales clasificaciones:

- **Blockchain pública:** Es un tipo de blockchain en la que sus participantes no tienen restricciones para leer los datos de la cadena de bloques ni para enviar transacciones para que sean incluidas en la cadena de bloques. En ellas es sencillo unirse a la red, son transparentes, están construidas con precaución para la operación en un entorno no confiable. Son ideales para aplicaciones totalmente descentralizadas.

- **Blockchain privada:** Es aquella en la que tanto los accesos a los datos de la cadena de bloques como el envío de transacciones para ser incluidas, están limitadas a una lista predefinida de entidades.

Otra forma de ver esta clasificación es que, en las blockchains públicas, cualquier nodo puede formar parte de la misma, o sea, no existe ninguna restricción que impida que nuevos nodos se sumen a la blockchain. En cambio, en blockchains privadas, para que un nuevo nodo se pueda unir a la misma, este debe ser admitido por los nodos ya pertenecientes a la

misma. Por esta razón, a las blockchains privadas también se las denomina como permisionadas (y no permisionadas a las blockchains públicas).

2.1.5. Algoritmos de consenso

Dado que blockchain es una red de nodos donde varios de ellos contienen una réplica de la información, es necesario que exista coordinación entre los nodos para que todos almacenen la misma información. Esta coordinación se logra con los algoritmos de consenso.

Los algoritmos de consenso permiten a los nodos de la blockchain determinar cuales son los bloques que componen la cadena y las transacciones que se almacenan en cada uno. Además de esto, están diseñados para soportar una X cantidad (depende de cada algoritmo) de nodos maliciosos en la red. Por ejemplo, nodos maliciosos podrían intentar añadir bloques o transacciones inválidas [9].

Existen varios algoritmos de consenso que se utilizan hoy en día y, a modo de ejemplo, a continuación se mencionan dos de ellos.

Proof of work (PoW)
PoW es de los algoritmos de consensos más conocidos por ser utilizado por Bitcoin. Este algoritmo requiere gran esfuerzo computacional por parte de los nodos de la blockchain para poder agregar información a la cadena. Esto se hace para evitar que nodos maliciosos agreguen información fraudulenta a la cadena [10].

Este algoritmo se aplica en blockchains públicas (donde no existe ninguna confianza entre los nodos de la red) y que implementan criptomonedas, ya que, el mismo se basa en un incentivo económico para su funcionamiento que se explica a continuación.

En una blockchain que utiliza PoW como algoritmo de consenso, existen nodos especiales llamados “mineros” que son quienes se encargan de recibir las transacciones realizadas por parte de los usuarios y generar los bloques de la cadena. Los mineros tienen una dificultad y una motivación para generar nuevos bloques.

La dificultad consiste en que existe un campo dentro de cada bloque llamado “nonce” cuyo valor debe ser determinado por el minero. El valor del campo nonce debe ser tal que haga que el hash del bloque cumpla con cierta condición (generalmente que comience con x cantidad de ceros). Para hallar el valor del campo nonce, los mineros deben probar valores de forma aleatoria, lo cual requiere grandes recursos computacionales.

La motivación para los mineros para realizar dicha tarea es económica, ya que, este tipo de algoritmo de consenso se utiliza en blockchains en las cuales se implementan criptomonedas. En estos casos, las transacciones que se almacenan en los bloques corresponden a transferencias de dinero entre los usuarios y los mineros reciben comisiones sobre esas transferencias cuando logran generar bloques nuevos que contienen las transacciones de dichas transferencias.

Cuando un minero logra generar un nuevo bloque, lo envía a otros nodos de la red, los cuales lo validan y lo agregan a su cadena en caso de ser válido. La validación de un bloque es una tarea muy sencilla y rápida (a diferencia de la generación).

Puede ocurrir que dos mineros logren generar dos bloques distintos (contienen distintas transacciones) al mismo tiempo. A esta situación se le llama fork y se ilustra en la imagen 2 [11].

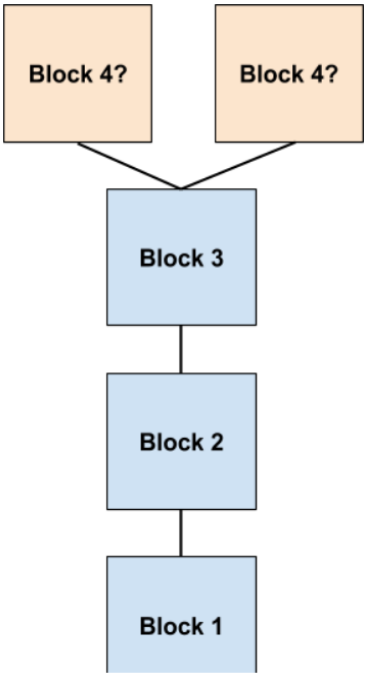

Imagen 2. Bifurcación de la cadena de bloques [10]

Cuando ocurre un fork, la red aceptará la cadena que tenga mayor cantidad de computo, lo cual es básicamente la cadena más larga. En el ejemplo de la imagen 2, ambas cadenas tienen el mismo largo, por lo cual, los mineros que deseen agregar un nuevo nodo, deberán elegir de forma arbitraria una de las dos cadenas. Una vez que un minero logre agregar un nuevo bloque a una de las dos cadenas, se resolverá el fork. Esto se muestra en la imagen 3.

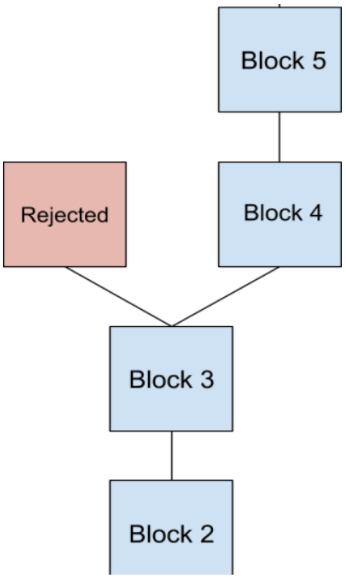

Imagen 3. Resolución fork de cadena [10].

Debido a lo anterior, para que un nodo o grupo de nodos pueda “dominar” la cadena de bloques, es necesario que posea por lo menos el 51% de poder de cómputo de la red. Si esto ocurre, dicho nodo o grupo de nodos, puede generar cadenas con mayor cantidad de cómputo (más largas) que contienen sólo bloques/transacciones que ellos desean, y los bloques generados por el resto de los mineros serán descartados por ser cadenas con menor cantidad de computo.

Practical Bizantine Fault Tolerant (PBFT)

Este algoritmo es utilizado principalmente en blockchains permisionadas y la cantidad de nodos maliciosos debe ser menor que el 33% del total de los nodos [12], [13]. Aquí existe un nodo líder o primario y nodos seguidores o secundarios. El líder es elegido por votación entre todos los nodos y puede cambiar, es decir, un nodo seguidor puede convertirse en líder y viceversa.

El mecanismo para alcanzar el consenso en este algoritmo se divide en cuatro fases:

1. Un cliente que desea registrar una transacción, envía la misma al nodo líder.
2. El nodo líder reenvía dicha transacción a todos los nodos seguidores.
3. Cada nodo seguidor ejecuta la transacción y envía el resultado al cliente.
4. Una vez que el cliente recibe m+1 (siendo m la cantidad máxima de nodos maliciosos soportada) respuestas iguales, se da por confirmada la transacción.

En la imagen 4, se representa el funcionamiento del algoritmo.

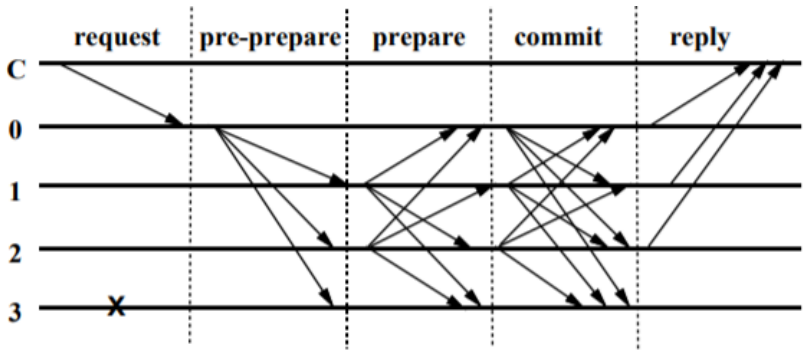

Imagen 4. Funcionamiento PBFT [12].

Antes de que los nodos envíen la respuesta al cliente (fases de prepare y commit), existe un gran intercambio de mensajes entre todos los nodos, por lo cual, la cantidad de mensajes crece exponencialmente al crecer el número de nodos. El objetivo de las fases pre-prepare y prepare es ordenar las transacciones, mientras que el de la fase de commit es confirmar las mismas.

2.1.6. Smart contracts

Un contrato inteligente es un programa que facilita, asegura, hace cumplir y ejecuta acuerdos registrados entre dos o más partes. Los mismos ayudan en la negociación y definición de tales acuerdos que causan que ciertas acciones sucedan como resultado de que se cumplan una serie de condiciones específicas [14].

En una definición más técnica, podríamos decir que los smart contracts son scripts (código ejecutable) que se almacenan en la blockchain y permiten la implementación de aplicaciones descentralizadas sobre esta tecnología. En estos scripts se implementan/programan los acuerdos entre varias partes y la ejecución de los mismos es realizada por algunos o todos los nodos de la red. El resultado de dichas ejecuciones es consensuado por los nodos de la red y almacenado en la cadena de bloques.

El resultado de los smart contracts, típicamente, depende únicamente de la información almacenada en la cadena al momento de ejecutarse el mismo, de forma tal que toda la lógica se encuentra dentro de la blockchain. Esta limitante puede ser sobrepasada si se utilizan oráculos [15]. Estos permiten obtener información que se encuentra fuera de la blockchain, como puede ser obtener la cotización de una moneda desde un servicio externo.

Cuando se construye una aplicación sobre blockchain, se crean smart contracts que implementan la lógica de dicha aplicación y, dado que la misma ejecución de los smart contracts es realizada por varios nodos, es que se dice que las aplicaciones construidas sobre blockchain son descentralizadas.

2.1.7. Funcionamiento de blockchain

Por lo general, al registrar una transacción en una blockchain, se sigue el siguiente flujo [16]:

1. Un usuario registra una transacción y la misma es enviada al resto de los nodos de la red. La transacción realizada está firmada con una clave privada del usuario para demostrar que la misma fue realizada por el usuario.
2. Los nodos de la red verifican la transacción.
3. Si la transacción es válida, la transacción es agregada a un bloque y los nodos ejecutan el algoritmo de consenso.
4. Una vez alcanzado el consenso, el bloque es agregado a la cadena y la transacción queda registrada.

Debido a la propiedad de inmutabilidad de blockchain, una vez que una transacción es registrada, la misma no puede ser borrada.

En la imagen 5 se ilustra el funcionamiento de blockchain.

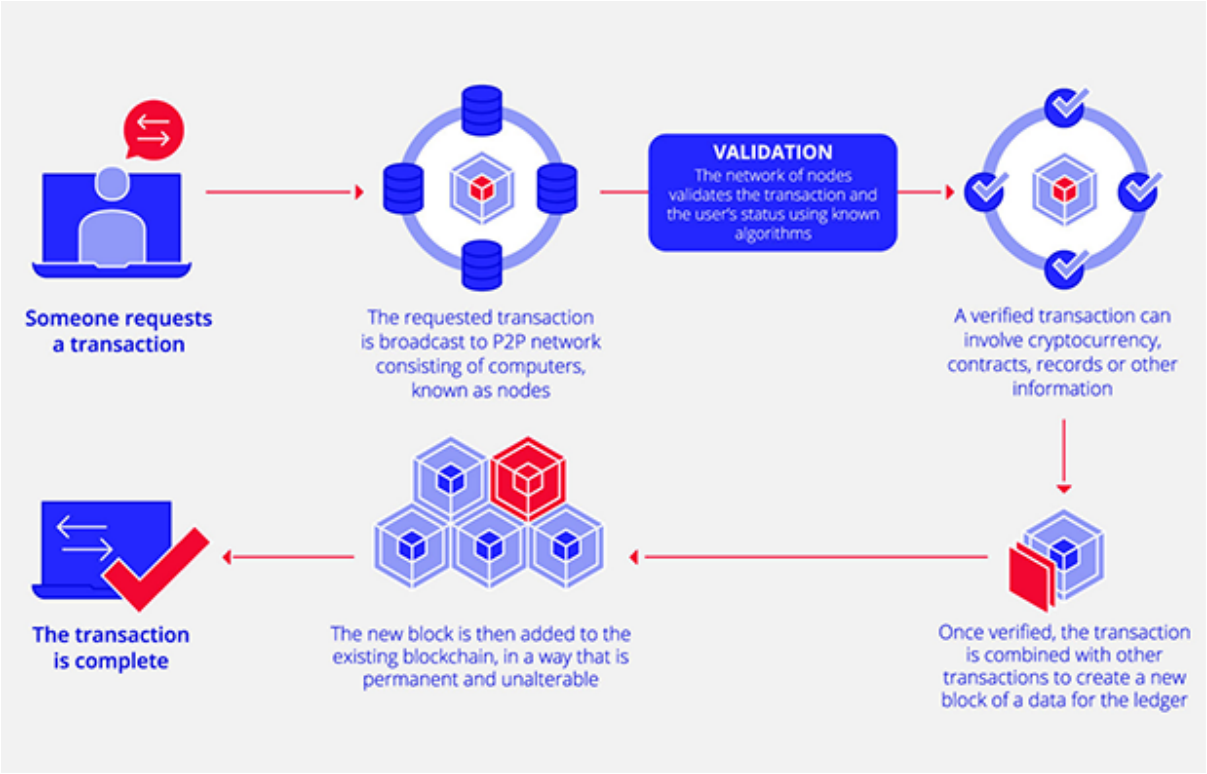

Imagen 5. Esquema de funcionamiento de blockchain. [16]

2.1.8. Casos de uso

En esta sección se presentan casos de usos comunes de la tecnología blockchain.

Criptomonedas
Son un sistema de dinero electrónico que permite transacciones entre pares sin necesidad

de involucrar a una entidad centralizada. Bitcoin, que dio origen a blockchain, es un tipo de criptomoneda [3].

Es una alternativa que busca abordar algunos problemas que padecen los medios de pago electrónicos tradicionales. Esta alternativa se enfoca en:

- Reducir costos adicionales de operatividad que genera la mediación de la entidad confiable.
- No permitir la reversión de pagos.
- Evitar incertidumbre por la falta de confianza entre los involucrados.
- Mitigar las posibilidades de fraude.

Logística

Dada la complejidad de los acuerdos en el área de logística y cadena de suministros, el intercambio de información a lo largo de los procesos es crucial. [1]

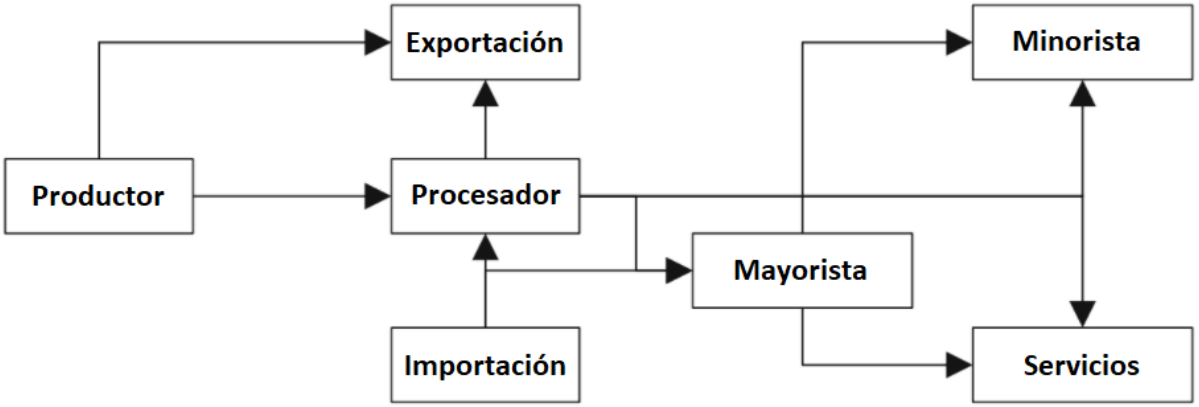

Imagen 6. Ejemplo de partes involucradas en una cadena de suministros. [3]

En la imagen 6 se ilustran distintas partes que están involucradas en una cadena de suministros. Estas partes intercambian bienes e información entre sí, y además tienen diferentes intereses comerciales o económicos.

Blockchain puede ayudar a resolver conflictos entre las partes involucradas y provee soluciones a requisitos claves del área:

- Interoperabilidad: coordinación del intercambio de información.
- Latencia: el intercambio de información debe ser lo suficientemente rápido para no retrasar el intercambio de productos.
- Integridad: la información no puede ser falsificada.
- Confidencialidad: debido a temas comerciales y diferencias de intereses entre los involucrados, cierta información debe poder mantenerse de forma confidencial.
- Escalabilidad: debe poder soportar una creciente cantidad de transacciones.

Votaciones

Las características de blockchain brindan soluciones a algunos problemas que se presentan en los sistemas de información para votaciones [1]:

- Las votaciones deben ser anónimas.
- Los votantes deben poder consultar si su voto es válido, sin que su voto sea revelado.
- La acción de determinar qué votos son válidos debería poder realizarse sin la necesidad de privilegios especiales.

2.2. Plataformas de blockchain

Las plataformas de blockchain son herramientas que permiten el desarrollo de aplicaciones descentralizadas sobre la tecnología blockchain [2]. Para permitir el desarrollo de dichas aplicaciones, estas plataformas generalmente proveen la posibilidad de utilizar smart contracts. Es en los smart contracts donde se implementa la lógica de negocio.

Estas plataformas, entre otras cosas, se encargan de manejar la cadena de bloques, cómo las transacciones se guardan en ella, y de implementar y ejecutar los algoritmos de consenso. Esto provee una abstracción de más alto nivel para quienes desarrollan sobre estas plataformas, ya que no deben preocuparse por estos aspectos de bajo nivel y así poder enfocarse en el desarrollo de smart contracts que contendrán la lógica de las aplicaciones [1].

Cabe destacar la diferencia entre blockchain y plataformas de blockchain. El libro *Architecture for Blockchain Applications* define las plataformas de blockchain de la siguiente manera:

“Una plataforma de blockchain es la tecnología necesaria para operar una blockchain. Esto incluye el software que opera los nodos, el almacén de datos de los nodos y cualquier otro software cliente utilizado para acceder a la red de la blockchain.” [3]

Se puede llegar a ver a blockchain como el concepto y a una plataforma de blockchain como la tecnología que lo implementa y facilita el desarrollo de aplicaciones sobre este concepto.

Las aplicaciones descentralizadas que se implementan sobre estas plataformas de blockchain son utilizadas por otras aplicaciones (aplicaciones tradicionales) externas que son quienes disparan operaciones en la aplicación distribuida. Esto se hace, generalmente, disparando la ejecución de un smart contract o una operación del mismo con ciertos parámetros.

En este proyecto se trabajó con dos plataformas de blockchain: Hyperledger Fabric y Corda, las cuales se describen a continuación.

2.2.1. Hyperledger Fabric

Hyperledger Fabric es un framework modular que permite la construcción de blockchains permisionadas y soporta smart contracts. Esta plataforma actúa como base para el

desarrollo de productos, soluciones y aplicaciones que están destinados a uso empresarial y privado [17].

Fabric ataca el hecho de que las blockchain tradicionales no admiten transacciones privadas y contratos confidenciales que son de suma importancia para las empresas (lo cual se ilustra en el ejemplo que se encuentra más adelante en esta sección). Transacciones privadas y contratos confidenciales se refiere a transacciones o smart contracts que son visibles solo por un subconjunto de nodos de la red y no por todos. Hyperledger Fabric se diseñó en respuesta a esto como una base modular, escalable y segura para ofrecer soluciones de blockchains empresariales.

En el resto del documento se menciona a Hyperledger Fabric y Fabric de forma indistinta.

Ejemplo de uso
Supongamos que hay un fabricante que desea enviar chocolates a un minorista específico o mercado de minoristas (por ejemplo, todos los minoristas de EE. UU.) a un precio específico, pero no quiere revelar ese precio en otros mercados (por ejemplo, minoristas chinos).

Dado que el movimiento del producto puede involucrar a otras partes, como aduanas, una empresa de transporte y un banco financiero, el precio privado puede revelarse a todas las partes involucradas si se utiliza una versión básica de la tecnología (como la descrita en la sección 2.1) blockchain para respaldar esta transacción.

Hyperledger Fabric aborda este problema manteniendo las transacciones privadas en la red; solo los participantes que necesitan saber conocen los detalles necesarios. La partición de datos en la cadena de bloques permite que los puntos de datos específicos sean accesibles sólo para las partes que necesitan saberlo.

A continuación se presentan los principales componentes y conceptos que hacen a Hyperledger Fabric y, en el anexo IV se presenta describe el flujo de transacciones que utiliza la plataforma.

2.2.1.1. Smart contracts

En esta plataforma, los smart contracts pueden ser escritos en tres lenguajes distintos: Javascript, Java y Go [18].

En Hyperledger Fabric, el estado del sistema es un conjunto de pares clave-valor y el mismo es leído y modificado por los smart contracts [19]. Cada ejecución de una operación de un smart contract da como resultado un conjunto de clave-valor que contiene aquellas claves que deben ser actualizadas. Además de lo anterior, cada ejecución también da como salida un conjunto de clave-valor que contiene aquellas claves (y sus valores) que leyó del estado del sistema. Este último conjunto es necesario para la validación de transacciones que se verá más adelante en este documento.

2.2.1.2. Eventos

Los smart contracts implementados en esta plataforma tienen la posibilidad de emitir eventos [20]. Estos eventos son útiles para que, las aplicaciones externas que utilizan la blockchain, puedan ser notificadas de lo que sucede en la blockchain. Además de los eventos que pueden ser disparados desde el código de los smart contracts, Fabric por defecto emite eventos cuando se genera un nuevo bloque y cuando una operación de un smart contract es ejecutada. Cualquiera de estos tres tipos de eventos son emitidos una vez que el mismo está confirmado por todos los nodos (o los necesarios) de la blockchain. Esto proporciona, a las aplicaciones externas que utilizan la blockchain, la certeza de que el evento es algo confirmado por todos los nodos necesarios y no un evento disparado por un único nodo sin que haya sido validado por los demás.

2.2.1.3. Canales

Fabric, como se mencionó en la sección 2.2.1, permite transacciones privadas y, para lograr esta funcionalidad, utiliza el concepto de “canal” [21]. Un canal puede ser visto como un grupo formado por un subconjunto de los nodos de la red. Toda transacción realizada en Fabric pertenece a un canal y, únicamente los nodos miembros de dicho canal, pueden ver la transacción.

2.2.1.4. Tipos de nodos

En Fabric existen dos principales tipos de nodos con roles distintos. Estos son los nodos Peers y Orderers. [22]

Peers
Los nodos peers contienen una copia de la cadena de bloques y smart contracts, y son los encargados de realizar la ejecución de estos últimos. La ejecución de los smart contracts genera transacciones que luego son enviadas a los nodos orderers para que sean colocadas en bloques.

Orderers
Los nodos orderers son los encargados de recibir transacciones, ordenarlas y generar nuevos bloques. Cada vez que se genera un nuevo bloque, estos nodos envían dichos bloques a los nodos peers para que lo agreguen a su cadena de bloques.

2.2.1.5. Interacción con aplicaciones externas

Hyperledger Fabric provee una SDK [23] implementada (hasta el momento) en Java y Node.js que puede ser utilizada por aplicaciones externas que requieran interactuar con las aplicaciones distribuidas construidas sobre esta plataforma. Esta SDK hace que dicha interacción sea realmente fácil y entre las principales funciones que permite realizar se encuentran:

- Disparar operaciones de smart contracts, ya sean operaciones que devuelven información del estado del sistema actual o modificarlo. En caso de operaciones que

modifican el estado del sistema, ofrece la posibilidad de saber cuando esta operación es confirmada a nivel de toda la blockchain.
- Escuchar eventos

Existen intenciones por parte de los desarrolladores de esta plataforma de proveer esta SDK en Python y Go en futuras versiones.

2.2.2. Corda

De forma similar a Hyperledger Fabric, Corda es una plataforma que permite la construcción de blockchains permisionadas, con foco en el ámbito empresarial y privado. [24], [25], [26]
A continuación se describen algunas de sus características:

- **Consenso:** Brinda la posibilidad de combinar diferentes mecanismos de consenso dentro de una misma blockchain.

- **Propagación de transacciones:** Las transacciones son propagadas únicamente a las partes involucradas. Esto ofrece privacidad y agilidad a la hora de confirmar transacciones. La agilidad para confirmar transacciones es consecuencia de evitar esperar por la confirmación por parte de nodos que no necesitan participar de dichas transacciones.

- **Lenguaje de programación:** Emplea un lenguaje de programación ampliamente utilizado y de alto nivel como Java.

- **Transacciones confirmadas en tiempo real:** Las transacciones no se agrupan en bloques para ser almacenadas en la cadena, sino que son las transacciones en sí mismas las que forman la cadena.

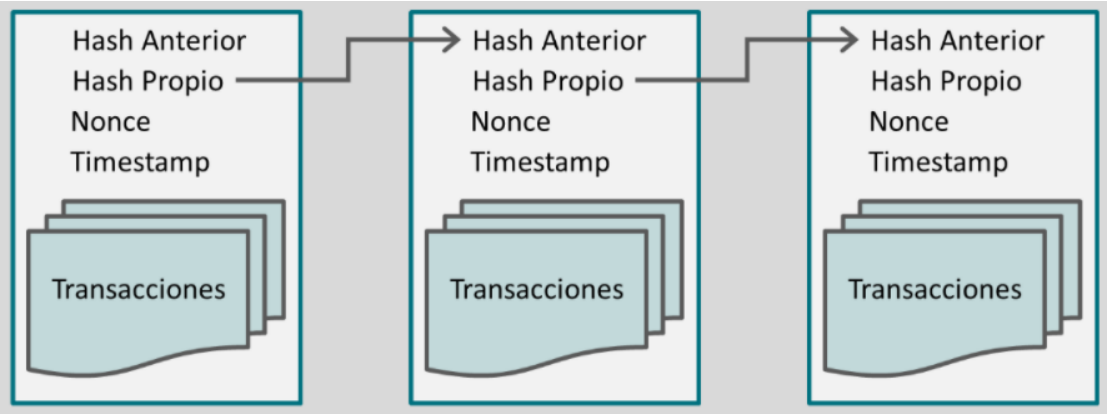

Imagen 7. Concepto de cadena de bloques [26].

En la imagen 7 se ilustra una cadena de bloques tradicional, donde cada bloque almacena una lista de transacciones. Corda aborda esto de una manera diferente y opta por no utilizar bloques. En su lugar, como se observa en la imagen 8, cada eslabón de la cadena contiene una única transacción:

Imagen 8. Cadena de transacciones en Corda.

Esto es un factor que influye en la velocidad en que las transacciones son confirmadas, ya que para que una transacción sea almacenada en el *ledger*, no se debe aguardar por otras transacciones para generar un nuevo bloque en la cadena [27].

La diferencia radica en que para aprobar una transacción, ésta no se comparte con toda la red, sino que se comparte solamente con las partes involucradas y son ellas las que validan la transacción. Una vez que todas las partes involucradas firman la transacción, ésta se registra en el ledger. En consecuencia, el proceso de confirmación de una transacción resulta más rápido que con la definición tradicional vista en la sección 2.1.7.

A continuación se presentan los principales componentes y conceptos que hacen a la plataforma. Algunos conceptos coinciden con los presentados en la definición general de blockchain en la sección 2.1, y otros son adaptaciones propias de la plataforma.

En el anexo V se presentan más conceptos de Corda, incluyendo: estados, transacciones, contratos, mecanismo de consenso y nodos especiales.

2.2.2.1. Ledger

El concepto de *ledger* en blockchain, como se menciona en la sección 2.1, hace referencia al lugar donde se registran las transacciones. Sin embargo, en el caso de Corda, el *ledger* está compuesto de una forma diferente. Esta plataforma no posee un único almacenamiento de datos que concentre toda la información del *ledger*, sino que por el contrario, cada nodo mantiene su propia base de datos donde almacena todos los datos con los que está involucrado [28]. Por ejemplo, si el nodo A le presta dinero al nodo B y quieren registrarlo en el ledger, entonces los nodos A y B serán los únicos nodos en la red que compartirán y/o almacenarán la transacción.

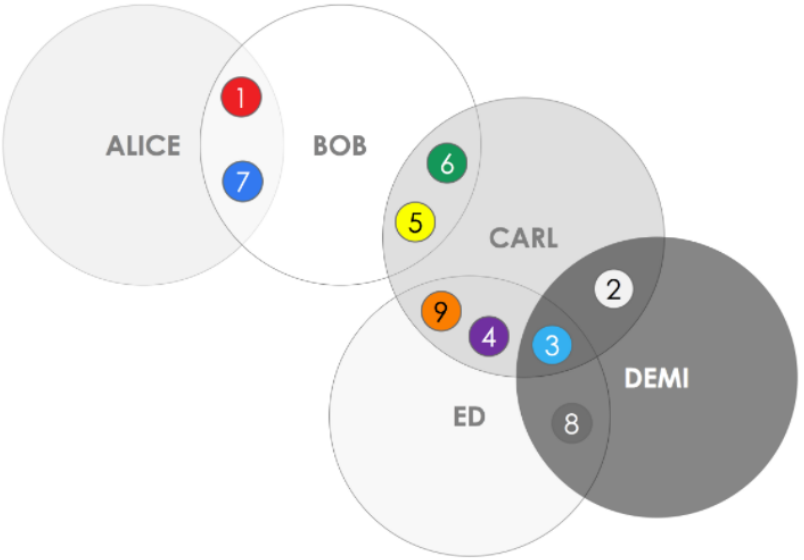

Imagen 9. Representación del ledger en Corda. [27]

En la imagen 9 se representa el *ledger* mediante un diagrama de Venn, donde cada conjunto es un nodo de la red, y las intersecciones contienen las transacciones en que los nodos participaron.

En Corda, el *ledger* es subjetivo desde el punto de vista de cada nodo y no hay nodos que posean una visión global de todo *ledger*. Este es un ejemplo de variación de los conceptos generales de blockchain, tal como se menciona en la sección 2.1.

2.2.2.2. Flujos

En Corda las comunicaciones entre nodos no se realizan a través de *broadcasts,* sino que, se establecen comunicaciones punto a punto entre nodos. Eso implica que para actualizar el *ledger* hay que coordinar y especificar qué información se va a enviar, a qué nodos y en qué orden [29].

Para automatizar este proceso de comunicación punto a punto Corda ofrece el concepto de *flujo*, el cual consta de una secuencia de pasos que llevan a una actualización del *ledger*.

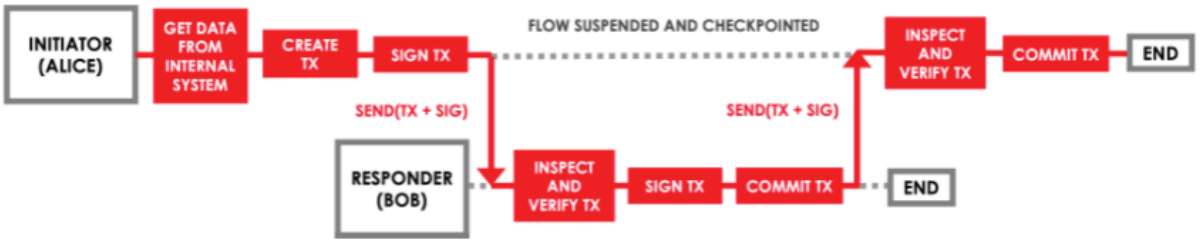

Imagen 10. Ejemplo de flujo en Corda. [28]

En la imagen 10, se ilustra el proceso de comunicación entre dos nodos cuando se genera una nueva transacción. En primera instancia, el iniciador crea una transacción, la firma y la envía a otro nodo participante de la transacción. Una vez que el segundo participante recibe

la transacción, éste la verifica y adiciona su firma, para luego retornar la transacción firmada al nodo que la inició.

Corda provee una librería de flujos que intenta cubrir las necesidades de ciertas tareas comunes, evitando que deban ser implementadas para cada aplicación. Algunas de los procesos comunes que se abstraen con esta librería son:

- Notarizar y almacenar una transacción
- Recolectar las firmas de los nodos participantes de una transacción
- Verificar una cadena de transacciones

Los flujos pueden ser ejecutados concurrentemente. A su vez, su implementación provee un sistema de *checkpoints* y persistencia de transacciones para poder pausar y reanudar su ejecución a lo largo del tiempo. Esto permite ejecutar flujos de larga duración de una forma segura, y garantizar que otros flujos menos extensos serán ejecutados en paralelo en caso de que haya capacidad ociosa.

2.2.2.3. Estados

Un estado es un objeto inmutable que representa información que es conocida por uno o más nodos de la red en determinado momento. La información que pueden contener es arbitraria y pueden representar cualquier clase de datos como préstamos, acciones, etc.

Complementando la sección 2.2.2.1, el *ledger* de cada nodo se puede ver como el conjunto de todos los estados que el nodo conoce actualmente. En la imagen 11 se puede ver una representación de algunos de los datos que contiene un estado.

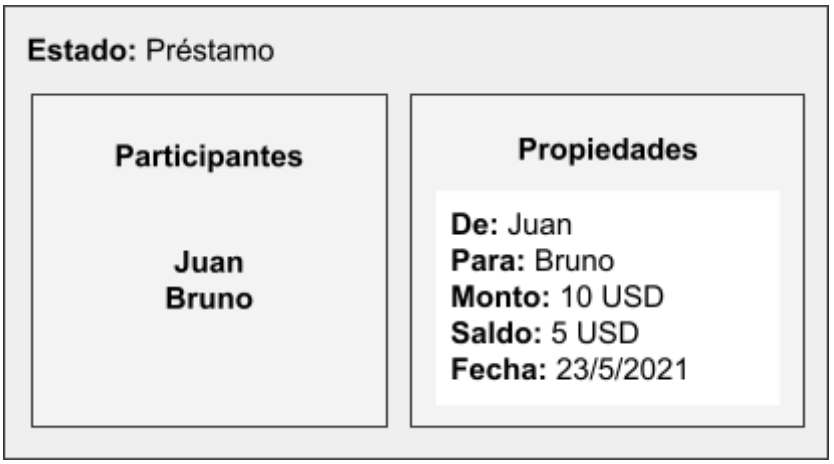

Imagen 11. Representación de un estado en Corda.

Como los estados son inmutables, éstos no se pueden modificar para reflejar un cambio en la realidad. En su lugar, los cambios se representan mediante lo que se conoce como **secuencia de estado**: se crea una nueva versión del estado que represente la nueva realidad y se marca el estado actual como histórico. En la imagen 12 se ilustra una secuencia de estados.

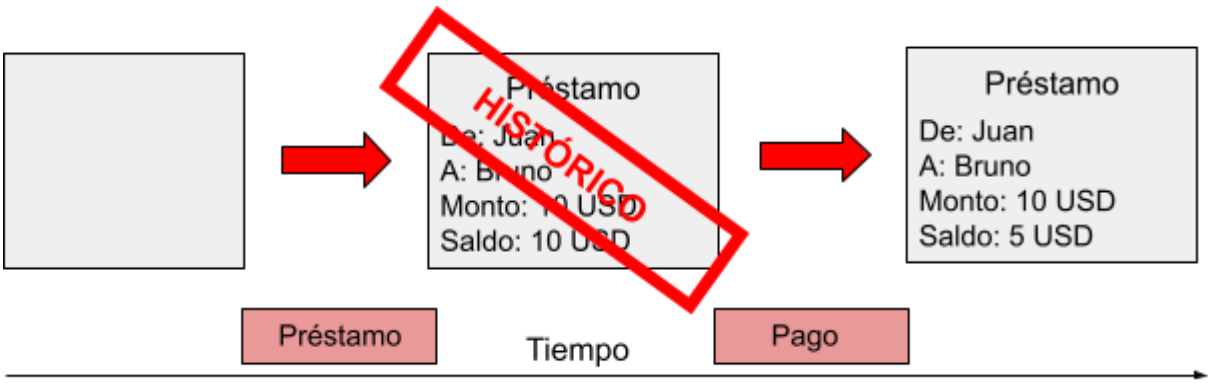

Imagen 12. Secuencia de estados en Corda.

2.2.2.4. Transacciones

A grandes rasgos, las transacciones son propuestas de modificaciones al *ledger*. Corda utiliza el modelo *UTXO (Unspent Transaction Output)*, donde los estados del *ledger* son inmutables. Con este modelo, las transacciones actualizan el *ledger* tomando un conjunto de estados existentes (*inputs*) y generando un conjunto de estados nuevos (*outputs*) que reflejan la nueva realidad.

De esta forma, el *ledger* evoluciona a lo largo del tiempo mediante la aplicación de transacciones. Éstas representan los saltos en la secuencia de estado presentada en la sección anterior.

Una de las principales características de Corda es que no posee una cadena de bloques, sino una cadena de transacciones. Para generar la cadena, cada transacción referencia a la transacción que generó el estado que se está referenciando como *input*, utilizando un *hash* del contenido de la transacción como identificador. Esto se ilustra en la imagen 13.

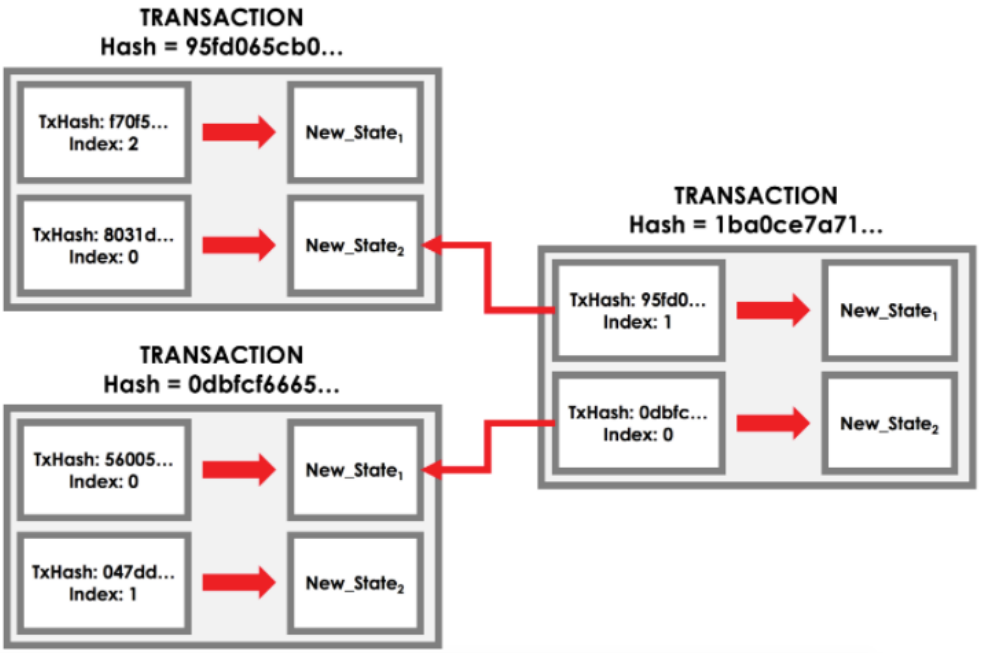

Imagen 13. Referencias entre transacciones.

Hasta que las transacciones no sean confirmadas, son meras propuestas de modificar el *ledger*. Para que una transacción se confirme, debe recibir las firmas de todos los nodos que participen de la misma, quienes adjuntan su firma a la transacción indicando que aprueban la propuesta de modificación luego de realizar el conjunto de validaciones necesarias. Además de obtener las firmas, las transacciones deben satisfacer un *contrato* para que éstas sean confirmadas.

La confirmación de una transacción tiene dos consecuencias:

- Los estados *inputs* son marcados como históricos.
- Los estados *outputs* pasan a ser parte del estado actual del *ledger*.

2.3. Interoperabilidad en blockchain

Según la Real Academia Española, la interoperabilidad se define como la *"capacidad de los sistemas de información, y por ende de los procedimientos a los que estos dan soporte, de compartir datos y posibilitar el intercambio de información y conocimiento entre ellos"* [30]. De forma complementaria, también la define como la *"habilidad de dos o más sistemas o de sus componentes para usarse de forma conjunta e intercambiable".*

Existen múltiples mecanismos y middlewares para interoperar sistemas de software en contextos fuera de blockchain, tales como web services SOAP [31], REST APIs [32], entre otros [33]. De forma similar a cómo surgieron estos mecanismos en su momento, la industria y la comunidad científica se encuentra desarrollando mecanismos para interoperar blockchains [34], [35].

Con el crecimiento de la adopción de blockchain se han ido introduciendo una gran variedad de plataformas; cada una con sus finalidades, funcionalidades, ventajas y desventajas [2].

Los resultados de la encuesta global del blockchain de 2018 realizada por PWC [36] arrojan que el 28% de los encuestados considera que la interoperabilidad entre blockchains es clave para el éxito en la adopción de esta tecnología en sus negocios. En particular, una de los principales desafíos que identifican es la falta de estándares para interoperar, ya que sin estos estándares, las organizaciones deben acordar las reglas y modelos a seguir. A su vez, el 41% de los encuestados considera que la incapacidad de las plataformas de blockchain de trabajar de forma conjunta, es una de las principales barreras para la adopción de esta tecnología.

La variedad de plataformas y casos de uso posibles pone sobre la mesa diversas de acciones que se pueden llegar a realizar en un contexto de blockchain; tales como transferir un activo, ejecutar software arbitrariamente a través de *smart-contracts*, entre otras. Sin embargo, estas plataformas son independientes y desconectadas. De esta forma existe una fragmentación tal que las funcionalidades quedan encapsuladas dentro de una misma blockchain, sin posibilidad de comunicarse con otra blockchain que posea una funcionalidad

complementaria. A modo de ejemplo, una pregunta que quedaría sin responder si no se hablara de interoperabilidad en blockchain sería: ¿Cómo ejecutar un smart contract en una blockchain B al transferir un activo en una blockchain A?

A continuación se presentan conceptos y características que llevan a la diversidad y heterogeneidad de plataformas de blockchain que existen.

2.3.1. Fragmentación y aislamiento de plataformas existentes

Cada plataforma de blockchain posee sus propias características y éstas pueden variar de una a otra generando una amplia diversidad. Por ejemplo, las plataformas pueden diferir en algunos aspectos como [37]:

- Públicas o privadas
- Permisionadas o no permisionadas
- Tecnología de implementación
- Mecanismo de almacenamiento de las transacciones
- Protocolos de consenso
- Tamaño de bloque

Esta diversidad genera que la decisión de qué plataforma utilizar para implementar una blockchain en determinado escenario, sea una tarea compleja. En ese sentido se han realizado estudios para facilitar esta tarea y generar guías que ayuden a decidir cuál es la plataforma más adecuada [2].

A modo de ejemplo, si bien las características de las plataformas Hyperledger Fabric y Corda utilizadas en este proyecto son bastante similares, también presentan algunas diferencias. Por ejemplo, desde un punto de vista conceptual, Corda no almacena las transacciones agrupadas en bloques como lo hace Fabric. Por otro lado, desde un punto de vista técnico, Fabric provee un SDK para la comunicación extra blockchain mientras en Corda se realiza de forma más artesanal.

2.3.2. Desafíos de interoperabilidad

En esta sección se mencionan algunos aspectos importantes de la interoperabilidad que deben ser abordados al proveer una solución que permita comunicar dos plataformas de blockchain.

Operaciones *cross-blockchain*
Las operaciones *cross-blockchain* son operaciones de las que participa más de una blockchain. Resulta clave identificar las distintas operaciones que se deben poder realizar entre las blockchains a interoperar. Por ejemplo, en contextos de criptomonedas es común transferir activos de una blockchain a otra; mientras que en un contexto de cadena de suministro puede interesar ejecutar un *smart contract* en otra blockchain para registrar un intercambio de mercadería.

Estas operaciones pueden involucrar una o más blockchains y su complejidad varía ampliamente. Por ejemplo, una transacción puede requerir ser confirmada en más de una blockchain para que la transacción se considere confirmada.

Por lo general, las operaciones *cross-blockchain* son realizadas a través de protocolos de comunicación *cross-blockchain*, los cuales se diseñan para satisfacer las necesidades de dichas operaciones [37]. Por ejemplo, en el ejemplo anterior, el protocolo debe proveer un mecanismo para aguardar por la confirmación de todas las blockchains.

Protocolo de comunicación *cross-blockchain*
Un protocolo de comunicación *cross-blockchain* (CBCP) [37] define el proceso con el cual un par de blockchains se comunican para sincronizar correctamente transacciones *cross-blockchain*. Estos protocolos existen tanto para comunicar blockchains implementadas con la misma plataforma (homogéneas) [38], como implementadas en distintas plataformas (heterogéneas) [39]. En este informe nos centraremos en la comunicación entre blockchains heterogéneas.

A continuación se describe cómo las plataformas se comunican con aplicaciones externas, de forma de lograr la interoperabilidad.

Comunicación de las plataformas con aplicaciones externas
Una parte clave de la comunicación *cross-blockchain* es la capacidad de las plataformas de comunicarse con sistemas externos. Existen plataformas como Hyperledger Fabric que proveen herramientas (SDKs) que resuelven esa comunicación de forma transparente para los desarrolladores, y otras plataformas como Corda que proveen los *building blocks* pero brindan a los desarrolladores la libertad de implementar la solución más conveniente de forma artesanal.

Estos mecanismos de comunicación por lo general se basan en el protocolo HTTP y es un punto de la interoperabilidad en el que es necesario aplicar mecanismos de seguridad, ya que queda fuera de la jurisdicción de los mecanismos de seguridad de las blockchains. Estos mecanismos dependen de la plataforma, ya que cada una propone su propia estratégia. Por ejemplo, Fabric provee un SDKs que resuelve la comunicación con aplicaciones externas de forma transparente. Este SDK incluye una API de alto nivel que permite enviar y recibir mensajes a la blockchain. Por otro lado, Corda no provee un SDK, si no que provee un servidor Spring Boot[1] en el cual se pueden implementar endpoints que permitan la comunicación desde aplicaciones externas.

A continuación se describe el problema de determinar cuándo una transacción que involucra a más de una blockchain, se considera confirmada.

Protocolos de consenso y finalidad de transacciones
Los mecanismos de consenso son otro punto importante de la interoperabilidad. Pueden haber casos donde es suficiente que haya consenso en cada una de las blockchains, pero también situaciones donde sea necesario un consenso que abarque a más de una blockchain y se deban combinar los resultados de ambos protocolos de consenso. Estos

[1] Spring Boot: https://spring.io/quickstart

casos son complejos de resolver e involucran protocolos de comunicación más sofisticados. [37]

Asociado al mecanismo de consenso se encuentra la finalidad de las transacciones. Es importante que la comunicación de una blockchain considere el momento en que una transacción es confirmada, de lo contrario pueden darse situaciones donde una transacción es confirmada en una blockchain y no en otra, generando inconsistencia en la información.

2.3.3. Niveles de interoperabilidad

El *National Interoperability Framework Observatory* de la Unión Europea define cuatro niveles de interoperabilidad (representados en la imagen 14): legal, organizacional, semántica y técnica [37], [40].

Imagen 14. Niveles de interoperabilidad. [40]

En este proyecto se trabaja únicamente sobre el nivel de interoperabilidad técnica.

Interoperabilidad legal
Trata sobre lograr que la legislación o reglas que regulan las distintas organizaciones, permitan interacciones entre dichas organizaciones. Este nivel de interoperabilidad facilita los objetivos de los niveles inferiores.

Implica por ejemplo revisar aspectos reglamentarios en cada una de las organizaciones que sean sobre restrictivas, contradictorias o desactualizados, que limiten las posibilidades de interoperar con otras organizaciones.

Interoperabilidad organizacional
En este nivel se definen los procesos de negocio a seguir para lograr los fines comunes entre las distintas organizaciones que interoperan. Implica generar procesos y documentación clara, que todas las partes entiendan y compartan, dando así un marco formal a los objetivos de la interacción entre las organizaciones.

Interoperabilidad semántica
Se trata de estandarizar el formato y significado de la información intercambiada entre todas las partes. El aspecto semántico se encarga de asegurar que el significado de los datos y

sus relaciones sean las deseadas; mientras que el aspecto sintáctico se encarga de definir los formatos en que se intercambia la información.

Por ejemplo, algunas de las actividades realizadas en este nivel son definir vocabularios, taxonomías, modelos y estructuras de datos.

Interoperabilidad técnica
Abarca las aplicaciones y la infraestructura que comunica los diferentes sistemas interoperantes. Este nivel incluye la especificación de interfaces de comunicación, protocolos, mecanismos de integración de datos, políticas de seguridad, entre otros.

Se intenta abordar con una visión global y no encapsulada en cada organización, además de utilizar una especificación técnica formal.

2.4. Soluciones de interoperabilidad existentes

Para obtener un panorama amplio de las soluciones de interoperabilidad propuestas hasta el momento, nos basamos principalmente en el paper [37]. Dicho paper fue publicado en mayo de 2020 y hace un gran relevamiento de las soluciones existentes. El mismo organiza las soluciones en tres grandes categorías, de las cuales algunas están a su vez divididas en subcategorías. Las tres categorías son Cryptocurrency-directed approaches, blockchain engines y blockchain connectors. Las soluciones contenidas en las dos primeras categorías no son necesarias para la comprensión del resto del documento, por eso las mismas se encuentran descritas en los Anexos I y II respectivamente. De la categoría blockchain connectors, únicamente la subcategoría Trusted relays es necesaria para la comprensión de este informe, por lo tanto, será la única que se describirá. El resto de las subcategorías son descritas en el Anexo III.

2.4.1. Blockchain connectors

Esta categoría está compuesta por soluciones que no entran dentro de las categorías “Cryptocurrency-directed approaches” ni “blockchain engines”. Esta categoría es dividida en cuatro subcategorías:

- Trusted Relays
- Blockchain Agnostic Protocols
- Blockchain of Blockchains
- Blockchain Migrators

2.4.1.1. Trusted relays

Trusted relays son entidades de confianza que se encargan de redireccionar transacciones de una blockchain a otra. Estas soluciones generalmente se utilizan en ambientes permisionados.

Dentro de esta categoría, se relevan tres soluciones: HyperLedger Cactus y dos soluciones propuestas en artículos, las cuales se describen a continuación:

HyperLedger Cactus

Esta solución, al momento de realizar el relevamiento de soluciones existentes, se encontraba aún en desarrollo (en fase de implementación). Según el whitepaper de la solución [34], la misma permitirá realizar transacciones cross chain entre distintas plataformas de blockchain existentes. Entre las plataformas que planea soportar estan Hyperledger Fabric, Hyperledger Besu, Corda y Quorum [41].

En esta solución, existe una red de validadores por cada plataforma que se desea interoperar. Estos validadores se encargan de detectar transacciones cross chain, firmarlas y enviarlas. Una transacción es válida si está firmada por un conjunto de validadores. A su vez, existe una entidad que se encarga de "routear" las transacciones cross chain.

Estos conceptos están representados de forma gráfica en la imagen 15. V1 a V6 son los validadores de cada una de las plataformas.

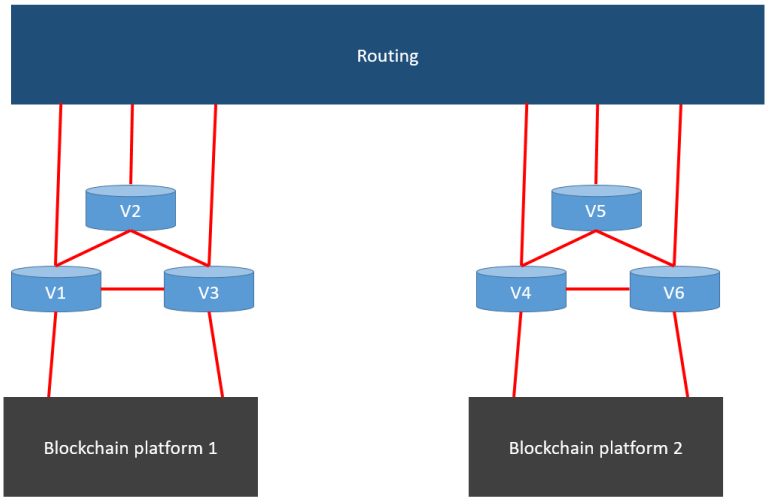

Imagen 15. Diagrama de Hyperledger Cactus

Abebe et al

En este artículo se propone un protocolo generalizado para la transferencia de datos entre blockchains permisionadas [42]. En el artículo, aplican el protocolo propuesto sobre dos blockchains construidas sobre HyperLedger Fabric aunque, según los autores, se podría extender a otras plataformas.

La solución propuesta introduce tres conceptos (representados en la imagen 16): system contracts, relay services y un protocolo de comunicación.

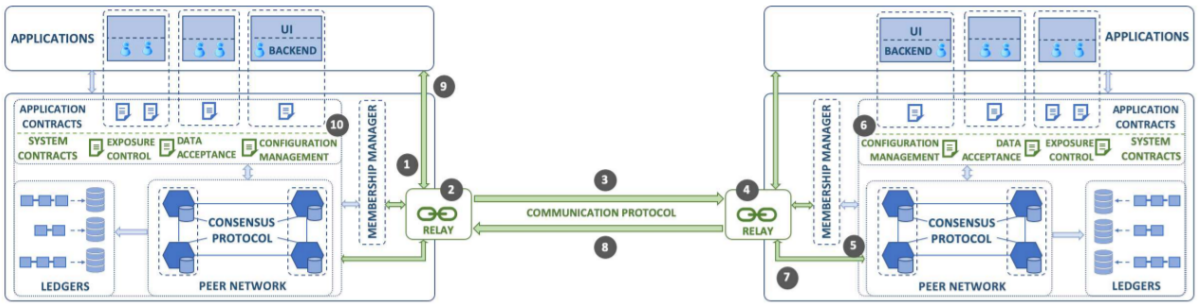

Imagen 16. Diagrama de arquitectura [42].

Los relay services (hay un relay service por cada blockchain) reciben las solicitudes de información de las aplicaciones y utilizan las blockchains para responder a dichas solicitudes. Si un relay service recibe una solicitud (query) que requiere información de otra blockchain que no es la blockchain a la cual representa, entonces se comunicará con el relay service de dicha blockchain. En dicha comunicación se utiliza el protocolo de comunicación y el mismo maneja los siguientes aspectos:

- Define pruebas que aseguran la identidad de quienes realizan las consultas (queries).
- Define pruebas que aseguran la veracidad de las respuestas a las solicitudes.
- Permite definir la blockchain y contrato destino al cual está destinada la consulta.

Los system contracts se encargan de controlar el acceso a la información mediante la aplicación de ciertas reglas y de esta forma se controla la exposición de la información.

Kan et al

En este artículo se propone un protocolo y una arquitectura (se muestran en la imagen 17) para que distintas plataformas de blockchain puedan interoperar (ejecutar transacciones cross chain) [43], pero dicho protocolo y arquitectura requieren que las plataformas existentes se adapten al mismo. Los autores probaron dicho protocolo/arquitectura con dos custom blockchains construidas por ellos mismos.

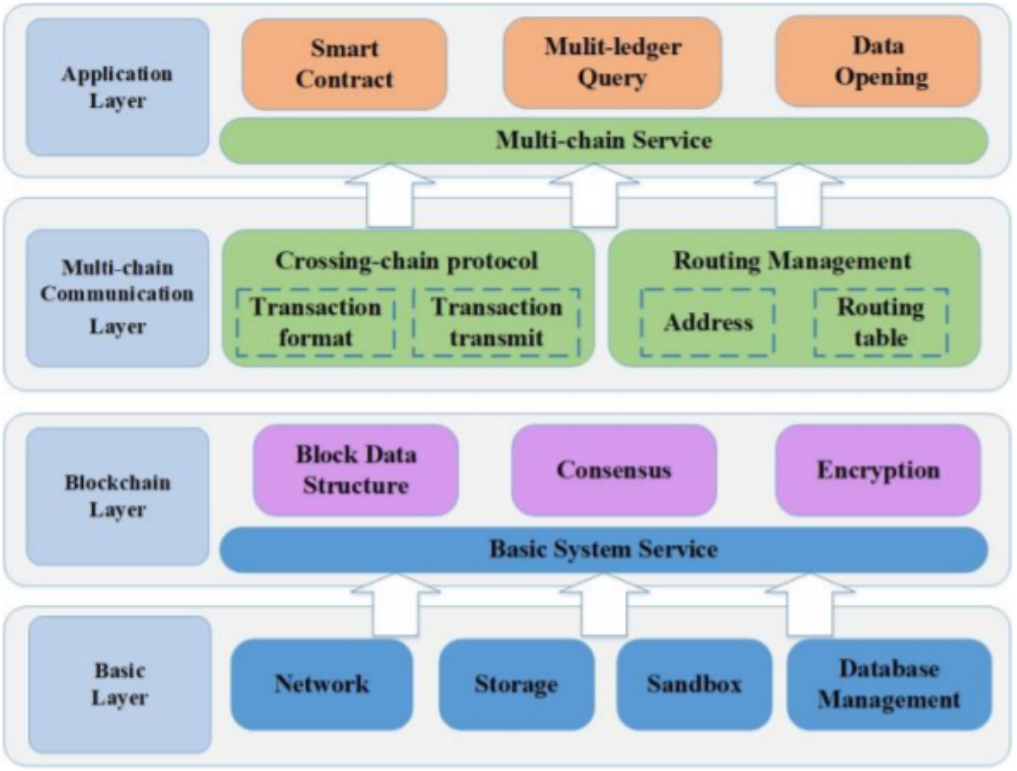

Imagen 17. Diagrama de arquitectura [43].

3. Análisis de requerimientos

En este capítulo se realiza un análisis de la problemática planteada al principio del proyecto.

3.1. Requerimientos del proyecto

El requisito principal del proyecto consiste en encontrar una forma de interoperar plataformas de blockchain y aplicar la misma sobre las plataformas Hyperledger Fabric y Corda. Para esto, se debe explorar soluciones existentes y, en caso de que alguna de ellas se adapte al escenario del proyecto, experimentar con ella. En caso contrario, se debe proponer una solución propia y realizar una implementación a modo de prueba de concepto.

Es importante mencionar que la solución de interoperabilidad buscada no debe modificar las plataformas de blockchain, sino utilizarlas tal cual están.

El segundo requerimiento consiste en implementar un escenario propuesto sobre ambas plataformas de blockchain e interoperar las mismas con la solución mencionada anteriormente. El escenario propuesto pertenece al contexto de la seguridad social en el que se registran intercambios de información entre organizaciones de seguridad social de distintos países. Dicho escenario es utilizado para aplicar la solución de interoperabilidad mencionada anteriormente y se describe en mayor detalle a continuación.

3.2. Descripción del escenario propuesto

Las organizaciones de seguridad social de distintos países frecuentemente intercambian información relativa a los años trabajados por una persona en un país o el estado de vida (si se encuentra con vida o no). Esta información e intercambio es necesario en situaciones donde las personas han trabajado en un país pero deciden cobrar su jubilación en otro.

Cuando un país necesita información de seguridad social de otro país, debe realizar una solicitud al país que corresponda y esperar una respuesta con dicha información. Las respuestas a las solicitudes demoran un cierto tiempo y es aquí donde se presentan malos entendidos entre las organizaciones. Dichos malos entendidos giran en torno a las fechas de solicitudes y respuesta. A modo de ejemplo, una organización podría sostener que realizó una solicitud de información en determinada fecha, mientras que la organización que recibió la solicitud, podría sostener que la misma fue realizada en otra fecha.

Debido a lo anterior, resulta necesario contar con un sistema que registre dichas solicitudes y respuestas de forma que estas puedan ser auditadas. A su vez, debido a que las distintas organizaciones podrían discrepar acerca de las fechas de solicitudes y respuestas, las características de descentralización e inmutabilidad que provee blockchain son muy útiles en este escenario para resolver dichas discrepancias.

Por otro lado, se debe contemplar la posibilidad de que un conjunto de países utilicen una aplicación construida sobre una plataforma de blockchain mientras que, otro conjunto de

países utilicen otra aplicación construida sobre otra plataforma para registrar sus intercambios. Es aquí donde la interoperabilidad resulta necesaria, ya que, los intercambios realizados entre países que utilizan distintas aplicaciones, deben quedar registrados en ambas aplicaciones.

Cuando un país envía una solicitud (o pedido) de información a otro, el país solicitante envía un paquete (al país al que le solicita) que contiene los detalles de la información que está solicitando. Este paquete no será guardado en las aplicaciones pero sí un hash del mismo. A su vez, cuando se realiza un pedido de información se define una fecha esperada de respuesta.

De igual manera que lo anterior, cuando un país responde un pedido de información, envía un paquete que contiene la información propiamente solicitada y el hash del mismo debe ser guardado en la aplicación.

En la imagen 18, se representa de forma gráfica el caso de uso descrito. En dicha imagen, a modo de ejemplo, se muestra que los países de América del sur utilizan una plataforma de blockchain, mientras que los de Europa utilizan otra. A su vez, una solicitud realizada por parte de un país de América, queda registrada en ambas blockchain al igual que la respuesta a dicha solicitud.

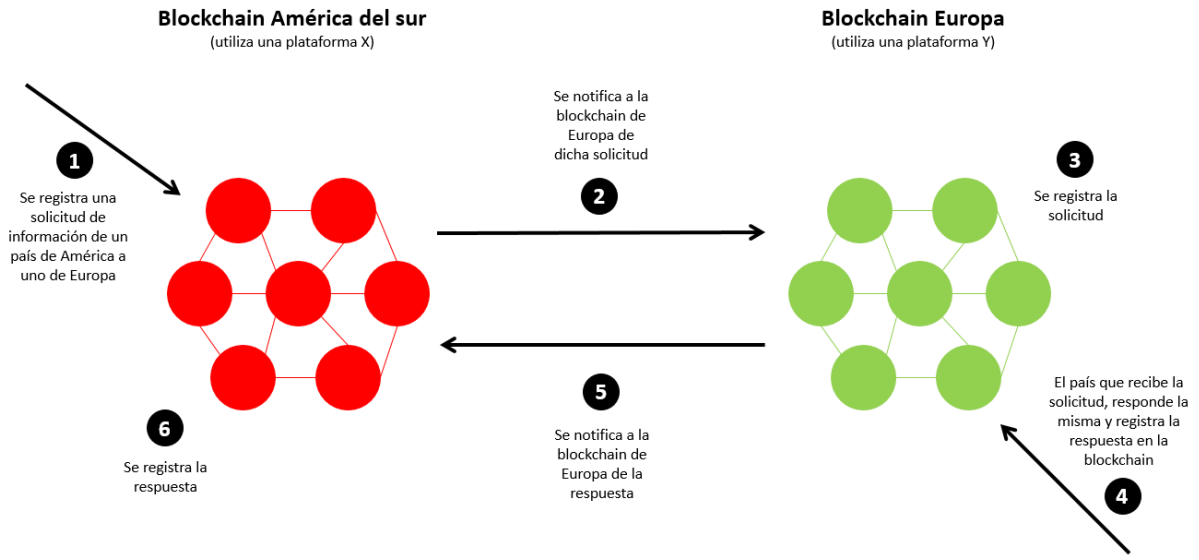

Imagen 18. Escenario de uso.

3.3. Requerimientos del escenario propuesto

Los requerimientos del escenario propuesto consisten en implementar dos sistemas que permitan registrar los intercambios de información mencionados en la sección anterior. Sobre estos sistemas, existen los requerimientos descritos en la tabla 2.

Req.	**Descripción**
R1	Confianza. La información almacenada relativa a los intercambios debe ser confiable y la misma no puede ser manipulada/modificada por ninguna de las partes.
R2	Inmutabilidad. Una vez registrado un intercambio, el mismo no puede ser borrado.

Tabla 2. Requerimientos no funcionales.

Debido a los requerimientos de la tabla 2, se opta por utilizar blockchain ya que sus características permiten lograrlos. A su vez, ambos sistemas deben ser implementados como aplicaciones descentralizadas sobre las plataformas Hyperledger Fabric y Corda. En la tabla 3 se describen cada uno de los requerimientos funcionales que debe soportar cada una de las aplicaciones descentralizadas.

Req.	**Operación**	**Parámetros**
R3	Ingresar un pedido de información	● ID que identifica de forma única al pedido ● País solicitante ● País al cual se realiza el pedido ● Fecha en la cual se realizó el pedido ● Fecha esperada de respuesta ● Hash del paquete enviado al realizar el pedido ● Tipo de pedido
R4	Ingresar la respuesta a un pedido de información	● ID del pedido ● Hash del paquete de respuesta ● Fecha de respuesta
R5	Obtener el listado de pedidos de información realizados. Esto incluye la respuesta (en caso que el pedido ya haya sido respondido)	Ninguno

Tabla 3. Requerimientos funcionales.

Con respecto al requerimiento R3, los pedidos de información pueden ser de cuatro tipo:

- **Personal Data**: consiste en datos personales de una persona (trabajador o jubilado).

- **Life Status**: consiste en información sobre el estado civil y “de vida” de una persona (o sea si la persona está viva o falleció).
- **Períodos trabajados**: consiste en la historia laboral de una persona, o sea donde trabajó y en qué periodos de tiempo.
- **Trabajador separado**: consiste en un pedido de autorización para que una persona vaya a trabajar al otro país por un período limitado pagando aportes de seguridad social en su país de origen.

Además de los requerimientos funcionales sobre cada aplicación, existen requerimientos de interoperabilidad entre ambas aplicaciones. Dichos requerimientos son descritos en la tabla 4.

Req.	**Descripción**
R6	Cuando se ejecuta la operación R3 (solicitud de información) entre países que utilizan distintas aplicaciones, dicha solicitud debe quedar registrada en ambas aplicaciones.
R7	Cuando se ejecuta la operación R4 (respuesta a una solicitud de información) entre países que utilizan distintas aplicaciones, dicha respuesta debe quedar registrada en ambas aplicaciones.
R8	Consistencia entre la información guardada en ambas blockchains. Las solicitudes de información (o respuestas a las mismas) que involucran a países que utilizan aplicaciones distintas, deben quedar guardadas en ambas aplicaciones de forma consistente y sin diferencias entre ambas.

Tabla 4. Requerimientos de interoperabilidad.

3.4. Alcance del proyecto

El alcance del proyecto comprende los siguientes puntos:

- Hallar una solución genérica que permita interoperar plataformas de blockchain.
- Los requerimientos R1 a R7

El requerimiento R8 queda fuera del alcance del mismo.

4. Solución propuesta

En este capítulo se presenta una solución propia a la problemática de la interoperabilidad entre plataformas de blockchain. El contenido del capítulo se organiza de la siguiente forma.

- En la sección 4.1 se presenta el diseño de la solución mencionada.
- En la sección 4.2 se presentan los argumentos del porqué de la solución propuesta.
- En la sección 4.3 se presenta una solución alternativa.

4.1. Diseño

La solución propuesta aplica a los niveles de interoperabilidad técnica y semántica mencionados en la sección 2.3.3 y se encuentra dentro de la categoría de *trusted relays.* La misma se basa en el uso de *gateways*, quienes actúan de nexo en la comunicación entre las distintas blockchains. Además, la solución requiere que las organizaciones que implementan las blockchains confíen en el gateway (lo cual es una característica de las soluciones dentro de la categoría trusted relays), ya que es un agente externo sobre el cual pueden no tener control.

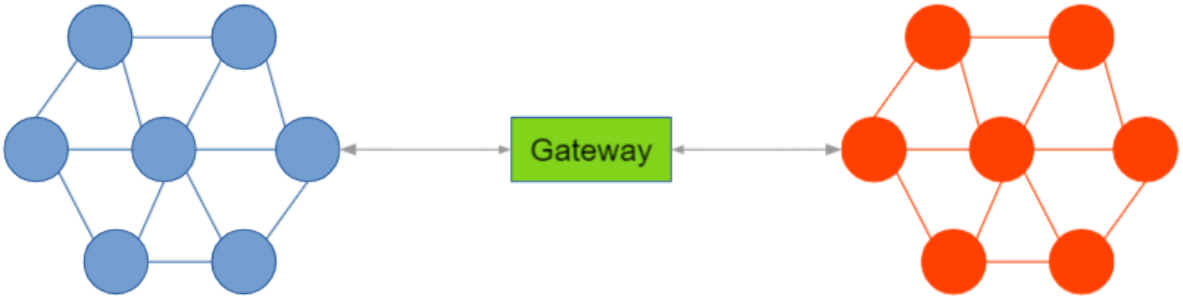

Imagen 19. Rol del gateway en la comunicación

A alto nivel, la responsabilidad del *gateway* es hacer llegar la información de una blockchain a otra y se ubica entre ambas blockchains, como se muestra en la imagen 19. Para esto, una blockchain origen notifica al *gateway*, mediante la emisión de un evento, que tiene información para enviar a una blockchain destino. Aquí el *gateway* recibe el pedido y retransmite la información a la blockchain destino, sin hacer ninguna modificación sobre dicha información.

En un nivel más de detalle y, como se ilustra en la imagen 20, el *gateway* está compuesto internamente por *conectores* y un *router*:

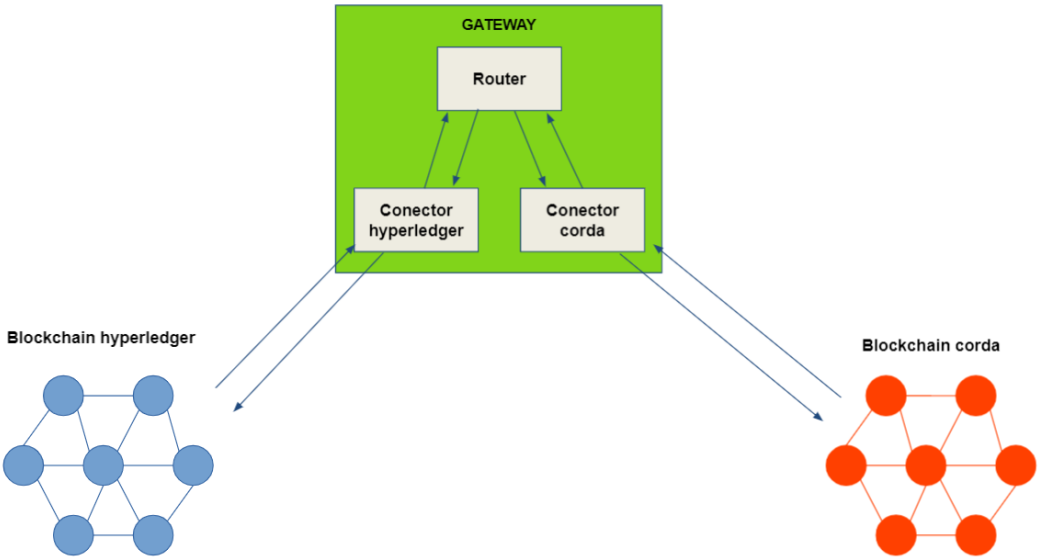

Imagen 20. Composición del gateway

A continuación se describe la función de cada componente.

Conector
El conector tiene la responsabilidad de comunicar la blockchain con el router. Dada la diversidad de plataformas de blockchain, el conector implementa un mecanismo de comunicación específico para la plataforma que se quiera integrar, y abstrae al router de las diferencias que existen entre las distintas plataformas proveyendo una interfaz genérica. Por cada blockchain que se desee interoperar, se debe implementar un conector que permita la comunicación con la misma.

A su vez, el conector es el punto de contacto de una blockchain al comunicarse con otras, ya que, cuando una blockchain interopera con otras, siempre lo hace a través del mismo.

En específico, las tareas que desempeña cada conector son las siguientes:

1. Cuando la blockchain asociada al conector emite un evento destinada a otra blockchain, el conector recibe dicho evento y lo envía al router.
2. Cuando el conector recibe un evento (destinado a la blockchain asociada al conector) desde el router, debe notificar a la blockchain de dicho evento.

Router
La función del router consiste en enrutar los eventos emitidos por las blockchains. Para esto, el router recibe un evento emitido por un conector origen y lo envía al conector destino correspondiente. Para que el router pueda realizar esta tarea, es necesario que, cuando las blockchains emiten los eventos, especifiquen la blockchain y smart contract al cual está destinado el mismo.

Los conectores y el router son los componentes que componen al gateway, y los mismos deben ser implementados como aplicaciones que se comunican mediante servicios REST. Debido a esto, los componentes del gateway pueden ser implementados en distintos

lenguajes de programación. Esto presenta una ventaja teniendo en cuenta que, algunas plataformas de blockchain, proveen SDKs implementadas en ciertos lenguajes, por lo cual, los conectores del gateway deben estar implementados en dichos lenguajes para poder hacer uso de las mismas (lo cual hace que los conectores deban ser implementados en distintos lenguajes).

Protocolo cross-blockchain
A continuación se describe el protocolo de comunicación cross-blockchain utilizado en esta solución. Dicho protocolo consta de 5 pasos:

1. La blockchain origen registra una transacción y emite un evento dirigido a su conector. Este evento incluye la información de la blockchain destino e información adjunta que sea necesaria.
2. El conector de la blockchain origen recibe el evento y lo envía al router.
3. El router obtiene la blockchain destino a partir del evento y lo envía al conector de dicha blockchain.
4. El conector de la blockchain destino envía el evento a la blockchain.
5. La blockchain destino recibe el evento y decide la acción a tomar, por ejemplo, registrar una transacción en su ledger.

4.1.1. Interfaces

Los conectores y el router deben respetar ciertas interfaces para poder aplicar esta solución. A continuación se describen estas interfaces.

Como se mencionó en la sección anterior, el conector debe realizar dos tareas. Para lograr la primera de ellas, debe invocar un servicio REST que será expuesto por el router en un puerto arbitrario (configurable). Dicha invocación será mediante el método POST y el body de la misma contendrá información de tipo JSON[2] con el formato mostrado en la imagen 21.

Imagen 21. Formato información endpoint router.

En la tabla 5, se describe cada uno de los campos presentes en la imagen 21.

[2] JSON: https://www.json.org/

Campo	Descripción
targetBlockchain	Contiene el nombre de la blockchain a la cual está destinado el evento
targetContract	Contiene el nombre del smart contract al cual está destinado el evento
data	Contiene información arbitraria relativa al evento

Tabla 5. Descripción de campos del endpoint.

Para lograr la segunda tarea, el conector debe exponer un servicio REST que será invocado por el router. Dicho servicio debe ser expuesto en un puerto cualquiera (el mismo será configurado en el router) y debe escuchar solicitudes POST. El campo body de dichas solicitudes contendrá información de tipo JSON y tendrá el formato que se muestra en la imagen 22.

Imagen 22. Información endpoint conector.

En la tabla 6, se describe cada uno de los campos presentes en la imagen 22.

Campo	Descripción
targetContract	Contiene el nombre del smart contract al cual está destinado el evento
data	Contiene información arbitraria relativa al evento

Tabla 6. Descripción campos endpoint conector.

Cuando el conector recibe un evento de este tipo por parte del router, debe disparar una operación en la blockchain que notifique a la misma de dicho evento. La forma en que el conector notifica a la blockchain, no queda definida en esta solución, ya que, dicha forma será distinta para cada plataforma y no es generalizable. Lo mismo sucede con la forma en que el conector recibe los eventos de su blockchain.

Los eventos que un conector envía al router deben ser eventos confirmados a nivel de toda la blockchain, es decir, el conector no debe enviar al router un evento que es emitido por un nodo de la red pero que no ha sido consensuado y confirmado por el resto de los nodos.

4.2. Precondiciones

Para poder aplicar la solución propuesta, es necesario que las blockchains a interoperar tengan la capacidad de emitir eventos. Si una blockchain no posee ninguna forma de notificar al *gateway* de lo que sucede en la misma, la solución propuesta no es aplicable. Junto con el requisito anterior, las blockchains a su vez deben proveer a aplicaciones externas alguna forma de disparar operaciones en la misma.

4.3. Guías de diseño

En esta sección se presentan las razones que llevaron al diseño de la solución propuesta.

Primeramente, teniendo en cuenta que las plataformas de blockchain presentan grandes diferencias en las funcionalidades que ofrecen (aún más en su implementación) y dentro de los requerimientos del proyecto estaba la restricción de que las mismas no podian ser modificadas, esto hace que sea necesario tener software intermediario entre las plataformas que haga posible la comunicación.

Luego, observando las soluciones de interoperabilidad relevadas, en varias de ellas (por ejemplo [34], [42], [44]) se encontró la presencia de entidades que se encargan de enviar mensajes o transacciones de una blockchain a otra, lo cual es similar a la función que cumple el gateway en la solución propuesta.

Por otro lado, debido a que la forma de interacción con cada plataforma de blockchain es diferente, parece adecuado tener un componente que se encargue de dicha tarea. En la solución propuesta, dicho componente son los conectores, lo cual es un concepto compartido con Hyperledger Cactus [33].

5. Implementación

En este capítulo se describe una implementación de la solución propuesta en el capítulo 4, utilizando las plataformas Corda y Fabric. El contenido del capítulo se organiza de la siguiente manera:

- En la sección 5.1 se describe la implementación del gateway.
- En la sección 5.2 se describe la construcción de las blockchains.
- En la sección 5.3 se describen las aplicaciones web que actúan como interfaces gráficas de las blockchains.
- En la sección 5.4 se describen las decisiones de implementación que se tomaron.

5.1. Instanciación de la solución

A lo largo de esta sección se menciona la *implementación* como una instancia de la solución propuesta en el capítulo 4. Esta implementación se realizó como prueba de concepto en el marco del escenario de uso planteado en el capítulo 3.

Un diagrama de arquitectura más detallado de la implementación se puede ver en la imagen 23.

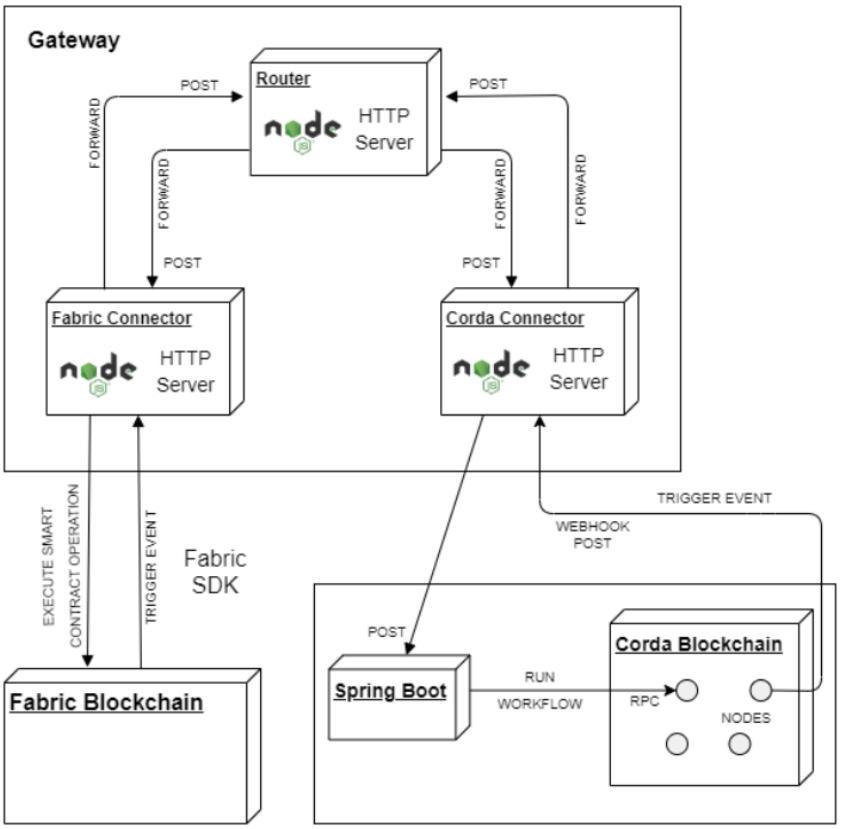

Imagen 23. Diagrama de arquitectura de la implementación.

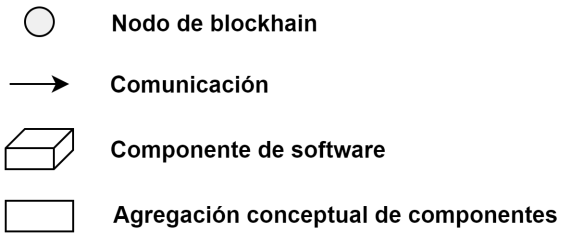

Imagen 24. Diagrama de arquitectura de la implementación

En el diagrama de la imagen 23 se pueden identificar los tres grandes componentes del diseño de la solución: el gateway y dos blockchains. En la imagen 24 se describe el significado de cada uno de los símbolos presentes en la imagen 23.

5.1.1. Gateway

El gateway es implementado como tres servidores Node.js: conector de Corda, conector de Fabric y Router. Los detalles de implementación de ambos conectores serán abordados en la siguiente sección, mientras que el funcionamiento del router se describe a continuación.

Router

El servidor determina a dónde reenviar los eventos mediante un archivo de configuración donde se especifican los distintos conectores junto con sus direcciones, de forma similar a una tabla de ruteo. En la tabla 7 se presenta un ejemplo de dicha configuración:

```
{
  "blockchains": {
    "fabric": {
      "connectorHost": "localhost",
      "connectorPort": 3001
      },
      "corda": {
        "connectorHost": "localhost",
        "connectorPort": 3002
      }
  }
}
```

Tabla 7. Archivo de configuración.

5.1.2. Comunicación entre blockchain y gateway

La comunicación entre el gateway y cada blockchain depende estrictamente de las plataformas utilizadas. Parte del propósito de los conectores es soportar el mecanismo de comunicación específico para cada plataforma. A continuación se describe la comunicación entre los conectores y las blockchains.

5.1.2.1. Corda

Para describir la comunicación entre Corda y su conector, diferenciamos dos casos:

- Comunicación desde el gateway hacia la blockchain.
- Comunicación desde la blockchain hacia el gateway.

A continuación se describen ambos casos.

Comunicación desde el gateway hacia la blockchain

Si bien Corda no provee un SDK que resuelva la comunicación con sistemas externos de forma transparente, provee un servidor Spring Boot (representado en la imagen 25) integrado con Corda, además de documentación y ejemplos. El objetivo de este servidor es exponer *endpoints* que permitan a sistemas externos comunicarse con la blockchain y, en este caso, se utiliza para comunicar el conector con la blockchain. El código fuente de este servidor se obtiene junto al código fuente de un proyecto base de Corda, el cual está disponible para su descarga en su sitio web.

El código fuente brindado por Corda contiene controladores de ejemplo que ilustran cómo invocar determinadas acciones en la blockchain mediante peticiones HTTP a este servidor. Dichos controladores fueron tomados como referencia para implementar nuevos controladores que cubrieran los requerimientos del escenario.

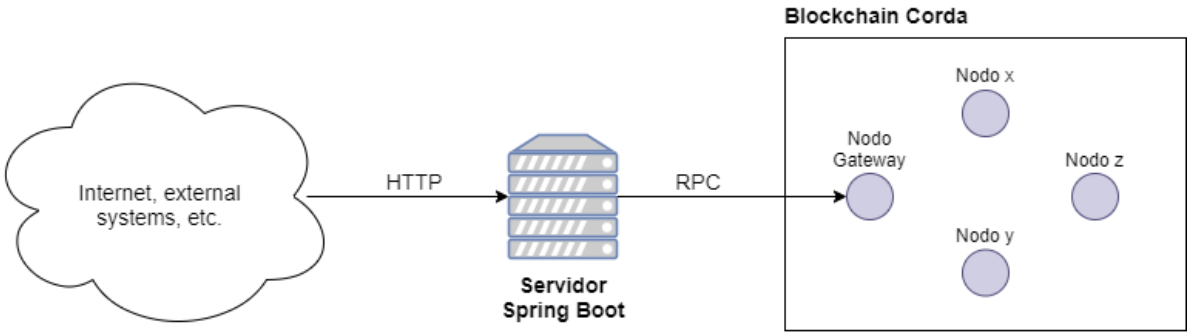

Imagen 25. Servidor que habilita la comunicación con la blockchain desde el exterior.

En dicho servidor Spring Boot se implementan controladores que procesan peticiones HTTP POST y se comunican con los nodos de la blockchain mediante *Remote Procedure Calls* (RPC) para iniciar los flujos.

A su vez, en Corda se creó un nodo especial, el cual denominamos *gateway.* El cometido de este nodo es realizar acciones en la blockchain en nombre del servidor Spring Boot, o dicho de otra forma, acciones solicitadas por aplicaciones externas. Este nodo es utilizado ya que

el servidor Spring Boot debe comunicarse con un nodo de la red para poder realizar acciones en la blockchain. El nodo gateway es creado especialmente para lograr la interoperabilidad, no siendo éste provisto por Corda. Además, no debe confundirse con el gateway de la sección 5.1.1.

En las imágenes 26 y 27 se muestra parte del código fuente de un controlador del servidor Spring Boot, y a continuación un pseudo-código con sus principales acciones:

- Recibir petición HTTP POST
- Extraer de la petición los datos mencionados en la sección 4.1.
- Conectarse con el nodo gateway mediante RPC
- Iniciar el flujo que corresponda según los datos de la petición

```
public Controller(NodeRPCConnection rpc) {
    this.proxy = rpc.proxy;
    String gatewayPartyString = "O=Gateway,L=Berlin,C=DE";
    CordaX500Name gatewayPartyName = CordaX500Name.parse(gatewayPartyString);
    this.gatewayParty = proxy.wellKnownPartyFromX500Name(gatewayPartyName);
}
```

Imagen 26. Conexión con el nodo gateway por RPC.

```
@PostMapping(value = "/transactions")
private ResponseEntity<String> recordExchange(@RequestBody Event event) throws IllegalArgumentException {
    String targetContract = event.getTargetContract();
    EventData data = event.getData();

    // Create a new state using the parameters given
    try {
        Class<? extends FlowLogic<SignedTransaction>> flowClass = Utils.getMappedFlow(targetContract);

        // Start the flow. It blocks and waits for the flow to return.
        SignedTransaction result =
            proxy.startTrackedFlowDynamic(
                flowClass,
                data,
                gatewayParty
            ).getReturnValue().get();

        return ResponseEntity
                .status(HttpStatus.CREATED)
                .body("Transaction id " + result.getId() + " committed to ledger.");
    } catch (InterruptedException | ExecutionException e) {
        return ResponseEntity
                .status(HttpStatus.BAD_REQUEST)
                .body(e.getMessage());
    }
}
```

Imagen 27. Controlador para el registro de transacciones en la blockchain.

Comunicación desde la blockchain hacia el gateway

En este caso la comunicación también es a través de peticiones HTTP POST. El conector expone un *webhook*, el cual es invocado por el nodo gateway cuando se registra una transacción *cross-chain* desde Corda.

La petición POST que se envía al *webhook* debe realizarse luego de que la transacción es confirmada en el ledger de Corda. Por esta razón es que la petición se realiza en el último paso del flujo *SSFlow*, el cual se describe en la sección 5.2.3 y fue implementado para llevar a cabo el registro de las transacciones que se originan en Corda. En las imágenes 28 y 29 se muestran una parte del código fuente del *SSFlow* y de la petición POST al webhook, respectivamente.

```
// Signing the transaction.
SignedTransaction signedTx = serviceHub.signInitialTransaction(txBuilder);

try {
    // Finalise the transaction and send it to the counterparties.
    getLogger().info("Forwarding transaction to peers");
    subFlow(new FinalityFlow(signedTx, sessions));
    getLogger().info("Transaction signed by all peers");
} catch (FlowException e) {
    getLogger().error(e.getMessage());
}

getLogger().info("Triggering cross-chain event");
RemoteSSTxService service = serviceHub.cordaService(RemoteSSTxService.class);
service.notifyRemoteTx(outputState, signedTx);

return signedTx;
```

Imagen 28. Código de SSFlow notificando la transacción cross-chain luego de confirmada.

```java
public void notifyRemoteTx(SSState state, SignedTransaction tx) {
    try {
        URL url = new URL(CONNECTOR_WEBHOOK);
        URLConnection con = url.openConnection();
        HttpURLConnection http = (HttpURLConnection) con;
        http.setRequestMethod("POST");
        http.setDoOutput(true);

        byte[] body = buildBody(state, tx);

        http.setRequestProperty("Content-Type", "application/json; charset=UTF-8");
        http.connect();

        try (OutputStream os = http.getOutputStream()) {
            os.write(body);
        }
    } catch (IOException ignored) {}
}
```

Imagen 29. Petición HTTP POST al webhook del conector.

5.1.2.2. Fabric

La comunicación entre el conector de Fabric y la blockchain se realiza utilizando la SDK provista por Hyperledger Fabric, tanto para detectar los eventos de los smart contracts como para notificar a la blockchain cuando llega un evento destinado a la blockchain de Fabric.

Las tareas que hace el conector al utilizar la SDK son las siguientes:

- Realiza la conexión a la blockchain. Para esto, necesita proveer la ruta del archivo de conexión que contiene parte de la topología de los nodos de la blockchain y el certificado del usuario con el cual se accedera a realizar operaciones en la blockchain.
- Se obtiene el smart contract que se desea.
- Se invoca una operación del smart contract obtenido o se pone a escuchar eventos del mismo.

A continuación se muestran las funciones (a nivel de código) que realizan cada una de estas tareas.

La función “loadFabricGateway” que se muestra en la imagen 30 es la encargada de realizar la conexión con Fabric. Esta función crea una variable miembro “fabricGateway” que almacena un objeto creado con la SDK de Fabric y permite la comunicación con la blockchain.

```
async loadFabricGateway() {
    const ccpPath = config.blockchains.fabric.connectionJSONPath;
    const user = config.blockchains.fabric.fabricUser;
    const walletPath = config.blockchains.fabric.walletPath;
    const wallet = new FileSystemWallet(walletPath);
    const userExists = await wallet.exists(user);

    if (!userExists) {
        throw new Error(`An identity for the user "${user}" does not exist in the wallet`);
    }

    this.fabricGateway = new Gateway();
    await this.fabricGateway.connect(ccpPath, { wallet, identity: user, discovery: { enabled: true, asLocalhost: true }});
}
```

Imagen 30. Conexión que realiza la conexión Fabric.

La función “sendEventToFabricBlockchain” notifica a la blockchain cuando se recibe un evento desde el router y la misma se muestra en la imagen 31. Esta función utiliza la variable miembro “fabricGateway” mencionada previamente.

```
async sendEventToFabricBlockchain(event) {
// notifies fabric blockchain about the received event
    const network = await this.fabricGateway.getNetwork(config.blockchains.fabric.fabricChannel)
    const contract = network.getContract(event.targetContract);
    const receiveEvtOper = config.blockchains.fabric.fabricReceiveEventOperation;

    await contract.submitTransaction(receiveEvtOper, JSON.stringify(event.data));
}
```

Imagen 31. Función que envía un evento a Fabric.

Por último, la función “listenBlockchainEvents” que se muestra en la imagen 32 detecta los eventos disparados desde Fabric. Esta función también utiliza la variable miembro “fabricGateway”.

```
async listenBlockchainEvents() {
// listen events on fabric blockchain
    const network = await this.fabricGateway.getNetwork(config.blockchains.fabric.fabricChannel);
    const contract = network.getContract(config.blockchains.fabric.fabricContract);
    const remoteEvent = config.blockchains.fabric.fabricRemoteRequestEvent;

    await contract.addContractListener('listener', remoteEvent, this.eventReceivedFromFabric.bind(this));
}
```

Imagen 32. Función que detecta eventos disparados por Fabric.

5.2. Implementación del escenario propuesto

En esta sección se describe cómo se implementó el escenario de seguridad social en cada una de las blockchains y como se aplicó la solución descrita en el capítulo 4 a dicho escenario.

5.2.1. Aplicación de la solución al escenario

Con el fin de mostrar la solución propuesta aplicada al escenario propuesto de seguridad social, en la imagen 33 se muestran la serie de pasos que ocurren para resolver los requerimientos R3 y R6 mencionados en las tablas 3 y 4 (capítulo 3) respectivamente.

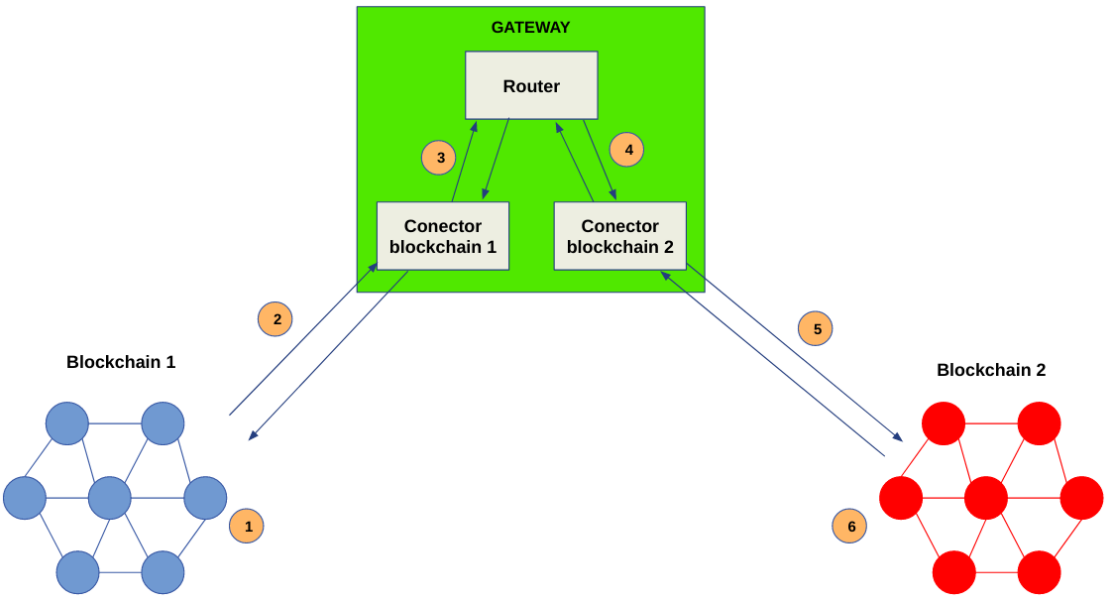

Imagen 33. Flujo de comunicación entre blockchains

Pedido de información

1) Una aplicación externa dispara el registro de un pedido de información en la blockchain 1.
2) La blockchain 1 registra dicho pedido y emite un evento dirigido a su conector. Este evento incluye la información de la blockchain destino (blockchain 2) y la información del pedido realizado.
3) El conector de la blockchain 1 recibe el evento y lo envía al router.
4) El router obtiene la blockchain destino (blockchain 2) a partir del evento y lo envía al conector de la blockchain 2.
5) El conector de la blockchain 2 envía el evento a la blockchain 2.
6) La blockchain 2 recibe el evento y registra el pedido de información en su ledger.

Respuesta a pedido de información

La respuesta a una solicitud de información ocurre de forma análoga al pedido, pero comenzando en la blockchain 2.

5.2.2. Hyperledger Fabric

En esta plataforma se implementó un smart contract que mantiene el listado de solicitudes de información realizados (junto con sus respuestas) y contiene las siguientes operaciones:

1. Registro de una solicitud de información
2. Registro de la respuesta de una solicitud
3. Recibir un evento de otra blockchain
4. Obtener el listado de solicitudes realizadas (junto con sus respuestas)

Las dos primeras operaciones son las que emiten los eventos destinados a la blockchain de corda y, la tercera operación, es la que es invocada por el conector cuando existe un evento

proveniente de Corda. La cuarta operación es utilizada por la aplicación web (que se describe más adelante en este capítulo) para mostrar las solicitudes de información realizadas.

En las imágenes 34 y 35, se muestra como se emiten eventos desde Fabric y como se reciben eventos desde el conector respectivamente.

```
7   async sendRemoteEvent(ctx, remoteEvent) {
8   // this operation is called by contract operations when they want to notify another blockchain about an event
9     await ctx.stub.setEvent('remoteEvent', Buffer.from(JSON.stringify(remoteEvent)));
10  }
```

Imagen 34. Código que emite eventos.

```
158   async remoteEventReceived(ctx, event) {
159   // this operation is called by gateway when there's an event destined to this smart contract
160     const evnt = JSON.parse(event);
161
162     if (evnt.messageType === 'request') {
163       await this.saveInformationRequest(ctx,
164                                         evnt.requestID,
165                                         evnt.senderInstitution,
166                                         evnt.receiverInstitution,
167                                         evnt.requestType,
168                                         evnt.expectedReplyDate,
169                                         evnt.packageHash,
170                                         evnt.requestDate);
171     } else if (evnt.messageType === 'response') {
172       await this.saveInformationRequestResponse(ctx,
173                                                 evnt.requestID,
174                                                 evnt.packageHash,
175                                                 evnt.responseDate);
176     } else {
177       throw new Error('Message type invalido');
178     }
179   }
```

Imagen 35. Código que recibe eventos desde el conector.

La función *remoteEventReceived* es la que es invocada por el conector para notificar a Fabric acerca de un evento y, en el parámetro *event* el conector coloca el valor del campo *data* mencionado en la sección 4.1.1.

5.2.3. Corda

Corda no cuenta con el concepto de smart contract pero cuenta con otros conceptos que, combinados, son equivalentes a un smart contract. Uno de estos conceptos es el de *flujo*, el cual consta de una secuencia de pasos que llevan a una actualización del *ledger*. Este concepto de flujo fue utilizado para implementar el escenario planteado y orquesta las acciones que se llevan a cabo en la blockchain; o bien como resultado de una transacción cross-chain recibida, o bien que generan una nueva transacción cross-chain.

Para lograr los objetivos del escenario de seguridad social se implementaron dos flujos custom: *SSFlow* y *RemoteSSFlow*. El primero es utilizado para registrar una transacción en la blockchain y notificar el evento al gateway, y el segundo es utilizado para recibir un evento del gateway y registrar una transacción en la blockchain a partir de su información.

El flujo *SSFlow* puede ser iniciado por cualquier nodo participante de la blockchain, mientras que el *RemoteSSFlow* es iniciado únicamente por el nodo especial gateway.

Los flujos en Corda son clases Java con un método *call* donde se incluye su lógica. Ambos flujos fueron implementados de cero para la implementación del escenario utilizando la API Java de Corda. [45]

SSFlow
Este flujo se corresponde con el paso número 2 de la comunicación cross-chain mencionado en la sección 5.2.1. El flujo es iniciado por un nodo de la blockchain cuando se quiere iniciar una transacción cross-chain desde Corda y tiene como particularidad que al finalizar el flujo, realiza una request HTTP POST al webhook expuesto por el conector de Corda, de forma de notificar el evento. Parte del código se muestra en la imagen 36 y un diagrama de secuencia se ilustra en la imagen 37.

Si bien no existen restricciones y cualquier nodo podría iniciar este flujo, en la implementación siempre es iniciado por un nodo específico ya que las acciones son realizadas desde una aplicación web, la cual se comunica con dicho nodo.

RemoteSSFlow
Este flujo se corresponde con el paso número 6 de la comunicación cross-chain mencionado en la sección 5.2.1. La finalidad de este flujo es registrar en Corda las transacciones iniciadas en Fabric y es iniciado por el nodo gateway de la blockchain, el cual se encarga de recibir los eventos provenientes del conector y transformarlas en transacciones en la blockchain. El código y diagrama de secuencia de las imágenes 36 y 37, a menos de ligeras modificaciones, reflejan los detalles de este flujo. Por ejemplo, este flujo no notifica al conector luego de registrar una transacción.

```java
/**
 * The flow logic is encapsulated within the call() method.
 */
@Suspendable
@Override
public SignedTransaction call() throws FlowException {
  // Flow started
  Utils utils = new Utils();
  ServiceHub serviceHub = getServiceHub();
  Party self = getOurIdentity();

  // Get nodes that will participate
  List<AbstractParty> allParties = utils.getAllParties();

  // Get the notary identity from the network map
  Party notary = serviceHub.getNetworkMapCache().getNotaryIdentities().get(0);

  // Get the transaction input state
  StateAndRef<SSState> refRequestState =
    utils.populateResponse(messageType, requestID);

  // Create the transaction output state
  SSState outputState = new SSState(
    senderInstitution,
    receiverInstitution,
    messageType,
    ...
  );

  // Build the transaction
  TransactionBuilder txBuilder = new TransactionBuilder(notary);
  if (refRequestState != null) {
    txBuilder.addInputState(refRequestState);
  }
  txBuilder.addOutputState(outputState, TemplateContract.ID);

  // Add participants to the transaction
  List<FlowSession> sessions = new ArrayList<>();
  for (AbstractParty pty: allParties) {
    FlowSession session = initiateFlow(pty);
    sessions.add(session);
  }

  // Signing the transaction
  SignedTransaction signedTx = serviceHub.signInitialTransaction(txBuilder);

  // Forwarding transaction to peers
  subFlow(new FinalityFlow(signedTx, sessions));
  // Transaction signed by all peers

  // Triggering cross-chain event
  RemoteSSTxService service =
    serviceHub.cordaService(RemoteSSTxService.class);
  service.notifyRemoteTx(outputState, signedTx);

  return signedTx;
}
```

Imagen 36. Código del flujo SSFlow

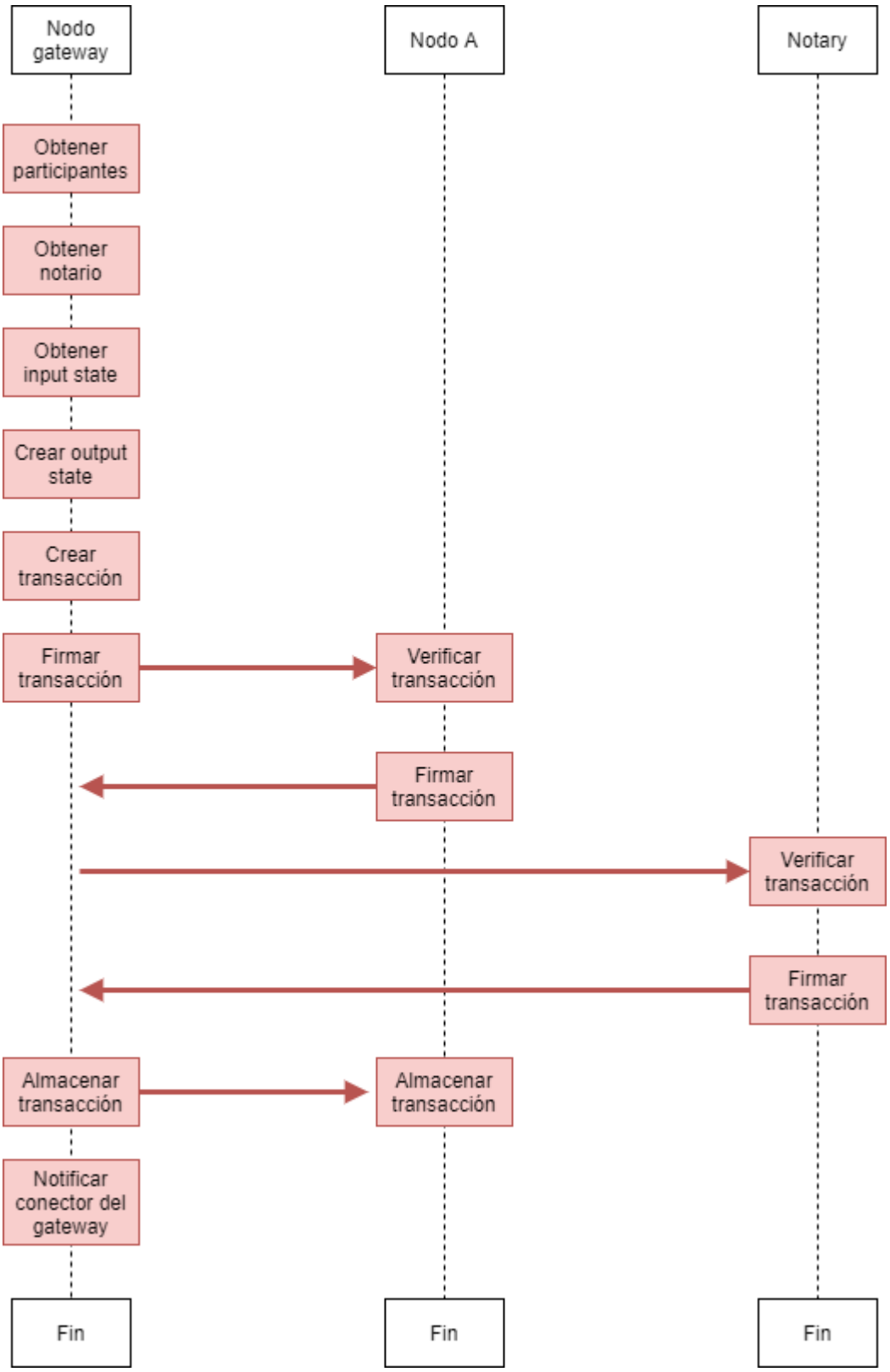

Imagen 37. Diagrama de secuencia de ambos flujos.

5.2.4. Formato de los mensajes intercambiados

La información presente en los eventos emitidos desde Fabric y Corda pueden tener dos posibles formatos dependiendo de si la operación realizada en la blockchain es una solicitud de información o una respuesta a una solicitud. En caso de tratarse de una solicitud de información, el formato es el que se muestra en la tabla 8:

```
{
  targetBlockchain: 'corda',
  targetContract: 'SocialSecurityExchange',
  data: {
    requestID: 'nj23r23js',
    senderInstitution: 'ESP',
    receiverInstitution: 'URU',
    requestType: 'WorkedPeriods',
    requestDate: 'Sun Jun 27 2021 14:51:52 GMT+0000 (UTC)',
    expectedReplyDate: '15/07/2021',
    status: 'pending',
    messageType: 'request',
    packageHash: '323zczAI',
    sourceBlockchain: 'fabric',
    sourceContract: 'socialSecurityExchange'
  }
}
```

Tabla 8. Formato de mensaje.

En la tabla 9 se describen cada uno de los campos:

Campo/s	**Descripción**
requestID	Corresponde al ID de la solicitud realizada
senderInstitution	Es el país que realiza la solicitud
receiverInstitution	Es el país a quien se le hace la solicitud
requestType	Indica el tipo de solicitud de información
requestDate	Es la fecha en que fue realizada la solicitud
expectedReplyDate	Es la fecha esperada de respuesta
messageType	Indica si se trata de una solicitud o una respuesta a una solicitud de información
packageHash	Es el hash del paquete de información enviado al realizar la solicitud
sourceBlockchain y	Indican la blockchain y smart contract de origen

sourceContract	

Tabla 9. Descripción campos.

En caso de tratarse de una respuesta a una solicitud, el formato es el representado en la tabla 10:

```
{
  targetBlockchain: 'corda',
  targetContract: 'SocialSecurityExchange',
  data: {
    messageType: 'response',
    requestID: 'nj23r23js',
    packageHash: 'sd923CSACA',
    responseDate: 'Sun Jun 27 2021 15:13:02 GMT+0000 (UTC)',
    sourceBlockchain: 'fabric',
    sourceContract: 'socialSecurityExchange'
  }
}
```

Tabla 10. Formato mensaje.

En la tabla 11 se describen cada uno de los campos.

Campo/s	**Descripción**
requestID	Es el ID de la solicitud que se está respondiendo
packageHash	Es el hash del paquete enviado al enviar la respuesta
responseDate	Es la fecha de la respuesta
messageType	Indica si se trata de una solicitud o una respuesta a una solicitud de información

Tabla 11. Descripción campos.

La información de un evento es recibida por el conector de la blockchain y es enviada al router sin modificarla. Esto es así porque, tanto la blockchain de Fabric cómo Corda, emiten la información de los eventos en el formato JSON definido en el capítulo 4, por lo cual no es necesario que los conectores realicen ninguna transformación.

5.3. Aplicación web

A modo de facilitar y hacer más amigable la interacción con las plataformas de blockchain, se implementó una aplicación web que permite realizar las siguientes operaciones sobre las mismas:

- Registrar una solicitud de información
- Registrar la respuesta de una solicitud

- Listar todas las solicitudes realizadas junto con sus respuestas

El frontend de esta aplicación web fue construido con Angular[3] y se generaron dos instancias de la misma, una para cada blockchain. Cada una de estas aplicaciones web se comunica con un servidor intermedio que hace posible la comunicación con la blockchain. Este servidor expone tres endpoints, que se corresponden con las tres operaciones que se pueden realizar desde la aplicación web. Dichos servidores son los que permiten la comunicación con las blockchains y efectivamente realizar las acciones en ellas. Ambos servidores exponen las mismas interfaces, pero son implementados de distinta forma.

En el caso de Fabric se implementó un servidor en Node.js, el cual utiliza el SDK de Fabric para la comunicación con la blockchain, de forma muy similar a como lo hace el conector. En la imagen 38 se muestra a modo de ejemplo como la aplicación Angular invoca el endpoint del servidor intermedio que registra una nueva solicitud de información y, en la imagen 39, se muestra el código del lado del servidor intermedio que atiende dicha invocación.

```
23      const data = {
24        requestID: this.requestID,
25        senderInstitution: this.senderInstitution,
26        receiverInstitution: this.receiverInstitution,
27        type: this.type,
28        date: this.date,
29        hash: this.hash
30      };
31
32      this.confirmando = true;
33
34      axios.post('/api/new-request', data)
35        .then(resp => {
36          this.confirmando = false;
37
38          if (resp.data.success) {
39            alert('El pedido se registro correctamente');
40          } else {
41            alert('Ocurrio un error al intentar registrar el pedido');
42          }
43        })
44        .catch(() => {
45          this.confirmando = false;
46          alert('Ocurrio un error al conectar con el servidor');
47        });
```

Imagen 38. Invocación de endpoint del servidor intermedio desde la aplicación Angular.

[3] Angular: https://angular.io/

```
server.post('/new-request', async function (req, res) {
    try {
        await contract.submitTransaction(   'internalInformationRequest',
                                            req.body.requestID,
                                            req.body.senderInstitution,
                                            req.body.receiverInstitution,
                                            req.body.type,
                                            req.body.date,
                                            req.body.hash);

        res.send({
            success: true
        });
    } catch (error) {
        res.send({
            success: false
        });
    }
});
```

Imagen 39. Código del servidor intermedio que atiende el endpoint de registro de solicitud.

En el caso de Corda, se utiliza el mismo servidor Spring Boot que ya utiliza la plataforma para la comunicación con el conector. Para esto se crearon tres nuevos controladores que implementan los tres endpoints mencionados previamente. La aplicación Angular invoca estos endpoints de la misma forma que con Fabric (imagen 38), y en la imagen 40 se muestra el código del controlador que atiende el endpoint de registro de solicitud.

```
@PostMapping(value = "/requests")
private ResponseEntity<String> recordRequest(@RequestBody EventData data) {
    data.setMessageType("request");
    return triggerTransaction(data);
}

private ResponseEntity<String> triggerTransaction(EventData data) {
    // Create a new state using the parameters given
    try {
        // Start the flow. It blocks and waits for the flow to return.
        SignedTransaction result =
                proxy.startTrackedFlowDynamic(
                        CreateRemoteSSTxFlow.class,
                        data
                ).getReturnValue().get();

        return ResponseEntity
                .status(HttpStatus.CREATED)
                .body("Transaction id " + result.getId() + " committed to ledger.");
    } catch (Exception e) {
        return ResponseEntity
                .status(HttpStatus.BAD_REQUEST)
                .body(getStackTrace(e.getCause()));
    }
}
```

Imagen 40.

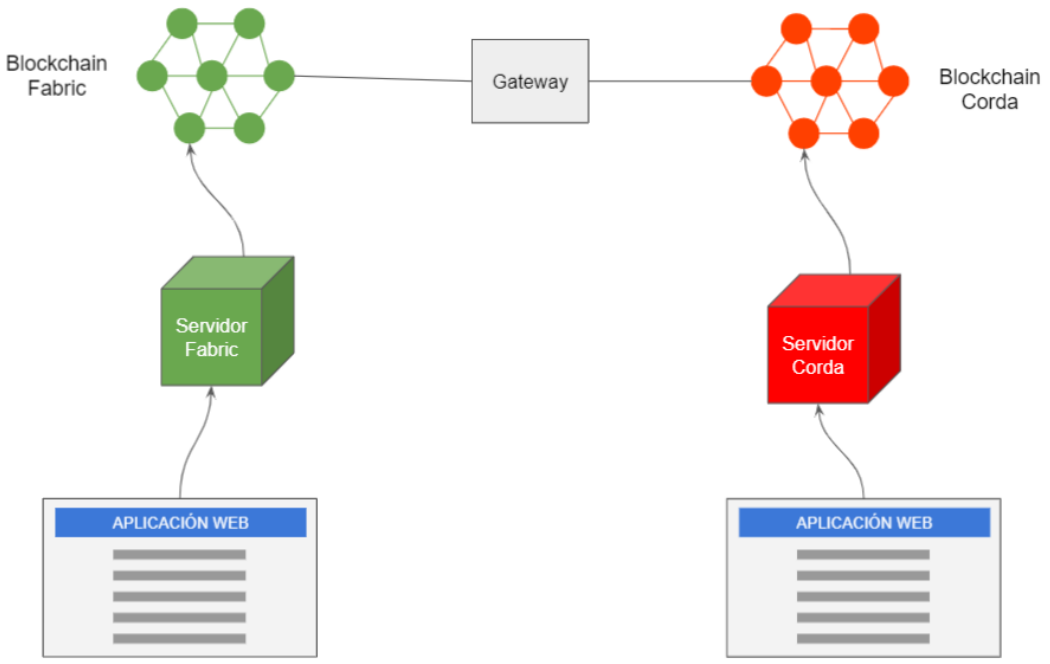

Imagen 41. Diagrama incluyendo aplicaciones web

En la imagen 41, además de las aplicaciones web y los servidores mencionados, también está representado el gateway a modo de tener una foto completa de todo lo realizado.

De esta forma, cuando desde una aplicación web se realiza una solicitud de información o se registra una respuesta que involucra a países de distintas blockchains, dicha solicitud/respuesta también queda visible en la otra aplicación web. Las imágenes 42 y 43 muestran cómo interactúa el usuario con la aplicación web.

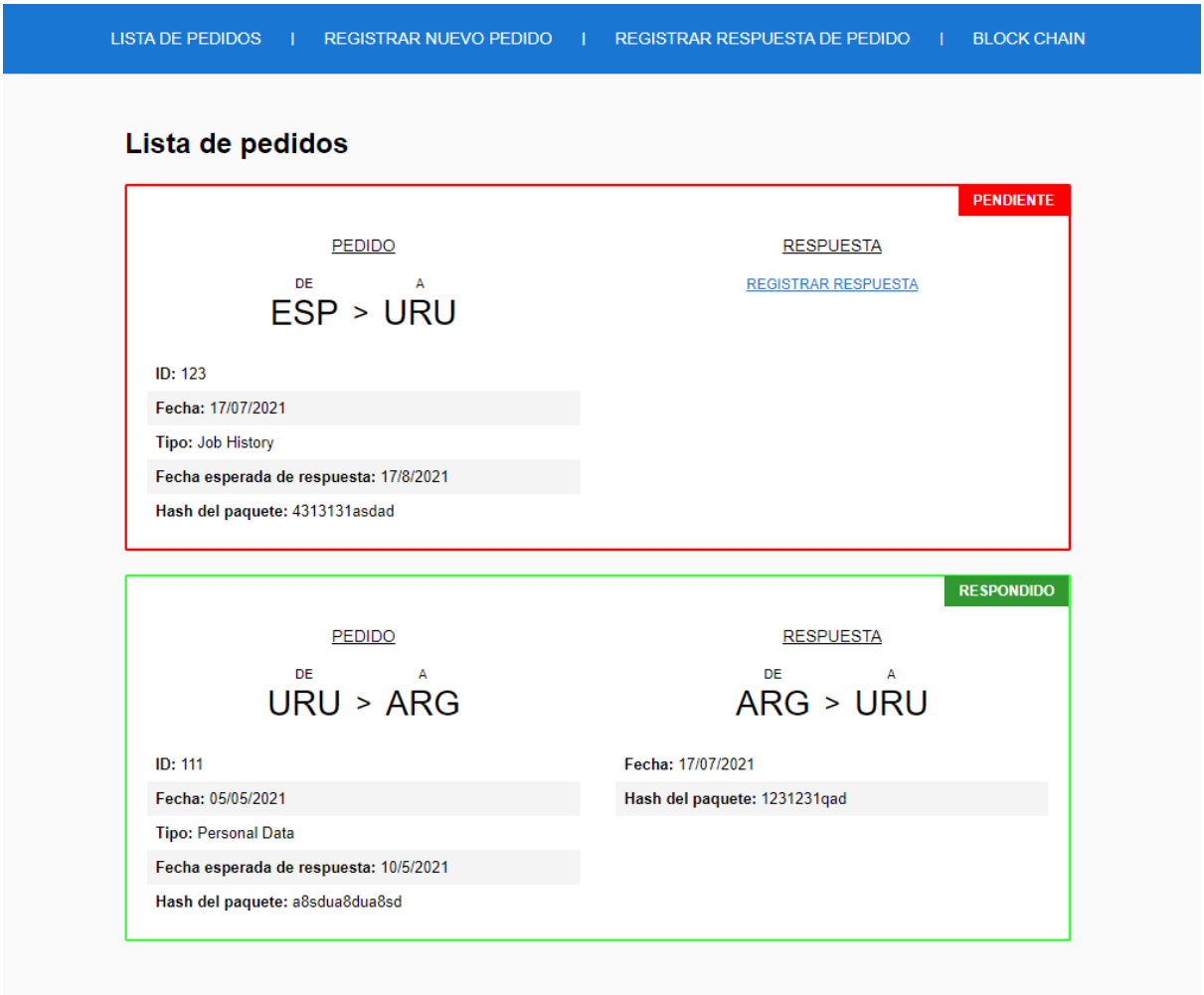

Imagen 42. Vista de pedidos de información

Imagen 43. Vista de formulario de pedido de información

5.3.1. Ejemplo de uso

A continuación se muestran las etapas por las que pasa una transacción cross-blockchain creada desde la aplicación web de la blockchain implementada con Corda. La transacción representa un pedido de información personal de una organización de seguridad social a otra. En caso de que la solicitud sea entre organizaciones de distintas blockchains, se quiere que la transacción quede registrada en ambas blockchains.

En primer lugar, se registra el pedido mediante la aplicación web de la blockchain origen. Para esto, se debe completar el formulario de registro de pedido, como se muestra en la imagen 45.

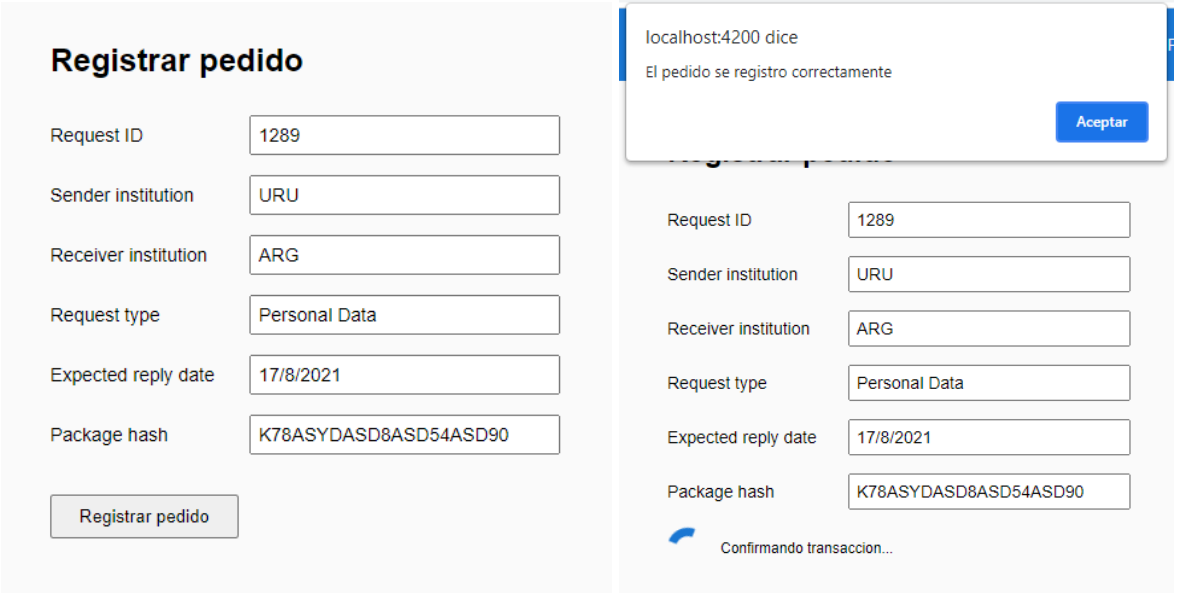

Imagen 45. Registro de pedido de información

Una vez la transacción se registró correctamente desde la aplicación web, podemos comprobar que la transacción efectivamente quedó registrada en el *ledger* de Corda. Para esto, hacemos una consulta mediante la interfaz de línea de comandos (*CLI*) del nodo *gateway*. En este caso, consultaremos por la lista de estados de tipo *SSState* registrados en el *ledger* de la blockchain. El concepto de estados en Corda se cubre en detalle en la sección 2.2.2.3. Los estados de tipo *SSState* fueron creados específicamente para esta implementación y representan los datos de los intercambios de información entre organizaciones que figuran en la tabla 9 de la sección 5.2.4. Para verificar que la transacción fue registrada exitosamente, se verifica que el último estado de tipo *SSState* en el *ledger* coincida con el pedido de información que se acaba de registrar.

La consulta se realiza ejecutando el siguiente comando en la *CLI* del nodo *gateway*:

run vaultQuery contractStateType: com.template.states.SSState

Imagen 46. Consulta en el ledger de Corda

En la imagen 46 se puede ver un estado registrado en la blockchain, que coincide con el intercambio de información iniciado desde la aplicación web. Dicho estado está asociado a una transacción cuyo *hash* se puede ver en la imagen.

Además de confirmar que la transacción quedó confirmada en la blockchain origen (Corda), veremos que se disparó un evento notificando al conector de Corda de una nueva transacción cross-blockchain.

Imagen 47. Log del conector de Corda

En la imagen 47 se puede ver la información recibida por el conector de Corda desde la blockchain, la cual coincide con la interfaz descrita en la sección 6.1. Esta información es reenviada al router para que continúe su trayectoria hacia la blockchain destino.

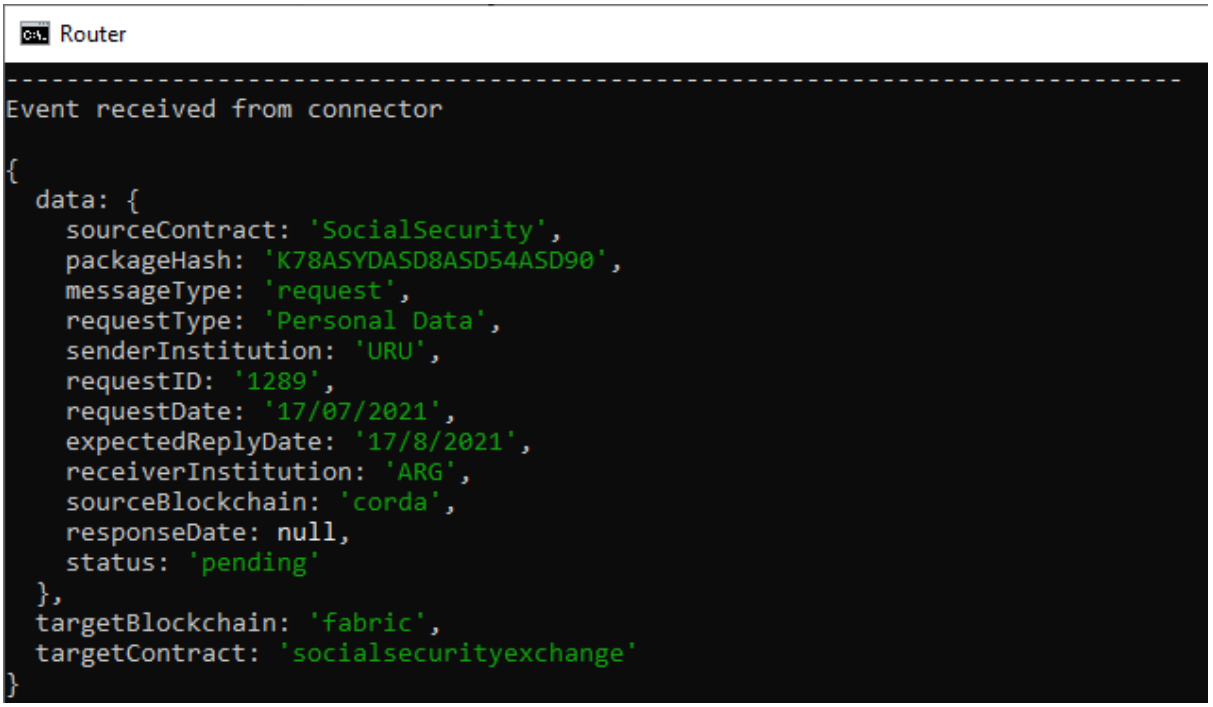

Imagen 48. Log del router

En la imagen 48 se puede ver la información recibida por el router desde el conector de Corda, la cual es exactamente la misma que este último recibió desde la blockchain.

Luego, el router redirecciona ese mensaje/evento al conector de Fabric y en la consola del conector (imagen 49) se puede ver que el evento llegó correctamente.

```
--------------------------------------------------------------------------------
Event received from router

{
  data: {
    sourceContract: 'SocialSecurity',
    packageHash: 'K78ASYDASD8ASDS4SD90',
    messageType: 'request',
    requestType: 'Personal Data',
    senderInstitution: 'URU',
    requestID: '1289',
    requestDate: '17/07/2021',
    expectedReplyDate: '17/08/2021',
    receiverInstitution: 'ARG',
    sourceBlockchain: 'corda',
    status: 'pending'
  },
  targetBlockchain: 'fabric',
  targetContract: 'socialsecurityexchange'
}
```

Imagen 49. Log del conector de Fabric

Una vez que esto ha ocurrido, la transacción realizada en la blockchain de Corda debe estar también registrada en Fabric. Esto lo vamos a comprobar creando un pequeño script en Javascript que utiliza la SDK de Fabric, obtiene el smart contract de seguridad social y ejecuta la operación que devuelve todas las solicitudes de información realizadas.

La invocación y el resultado de dicho script (query.js) es el siguiente:

```
juan@juan:~/go/src/github.com/hyperledger/fabric-samples/socialSecurity$ node query.js
Wallet path: /home/juan/go/src/github.com/hyperledger/fabric-samples/socialSecurity/wallet
Transaction has been evaluated, result is: [{"expectedReplyDate":"17/08/2021","receiverInstitution":"ARG","
requestDate":"17/07/2021","requestID":"1289","requestPackageHash":"K78ASYDASD8ASDS4SD90","requestType":"Per
sonal Data","responseDate":null,"responsePackageHash":null,"senderInstitution":"URU","status":"pending"}]
```

Imagen 50. Consulta al ledger de Fabric

En la imagen 50 se puede ver que la solicitud de información realizada en la blockchain de Corda también quedó registrada en Fabric y, como en la blockchain de Fabric no existía ninguna solicitud de información, la única solicitud de información registrada en Fabric es la que se realizó en Corda.

A su vez, si utilizamos la aplicación web conectada a la blockchain de Fabric, vemos la misma información (imagen 51) que en la prueba anterior pero de forma más amigable.

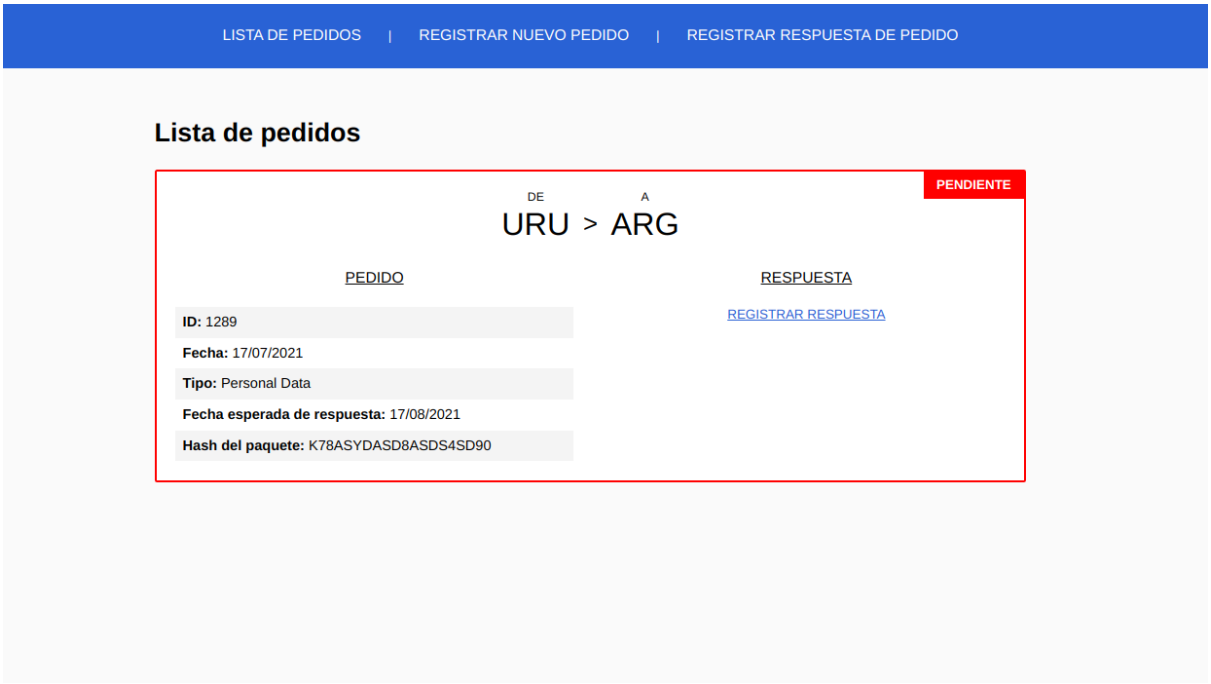

Imagen 51. Transacción confirmada en la aplicación web de Fabric

5.4. Decisiones de implementación

En esta sección se mencionan las decisiones que se tomaron al momento de realizar la implementación.

Para la implementación del gateway se eligió JavaScript por ser la tecnología con la que el equipo está más familiarizado. Con el mismo criterio fueron elegidos los lenguajes para las implementaciones en las blockchains y la aplicación web; de los tres lenguajes (Go, Java y JavaScript) en los que se pueden implementar los smart contracts en Fabric se optó por JavaScript, entre los que ofrece Corda (Java y Kotlin) se optó por Java y para la aplicación web se optó por Angular.

A continuación se mencionan otras decisiones de implementación que se tomaron a lo largo del proyecto:

Gateway

- Implementar conectores y router en Node.js.
- En el router se utiliza un archivo de configuración que contiene las direcciones de los conectores de cada blockchain. Este archivo cumple la función de tabla de ruteo.
- El conector de Corda expone un endpoint (webhook) que recibe los eventos de la blockchain.
- El conector de Fabric utiliza la SDK provista para interactuar con la blockchain.

Corda

- Tener un nodo especial “gateway” a través del cual se realiza la comunicación con Fabric.

- Implementar dos flujos, uno para las operaciones iniciadas en Corda y otro para las operaciones iniciadas en Fabric. Estos flujos implementan las operaciones requeridas para el escenario de seguridad social.

Fabric

- Implementar un smart contract que soporta las operaciones requeridas para el escenario de seguridad social.

Aplicación web

- Implementación de servidores intermedios que hacen posible la comunicación con las blockchains.

6. Evaluación de la solución

En este capítulo se realiza una evaluación de la solución propuesta e implementada intentando mencionar los puntos fuertes y débiles de la misma.

La solución propuesta básicamente posibilita la comunicación entre dos plataformas de blockchain mediante el intercambio mensajes (eventos) y se basa en el hecho de que las blockchains confían en el gateway como se mencionó en el capítulo 4. Dicha confianza es necesaria para lograr la interoperabilidad y los autores del artículo *SoK: Communication Across Distributed Ledgers* [46] demuestran que no es posible la interoperabilidad sin una tercera parte confiable.

Algunas de las alternativas relevadas durante el proyecto proponen soluciones o protocolos que intentan mitigar el impacto de que las terceras partes que proveen la interoperabilidad no sean confiables. Un ejemplo de esto es el protocolo propuesto en el artículo *Decentralized Cross-Blockchain Asset Transfers* [47], donde la blockchain destino se comunica con la blockchain origen y realiza cálculos criptográficos que le permiten asegurar si una transacción fue confirmada en la blockchain origen. De esta forma, la blockchain destino puede recibir un evento de un gateway no confiable, pero cuenta con un mecanismo para confirmar si la transacción recibida efectivamente ocurrió en el origen.

El intercambio de eventos permite a una blockchain saber lo que ocurre en otras y realizar las acciones que desee. Acá cabe resaltar que, cuando una blockchain recibe un evento de otra, es la propia blockchain (la que recibe el evento) quien decide que hacer, incluso podría no hacer nada e ignorar el evento.

Por otro lado, se puede decir que, como las plataformas de blockchain han sido construidas de forma completamente independiente (sin seguir ningún estándar en común), con diferencias en las funcionalidades que ofrecen y las implementaciones de las mismas, resulta necesario tener algún software intermediario para lograr la interoperabilidad. En nuestra solución, dicho software son los conectores y el router.

6.1. Puntos fuertes

Como se mencionó anteriormente, esta solución permite que dos blockchains se puedan comunicar. Eso creemos que es el primer paso hacia la interoperabilidad y la principal virtud de esta solución, ya que en las condiciones actuales las plataformas de blockchain no son capaces de comunicarse.

Portabilidad
Dado que el gateway está compuesto por servidores web, brinda flexibilidad sobre la infraestructura necesaria para implementar la solución en una determinada realidad. Los servidores correspondientes al router y conectores. En caso de seguir el diseño presentado en la sección 4.3, pueden estar alojados en infraestructuras distribuidas e incluso pertenecientes a diferentes organizaciones.

A pesar de que estos servidores fueron implementados utilizando Node.js, no existen restricciones en cuanto a las tecnologías en que éstos pueden estar implementados.

Escalabilidad
Es sencillo añadir nuevas plataformas a la solución de interoperabilidad. Basta con incluir un servidor web que implemente el conector específico para la plataforma que se quiere añadir e incluir este conector a la lista de conectores alcanzables por el router.

6.2. Puntos débiles

Además de puntos fuertes, también creemos que la solución propuesta presenta aspectos débiles que, por razones de alcance de proyecto, no pudieron ser abordados. Estos aspectos pueden ser tratados como trabajo futuro y se los menciona a continuación.

Único punto de fallo
Como punto débil de la solución propuesta, se puede señalar que en los conectores y el router existe un único punto de fallo y eso va en contra de una de las propiedades de blockchain. Para mitigar/resolver esto, se podría tener varios conectores por cada plataforma y varios routers. En ese escenario, se tendría que definir una forma de comunicación que asegure que un mensaje solo llegue una vez a la blockchain destino o, si llega más de una vez, que el mismo solo sea tenido en cuenta una vez. Esto se considera como posible trabajo futuro.

Confirmacion de recepcion de eventos
En el contexto de este proyecto, cuando una blockchain emite un evento, no existe un mecanismo que asegure la llegada del evento a la blockchain destino. Es decir, si ocurre un error en la propagación del evento desde la blockchain origen a la destino, el evento se pierde. Sería deseable contar con un mecanismo que contemple estos posibles errores y asegure la llegada del evento a la blockchain destino.

6.3. Comparativa con otras soluciones

En esta sección se realiza una comparación de la solución propuesta con las soluciones relavadas que más se acercan a los requerimientos del proyecto.

6.3.1. Hyperledger Cactus

La solución provista por Hyperledger Cactus era la que más se adecuaba al proyecto al momento del relevamiento de soluciones existentes. Asimismo, el diseño (imagen 52) de esta solución tiene algunas diferencias con la solución propuesta en el proyecto [48].

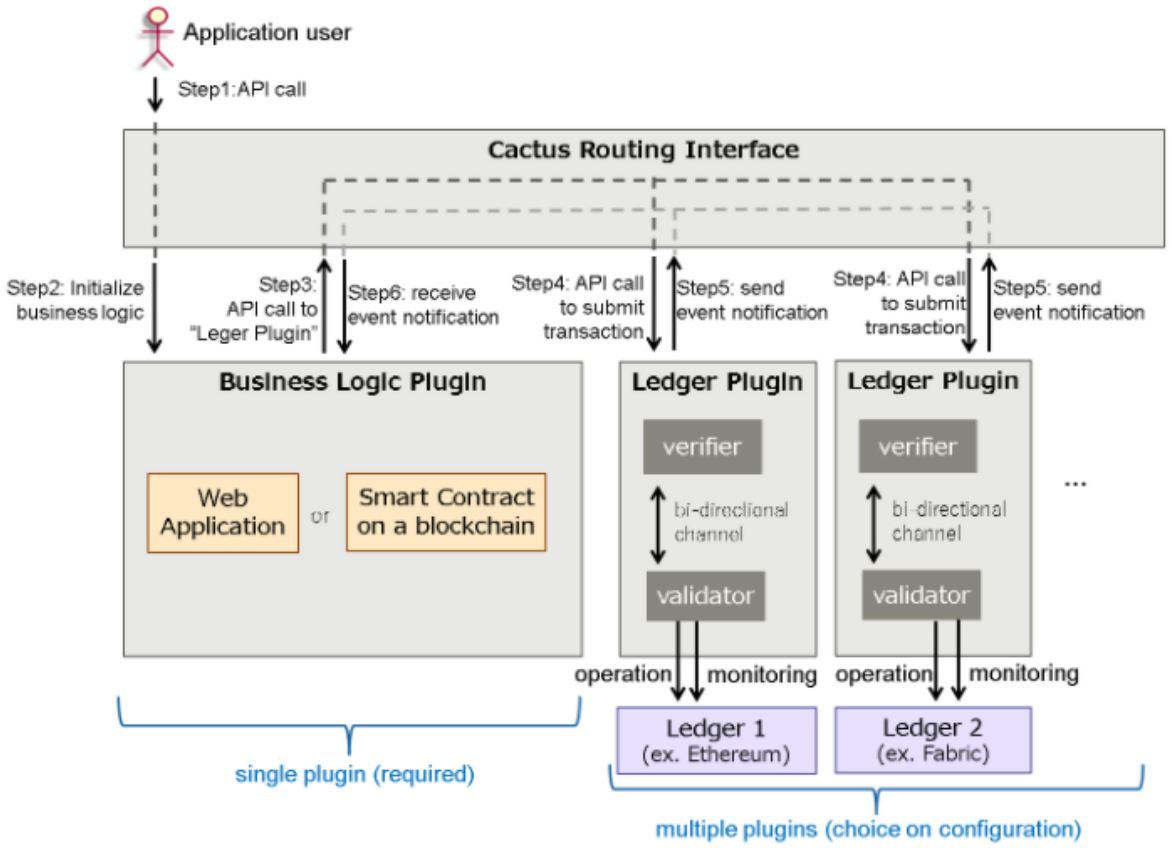

Imagen 52. Arquitectura de Hyperledger Cactus.

A continuación se mencionan algunas diferencias y similitudes entre Hyperledger Cactus y la solución propuesta.

Diferencias
A continuación se describen las diferencias.

- **Cactus Routing Interface**
  Toda transacción cross-chain se inicia con una llamada al módulo Routing Interface de Cactus. Este módulo determina qué acción se debe realizar y se encarga de comunicar a la blockchain correcta.

  La diferencia radica en que con la solución propuesta en este proyecto, las transacciones se inician invocando una operación directamente a una de las blockchains, sin pasar por una entidad externa (como el routing interface de Cactus). Esto puede ser una ventaja, ya que, las aplicaciones externas que usan las blockchains lo hacen de la misma forma que lo harían normalmente y no se ven afectadas por la solución.

- **Validadores**
  Los validadores (imagen 53) forman una red de nodos externos a una blockchain, los cuales tienen acceso al estado del ledger, y tienen el objetivo de validar las transacciones cross-chain. Corren su propio algoritmo de consenso, independiente del algoritmo utilizado por la blockchain internamente. La solución propuesta en el

proyecto no cuenta con esta capa de validación y se basa en que el gateway es una entidad de confianza.

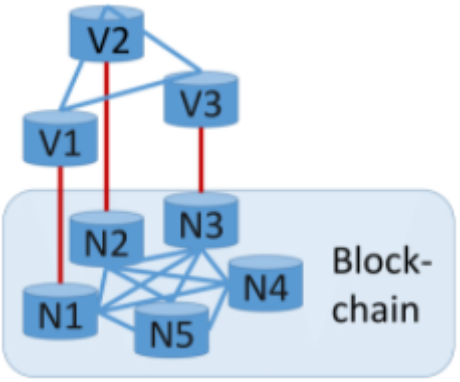

Imagen 53. Diagrama de validadores en Hyperledger Cactus

Similitudes
A continuación se describen las similitudes.

- ***Pluggable connectors***
  Posee conectores específicos para cada plataforma de blockchain involucrada en la interoperabilidad. Estos conectores a su vez son *pluggables*, pudiendo fácilmente incluir nuevos conectores/plataformas y también removerlos.

  La solución propuesta en el proyecto comparte este concepto.

6.3.2. Abebe Et Al

En el artículo *Enabling Enterprise Blockchain Interoperability with Trusted Data Transfer* [42] los autores proponen una solución basada en *Trusted Relays*. Esta solución (imagen 54) también presenta similitudes y diferencias con la solución propuesta en el proyecto.

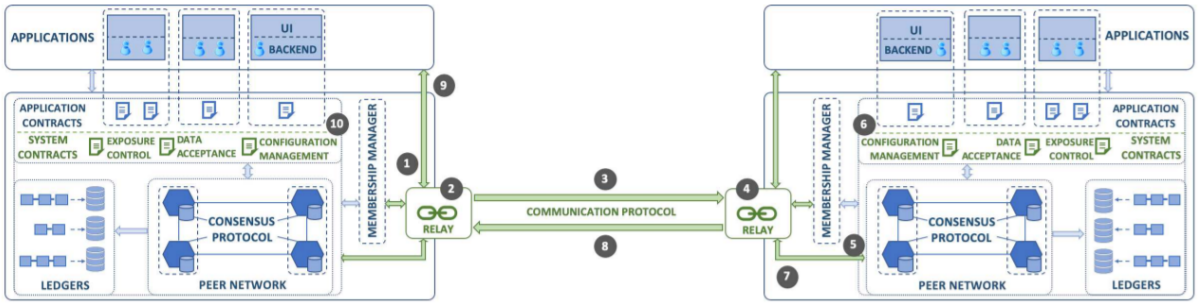

Imagen 54. Diagrama de arquitectura.

Diferencias
A continuación se describen las diferencias.

- Las transacciones se inician a través de los *relays*. Para esto, las aplicaciones solicitan la ejecución de una transacción al *relay* que le corresponde. De esta

manera, y de forma similar a Hyperledger Cactus, las transacciones deben pasar a través de la solución de interoperabilidad desde su inicio.

La diferencia radica en que con la solución propuesta en este proyecto, las transacciones se inician invocando una operación directamente a una de las blockchains, sin pasar por una entidad externa. A esto le vemos la ventaja que, las aplicaciones externas son agnósticas a la solución de interoperabilidad, ya que, la forma en que la mismas interactúan con las blockchains se ve afectada.

- Los autores proponen un protocolo de comunicación entre los *relays*. Este protocolo está fuertemente enfocado en la consulta de datos de una blockchain a otra, aunque puede ser modificado para otros propósitos (como la invocación de operaciones) reutilizando varios componentes.

- La solución se enfoca en el pedido de información desde una blockchain a otra. Por lo general, se trata de información faltante para poder ejecutar completamente una transacción en una blockchain.

  La solución propuesta en este proyecto simplemente ejecuta una transacción en una blockchain remota y no recibe una respuesta síncrona. Además, no hay limitantes en las acciones realizadas por las transacciones.

Similitudes
A continuación se describen las similitudes.

- La comunicación entre dos blockchains se realiza a través de *relays* y son estos quienes se comunican entre sí punto a punto. En la solución propuesta se cuenta con un concepto equivalente (*gateways*), los cuales proveen la comunicación entre blockchains.

7. Gestión del proyecto

En este capítulo se describe el proceso llevado a cabo para la realización del proyecto.

7.1. Organización del proyecto

Para la realización de este proyecto, se plantearon reuniones cada dos semanas con los tutores para de esta manera iterar el trabajo en períodos cortos y plantear objetivos alcanzables.

El proyecto se organizó en varias fases siguiendo un modelo en cascada. Las fases del mismo fueron las siguientes:

1. Investigación de la tecnología blockchain
2. Relevamiento de soluciones existentes de interoperabilidad
3. Diseño de la solución propuesta
4. Implementación de la solución
5. Documentación

Dichas fases se describen a continuación.

1. Investigación de la tecnología blockchain
Esta fue la primera fase del proyecto y durante la misma se adquirió información relativa al funcionamiento de blockchain y las plataformas de blockchain. Durante esta fase se recopiló la mayoría de la información presentada en el capítulo 2.

Esto fue necesario debido a que los integrantes del equipo no contaban con ningún conocimiento previo sobre blockchain y esto era un requisito para luego poder adentrarse en la problemática de la interoperabilidad.

2. Relevamiento de soluciones existentes
Una vez entendido el funcionamiento de blockchain, se procedió a relevar soluciones existentes buscando alguna que pudiera resolver el escenario planteado en el proyecto.

Durante esta fase se observó que la problemática de la interoperabilidad comenzó a ser atacada activamente en los últimos años, existiendo una gran cantidad de soluciones propuestas. A su vez, se observó que muchas de esas soluciones solo existen a nivel teórico y no cuentan con implementaciones de referencia, existiendo así una brecha entre el campo teórico y práctico.

El resultado de esta fase fue que no se encontró ninguna solución que se adaptara al escenario planteado y las soluciones relevadas se mencionan en la sección 2.3.4.

3. Diseño de la solución propuesta
Debido al resultado de la fase anterior, se procedió al diseño de una solución propia que permitiera interoperar dos plataformas de blockchain. Para el diseño de la misma, se tuvo

en cuenta las soluciones propuestas por otros autores que fueron relevadas en la fase anterior.

El diseño de esta solución se hizo de forma iterativa y en conjunto con los tutores, siendo la solución resultante la presentada en el capítulo 4.

4. Implementación de la solución
En esta fase, se implementó la solución previamente diseñada. A su vez, se implementó el escenario de seguridad social (descrito en el capítulo 3) sobre las plataformas Hyperledger Fabric y Corda, para así aplicar la solución de interoperabilidad sobre ambas plataformas.

5. Documentación
Una vez culminadas las fases anteriores, se procedió a la elaboración del presente informe con el fin de documentar todo lo realizado durante dichas fases.

Nuevamente, la elaboración del presente informe se realizó de forma iterativa, recibiendo correcciones por parte de los tutores en cada interacción.

7.2. Cronograma

En la imagen 55 se presenta un diagrama que contiene la duración de cada una de las fases del proyecto.

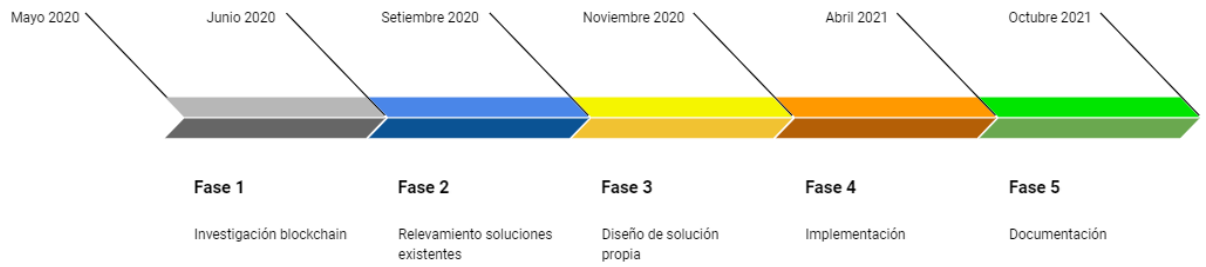

Imagen 55. Cronograma.

8. Conclusiones y trabajo futuro

En este capítulo se presentan las conclusiones del proyecto (sección 8.1) y los posibles trabajos a futuro (sección 8.2).

8.1. Conclusiones

El principal objetivo de este proyecto era encontrar una forma genérica de interoperar distintas plataformas de blockchain. Dicho objetivo se considera logrado ya que la solución propuesta permite una comunicación básica entre dos plataformas de blockchain, sin necesidad de modificarlas.

En cuanto a los objetivos específicos y aportes mencionados en las secciones 1.1 y 1.2, los mismos también fueron cumplidos y a continuación se detalla cómo se cubrieron cada uno de ellos.

Dado que los miembros del equipo no contaban con conocimientos previos de blockchain, fue necesario el aprendizaje de las bases de la tecnología. Como parte del proceso, se logró organizar los conocimientos adquiridos (presentados en el capítulo 2), siendo este uno de los aportes del proyecto.

El objetivo específico 1 se logró mediante el relevamiento de soluciones de interoperabilidad existentes presentado en la sección 2.4. De las soluciones relevadas, se puso mayor foco en aquellas que permitían interoperabilidad entre plataformas no relacionadas a criptomonedas, ya que eran las de mayor interés para los objetivos del proyecto.

El objetivo específico 2 se logró mediante la especificación (e implementación de referencia) de una solución de interoperabilidad entre plataformas de blockchain, agnóstica a la tecnología. Con esta solución, una blockchain es capaz de emitir un evento destinado a otra blockchain y que de esta forma exista comunicación entre ellas. Para esto, se utiliza un gateway que actúa de intermediario entre las distintas plataformas. Este gateway está compuesto por conectores (uno por cada plataforma a interoperar) y un router. Los conectores se encargan de resolver la comunicación con cada plataforma, absorbiendo las diferencias que existen entre las distintas plataformas y así proveer una interfaz genérica al router, quien se encarga de enrutar los eventos entre los conectores. Para que esta solución pueda ser aplicada, es necesario que las blockchain tengan la capacidad de enviar información a aplicaciones externas cuando sea necesario. A su vez, es necesario que aplicaciones externas puedan disparar operaciones en las blockchains.

El objetivo específico 3 se logró mediante la implementación de un escenario de intercambio de información de seguridad social tanto en Hyperledger Fabric como en Corda, descrito en el capítulo 5. Los sistemas construidos en ambas plataformas interoperan utilizando la implementación de referencia de la solución propuesta.

A continuación se comparten algunos puntos de vista y previsiones personales acerca de la interoperabilidad entre plataformas de blockchain.

En base al relevamiento de soluciones de interoperabilidad existentes, se considera que Hyperledger Cactus es la solución más madura dadas las funcionalidades que ofrece y las plataformas que soporta. Además se la percibe como la solución con mayor proyección a futuro dado que cuenta con el apoyo e interés de grandes empresas.

Por otro lado creemos que, si la tecnología blockchain despierta suficiente interés y adopción, en el mediano o largo plazo la interoperabilidad entre plataformas de blockchain se resolverá mediante el uso de estándares (al cual deberán adaptarse las plataformas). En el corto plazo, la interoperabilidad deberá lograrse mediante el uso de diferentes soluciones, las cuales demostrarían interés por la problemática y contribuirían a la aparición de dichos estándares. Es aquí donde se espera que este proyecto aporte valor.

8.2. Trabajo futuro

En esta sección se presentan algunas funcionalidades o características de la solución que quedaron fuera del alcance de este proyecto y aportarían valor al proyecto en caso de abordarse en el futuro.

8.2.1. Seguridad

Actualmente la información transmitida entre las blockchains viaja en texto plano y una blockchain no tiene la posibilidad de verificar que el origen del mensaje es el correcto. Para esto sería deseable que los datos transmitidos por el gateway sean cifrados con algún mecanismo de cifrado adecuado. Además, que los eventos emitidos por las blockchains sean firmados aportaría un valor de confiabilidad extra. De esta forma, una blockchain podría asegurarse de que los mensajes que recibe desde el gateway son válidos, disminuyendo así el grado de confianza necesaria sobre el gateway. Una posible implementación sería utilizando HMAC[4].

8.2.2. Alta disponibilidad

La implementación realizada soporta un único gateway, generando así un único punto de fallo. Es deseable contar con un mecanismo que soporte redundancia en los gateways y descentralice la comunicación.

8.2.3. Confirmación de recepción de eventos

La solución implementada no contempla posibles errores en la propagación de eventos desde la blockchain origen a la blockchain destino. Sería necesario contar con un mecanismo que maneje dichos errores y asegure la llegada del evento a la blockchain destino, de forma de asegurar la consistencia de los datos entre ambas blockchains. Una implementación posible sería un escenario de two-phase commit.

[4] HMAC: Keyed-Hashing for Message Authentication. https://datatracker.ietf.org/doc/html/rfc2104

8.2.4. Hyperledger Cactus

A la fecha de finalización del proyecto, Hyperledger Cactus soporta las plataformas Fabric y Corda, por lo que sería interesante realizar una implementación del escenario sobre dicha herramienta y compararla con la solución propuesta en este proyecto. Dicha comparación sería importante para ponderar la solución brindada respecto a una solución referente en la industria.

8.2.5. Diseño alternativo

La solución descrita en la sección 4.1 se basa en tener un único gateway que debe ser ajeno a las blockchains a interoperar. En un ambiente empresarial donde cada organización utiliza una plataforma de blockchain, el diseño anterior puede presentar un problema con respecto a donde o quien debe alojar el gateway. Teniendo en cuenta este escenario, se propuso una alternativa al diseño anterior que se ajusta a dicho escenario. Este diseño no fue implementado durante el proyecto por simplicidad, y su implementación se considera trabajo futuro.

El diseño alternativo se ilustra en la imagen 56 y en el mismo cada blockchain tiene su propio gateway, donde cada uno está compuesto por un conector y un router.

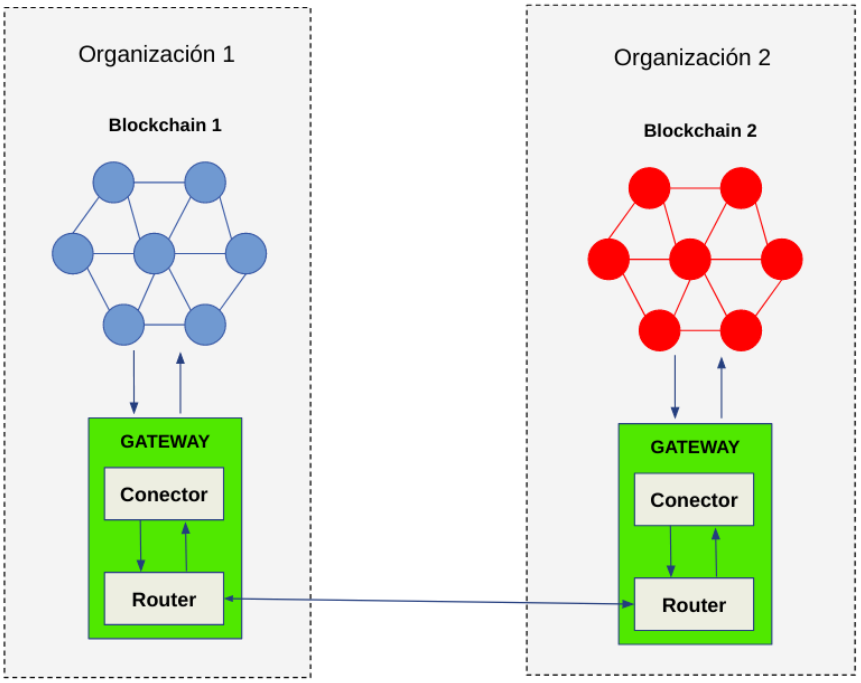

Imagen 56. Diseño alternativo

En este caso, cuando una blockchain emite un evento, lo que sucedería es lo siguiente.

1. El conector del gateway correspondiente a la blockchain origen recibe el evento y lo envía al router del mismo gateway.
2. El router del gateway de la blockchain origen envía el evento al router del gateway de la blockchain destino.
3. El router del gateway de la blockchain destino envía el evento a su conector.
4. El conector del gateway de la blockchain destino notifica a la blockchain.

Este enfoque tiene la ventaja de que cada organización puede encargarse de implementar su propio gateway, lo cual le otorga control sobre el router. Sería de interés una comparativa entre ambos enfoques.

9. Referencias

[1] Xiwei Xu, Ingo Weber, y Mark Staples, *Architecture for Blockchain Applications*.
[2] M. Pereira, M. Toscano, P. Villar, I. J. Barreiro, y I. G. Llambías, «Plataformas blockchain y escenarios de uso».
[3] S. Nakamoto, «Bitcoin: Un Sistema de Efectivo Electrónico Usuario-a-Usuario», p. 9.
[4] «blockchain». https://dictionary.cambridge.org/dictionary/english/blockchain (accedido may 17, 2021).
[5] «Making sense of bitcoin, cryptocurrency and blockchain». https://www.pwc.com/us/en/industries/financial-services/fintech/bitcoin-blockchain-cryptocurrency.html (accedido may 17, 2021).
[6] «Blockchain Explained». https://www.investopedia.com/terms/b/blockchain.asp (accedido ago. 03, 2021).
[7] «¿Qué es un nodo?» https://academy.bit2me.com/que-es-un-nodo/ (accedido ago. 03, 2021).
[8] «Cuántos tipos de blockchain hay». https://academy.bit2me.com/cuantos-tipos-de-blockchain-hay/ (accedido ago. 04, 2021).
[9] «¿Qué es un Algoritmo de Consenso?» https://academy.binance.com/es/articles/what-is-a-blockchain-consensus-algorithm (accedido ago. 03, 2021).
[10] «Proof of Work (PoW)». https://www.investopedia.com/terms/p/proof-work.asp (accedido ago. 03, 2021).
[11] «Understanding Proof-of-Work: Achieving Consensus and the Double Spend Attack». https://medium.com/coinmonks/understanding-proof-of-work-achieving-consensus-and-the-double-spend-attack-f822ab68e20b (accedido sep. 02, 2021).
[12] M. Castro y B. Liskov, «Practical Byzantine Fault Tolerance».
[13] «practical Byzantine Fault Tolerance(pBFT)». https://www.geeksforgeeks.org/practical-byzantine-fault-tolerancepbft/ (accedido ago. 10, 2021).
[14] «Smart Contracts: ¿Qué son, cómo funcionan y qué aportan?» https://academy.bit2me.com/que-son-los-smart-contracts/ (accedido ago. 03, 2021).
[15] «Extender contratos inteligentes de blockchain con lógica que está fuera de la cadena». https://developer.ibm.com/es/tutorials/cl-extend-blockchain-smart-contracts-trusted-oracle/ (accedido sep. 04, 2021).
[16] «What Is Blockchain and How Does It Work?» https://www.thesslstore.com/blog/what-is-blockchain-how-does-blockchain-work/ (accedido sep. 11, 2021).
[17] «Hyperledger Fabric». https://www.hyperledger.org/use/fabric (accedido ago. 04, 2021).
[18] «Smart Contracts and Chaincode — Hyperledger documentation». https://hyperledger-fabric.readthedocs.io/en/release-2.2/smartcontract/smartcontract.html (accedido ago. 04, 2021).
[19] «Ledger — Hyperledger documentation». https://hyperledger-fabric.readthedocs.io/en/release-2.2/ledger/ledger.html#world-state (accedido ago. 04, 2021).
[20] «Chaincode Events - Hyperledger Fabric Documentation». https://ibm.github.io/hlf-internals/shim-architecture/interaction-flow/chaincode-events/ (accedido ago. 04, 2021).
[21] «Channels — hyperledger-fabricdocs master documentation». https://hyperledger-fabric.readthedocs.io/en/release-2.2/channels.html (accedido oct. 19, 2021).
[22] «Peers — hyperledger-fabricdocs master documentation». https://hyperledger-fabric.readthedocs.io/en/release-2.2/peers/peers.html (accedido ago. 03, 2021).
[23] «Hyperledger Fabric SDKs — Hyperledger documentation».

https://hyperledger-fabric.readthedocs.io/en/release-2.2/fabric-sdks.html (accedido ago. 04, 2021).
[24] «R3 Corda: What Makes It Different», *Corda*, oct. 25, 2016. https://www.corda.net/blog/r3-corda-what-makes-it-different/ (accedido ago. 06, 2021).
[25] «Corda: An Introduction», [En línea]. Disponible en: https://docs.corda.net/en/pdf/corda-introductory-whitepaper.pdf
[26] «Corda | Multi-Party Application Development Platform for Business». https://www.corda.net/ (accedido jun. 13, 2021).
[27] R. Brown, «Corda Top Ten Facts #7: Both a blockchain and not a blockchain», *Corda*, sep. 06, 2018. https://medium.com/corda/corda-top-ten-facts-7-both-a-blockchain-and-not-a-blockchain-9d8104db056c (accedido ago. 06, 2021).
[28] «Ledger». https://docs.corda.net/docs/corda-os/4.8/key-concepts-ledger.html (accedido may 23, 2021).
[29] «Flows». https://docs.corda.net/docs/corda-os/4.7/key-concepts-flows.html (accedido jun. 13, 2021).
[30] «Definición de interoperabilidad - Diccionario panhispánico del español jurídico - RAE». https://dpej.rae.es/lema/interoperabilidad (accedido jun. 13, 2021).
[31] «SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)». https://www.w3.org/TR/soap12/ (accedido sep. 19, 2021).
[32] «Fielding Dissertation: CHAPTER 5: Representational State Transfer (REST)». https://roy.gbiv.com/pubs/dissertation/rest_arch_style.htm (accedido sep. 19, 2021).
[33] M. Middleware y E. Curry, «1».
[34] *Hyperledger Cactus Whitepaper*. Accedido: sep. 06, 2021. [En línea]. Disponible en: https://github.com/hyperledger/cactus/blob/e7b60a04051d0590203482430e5161d5f6a31d24/whitepaper/whitepaper.md
[35] «Cosmos: The Internet of Blockchains». https://cosmos.network (accedido ago. 03, 2021).
[36] Pricewaterhouse Coopers, «PWC Global Blockchain Survey 2018». [En línea]. Disponible en: https://www.pwccn.com/en/research-and-insights/publications/global-blockchain-survey-2018/global-blockchain-survey-2018-report.pdf
[37] R. Belchior, A. Vasconcelos, S. Guerreiro, y M. Correia, «A Survey on Blockchain Interoperability: Past, Present, and Future Trends», Accedido: ago. 03, 2021. [En línea]. Disponible en: https://arxiv.org/abs/2005.14282v3
[38] «IBC Overview | Cosmos SDK». https://docs.cosmos.network/master/ibc/overview.html (accedido sep. 23, 2021).
[39] «Interledger: Interledger Protocol V4 (ILPv4)». https://interledger.org/rfcs/0027-interledger-protocol-4/ (accedido sep. 23, 2021).
[40] «Interoperability layers | Joinup». https://joinup.ec.europa.eu/collection/nifo-national-interoperability-framework-observatory/3-interoperability-layers (accedido jun. 13, 2021).
[41] «Hyperledger Cactus», *Hyperledger*. https://www.hyperledger.org/use/cactus (accedido ago. 03, 2021).
[42] A. Ermyas *et al.*, «Enabling Enterprise Blockchain Interoperability with Trusted Data Transfer.»
[43] L. Kan, Y. Wei, A. H. Muhammad, W. Siyuan, G. Linchao, y H. Kai, «A Multiple Blockchains Architecture on Inter-Blockchain Communication.»
[44] V. Buterin, «Notary Schemes». [En línea]. Disponible en: https://www.r3.com/wp-content/uploads/2017/06/chain_interoperability_r3.pdf
[45] «API: Flows». https://docs.r3.com/en/platform/corda/4.7/open-source/api-flows.html (accedido oct. 23, 2021).
[46] A. Zamyatin *et al.*, «SoK: Communication Across Distributed Ledgers», 1128, 2019. Accedido: jul. 11, 2021. [En línea]. Disponible en: http://eprint.iacr.org/2019/1128
[47] M. Sigwart, P. Frauenthaler, C. Spanring, y S. Schulte, «Decentralized Cross-Blockchain

Asset Transfers», *ArXiv200410488 Cs*, abr. 2020, Accedido: jul. 11, 2021. [En línea]. Disponible en: http://arxiv.org/abs/2004.10488
[48] *hyperledger/cactus*. Hyperledger, 2021. Accedido: jul. 25, 2021. [En línea]. Disponible en: https://github.com/hyperledger/cactus/blob/539a801d28b5143ea685c0719c55869540f89983/whitepaper/whitepaper.md
[49] «¿Qué es una cadena lateral o sidechain?» https://academy.bit2me.com/que-es-cadena-lateral-sidechain/ (accedido ago. 03, 2021).
[50] «Sidechains, Drivechains, and RSK 2-Way peg Design | RSK - Smart Contract Platform Secured by the Bitcoin Network». https://blog.rsk.co/es/noticia/sidechains-drivechains-and-rsk-2-way-peg-design/ (accedido ago. 03, 2021).
[51] «What is a Hashed Timelock Contract?» https://www.investopedia.com/terms/h/hashed-timelock-contract.asp (accedido ago. 03, 2021).
[52] T. Hardjono, A. Lipton, y A. Pentland, «Toward an interoperability architecture for blockchain autonomous systems».
[53] «Hyperledger Quilt - Hyperledger Quilt - Hyperledger Confluence». https://wiki.hyperledger.org/display/quilt/Hyperledger+Quilt (accedido ago. 03, 2021).
[54] G. Wang, «SoK: Exploring Blockchains Interoperability», 537, 2021. Accedido: sep. 26, 2021. [En línea]. Disponible en: http://eprint.iacr.org/2021/537
[55] «Overledger». https://www.quant.network/ (accedido ago. 03, 2021).
[56] Z. Liu y Y. Xiang, «Hyperservice: Interoperability and programmability across heterogeneous blockchains».
[57] «What is Overline?» https://docs.overline.network/docs/block-collider-core-technology (accedido ago. 03, 2021).
[58] M. Amiri, D. Agrawal, y E. A. Mohammad Amr, «A Cross-Application Permissioned Blockchain.»

Anexo I. Soluciones orientadas a criptomonedas

Esta categoría está compuesta por soluciones que generalmente se utilizan para transferencia de criptomonedas de una blockchain a otra. Estas soluciones aplican a blockchains públicas (no permisionadas) y se utilizan mayoritariamente en blockchains homogéneas (blockchains construidas sobre la misma plataforma). Esta categoría es dividida en cuatro subcategorías:

- Sidechains
- Notary Schemes
- Hashed timelocks contracts
- Combined solutions

Sidechains [49]
También llamadas

- Relay chains
- Secondary chains
- Satellite chains
- Child chains
- Sub chains

En este tipo de solución, se utiliza una tercera blockchain (además de las dos que se quiere interoperar) que actúa como intermediaria y se le llama sidechain. El principal uso de esta solución es el mismo que la categoría padre (transferencia de assets de una blockchain a otra). En este contexto, el término ‘asset’ generalmente se puede entender como cripto moneda.

A la blockchain desde la cual se va a realizar la transferencia se le denomina source blockchain y target blockchain a la blockchain a la cual se realiza la transferencia. Los assets son transferidos desde la source blockchain a la sidechain y luego de la sidechain a la target chain. Para estas transferencias se utilizan distintos tipos de mecanismo para garantizar la seguridad, siendo el más común two way peg [50]. Estos algoritmos están totalmente ligado al hecho de que lo que se realiza entre ambas blockchains es una transferencia de assets.

Notary schemes [44]
Un ‘notary’ es una entidad que monitorea varias blockchains y dispara transacciones en una de ellas de acuerdo a los eventos que ocurren en las demás. También existe la posibilidad de tener un conjunto de notaries, lo que permite descentralizar en algo la cuestión.

Estas soluciones simplifican bastante las transacciones entre blockchains pero a cambio de eso es necesario una entidad (o entidades) que son de confianza. En este caso, en contrario a sidechains, la solución en lugar de ser una extensión a las blockchains, es un software de un terciario que realiza operaciones en las mismas.

Hashed timelocks contracts (HTLCs) [51]
El objetivo de estas soluciones es el mismo que la anterior pero evitando la centralización. Proveen un mecanismo para el intercambio de assets entre dos actores (o a veces más).

Atomic swaps están dentro este concepto y permiten el intercambio entre actores que están en distintas blockchain. A continuación explicamos cómo funciona este mecanismo.

Un ‘atomic swap cross-chain’ es una tarea de coordinación distribuida en la que varios actores intercambian assets en varias blockchains. Tomemos como ejemplo la siguiente situación:

- Carol quiere vender su auto por bitcoins.
- Alice quiere comprar ese auto pero quiere pagar con ‘alt-coins’ (otra criptomoneda distinta de bitcoin).
- Bob está dispuesto a cambiar ‘alt-coins’ por bitcoins.
- Ningún actor confía en los demás.

En este escenario existen tres blockchains: una en la cual se manejan bitcoins, otra en la cual se manejan ‘alt-coins’ y otra en la cual se manejan títulos de propiedad.

Lo que propone atomic swaps es lo siguiente

- Alice transferirá ‘alt-coins’ a Bob
- Bob transferirá bitcoins a Carol
- Carol transferirá el título de propiedad de su auto a Alice

Para lograr esto, se tendrá lo siguiente:

- Una función criptográfica H que dada una clave secreta s, genera un hash h = H(s). Dicha función debe ser irreversible.
- Un timelock t es el tiempo/duración durante el cual un smart contract es válido.

Entonces, para lograr lo mencionado anteriormente, se hará lo siguiente:

1. Alice creará aleatoriamente una clave secreta s y publicará en la alt-coins blockchain un smart contract con los alt-coins necesarios para comprar el auto. Dicho contrato tendrá hash h y timelock t. Esto significa que, si alguien dentro del tiempo t, llama al smart contract con un valor x tal que h = H(x), entonces el smart contract transferirá los alt-coins a esa persona. Claramente x debe ser igual a s que solo Alice conoce.
2. Bob, cuando haya detectado que el smart contract anterior fue publicado, publicará en la blockchain de bitcoins un smart contract que transferirá los bitcoins equivalente a los alt-coins del contrato anterior. Este nuevo contrato tendrá el mismo hash que el anterior y un timelock menor. Este contrato funciona igual que el anterior.
3. Carol, una vez que haya detectado que el contrato anterior fue publicado, publicará en la blockchain de títulos de propiedad un smart contract que transfiere el título de propiedad de su auto. Este contrato tendrá el mismo hash que los anteriores y un timelock menor que el contrato del punto 2.
4. Alice, una vez que detecta que el anterior contrato fue publicado, llama al mismo con s y adquiere el título de propiedad del auto. Alice, al realizar esta acción, revela s a Carol.
5. Carol ahora conoce s por lo cual puede llamar al contrato publicado por Bob y recibirá bitcoins. Al realizar esto, Carol revela s a Bob.
6. Bob llamará al contrato publicado inicialmente por Alice y recibirá alt-coins.

Esta es la forma en que funciona atomic swaps y el mecanismo descrito no parece generalizable a otros casos que no sean transferencia de assets.

Combined solutions

Este tipo de soluciones se utiliza para el mismo objetivo que las anteriores. Las soluciones dentro de este tipo varían pero lo que tienen en común es que mezclan/utilizan los conceptos mencionados en las categorías anteriores.

Anexo II. Blockchain engines

Este tipo de soluciones proveen una infraestructura que permite el desarrollo de blockchains independientes (cada una con un cierto propósito distinto) y brindan la posibilidad de que dichas blockchains puedan interoperar. Las dos grandes soluciones dentro de este tipo son Cosmos y Polkadot. Cosmos permite interoperabilidad entre blockchains construidas con su tecnología y con blockchains basadas en Tendermint [2] (este es el cambio que se mencionaba anteriormente, esto en un principio no era posible). Polkadot permite interoperabilidad entre blockchains construidas con su tecnología y con blockchains basadas en Substrate.

Estas soluciones no resuelven completamente la problemática porque existen restricciones en las blockchains que se pueden interoperar, por lo cual sigue existiendo fragmentación.

Cosmos [35]
Esta solución es un framework que permite la construcción de blockchains paralelas e independientes que pueden trabajar en conjunto. Para la conexión entre blockchains se utiliza un protocolo llamado IBC, el cual permite entre otras cosas permite transferencias de assets y consultar información entre blockchains.

Anexo III. Blockchain connectors

Blockchains agnostic protocols

Estas soluciones son protocolos propuestos que permiten la comunicación entre distintas blockchains. Debido a que estas soluciones son propuestas de protocolos, las plataformas existentes deberían adaptarse a los mismos para poder ser utilizados.

Las siguientes tres soluciones se encuentran dentro de esta categoría.

Hardjono et al [52]

En este artículo se define una arquitectura para blockchains que permitiría interoperabilidad entre las mismas. El diseño de dicha arquitectura está totalmente basado en el diseño que se utilizó para Internet, ya que, según los autores, los problemas que se enfrentan hoy al intentar resolver la interoperabilidad en blockchains son análogos a los problemas por los que tuvo que pasar Internet. Es por eso que se basan en ese diseño para aprovechar el conocimiento/lecciones aprendidas con Internet.

Al definir la arquitectura previamente mencionada, también se logra que componentes comunes a todas las blockchains puedan comenzar a ser estandarizados, lo cual facilita el desarrollo, mejora la reusabilidad y permite mayor interoperabilidad.

En la arquitectura definida cada blockchain es vista como un sistema autónomo o una unidad de conexión. Dichos sistemas tienen un control central con topología distribuida. Dentro de dichos sistemas existen gateways, los cuales son nodos/entidades capaces de validar o ejecutar transacciones cross-blockchain. A su vez, los gateways pueden ser descubiertos y pueden redireccionar transacciones hacia otras blockchains..

Interledger protocol (ILP) [39]

Es un protocolo (discutido por la W3C) que permite realizar pagos entre distintas plataformas que implementan el protocolo.

Hyperledger Quilt [53]

Esta solución es una implementación open-source en Java de ILP. Es compatible con otras implementaciones de ILP como Interledger Rust o InterledgerJS.

Blockchain of blockchains

Estas soluciones operan sobre distintas plataformas de blockchain heterogéneas permitiendo interoperabilidad entre las mismas. Dichas soluciones poseen una blockchain principal (*main chain*), donde cada uno de sus participantes es otra blockchain (*subchain*). El propósito de la blockchain principal es registrar la actividad de cada *subchain*, actuando como un notario. [54]

Overledger [55]

Esta es una solución que corre sobre distintas plataformas de blockchains heterogéneas y permite interoperabilidad entre las mismas. Para soportar las distintas plataformas, utiliza un

conector desarrollado particularmente para cada plataforma. También existe la posibilidad de que, en lugar de implementar dicho conector a medida, se realice un refactor de la plataforma de blockchain para adaptarla a un cierto estándar y de esa forma poder funcionar con overledger.

Esta solución se ha vuelto popular y poco tiempo atrás firmó un acuerdo con la SIA (una compañía italiana de pagos en Europa que tiene una red de 500 bancos) para permitir interoperabilidad entre la blockchains de los bancos. Esta solución no es de código abierto (es una solución propietaria) y paga.

HyperService [56]
Esta solución es producto de un estudio académico y permite desarrollar dapps (aplicaciones distribuidas) que utilizan varias blockchains heterogéneas. La idea de la solución es proporcionar un lenguaje único (HSL) que permite escribir programas (smart contracts) que afectan a las distintas blockchains.

La solución propuesta cuenta con una implementación (que está disponible en un repositorio github) y a modo de demostración, aplicaron la solución para una blockchain Ethereum y otra custom blockchain construida sobre Tendermint. Al parecer, esas son las únicas dos plataformas soportadas hasta el momento. Según los autores, sería sencillo agregar soporte para nuevas plataformas de blockchains.

Block collider [57]
Esta solución permite comunicación entre smart contracts de blockchains heterogéneas. A su vez, provee una tercera blockchain (además de las que se interoperan) cuyos bloques referencian los últimos bloques de cada una de las blockchain interoperadas. Esto permite una visión unificada de todas las blockchains.

Este proyecto es open source y actualmente soporta varias plataformas de criptomonedas. La idea de su funcionamiento de interoperabilidad es generalizable pero los casos de usos a los que se ha aplicado (y para los cuales parece estar pensado) son escenarios de criptomonedas.

CAPER [58]
Este proyecto no es una solución de interoperabilidad entre blockchains heterogéneas sino que es una plataforma de blockchain que puede ser usada por varias dapps y se pueden realizar transacciones cross applications entre dichas dapps. Claramente, acá no se está atacando el problema de la interoperabilidad entre distintas plataformas de blockchain sino que, lo que se busca es permitir interoperabilidad entre aplicaciones distribuidas que corren sobre la misma plataforma de blockchain.

Blockchain migrators

Estas soluciones aplican para los casos en los que se tiene una plataforma de blockchain y se quiere pasar a utilizar otra plataforma. Estas soluciones permiten la migración de información de una plataforma a otra y, a futuro, se prevé la posibilidad de migrar smart

contracts. Para realizar dicha migración, en algunos casos se utiliza un programa central que es ejecutado por un usuario.

Anexo IV. Flujo de transacciones en Hyperledger Fabric

La arquitectura modular de Hyperledger Fabric separa el flujo de procesamiento de transacciones en tres fases y este flujo se inicia cuando una aplicación externa quiere disparar una operación de un smart contract. A continuación se describen estas tres fases.

Fase 1
La aplicación externa se conecta a un conjunto de peers que validan la operación que dicha aplicación quiere realizar. Para esto, estos peers realizan la ejecución de la operación del smart contract, lo cual da como resultado un conjunto de transacciones que se deben registrar en la cadena de bloques. Dichos nodos devuelven a la aplicación externa una respuesta firmada que contiene estas transacciones y los pares clave-valor que se leyeron del estado del sistema. Estos pares clave-valor son incluidos en la respuesta porque son requeridos en la fase tres. Cabe destacar que los nodos simplemente realizan una simulación de la ejecución sin aplicar los resultados (transacciones) de la misma.

Una vez que los nodos necesarios (los cuales se definen en políticas de Fabric) han validado la operación, la aplicación externa envía las respuestas de todos los nodos al nodo orderer.

Fase 2
El nodo orderer recibe dichas respuestas de distintas aplicaciones externas y se encarga de comprobar si las transacciones contenidas en esas respuestas están validadas por los nodos necesarios. Las transacciones que cumplan con lo anterior serán ordenadas y colocadas dentro de un bloque y, cada vez que se genera un nuevo bloque, el nodo orderer lo envía a los nodos peers para que lo agreguen a su cadena.

Fase 3
En esta fase, los nodos peers reciben bloques por parte del nodo orderer, lo agregan a su cadena local y validan cada una de las transacciones contenidas en el mismo. Dicha validación incluye, entre otras cosas, comprobar que las transacciones están validadas por todos los nodos necesarios y que el conjunto de pares claves-valor leído del estado del sistema en la fase uno coincide con el estado actual (para que la transacción pueda ser aplicada). Las transacciones que no pasen esta validación, son igualmente mantenidas dentro del bloque pero marcadas (con una bandera) como inválidas y sus cambios no son aplicados al estado del sistema.

En la imagen 57, se muestra un diagrama el cual muestra de forma gráfica lo que sucede en estas tres fases.

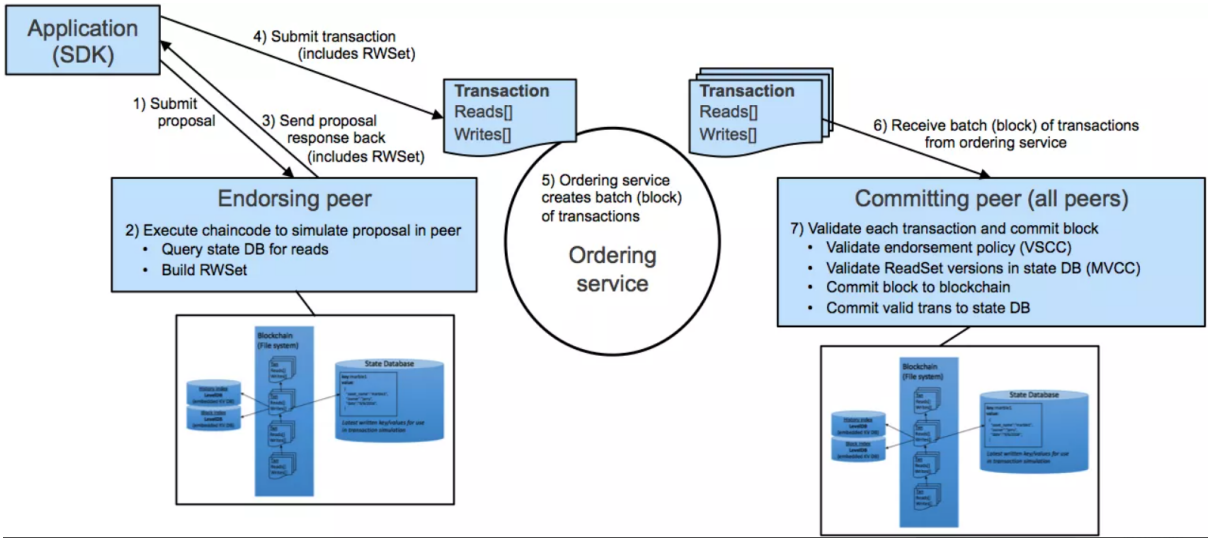

Imagen 57.

Como ya se mencionó anteriormente, las aplicaciones externas utilizan la SDK de Fabric para disparar operaciones de smart contracts y, las tareas realizadas en la fase 1 por parte de la aplicación externa, son realizadas por la SDK, lo cual simplifica mucho el trabajo de los desarrolladores de dichas aplicaciones.

Anexo V. Conceptos de Corda

Contratos
Como se menciona en la sección anterior, para que una transacción sea confirmada no sólo debe ser firmada por todos los pares involucrados, sino que debe satisfacer las reglas de un contrato.

El hecho de que una transacción sea considerada *contractualmente válida* se define como:

- La transacción especifica el tipo de contrato a utilizar.
- El contrato toma la transacción como entrada y, basándose en las reglas definidas en el contrato, determina si es válida o no.
- La transacción es válida si y sólo si el contrato determina que todos los *input states* y todos los *output states* son válidos.

Los contratos son piezas de software escritas en los lenguajes *Java* o *Kotlin*, y su ejecución es determinista.

El código de estos contratos tiene total acceso a la información almacenada en la transacción y sus estados asociados, y es sólamente en esa información en lo que se basa para determinar la validez.

Finalmente, si el contrato determina que una transacción no es válida, ésta no es registrada en el *ledger*.

Consenso
Corda subdivide el concepto de consenso en dos: consenso de validez y consenso de unicidad. Para que una transacción sea considerada válida y almacenada en el *ledger*, debe alcanzar ambos tipos de consenso.

<u>Consenso de validez</u>
Es el proceso de validar si una nueva transacción y todas las transacciones en la cadena de transacciones cumplen con las siguientes condiciones:

- Son contractualmente válidas
- Tienen todas las firmas requeridas.

Este proceso es realizado en conjunto por todos los nodos involucrados en la transacción.

Es importante destacar que no solo se valida la nueva transacción, sino también todas las transacciones previas en la cadena de transacciones. Esto es debido a que se desea confirmar que el nuevo estado del *ledger* es alcanzado mediante una serie de transacciones válidas.

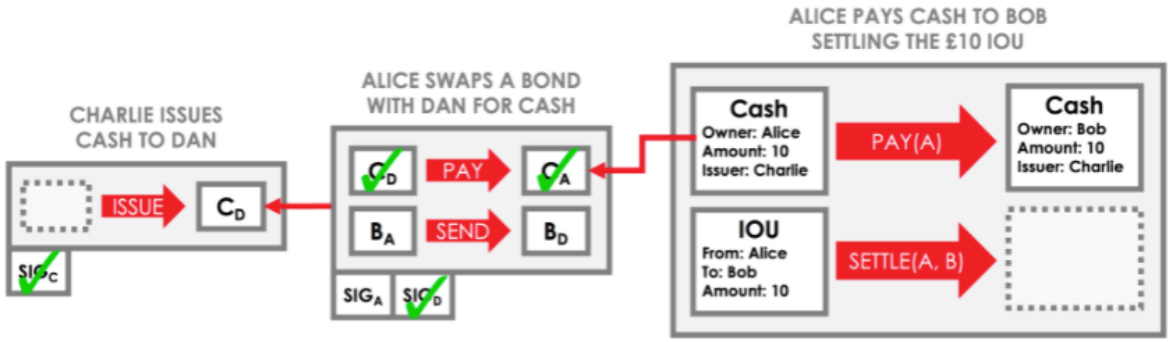

Imagen 58. Validación de transacciones en Corda.

En la imagen 58, Alice le entrega dinero a Bob, saldando su deuda. En ese caso, lo que se hace es validar que el dinero que Alice le entrega, fue obtenido de una forma válida.

Consenso de unicidad

Es el proceso de asegurar que las transacciones no utilicen un mismo *input state* más de una vez, siendo éste el problema conocido como *double spend*.

Esta tarea es realizada por un nodo especial conocido como *Notario*.

Notarios

Es un tipo de nodo especial en Corda que provee el servicio de consenso de unicidad, siendo su tarea verificar que no hayan *double spends* y firmar las transacciones. Se pueden agrupar formando *clusters* de notarios.

El notario logra esto verificando que no ha firmado anteriormente otra transacción donde se haya utilizado el mismo *input state* que se está queriendo utilizar en la nueva transacción a verificar.

El notario a su vez determina la finalidad de una transacción, ya que ningún nodo puede almacenar una transacción hasta que ésta no posea la firma del notario.