UNIVERSITAT OBERTA DE CATALUNYA

Máster universitario en seguridad de las tecnologías de la información y las comunicaciones

TRABAJO FIN DE MÁSTER

Proceso de portabilidad numérica entre teleoperadoras basado en Hyperledger Fabric

Por:

Francisco Caballero Girol

Tutora y profesora colaboradora:

María Francisca Hinarejos Campos

Madrid, enero de 2020

A mi mujer, Bego, pues cada uno de sus logros demuestra que las ganas y el esfuerzo son el binomio ideal para conseguir nuestros objetivos.

A mis hijos, Mario y Clara, porque al regalarme tantas sonrisas de manera desinteresada me hacen sentir muy afortunado.

A mi madre, Carmen, y a mis hermanos, Raúl y Carlos, por estar siempre cerca y hacerme sentir su cariño.

Y a ti, papá, que me enseñaste que hacer las cosas bien no es una opción, sino una obligación. Gracias por cada enseñanza que no supe valorar en su momento y que hoy están presentes en cada proyecto de mi vida.

RESUMEN EJECUTIVO

Cada día en España se realizan una media de 25000 solicitudes de portabilidad numérica entre las operadoras de telefonía nacional. Una solicitud de portabilidad consiste en el deseo del propietario de una línea de teléfono de cambiar de operador, sin la necesidad de tener que cambiar el número de teléfono asociado a dicha línea.

Para que una portabilidad se lleve a cabo ha de existir una coordinación en las tareas a realizar entre todos los actores del proceso. Las operadoras de telefonía se han organizado en forma de consorcio, definiendo una serie de reglas de negocio, sistemas centrales de datos y protocolos de comunicación para que las portabilidades lleguen a buen puerto. El sistema centralizado que soporta las solicitudes de portabilidad tiene una serie de riesgos inherentes a este tipo de arquitecturas que aumentan las probabilidades de sufrir problemas que impidan su correcto funcionamiento.

Con el fin de mejorar la eficiencia del proceso y mitigar los posibles fallos, este trabajo realiza un estudio de cómo se llevan a cabo las portabilidades de numeración telefónica fija en España y plantea una alternativa basada en la cadena de bloques permisionada Hyperledger Fabric.

Fabric es una tecnología relativamente nueva que permite el registro de transacciones inmutables entre miembros de una red privada con identidades reconocidas entre sí. El rendimiento, la escalabilidad de la red, el nivel de confianza entre miembros o la protección de los datos sensibles son otras características de Hyperledger Fabric que facilitan su incorporación a casos de negocio en el mundo industrial.

ÍNDICE

Contenido

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

El mundo de las telecomunicaciones lleva inmerso, desde sus comienzos, en una evolución y revolución tecnológica tan rápida que pocos sectores pueden decir lo mismo. El *Big Bang* de este sector comienza allá en 1833 con la aparición del telégrafo, sustituyendo a cartas y postales como medio tradicional de comunicación entre personas alejadas físicamente entre sí. La llegada del teléfono en 1820, y el establecimiento de la primera llamada a larga distancia en 1920 supusieron el inicio de la nueva era de telecomunicaciones.

La tecnología siguió evolucionando y poco a poco las comunicaciones abandonaron los sistemas analógicos para hacer uso de componentes digitales. La llegada de las computadoras permitió que las comunicaciones cada vez fueran más rápidas y menos costosas, y junto con la revolución que supuso el nacimiento de Internet, el intercambio de mensajes empezó a realizarse a través de diversas tecnologías, como los emails y los chats. Hoy día, la inteligencia artificial, la llegada del 5G, IoT, Edge Computing, Big Data o Deep Learning son algunas de las tecnologías que ya han abrazado las operadoras telefónicas, permitiéndoles ofrecer nuevos servicios de mayor calidad, haciendo un uso más eficiente de los recursos y disminuyendo los costes de producción.

Y es que las personas cada vez tenemos la necesidad de estar más conectados digitalmente, recibiendo y produciendo cada día cantidades ingentes de información. Para poner en perspectiva el volumen de datos que circula por las redes de telecomunicaciones del mundo, basta con echar un vistazo a la infografía titulada *"Los datos nunca duermen"*, de la consultora Domo [1], en la cual se analizaron las interacciones de los usuarios conectados a Internet **durante un minuto** en 2018. Los datos son los siguientes:

- Se visionaron 97.222 horas de vídeo en Netflix
- Se cuelgan 49.380 posts en Instagram
- Se escuchan en Spotify 750.000 canciones
- 4.333.560 visualizaciones de YouTube
- Se publican 473.400 tuits
- Se realizan 176.220 llamadas de Skype

Volviendo al mundo exclusivo de las comunicaciones, el aumento del catálogo de servicios que ofrecen las operadoras telefónicas (4G, geolocalización, videollamada…) conlleva a su vez un incremento del número de líneas de teléfono activas en nuestro país. La siguiente tabla muestra la variación, i.e.: altas y bajas, de los distintos tipos de líneas de telefonía de nuestro país durante el mes de septiembre de 2019.

LÍNEAS TELEFONÍA MÓVIL		**LÍNEAS TELEFONÍA FIJA**	
Variación mensual	**62.586**	Variación mensual	**-14.914**
Parque total	**54.023.533**	Parque total	**19.232.874**
Tasa variación del parque total (sep'18-sep'19)	**1,19%**	Tasa variación del parque total (sep'18-sep'19)	**-0,72%**

Tabla 1. *Datos extraídos de la página oficial de la Comisión Nacional de los Mercados y la Competencia, CNMC.* **[2]**

Sólo en un mes ha habido un incremento neto de más de cuarenta y siete mil líneas. Con un parque de líneas total que supera los 73 millones de líneas, los equipos de marketing de las teleoperadoras diseñan cada día nuevas estrategias para captar clientes y "robarlos" a la competencia, consiguiendo así no solo un aumento de sus beneficios sino una minoración en los de sus competidores de mercado. Este cambio de operadora conservando el número de teléfono es conocido habitualmente como p*ortabilidad,* y su definición, según la Comisión Nacional de los Mercados y la Competencia es [3]:

"...el proceso que permite a un cliente darse de baja en el operador que le presta servicio (donante) y simultáneamente solicitar el alta con otro operador (receptor), conservando su número"

En la siguiente tabla se muestra el número de portabilidades de líneas fijas y móviles ocurridas durante el último año:

MES/AÑO	PORTABILIDADES LÍNEA FIJA	PORTABILIDADES LÍNEA MÓVIL
septiembre 2018	217.139	700.707
octubre 2018	237.147	707.710
noviembre 2018	189.169	633.107
diciembre 2018	184.888	537.970
enero 2019	189.243	669.227
febrero 2019	171.146	598.155
marzo 2019	177.765	621.139
abril 2019	181.675	560.973
mayo 2019	174.970	569.874
junio 2019	145.476	553.680
julio 2019	180.138	627.499
agosto 2019	167.654	557.243
septiembre 2019	193.698	669.791

Tabla II. Datos mensuales de portabilidades extraídos de la página oficial de la CNMC [1]

En el proceso de portabilidad de una línea distinguimos a los siguientes participantes:

- **Cliente:** es el propietario de la línea que desea portar
- **Operadora donante:** es la operadora que pierde un cliente
- **Operadora receptora:** es la operadora que gana un cliente
- **Operadoras terceras:** aquellas que no intervienen directamente en la portabilidad pero que han de estar informados del cambio

La portabilidad de una línea conlleva una serie de operaciones que varían dependiendo de los servicios que el cliente tenga contratados y haya querido y podido portar. En esencia, las tareas que se llevan a cabo durante la portabilidad de una línea son:

- **Baja de los servicios contratados.** La operadora donante deberá actualizar su parque de clientes con el fin de reflejar la baja de la línea. Esto implicará registrar los cambios en la base de datos, modificación de las centralitas de red, desinstalación física de componentes instalados (router, nodos de telefonía…).

- **Alta de los nuevos servicios contratados.** Del mismo modo, la operadora receptora deberá dar de alta al cliente y realizar las acciones necesarias para prestar los servicios contratados por el cliente.
- **Portabilidad del número de teléfono.** Implica notificar a todas las operadoras que el número de teléfono a portar ahora pertenece a otra compañía, para que cada una de ellas realice los cambios en sus centralitas y nodos de telefonía necesarios para poder enrutar las llamadas al número portado.

Todas estas tareas han de realizarse durante un periodo determinado que actualmente se encuentra fijado en 24h, tal y como recomienda la Unión Europea.

La portabilidad es un derecho de todos los usuarios, y las operadoras están obligadas a facilitar el cambio. Mediante la Resolución de 15 de julio de 2004, el Consejo de la Comisión del Mercado de las Telecomunicaciones aprobó la Circular 2/2004 de conservación de la numeración [4], sentándose así las bases del modelo técnico, económico y organizativo que deberían seguir las operadoras para garantizar la ejecución de la portabilidad. En particular, se estableció la obligación a las operadoras de financiar el sistema centralizado de portabilidad fija, denominado Entidad de Referencia y más adelante **Asociación de Operadoras para la Portabilidad, AOP.** Del mismo modo, desde la publicación de la Circular 1/2008 sobre conservación y migración de numeración telefónica [5], las operadoras de telefonía móvil tienen la misma obligación de gestionar las operaciones de portabilidad de una manera centralizada, dando lugar a la **Asociación de Operadoras para la Portabilidad Móvil, AOPM.**

Así, estas dos asociaciones de teleoperadoras se encargan de velar y garantizar las portabilidades de los clientes, redefiniendo de manera constante sus procesos para adaptarlos a las exigencias de la CNMC y Unión Europea. Para poder coordinarse con las operadoras, AOP y AOPM mantienen bases de datos maestras en las que se relacionan los números de teléfono con la operadora actual de dicha línea. Cuando se realiza una portabilidad, AOP y AOPM actúan de intermediarios de las comunicaciones, exponiendo servicios de comunicación consumidos por las operadoras para que intercambien la información necesaria, actualizando al mismo tiempo las bases de datos maestras.

Los sistemas centralizados, como los que soportan los procesos de portabilidad en España, facilitan gran parte de la gestión de un proceso al recaer ésta sobre un elemento central. Esta arquitectura no está exenta de inconvenientes, tal y como veremos más adelante en el documento.

2. ACRÓNIMOS Y DEFINICIONES

- **AOP:** Asociación de Operadoras para la Portabilidad
- **AOPM:** Asociación de Operadoras para la Portabilidad Móvil
- **ASP:** Tipo de mensaje en el proceso de portabilidad que corresponde a una aceptación de solicitud de portabilidad
- **CA:** Autoridad de certificación. Entidad de confianza, responsable de emitir y revocar los certificados, utilizando en ellos la firma electrónica, para lo cual se emplea la criptografía de clave pública
- **Chaincode:** agrupación de smart contracts para su despliegue. También es considerado como un smart contract a bajo nivel
- **CNMC:** Comisión Nacional del Mercado de las Comunicaciones
- **CP:** Tipo de mensaje en el proceso de portabilidad que corresponde a una confirmación de portabilidad
- **DLT:** Tecnología de contabilidad distribuida. **(**Distributed Ledger Technology). Hyperledger Fabric y Bitcoin son implementaciones distintas de una DLT
- **DSP1:** Tipo de mensaje en el proceso de portabilidad que corresponde a una denegación de solicitud de portabilidad
- **DSP2:** Tipo de mensaje en el proceso de portabilidad que corresponde a una denegación de solicitud de portabilidad
- **ER:** Entidad de referencia. Es como se conoce también a la AOP y AOPM
- **Hyperledger:** Plataforma de código abierto para implementar redes permisionadas usando cadenas de bloques
- **Ledger:** libro de contabilidad. En el mundo tecnológico, hace referencia al repositorio donde se encuentran las transacciones y los datos de una red DLT.
- **MSP:** Servicio proveedor de membresía (Membership Service Provider). Componente de Hyperledger Fabric que ofrece una abstracción de la arquitectura de operaciones de miembros de la red
- **NRN:** Número de enrutamiento de red (Network Routing Number)
- **PBFT**: Practical Byzantine Fault Tolerance. Es un algoritmo de consenso para consorcios de empresas donde los miembros confían entre sí de manera parcial
- **Peer:** par o nodo de una red P2P, que se caracterizan por tener una topología en estrella
- **PNC:** Tipo de mensaje en el proceso de portabilidad que indica que una portabilidad no es cancelable
- **PoC:** Prueba de concepto (Proof of Concept)
- **Proceso batch:** Secuencia de órdenes que se ejecuta con posterioridad a la recepción de los datos que han de ser tratados.

- **SFTP:** *Secure File Transfer Protocol.* Protocolo de transferencia segura de ficheros
- **Smart contract:** scripts que contienen reglas de negocio acordadas entre varias partes y que se ejecutan cuando se cumplen una serie de condiciones
- **SP:** Tipo de mensaje en el proceso de portabilidad que corresponde a una solicitud de portabilidad
- **TFM:** Trabajo de Fin de Máster
- **TPS:** Transacciones Por Segundo
- **Zero-knowledge-proof:** Prueba de conocimiento nulo. Es un protocolo que permite conocer si una parte conoce un secreto sin que ésta deba revelarlo.

3. OBJETIVOS

A la hora de elegir un tema para el TFM me exigí a mí mismo que se deberían cumplir las siguientes premisas:

- Debía estar centrado en un área de conocimiento o temática que desconociera. La investigación sobre algo desconocido sería más provechosa y divertida que ahondar en alguna temática ya familiar.
- El TFM debía dar solución a un problema o escenario real. Me gusta pensar que el trabajo aquí realizado puede ser el punto de partida para resolver o mejorar un problema existente.

El hecho de trabajar en una empresa de telecomunicaciones me ha permitido ser conocedor de situaciones o problemas técnicos que no siempre son resueltos de la manera más idónea. Los costes, el tiempo de desarrollo y, por qué no, el desconocimiento de las personas encargadas de encontrar la solución, son variables que afectan de manera directa a la decisión tomada. Ser conocedores de todas las soluciones que nos brinda la tecnología actual nos amplía nuestro abanico de posibilidades para afrontar este tipo de problemas o de, al menos, tenerlas en cuenta para aproximarnos lo más posible a la solución más ideal.

El tema para este TFM fue elegido cuando conocí el modelo centralizado en el que mi empresa interviene a la hora de gestionar las portabilidades de líneas telefónicas. Me pregunté si sería posible optimizarlo. Teniendo en cuenta tal escenario, este trabajo se centra en el estudio del propio proceso de portabilidad fija con el fin de adaptarlo a la red distribuida Hyperledger Fabric. El motivo de usar esta tecnología, y no otra cadena de bloques, es que desde su concepción Hyperledger Fabric nació con el fin de crear consorcios de entidades que se reconocen entre sí, i.e., no permitiendo la unión a esta red a cualquier persona u organización externa al negocio. Existen otras redes, como la especialización privada Ethereum Quorum, que también encajarían en el escenario que aquí tratamos sobre las portabilidades, aunque la superior velocidad de transacción de Fabric con respecto a Quorum [6] (3500tps frente a unos cientos de tps) y el uso por parte de Ethereum de un lenguaje no estándar como es Solidity para implementar los chaincodes, fueron los motivos que hicieron que me inclinara a estudiar Hyperledger Fabric como tecnología a utilizar para dar solución al problema que en este trabajo se describe.

Así pues, los siguientes objetivos limitarán el alcance de este proyecto. Todo aquello que no se indique explícitamente no estará contemplado, bien por limitaciones temporales en la realización del proyecto o porque realmente no encaja en el alcance descrito.

- Estudio de los procesos de portabilidad de numeración de líneas fijas sin acceso mayorista.
- Reconocer los requisitos principales de los procesos de portabilidad de numeración y sus deficiencias.
- Conocer el estado del arte de Hyperledger Fabric, su despliegue y su configuración.
- Diseño e implementación de una PoC acerca de un proceso de portabilidad implementado con el framework Hyperledger Fabric.
- Mejorar y simplificar el proceso, aumentando los tiempos que tienen las operadoras para tomar sus decisiones en cuanto al tratamiento de las portabilidades.
- Estudio de la confidencialidad y/o privacidad de los datos intercambiados por los distintos actores que intervienen en un proceso de portabilidad, comprobando que se mantiene o se puede implementar dicha confidencialidad en la PoC.

4. CONTEXTUALIZACIÓN Y JUSTIFICACIÓN

Las arquitecturas de red centralizadas, como la que soporta el proceso de portabilidad numérica en España, adolecen de ciertos problemas e inconvenientes que van ligados a la naturaleza innata de la centralización.

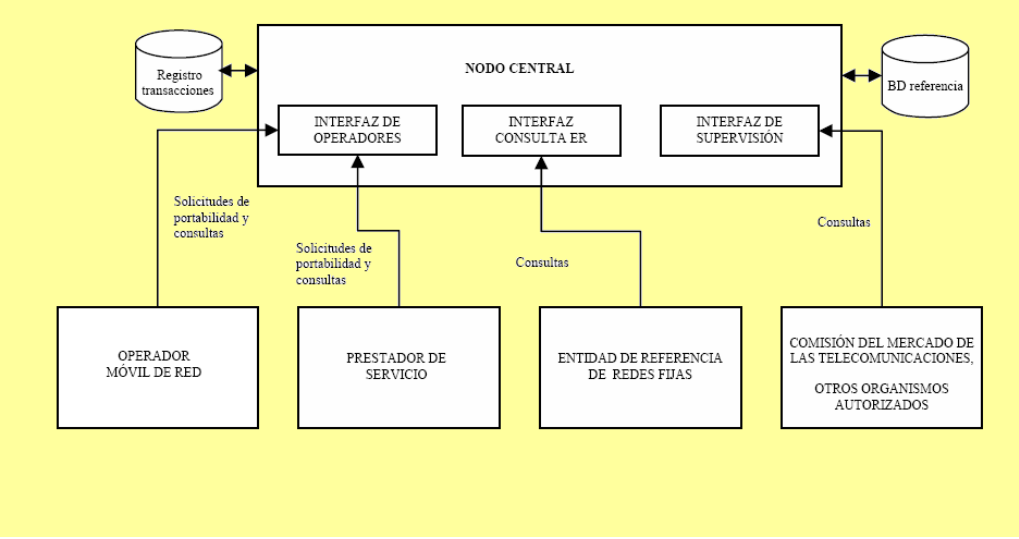

Figura 1. El nodo central, propiedad de la AOP, mantiene los datos y las interfaces que permiten a las operadoras realizar las solicitudes de portabilidad.

En la Figura 1 vemos como el nodo central, que reside en los sistemas de la AOP, pone a disposición de las operadoras de telefonía y red interfaces para enviar y recibir información acerca de las solicitudes de portabilidad. Los datos, por su parte, están almacenados en bases de datos también centralizadas en los sistemas de la AOP, y su actualización se realiza siempre desde los sistemas propios de esta entidad.

Esta arquitectura centralizada tiene las siguientes desventajas:

- **Poco estable.** Cualquier fallo en el nodo central provoca que el proceso quede completamente inoperativo.
- **Problemas de seguridad.** Los nodos centrales suelen ser objetivo de ataques con el fin de desestabilizar un sistema.
- **Lentitud.** Los sistemas centrales tienen una cantidad limitada de recursos, lo que a menudo provoca que sean cuellos de botella e impidan acelerar el proceso.
- **Problemas de escalabilidad.** Intentar mejorar el rendimiento, aumentar la tolerancia a fallos o disminuir los problemas de seguridad escalando un nodo central, es a menudo un proyecto complejo y costoso.

Aprovechando el hecho de trabajar con gente conocedora de este proceso, realicé una serie de reuniones con compañeros que llevan el mantenimiento de los sistemas que mantienen comunicación con la entidad AOP, recibiendo y enviando diariamente datos relacionados con las portabilidades fijas. Comentaron que esta entidad juega el papel de intermediario entre las operadoras, realizando algunas validaciones de formato de los mensajes que se envían, estableciendo la hora de envío de todas las comunicaciones pendientes a cada una de ellas, y aplicando algunas reglas de negocio, además de mantener la información maestra acerca del operador actual de cada línea de teléfono y de las transacciones que se han ido realizando. Que la base de datos sea centralizada implica que las operadoras asumen que la información que contiene es correcta, sin tener medios para comprobar que las portabilidades que se van realizando son lícitas o no. Dicho de otro modo, no hay nadie que audite ni pueda garantizar que la información almacenada en esta base de datos no está siendo manipulada.

Los intercambios de información entre el nodo central de AOP y las operadoras se realizan a través de ficheros utilizando el protocolo SFTP. Para cada operador, el nodo central habilita un “casillero” en el cual deposita ficheros que contienen los mensajes que desea hacer llegar a la operadora de telefonía de turno. De manera análoga, la operadora que quiere enviar un mensaje al nodo central lo hace dejando un fichero con un formato acordado en su casillero. Estos envíos no van firmados, por lo que no se puede garantizar el origen de un fichero alojado en el casillero asignado a una operadora.

Ya en el nodo central, los ficheros son tratados mediante un proceso *batch*, lo que obliga a las operadoras a depositarlos en su casillero antes de una hora determinada. El procesamiento batch tiene mayor capacidad que un procesamiento online, lo que permite tramitar muchos más mensajes por unidad de tiempo. La desventaja de esta forma de trabajo es que los receptores de la información contenida en

dichos mensajes han de esperar hasta que el nodo central ha procesado todos los mensajes para comprobar en su casillero las respuestas a los mensajes que dejaron en su momento. Estos tiempos de espera, inevitables dada la solución actual, dejan poco margen de tiempo a las operadoras para definir una estrategia comercial que evite la baja de productos y servicios de sus clientes: actualmente, las operadoras son notificadas de las portabilidades donantes a 07:00am por parte de la AOP, y están obligadas a dar una respuesta a esa solicitud (aceptación/rechazo) antes de las 14:00 de ese mismo día.

En ocasiones puede ocurrir que algunos de los ficheros que deberían ser generados por los sistemas de la AOP no llegan a los casilleros de las operadoras por causa mayor (fallos del sistema, incomunicaciones…). En estos casos las operadoras, antes de realizar las portabilidades, deben consultar a la AOP con el fin de obtener la información almacenada en su base de datos, y así poder operar. Este proceso es conocido como “Adquisición de conocimiento de numeración portada”, y su fin es evitar incoherencias entre las bases de datos de información de las operadoras y la AOP. Para ello, las primeras están obligadas a refrescar sus bases de datos a través de este proceso cada tres días.

Visto el escenario en el que transcurre el proceso de portabilidad fija, su implementación utilizando una red permisionada de bloques otorgaría una serie de beneficios a los participantes y simplificaría el proceso. La solución propuesta debería cumplir los siguientes requisitos:

- **Privacidad de la red**. Dado que solo las operadoras de telefonía pueden realizar portabilidades, debemos garantizar que la cadena de bloques elegida sea privada.
- **Seguridad e integridad.** Se debe poder comprobar en cualquier momento quién realizó una modificación de una portabilidad, garantizando que solo miembros pertenecientes a la red realizan modificaciones.
- **Transparencia.** La cadena de bloques ha de permitir a los participantes comprobar que las transacciones realizadas son correctas y los datos no han sufrido manipulaciones.
- **Lógica de negocio en la red.** Con el fin de poder realizar las validaciones que actualmente lleva a cabo la AOP, la cadena de bloques que utilicemos deberá permitir definir contratos inteligentes entre las operadoras que nos permitan albergar la lógica del proceso.
- **Permisos.** Dependiendo de su rol, deberá poder limitarse qué acciones puede realizar y cuáles no.
- **Privacidad de los datos.** Las solicitudes de portabilidad manejan datos personales del cliente de carácter sensible. La solución ofrecida debe permitir que este tipo de datos solo puedan ser consultados por organizaciones autorizadas que formen parte de la red.

Con estos requisitos, y tras búsquedas en la red y comparaciones con otras, planteamos la solución utilizando la cadena de bloques Hyperledger Fabric. Según su sitio web [7]

"Hyperledger Fabric es un marco de trabajo de ledger (libro mayor) distribuido autorizado de tipo empresarial para el desarrollo de soluciones y aplicaciones. Su diseño versátil y modular satisface una amplia gama de casos de uso. Ofrece un enfoque único de consenso que garantiza el rendimiento a escala mientras preserva la privacidad"

Hyperledger Fabric es uno de los múltiples proyectos que existen en Hyperledger, una plataforma de código abierto creada para avanzar en las tecnologías blockchain. Fue iniciada en 2015 por la fundación Linux, y actualmente cuenta entre sus miembros gigantes de la industria como IBM, Intel o SAP. Bajo esta plataforma, como decíamos, se encuentran varios proyectos, algunos aún en fase de desarrollo y otros mucho más avanzados como Hyperledger Fabric. Algunos de estos proyectos son:

- **Hyperledger Besu.** Cliente de la cadena de bloques Ethereum escrito en java.
- **Hyperledger Burrow.** Provee un cliente blockchain modular con un intérprete autorizado de contratos inteligentes incorporado a la especificación de la máquina virtual Ethereum (EVM).
- **Hyperledger Fabric.** Libro mayor distribuido que ofrece modularidad, versatilidad y opciones de privacidad para satisfacer un amplio conjunto de casos de uso de la industria.
- **Hyperledger Indy.** Libro mayor distribuido construido para identidades descentralizadas. Provee herramientas y librerías para crear y usar identidades digitales independientes definidas en otras cadenas de bloques.
- **Hyperledger Iroha.** Plataforma blockchain con su propio y único modo de consenso y algoritmos de servicios de ordenación, soportando multifirma y con un modelo autorizado basado en roles.
- **Hyperledger Sawtooth.** Plataforma para construir, desplegar y ejecutar ledgers distribuidos, que incluye el algoritmo de consenso "Prueba de tiempo transcurrido" (PoET).

5. ANÁLISIS

5.1. PROCESO DE PORTABILIDAD FIJA

Dependiendo del tipo de portabilidad telefónica a realizar, ésta se realiza mediante proceso básico o asegurado. Que se lleve a cabo por uno u otro depende de si la portabilidad tiene asociado acceso mayorista (desagregación del bucle o acceso indirecto bitstream) o no, lo que implica que la portabilidad se realice en 24 o en 72 horas. Con el fin de facilitar la comprensión del proceso y la posterior prueba de concepto, se va a describir el caso más simple de portabilidad, sin operadoras

revendedoras ni provisión de acceso mayorista. Tampoco se entrará en el detalle del cupo máximo de peticiones por operador, aunque se hará una mención sobre éste. Para conocer el detalle exacto del proceso se puede consultar la especificación técnica del procedimiento [8].

La secuencia de acciones que se llevan a cabo durante la ejecución del proceso básico de portabilidad es la siguiente:

PASO 0: Inicio del proceso de portabilidad

Una vez que la operadora receptora ha llegado a un acuerdo con el cliente en término de contratos y condiciones del servicio a prestar, deberá iniciar el proceso de portabilidad el mismo día que el cliente realizó la solicitud.

PASO 1: Entrega de las solicitudes a la AOP

La operadora receptora incluirá en un fichero todas las Solicitudes de Portabilidad, SP, y lo alojará en su casillero de la AOP entre las 20:00 y las 21:00. Dentro del fichero se podrán adjuntar tantos mensajes SP como solicitudes de portabilidad tenga pendiente el receptor de tramitar.

Dentro del mensaje, la operadora receptora indicará los datos necesarios para realizar la portabilidad, tales como la fecha de solicitud, número a portar, identificador y nombre y dirección del abonado, operadora receptora, operadora donante, ventana de cambio, etc.

PASO 2: Validación de mensajes SP por la AOP

Los sistemas de la AOP procesarán los mensajes de tipo SP de las operadoras según el orden de llegada. Se validará la estructura de cada uno de los mensajes SP, comprobando que la fecha de la ventana de cambio indicada en la petición se encuentra dentro del plazo mínimo y máximo para portar, siendo el mínimo de un día y el máximo 30 días.

La AOP generará un fichero con mensajes de Denegación de Solicitud de Portabilidad, DSP1, para cada receptor, que indicarán las solicitudes que han sido denegadas y el motivo. Este fichero será depositado en el casillero de cada operadora receptora antes de las 07:00h. Si no hubiera ninguna solicitud errónea, el fichero no tendrá ningún mensaje DSP1.

Para las solicitudes validadas correctamente y, una vez comprobado que no se supera el cupo máximo de solicitudes que puede recibir un operador, la AOP dejará en un fichero localizado en el casillero del operador receptor los acuses de recibo de las solicitudes de portabilidad. Del mismo modo, se generará un único fichero por cada operador donante, donde se incluirán todas las solicitudes de

portabilidad SP en las que el operador participa como donante. Este fichero deberá estar disponible antes de las 07:00h.

PASO 3: Validación de solicitudes por el operador donante

La operadora donante recoge el fichero de solicitudes de portabilidad de su casillero asignado y realiza las comprobaciones pertinentes. En este caso, las validaciones que realiza la operadora donante son en referencia a, por ejemplo, si el cliente tiene alguna otra solicitud en vuelo, e.g: una modificación de su servicio de telefonía, que hace inviable la portabilidad. Si esto ocurre, la operadora dejará en su casillero un fichero con los mensajes de Denegación de solicitud de portabilidad 2, DSP2. Para las solicitudes de portabilidad aceptadas dejará otro fichero con mensajes de Aceptación de Solicitud de Portabilidad, ASP. Ambos ficheros deberán depositarse en el casillero de la operadora antes de las 14:00h.

PASO 4: Generación de confirmaciones CP

La AOP generará ficheros con los acuses de recibo para las operadoras donantes a partir de las 14:00h, tanto de las aceptaciones como de las denegaciones, dejando estos ficheros en los casilleros de las operadoras antes de las 15:00h. Asimismo, antes de las 15:00h la AOP dejará los ficheros de mensajes ASP y DSP2 en los casilleros de las operadoras receptoras. Por último, la AOP generará y pondrá a disposición de todas las operadoras receptoras, donantes y terceras, un fichero que contendrá los mensajes confirmaciones de portabilidad, CP.

PASO 5: Portabilidades no cancelables (PNC)

La AOP remitirá un fichero que contendrá mensajes de las Portabilidades No Cancelables, PNC, que son aquellas solicitudes de portabilidad confirmadas con anterioridad para las cuales no se ha recibido su cancelación por parte del operador donante. Estas portabilidades no cancelables deberán ser realizadas en las ventanas de cambio del día siguiente a su publicación.

Las solicitudes de portabilidad podrán ser canceladas antes de las 19:00h del día anterior a la ventana de cambio, tanto si es la ventana por defecto (08:00h), como si ésta difiere por voluntad del cliente.

A partir de las 20:00h, la AOP enviará a los casilleros de todas las operadoras el fichero con los mensajes PNC.

PASO 6: Ejecución de la portabilidad

En la ventana de cambio especificada en los distintos mensajes intercambiados, las operadoras donante y receptora deberán realizar los cambios necesarios en sus sistemas y componentes de red para que la portabilidad se lleve a cabo. El resto de las operadoras deberán hacer las modificaciones pertinentes en la red para reflejar los encaminamientos de las llamadas hacia los números portados.

La Figura 2 muestra el resumen de las acciones descritas anteriormente:

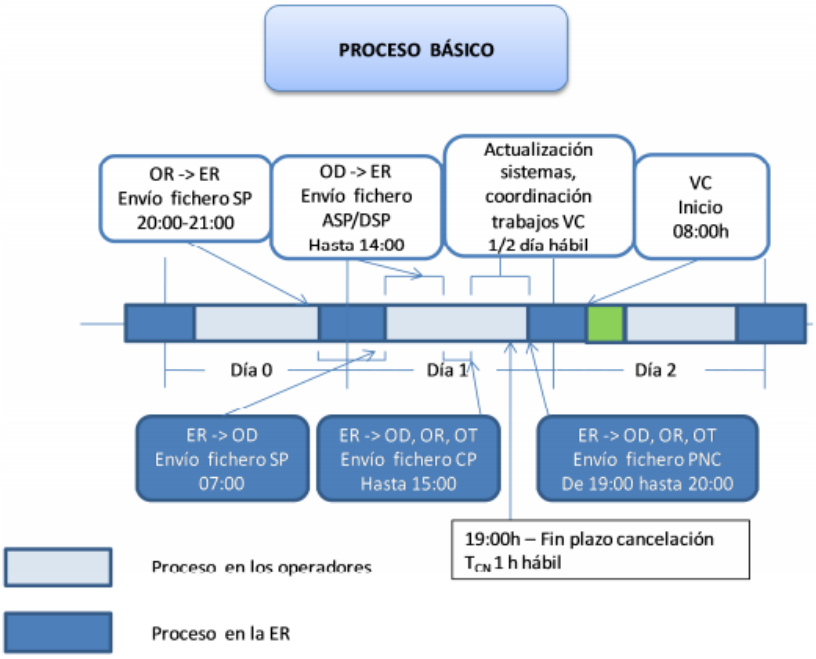

Figura 2. El proceso de portabilidad tiene prefijados unos momentos del día en el que se envían y reciben ficheros con solicitudes y cambios de portabilidades numéricas.

5.2. HYPERLEDGER FABRIC

Una *blockchain* o cadena de bloques es, en esencia, un listado de transacciones (modificaciones de datos) agrupadas en bloques, que permanecen inmutables una vez se han realizado. El símil que la comunidad de blockchain siempre utiliza es el del libro de cuentas (*ledger*, en inglés) de una empresa: en él se van escribiendo cada una de las transacciones económicas que recibe o emite la empresa en cuestión. Una vez escrita una transacción en el libro, ésta ya no puede ser alterada. En un momento dado, podemos verificar si el balance contable de la empresa reflejado es correcto recorriendo cada una de las transacciones en orden desde el inicio, sumando o restando en el caso de ingresos o gastos.

La blockchain más conocida y primera en utilizarse ha sido Bitcoin, la cual es utilizada para realizar transferencias de la criptomoneda que lleva el mismo nombre que la red. Posteriormente, Ethereum nació con características similares a Bitcoin, pero añadiendo los s*mart contracts* para crear plataformas de aplicaciones distribuidas, cuyos antecesores fueron los scripts de Bitcoin. Entre otras similitudes, Bitcoin y Ethereum se caracterizan por ser redes públicas y anónimas. i.e., cualquiera que disponga de un PC y un cliente de una de estas redes puede participar en ella sin desvelar su identidad. Estas características de Bitcoin y Ethereum obligan a las redes a funcionar basándose en algoritmos de verificación de transacciones costosos, tanto a nivel energético, debido a la alta computación, como a nivel de tiempo de proceso, lo que no las hace del todo atractivas para que las empresas las utilicen en sus casos de negocio. Además, en la mayoría de las ocasiones las empresas necesitan conocer a los participantes de sus operaciones, por lo que el anonimato no es una opción. En particular, las empresas requieren poder interaccionar con otros participantes que tienen un objetivo común, pero entre los cuales no hay una confianza total, tal y como ocurre entre empresas que intercambian mercancías, información, etc.

Dicho esto, los requisitos generales de una blockchain para el mundo empresarial son:

- Poder identificar a los participantes
- Redes privadas o restringidas
- Alto nivel de procesamiento de transacciones sin pérdida de rendimiento
- Baja latencia en el procesamiento de transacciones
- Transparencia de las operaciones, pero permitiendo privacidad y confidencialidad de los datos

Algunas blockchains del mercado ya han iniciado las tareas para adaptarse a las necesidades de las empresas, como es el caso de *Quorum,* una especialización de Ethereum. Otras, como Hyperledger Fabric, nacieron ya con la idea de poder ser utilizadas y adaptadas al entorno empresarial.

Como ya se ha indicado en el documento, Fabric es un proyecto bajo el paraguas de Hyperledger, un proyecto colaborativo de importantes empresas bajo la supervisión y gobierno de la fundación Linux. Fabric es, formalmente "*una plataforma de libro mayor distribuido permisionado de ámbito empresarial que posee algunas capacidades clave diferenciadoras sobre el resto de las plataformas blockchain"* [7]. Dos son las particularidades de Hyperledger Fabric que le hacen diferenciarse de la mayoría del resto de redes blockchain:

- Es una red de ámbito privado, por lo que solo pueden acceder e interaccionar con ella aquellas organizaciones y/o usuarios que estén autorizados para ello. El uso de certificados x.509 emitidos por autoridades de Certificación reconocidas, CA, es el mecanismo base para lograr esta privacidad.
- Dada su privacidad y reconocimiento mutuo entre las partes que participan en la red, el algoritmo de verificación de transacciones no se basa en una prueba de esfuerzo, sino en el "consenso" entre las partes a la hora de decidir la validez y el orden de las transacciones realizadas en la red. Este consenso es alcanzado en última instancia cuando el orden y los resultados de las transacciones de un bloque cumplen con las verificaciones indicadas en las políticas de consenso definidas entre todas las partes.

ARQUITECTURA DE HYPERLEDGER FABRIC

En una red Fabric encontramos los siguientes componentes que conforman la red:

1. **Peers.** Una red blockchain está compuesta principalmente por un conjunto de peers, pertenecientes a las organizaciones que forman la red. Son el elemento básico de la red porque almacenan el ledger y la lógica de negocio o smart contracts.
2. **Ledger.** Es la "base de datos" de la red. En ella se almacena la información de los "assets" o activos que se manejan, así como una cadena de bloques, conteniendo cada uno de ellos las transacciones que modificaron dichos activos. Estos activos pueden representar bienes tangibles, como vehículos, teléfonos, viviendas, etc., o bienes más abstractos como contratos de propiedad, testamentos, etc.
   Así, las transacciones realizadas en este tipo de redes son modificaciones sobre estos bienes, las cuales se van aplicando y almacenando de manera agrupada en bloques encadenados de manera secuencial, de ahí el origen del término "blockchain". El llamado "bloque génesis" corresponde con el primer bloque generado al crear la red para iniciar la cadena de bloques.

3. **Chaincodes/smart contracts.** Aunque con ligeras diferencias, son los conocidos como smart contracts en redes como las de Ethereum. Son programas informáticos que permiten definir lógica de negocio que luego podrá ser ejecutada por las aplicaciones que se conecten a la red. Se instancian dentro de un contenedor aislado en los peers como medida de seguridad para prevenir cambios indeseados en el ledger. Pueden estar escritos en lenguajes reconocidos como JavaScript, Java o Go.
4. **Ordering Service.** Es el servicio de ordenación y empaquetamiento de transacciones en bloques de Fabric. Recibe la ejecución de las transacciones de los peers y las organiza y ordena en bloques que, una vez definidos, permanecen inmutables.
   Al ser modular, Fabric permite decidir qué tipo de algoritmo de consenso utilizar según las necesidades de la red. Así, para pruebas de concepto y testing de los smart contracts suele utilizarse el algoritmo de consenso *Solo*¸ compuesto por un único nodo que realiza la ordenación de las transacciones. En entornos productivos, Fabric aconseja el uso de los algoritmos de ordenación *Raft* o *Kafka* pues, al estar implementados por varios nodos, toleran la caída de uno o varios de éstos sin que el servicio se vea afectado.
5. **Membership Service Provider.** Este módulo es el responsable de asociar las entidades que participan en la red con identidades digitales. Tanto los clientes, como los peers o ordering service nodes están identificados mediante certificados digitales. En su implementación, Fabric provee de una CA propia llamada *Fabric-CA* para emitir los certificados de cada uno de los componentes de la red.
6. **Canal.** Un canal es una "subred" privada de comunicación entre dos o más miembros de una red de Hyperledger Fabric, con el propósito de compartir transacciones de manera confidencial y privada con respecto al resto de miembros que no pertenecen al canal. Un canal está compuesto de miembros (organizaciones), peers, el ordering service y uno o varios smart contracts desplegados en el propio canal para que puedan ser invocados por los miembros.
7. **Base de datos.** Mientras que la cadena de bloques de transacciones del ledger es almacenada en un fichero, el estado actual de los activos u objetos de negocio son almacenados en una base de datos. Por defecto, Hyperledger Fabric utiliza LevelDB, ideal para objetos simples de tipo clave-valor. Para objetos más complejos estructurados en formato JSON, Hyperledger Fabric también puede ser configurado para usar CouchDB, que soporta consultas para obtener estos objetos más complejas.
8. **Políticas de validación.** Fabric utiliza distintas políticas para validar quién se puede conectar al canal, quién puede invocar a un smart contract, quién puede acceder a datos sensibles, etc.

Las *endorsement policies,* por ejemplo, permiten definir qué peers han de acordar los resultados de una transacción (i.e. la ejecución de un smart contract) antes de que pueda ser añadida al ledger.

Veamos un ejemplo de la arquitectura de una red de Hyperledger Fabric:

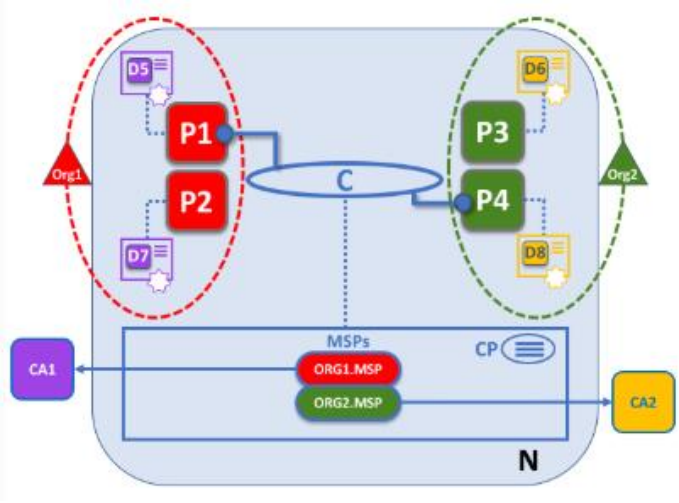

Figura 3. Red compuesta por dos organizaciones, participando cada una con dos nodos

En el esquema podemos ver dos organizaciones, *Org1* y *Org2*, pertenecientes a un único canal *C*, que aportan a la red dos peers cada una (*P1-P2* de Org1 y *P3-P4* de Org2), cada uno de ellos con su identidad emitidas por las CA *CA1* y *CA2*. Estas CA están registradas en el módulo MSP (Membership Service Provider), lo que permite reconocer y asignar el rol dentro de la red a cada uno de los componentes.

Como vemos en el esquema simplificado de la red, ésta no tiene componentes centralizados. Los recursos de la red son contribuciones individuales de cada una de las organizaciones que colabora en el mantenimiento de la red. Esta filosofía es la que permite que la red siga funcionando aunque una de las organizaciones deje de contribuir o participar en la red.

INTERACCIÓN CON EL LEDGER. LOS SMART CONTRACTS

En general, cuando dos o más partes quieren realizar algún tipo de transacción, previamente han de definir cuáles van a ser las reglas y limitaciones de dicho intercambio. En el caso de la portabilidad de un cliente, la empresa receptora y donante deberán acordar la fecha y hora del cambio, los motivos por el cual se puede rechazar una portabilidad, los datos necesarios para realizar la transacción, etc. A toda esta información conocida por ambas partes se la conoce como **lógica de negocio.** En Fabric,

toda esta lógica podemos convertirla en programas informáticos distribuidos capaces de ser ejecutados por aplicaciones propiedad de cada una de las organizaciones que intervienen en la red. Son los llamados *smart contracts*, los cuales son almacenados y ejecutados en los peers cuando una aplicación o usuario se conecta a ellos para interaccionar con el ledger. La ejecución exitosa de un smart contract deriva en una transacción y en una actualización del ledger, tal y como veremos más adelante.

Un peer puede almacenar una o varias instancias de ledgers y de smart contracts, puesto que es posible que un peer pertenezca a uno o más canales, con el fin de poder realizar transacciones de manera privada entre miembros de la red. A su vez, un smart contract puede estar desplegado en varios peers. Dependiendo de las políticas de configuración, la transacción fruto de la ejecución de un smart contract puede exigir que varios peers den validez a su ejecución, entendiendo como validez la firma de la transacción.

Grosso modo, los pasos de la ejecución de un smart contract hasta que se actualiza el ledger son los siguientes:

1. Una aplicación cliente A1, se conecta al peer P1.
2. A continuación, A1 realiza una propuesta de transacción, *proposal,* una funcionalidad del smart contract S1 almacenado en P1 que, si todo va bien, actualizará el ledger.
   2.1. P1 invoca a S1 con el proposal enviado por A1.
   2.2. S1 genera la respuesta de propuesta de actualización. Si S1 está configurado para que varios peers validen la propuesta (endorsement policies), todos ellos recibirán la propuesta y generarán la respuesta de propuesta de actualización. A estos peers se les conoce como *endorsing peers.*
3. P1 (y todos los endorsing peers) devuelve a A1 la respuesta propuesta de actualización. En este punto, aún no se ha actualizado el ledger.
4. La aplicación A1 envía una respuesta de transacción a un nodo del *ordering service.* Este servicio de Hyperledger Fabric crea bloques de transacciones, los cuales serán distribuidos a todos los peers que contienen una instancia del ledger para que validen el bloque y lo añadan a la cadena.
5. Una vez P1 ha aplicado el bloque de transacciones al ledger, notifica de manera asíncrona a A1 que la actualización se ha realizado con éxito.

ANONIMATO Y PRIVACIDAD

Pese a ser una red permisionada en la cual los participantes se reconocen mutuamente, hay ocasiones en las que se requiere que no se conozca la identidad del emisor de una transacción, pero garantizando que es uno de los miembros permitidos de la red. Por otro lado, hay casos de negocio que requieren que solo las partes involucradas directamente en una transacción sean las que puedan consultar ciertos datos relacionados con dicha transacción, impidiendo que el resto de los participantes pueda acceder a ellos.

Para otorgar anonimato a los participantes, Hyperledger Fabric provee de una implementación denominada "Idemix" que permite la autenticación de usuarios manteniendo su anonimato, pudiendo realizar transacciones sin revelar su identidad. Para ello, la CA de Fabric emitirá unas credenciales "de tipo Idemix" al usuario que quiera acceder a la red. Estas credenciales son similares a un certificado de tipo X.509 salvo por el esquema de la firma. Este certificado permite al verificador (MSP) autenticar al usuario sin revelar los atributos de su identidad, aplicando para ello una comprobación de tipo "zero-knowledge proof",

Por su parte, la confidencialidad de los datos pertenecientes a una transacción puede conseguirse en Fabric de dos maneras:

1) **Canales privados.** Aquellas partes que quieran realizar transacciones privadas pueden habilitar un canal privado entre ellos dentro de la red, impidiendo que el resto de los participantes tenga constancia de dichas transacciones.
2) **Colecciones privadas de datos.** Existe la posibilidad de crear colecciones privadas de datos que solo ciertas organizaciones puedan consultar. Estos datos son almacenados en una parte de la base de datos de los peers de aquellas organizaciones que tienen permitida su consulta. Por contra, en los ledgers de todos los peers del canal se distribuirá un hash de la colección privada, el cual sirve de evidencia de la transacción y es usado para la validación de ésta.

La decisión de usar canales privados o colecciones privadas de datos (o ambas) depende de cada escenario. Se usarán canales privados cuando toda la transacción deba ser confidencial, mientras que las colecciones privadas se utilizarán cuando el resto de los participantes pueda consultar la transacción, pero no algunos de los datos que incluye.

6. DISEÑO DE LA PRUEBA DE CONCEPTO (PoC)

Con el fin de solventar los problemas de arquitectura del proceso de portabilidad en España, vamos a realizar una prueba de concepto en la que diseñaremos una red con Hyperledger Fabric para poder tramitar solicitudes de portabilidad entre operadoras de telefonía. Así, una operadora receptora podrá realizar una solicitud de portabilidad de una línea telefónica que pertenezca en ese momento a otra operadora. La operadora donante será consciente de la solicitud de portabilidad y deberá decidir si acepta o rechaza la portabilidad. En caso de ser aceptada, la operadora receptora deberá ejecutar el cambio y notificar al resto de operadoras.

6.1. MODELO Y ENTIDADES

En el escenario de nuestra prueba existirán los siguientes actores como participantes en la red (ver Figura 5):

OPERADORA

Las operadoras estarán identificadas por un identificador único y una descripción. Serán las encargadas de realizar las solicitudes de portabilidad, así como de aceptarlas o rechazarlas. Serán dadas de alta en el sistema por un administrador de la red de Hyperledger Fabric. Consultarán a la red el listado de portabilidades receptoras y donantes que tengan pendientes para tramitarlas.

AOP

Es una entidad transversal al proceso. No intervendrá en las portabilidades que se den entre operadoras, pero tendrá capacidad de acceder al estado de las portabilidades y la numeración, con el fin de obtener informes y revisar que no se están cometiendo irregularidades malintencionadas por parte de las operadoras como, por ejemplo, el rechazo continuado de solicitudes de portabilidad.

El siguiente esquema muestra el modelo de entidades de la PoC:

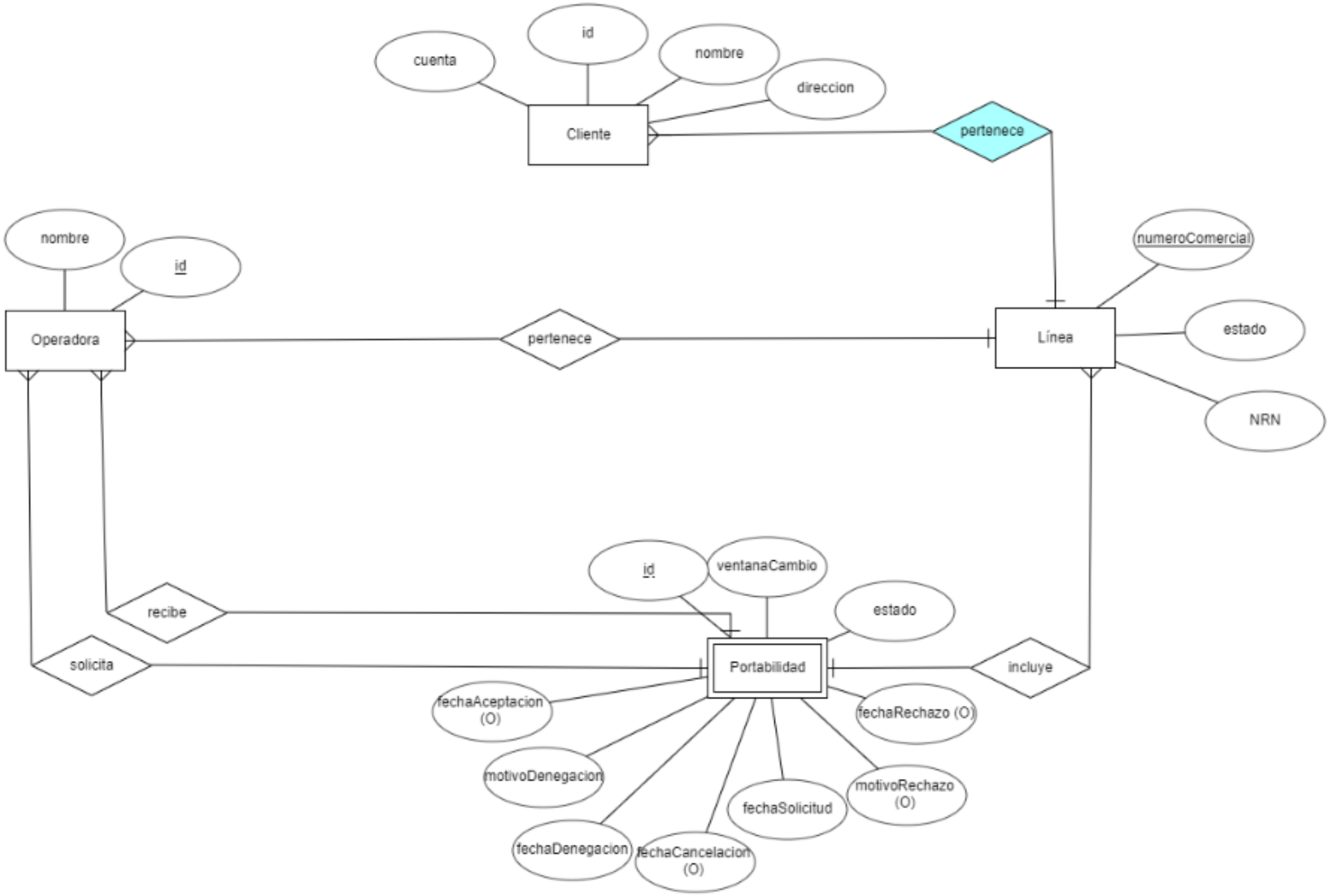

Figura 4. Operadoras, líneas y portabilidades son las entidades que conforman el modelo de la PoC.

En el modelo E-R podemos ver las siguientes entidades:

- **Operadora.** Contiene información acerca de una operadora. Para la prueba de concepto, únicamente tendremos el nombre de la operadora como atributo de ésta. Su identificador es un código de dos cifras prefijado por un administrador de la red.
- **Línea.** Representa el concepto de línea telefónica según el punto de vista de una operadora.
    - Se identificará por el número comercial de 9 dígitos. Los números de teléfono son asignados por "rangos" a las operadoras, los cuales son conocidos por todas ellas. Así, cuando un usuario realiza una llamada a un destino perteneciente a su misma operadora, la centralita origen sabe que el destinatario corresponde a su rango, por lo que enruta la llamada a la centralita correspondiente. En caso de que el número destino no pertenezca al rango asignado a la operadora origen, la centralita enrutará la llamada a aquella que conecta con la operadora propietaria del número destino.
      En caso de un número portado, la operadora que inicia la llamada no se fijará en el número comercial y sí en el campo NRN, que indicará dónde redirigir la llamada.
    - Contendrá el identificador del titular de la línea (DNI, CIF…)
    - En el campo NRN se almacenará información del enrutamiento para poder encaminar las llamadas entre centralitas. Es un campo cumplimentado por la operadora receptora cuando se realiza la portabilidad de un número. Está formado por un código de 6 dígitos, donde los dos primeros indican el identificador del operador que da servicio a la línea. Los siguientes dos dígitos indican la provincia, y los dos últimos quedan libres para organización interna del operador receptor.
      Si el campo no está cumplimentado el enrutamiento se realiza en base al rango del número comercial.
- **Portabilidad.** Representa la solicitud del cambio de número de teléfono de una operadora a otra. Los campos más destacados son:
    - Su campo "estado" tomará valores similares a los tipos de mensajes enviados en el proceso de portabilidad tradicional:
        - "SP" indica que la portabilidad se ha solicitado por parte de la operadora receptora. Es el estado inicial de cualquier portabilidad.
        - "ASP" indica que la portabilidad ha sido aceptada por la operadora donante

- "DSP" indica que la portabilidad se ha rechazado por parte de la operadora donante. Es un estado final de la portabilidad.
- "CNP" indica que la portabilidad ha sido cancelada a petición de la operadora donante. Es un estado final de la portabilidad.
- "PR" indica que la portabilidad ha sido realizada. Es un estado final de la portabilidad.
- "ER" indica que ha sucedido alguna anomalía con la portabilidad y ha de ser revisada por las operadoras.

- Dispone de campos para indicar la fecha de solicitud de la portabilidad, la fecha del rechazo de la operadora donante si la hubiere, fecha de cancelación si la hubiere, etc.

- **Cliente.** Modela la información del cliente, propietario de una o varias líneas telefónicas. Almacena información de carácter personal, como el documento de identidad, su nombre completo, la dirección y el número de cuenta. Esta información sensible solo podrá ser consultada por las operadoras, nunca por la AOP.

Dado que, por cuestiones de tiempo, no podemos adaptar la PoC a los intervalos de tiempo reales que maneja una portabilidad, superiores a 24h, vamos a reducir la escala de tiempo de tal manera que cada minuto real representará un día en la PoC, coincidiendo el inicio y el fin de un nuevo día de portabilidad con el inicio y el fin de un nuevo minuto real. La figura 6 representa dos días de la PoC y su equivalencia en minutos reales:

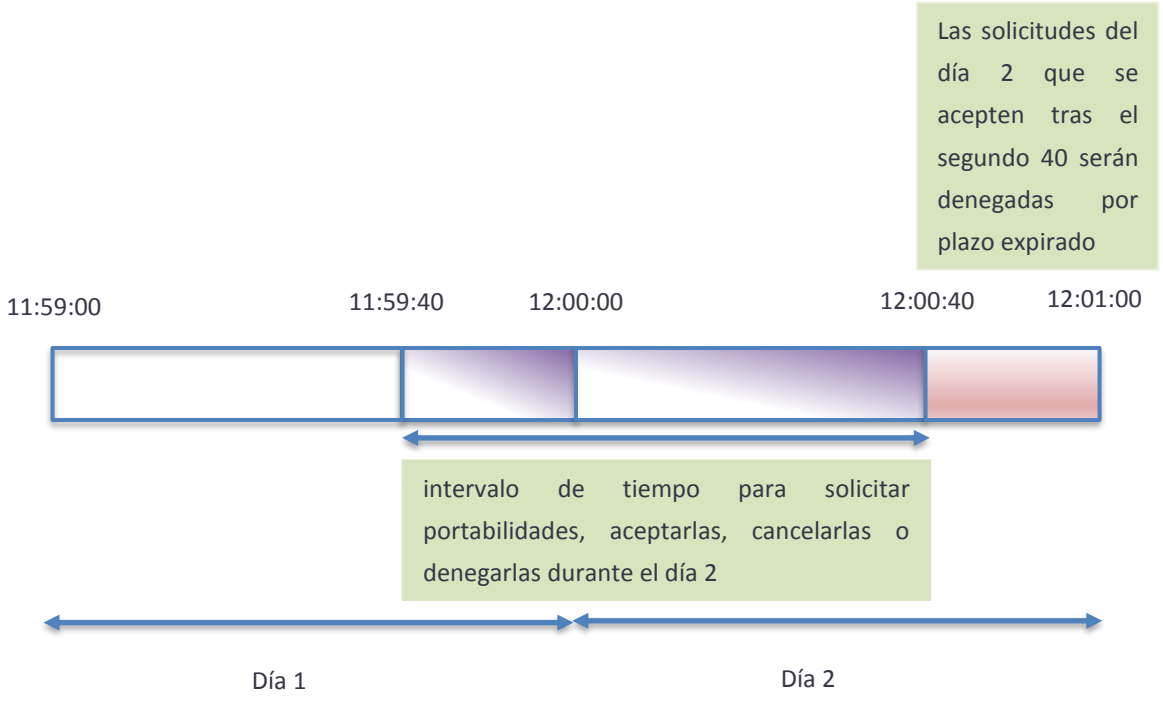

Figura 5. Ventana de portabilidad definida para la PoC

Con esta escala, el proceso de portabilidad se deberá llevar a cabo en días de 60''. Las condiciones serán las siguientes:

- Una operadora receptora podrá realizar una solicitud de portabilidad sobre una línea de otro operador. Se deberá validar en la propia red que no se solicita una portabilidad de una línea perteneciente a la operadora receptora ni que se encuentre en otro proceso de portabilidad. Esta solicitud deberá llegar antes del segundo 40 de cada minuto.
- Aquellas portabilidades que se soliciten posterior al segundo 40 se tratarán como si la petición hubiese sido realizada en el siguiente minuto.
- La operadora donante aceptará o denegará solicitudes de portabilidad. Tendrá hasta el segundo 40 para aceptar o denegar las solicitudes de ese día. Para ello, deberá consultar periódicamente si hay alguna portabilidad en vuelo en la que participe como donante y decidir qué hacer con ella. Si se acepta o deniega una solicitud del día más allá del segundo 40 dicha solicitud se denegará por tiempo expirado.
- La operadora receptora podrá cancelar la portabilidad antes del segundo 40 de ese día. Pasado ese momento, la cancelación no será posible y se marcará como denegada por tiempo expirado.
- Una vez se ha aceptado la solicitud de portabilidad, tanto la operadora donante como la receptora progresarán la portabilidad, realizando la baja y el alta pertinente de la línea en cada uno de sus parques.
- La AOP podrá consultar en cualquier momento el estado de las portabilidades, pero no podrá acceder a los datos sensibles, como pudieran ser el identificador del titular y su nombre.

Para la PoC se han unificado el momento máximo de aceptación/denegación de solicitudes de portabilidad y el momento máximo de cancelación, ofreciendo a las operadoras donantes más tiempo para realizar ofertas de retención a los clientes que han solicitado la portabilidad. En un entorno real, sería sencillo establecer las mismas limitaciones horarias que tiene actualmente el proceso de portabilidad.

6.2. TOPOLOGÍA DE LA RED

El siguiente esquema representa la arquitectura de red que deberá soportar la PoC.

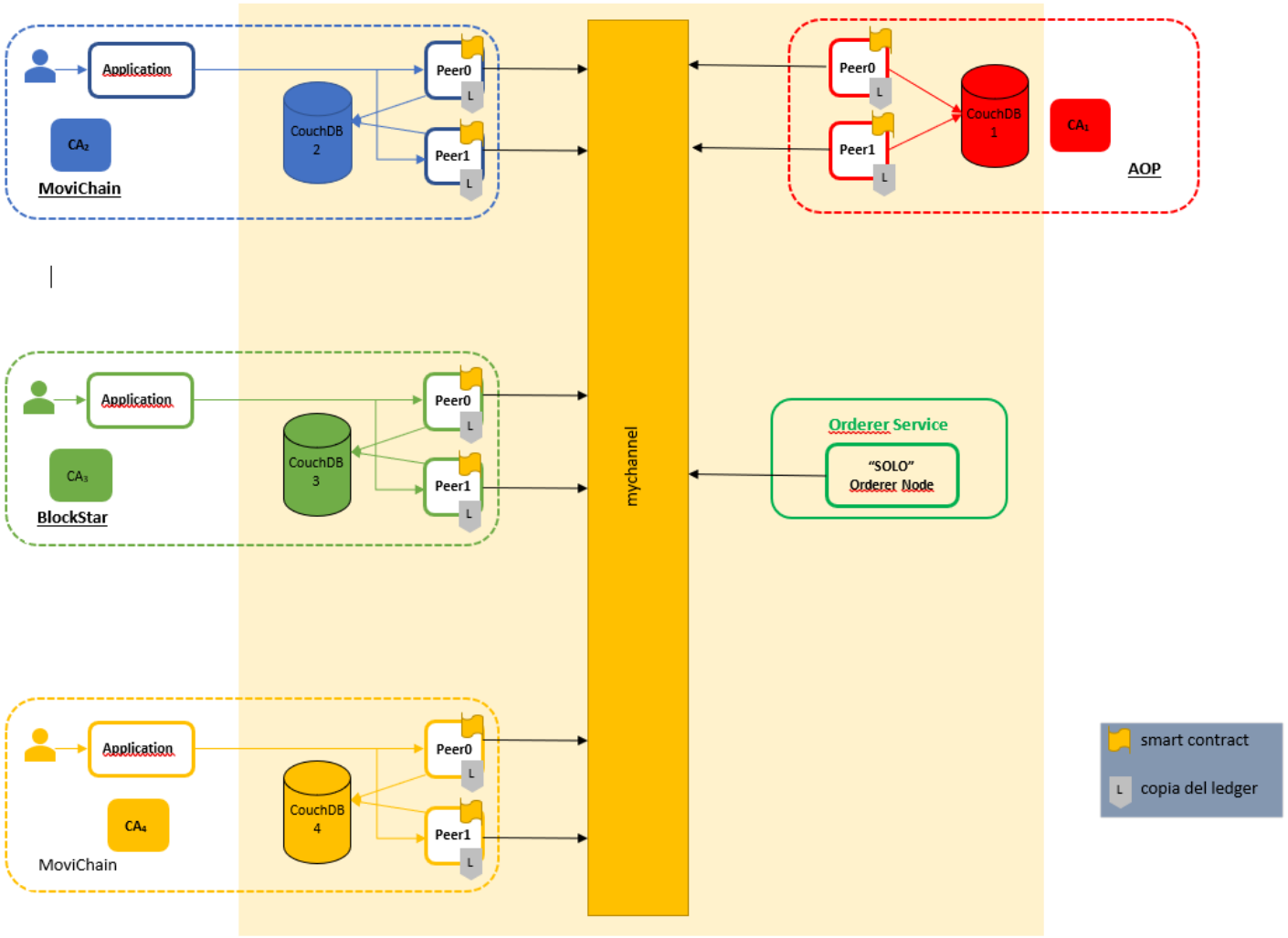

Figura 6. Actores, componentes de red y relaciones de la PoC

La red estará formada por tres operadoras: *MoviChain, BlockStar y MasLedger*. También participará en la red AOP, que llevará a cabo la inicialización de la red, pero en ningún caso hará ninguna transacción, solo consultas al ledger para ver el estado de las portabilidades.

Cada operadora aportará a la red dos peers, *peer0* y *peer1,* los cuales albergarán una copia del ledger. Además, tendrán instalado un smart contract denominado "*portabilityManagement",* que contendrá las validaciones y lógica de negocio del proceso de portabilidad. Estos peers se comunicarán a través del canal *"mychannel"*, al cual se conectarán con la identidad que les proveerá la CA correspondiente de la operadora a la que pertenecen.

Cada operadora tendrá una instancia de base de datos CouchDB en el que se almacenarán los datos de las portabilidades, líneas y clientes, a las cuales se conectará el smart contract para devolver los datos solicitados por las aplicaciones de usuario que deseen interactuar con la red.

Con respecto al Ordering Service, estará formado por un único nodo, conectado a la red cuyo algoritmo de consenso implementado será "solo".

6.3. CONFIDENCIALIDAD Y PRIVACIDAD

Pese a ser una red permisionada, habrá algunos datos que no deberán poder ser consultados por algunos de los participantes. En concreto, la AOP podrá consultar el estado de las portabilidades, pero no podrá conocer datos sensibles del titular de la línea, tales como su nombre completo, dirección o número de cuenta.

Para cumplir este requisito haremos uso de las colecciones privadas de datos de Fabric. En este caso no es posible usar canales privados, pues necesitaríamos un canal privado por cada par de operadoras. Además de necesitar (n·(n-1)) /2 canales privados, siendo n el número de operadoras, los canales privados impedirían a las operadoras que no participan en una portabilidad consultar datos relevantes como sería el caso del NRN.

Debido a esto, crearemos una colección de datos privados y la configuración necesaria para autorizar a las tres operadoras a consultar estos datos sensibles desde sus respectivas aplicaciones. Estos datos serán almacenados en un espacio alternativo de las bases de datos de los peers, y no viajarán al ordering service de una manera "visible", sino en formato "hash", con el fin de garantizar que ningún nodo de la red no autorizado pueda consultarlos.

7. IMPLEMENTACIÓN Y DESPLIEGUE

Para implementar la PoC haremos uso de las herramientas facilitadas por Hyperledger para emitir certificados, creación del bloque génesis, transacciones para unir los peers a la red, etc. En cuanto a la estructura física de la red, haremos uso de las imágenes de Docker publicadas por Hyperledger para crear instancias de peers, CAs, ordering service nodes, etc.

7.1. PREPARACIÓN DEL ENTORNO

Tanto la red de Fabric como las aplicaciones que se conectan a la red se ejecutarán dentro de la misma máquina. Siguiendo las recomendaciones de Fabric, instalamos una máquina con Ubuntu 18.04, sobre la cual instalaremos distintas herramientas software que son necesarias para la ejecución correcta de todas las partes de la PoC. A continuación, listamos las dependencias necesarias, su utilidad y los comandos y configuraciones a lanzar para su instalación y configuración:

- **curl**
  Nos permite el lanzamiento de peticiones HTTP desde consola.
- **docker**
  Permite el despliegue de aplicaciones dentro de contenedores de software predefinidos.
- **docker-compose**
  Es una extensión de docker con la cual podemos definir, configurar y ejecutar múltiples contenedores a partir de un fichero de configuración.
- **golang-go**
  Lenguaje de programación concurrente inspirado en la sintaxis de C.
- **nodejs**
  Servidor de aplicaciones escrito en JavaScript.
- **Python**
  Popular lenguaje de programación en entorno Unix.
- **Fabric samples**
  Binarios y ejemplos de Hyperledger Fabric para facilitar el diseño e implementación de nuestra red.

La ejecución del siguiente script, diseñado para esta PoC, realiza la instalación y configuración de cada una de las dependencias anteriores, dejando la máquina preparada para ejecutar una red de Hyperledger Fabric. Para ello es necesario que la máquina virtual tenga acceso a Internet, ya que se descargarán recursos y programas que son necesarios para ejecutar la red. También descargará los binarios y los ejemplos de Hyperledger Fabric:

```
#curl
sudo apt install curl
#docker
sudo apt-get update
sudo apt-get remove docker docker-engine docker-io
sudo apt install docker.io
sudo apt-get install apt-transport-https ca-certificates software-properties-common
#docker-compose
sudo curl -L "https://github.com/docker/compose/releases/download/1.24.1/docker-compose-
$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
sudo usermod -aG docker $USER
#golang-go
wget https://dl.google.com/go/go1.13.1.linux-amd64.tar.gz
tar -xvf go1.13.1.linux-amd64.tar.gz
sudo mv go /usr/local/
echo export GO_HOME=/usr/local/go >> $HOME/.profile
echo export GOPATH=$HOME/hyperledger >> $HOME/.profile
echo export PATH=$GOPATH/bin:$GO_HOME/bin:$PATH >> $HOME/.profile
source $HOME/.profile
#nodejs
wget https://nodejs.org/dist/v10.16.3/node-v10.16.3-linux-x64.tar.xz
tar -xvf node-v10.16.3-linux-x64.tar.xz
sudo mv node-v10.16.3-linux-x64 /usr/local/node
echo export NODE_HOME=/usr/local/node >> $HOME/.profile
echo export PATH=$NODE_HOME/bin:$PATH>> $HOME/.profile
source $HOME/.profile
sudo apt-install build-essential
#python
sudo apt install python
#hyperledger binaries and samples
mkdir hyperledger
cd hyperledger
curl -sSL http://bit.ly/2ysbOFE | bash -s
```

Una vez ejecutado el script, si no ocurrió ningún problema, el entorno estará listo para empezar a configurar y levantar la red de Hyperledger Fabric.

7.2. IDENTIFICACIÓN DE LOS COMPONENTES DE LA RED

Con el entorno ya creado, pasamos a definir de manera más concreta cada una de las partes de la red definida en la Figura 6 del apartado anterior, así como las características de ésta:

- **MSP.** Definiremos cuatro Membership Provider. Tres de ellos, MoviChain, BlockStar y MasLedger corresponderán a las operadoras que participan en la red. AOP, por su parte, representará a la entidad de referencia de los procesos de portabilidad actual. Cada uno de ellos dispondrá de una CA para la emisión y validación de certificados a utilizar en la red.
- **Peers.** Para cada una de las operadoras definiremos dos peers (peer0, peer1) para participar en la red, a los cuales se conectarán las aplicaciones de cada operadora para interactuar con el ledger.
- **CAs.** Crearemos una instancia de una autoridad de certificación para cada organización, con el fin de poder crear y validar usuarios que se conecten a la red. Para ello haremos uso de una de las imágenes de Docker provistas por Hyperledger para tal fin.
- **Orderer.** El ordering service de la red realizará la ordenación de las transacciones según la implementación indicada, mantendrá el listado de organizaciones que pueden crear canales y acceder a ellos, aplicará políticas de seguridad en cuanto a la modificación del ledger, etc. Para la PoC se utilizará la implementación "Solo", la cual estará soportada por un único nodo.
- **Canales.** Las operadoras realizarán sus comunicaciones y transacciones a través de un canal dentro de la red. En el caso de la portability-network, existirá un único canal denominado *mychannel*.
- ***portabilityManagement*.** Este será el nombre del smart contract que contendrá la lógica de negocio de nuestro proceso de portabilidad. Estará escrito en JavaScript y desplegado y configurado en cada uno de los nodos.

7.3. CREACIÓN DE MATERIAL CRIPTOGRÁFICO Y ARTEFACTOS DE LA RED

Para poder montar la red necesitamos que cada uno de los nodos que la forman estén debidamente identificados y reconocidos dentro ella. Por definición, todas las comunicaciones que se realizan dentro de Fabric se hacen de manera segura, utilizando para ello certificados digitales de tipo X.509. Fabric nos ofrece una herramienta llamada *cryptogen,* que generará todos los certificados necesarios a partir de un fichero *crypto-config.yaml,* en el cual identificaremos las organizaciones y los miembros que forman parte de la red portability-network:

```
OrdererOrgs:
  # ---------------------------------------------------------------------------
  # Orderer
  # ---------------------------------------------------------------------------
  - Name: Orderer
    Domain: example.com
    Specs:
      - Hostname: orderer
# ---------------------------------------------------------------------------
# "PeerOrgs" - Definition of organizations managing peer nodes
# ---------------------------------------------------------------------------
PeerOrgs:
  - Name: AOP
    Domain: aop.com
    Template:
      Count: 2
    Users:
      Count: 1
  - Name: MoviChain
    Domain: movichain.com
    Template:
      Count: 2
    Users:
      Count: 1
  - Name: BlockStar
    Domain: blockstar.com
    Template:
      Count: 2
    Users:
      Count: 1
  - Name: MasLedger
    Domain: masledger.com
    Template:
      Count: 2
    Users:
      Count: 1
```

En el fichero hemos indicado que existirá un ordering service node y cuatro organizaciones, haciendo referencia al dominio de cada una de ellas y el número de peers (Template Count) que tendrá cada organización. Además, indicamos que queremos 1 certificado de usuario para cada organización, que será el administrador. Lanzando el siguiente comando, se generarán los certificados necesarios para los componentes indicados en el fichero de configuración:

```
cryptogen generate --config=./crypto-config.yaml
```

La salida de la ejecución anterior genera una estructura de directorios para cada organización y componente en la que encontraremos los certificados necesarios para la interconexión de la red. En el siguiente apartado veremos cómo hacer referencia a los certificados generados.

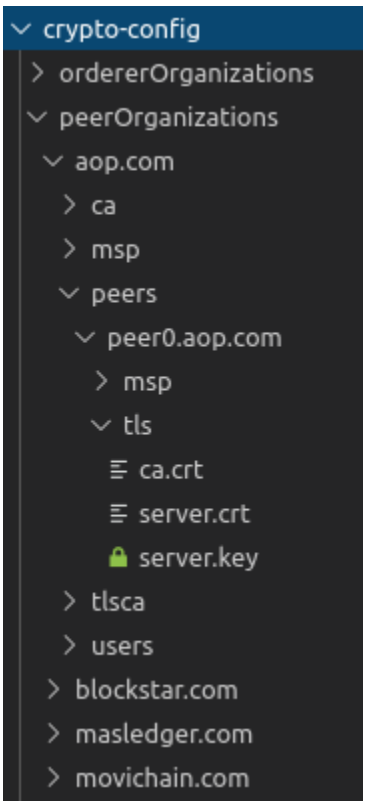

Figura 7. Directorios generados por cryptogen

Con los certificados ya generados pasamos a la creación del bloque génesis, del canal de comunicación, y de los *anchor peers,* que serán los nodos de comunicación de la red por organización. Para poder crear estos artefactos debemos definir un fichero *configtx.yaml* en el cual detallaremos a alto nivel las organizaciones (MSPs) que forman parte de la red, la configuración del Ordering Service, las políticas que aplicarán para las aplicaciones y el canal, etc. Vamos a revisar cada sección del fichero, resaltando la información más importante.

Organizations

En esta sección se define cada una de las entidades que forma parte de la red. Para cada entidad estableceremos su nombre, la localización de su identificación como miembro de la red (MSP) y las políticas de lectura y escritura en la red. La configuración para las organizaciones AOP y BlockStar sería la siguiente:

```
Organizations:
- &AOP
    # DefaultOrg defines the organization which is used in the sampleconfig
    # of the fabric.git development environment
    Name: AOPMSP
    # ID to load the MSP definition as
    ID: AOPMSP
    MSPDir: crypto-config/peerOrganizations/aop.com/msp
    Policies:
      Readers:
        Type: Signature
        Rule: "OR('AOPMSP.admin', 'AOPMSP.peer', 'AOPMSP.client')"
      Writers:
        Type: Signature
        Rule: "OR('AOPMSP.admin', 'AOPMSP.client')"
      Admins:
        Type: Signature
        Rule: "OR('AOPMSP.admin')"
    AnchorPeers:
      # AnchorPeers defines the location of peers which can be used
      # for cross org gossip communication.  Note, this value is only
      # encoded in the genesis block in the Application section context
      - Host: peer0.aop.com
        Port: 11051

- &MoviChain
    # DefaultOrg defines the organization which is used in the sampleconfig
    # of the fabric.git development environment
    Name: MoviChainMSP
    # ID to load the MSP definition as
    ID: MoviChainMSP
    MSPDir: crypto-config/peerOrganizations/movichain.com/msp

    # Policies defines the set of policies at this level of the config tree
    # For organization policies, their canonical path is usually
    #   /Channel/<Application|Orderer>/<OrgName>/<PolicyName>
    Policies:
      Readers:
        Type: Signature
        Rule: "OR('MoviChainMSP.admin', 'MoviChainMSP.peer', 'MoviChainMSP.client')"
      Writers:
        Type: Signature
        Rule: "OR('MoviChainMSP.admin', 'MoviChainMSP.client')"
      Admins:
        Type: Signature
        Rule: "OR('MoviChainMSP.admin')"
    AnchorPeers:
      # AnchorPeers defines the location of peers which can be used
      # for cross org gossip communication.  Note, this value is only
      # encoded in the genesis block in the Application section context
      - Host: peer0.movichain.com
        Port: 13051
```

Para cada una de las organizaciones definimos su identificador MSP, así como la ruta en la que se pueden encontrar los artefactos criptográficos generados con la herramienta cryptogen. Indicaremos también el anchor peer o nodo de comunicación para cada organización dentro del canal. Por último, estableceremos las políticas de validación a aplicar. Estas políticas determinan quién ha de firmar una petición para que ésta sea válida. Hemos definido para cada organización una política "Readers" (cualquier miembro de la organización), "Writers" (válida si el miembro sobre el que se aplica la validación es un administrador o un cliente), y "Admins" (válida si el miembro es administrador). La herramienta *configtxgen*, que consume este fichero configtx.yaml, genera por defecto unas políticas de validación, aunque en nuestro caso hemos definido las nuestras propias.

Orderer

En este apartado configuraremos el Ordering Service. Como ya se ha mencionado, será la implementación del algoritmo "Solo" la que se utilizará para la PoC. Estableceremos un tiempo máximo de bloque de transacciones de 2s, un número máximo de transacciones por bloque de 10 y un tamaño máximo de bloque de 99MB.

```
Orderer: &OrdererDefaults
  # Orderer Type: The orderer implementation to start
  # Available types are "solo","kafka"  and "etcdraft"
  OrdererType: solo
  Addresses:
    - orderer.example.com:7050
  # Batch Timeout: The amount of time to wait before creating a batch
  BatchTimeout: 2s
  # Batch Size: Controls the number of messages batched into a block
  BatchSize:
    # Max Message Count: The maximum number of messages to permit in a batch
    MaxMessageCount: 10

    # Absolute Max Bytes: The absolute maximum number of bytes allowed for
    # the serialized messages in a batch.
    AbsoluteMaxBytes: 99 MB
    # Preferred Max Bytes: The preferred maximum number of bytes allowed for
    # the serialized messages in a batch. A message larger than the preferred
    # max bytes will result in a batch larger than preferred max bytes.
    PreferredMaxBytes: 512 KB
```

Channel

En esta sección del fichero podremos definir el conjunto de políticas que aplica sobre un canal:

```
Channel: &ChannelDefaults
  # Policies defines the set of policies at this level of the config tree
  # For Channel policies, their canonical path is
  #   /Channel/<PolicyName>
  Policies:
    # Who may invoke the 'Deliver' API
    Readers:
      Type: ImplicitMeta
      Rule: "ANY Readers"
    # Who may invoke the 'Broadcast' API
    Writers:
      Type: ImplicitMeta
      Rule: "ANY Writers"
    # By default, who may modify elements at this config level
    Admins:
      Type: ImplicitMeta
      Rule: "MAJORITY Admins"
```

Utilizando los tipos de reglas “ImplicitMeta” definimos que las políticas “Readers” “Writers” son válidas si se cumple cualquier política del mismo nombre, i.e., aquellas que hemos definido previamente para cada organización. Para el caso de la política "Admins", bastará con que se cumplan la mayoría de políticas con este nombre. Para ampliar información acerca de las políticas de validación de Fabric puede consultarse la documentación oficial [9].

Perfiles

Los perfiles nos permiten relacionar la configuración anterior con el fin de generar distintos artefactos. Hemos definido dos perfiles en este caso:

- **OrgsOrdererGenesis:** utilizado para la creación del bloque génesis. Relaciona las políticas de validación del canal definidas y la configuración del Ordering Service con el consorcio de empresas definido “SampleConsortium”, formado por las cuatro organizaciones definidas en el mismo fichero.
- **OrgsChannel:** es utilizado para la creación del canal y las transacciones de los anchor peers. Hace referencia a las políticas de validación del canal y de las organizaciones definidas.

```
Profiles:
  OrgsOrdererGenesis:
    <<: *ChannelDefaults
    Orderer:
      <<: *OrdererDefaults
      Organizations:
        - *OrdererOrg
      Capabilities:
        <<: *OrdererCapabilities
    Consortiums:
      SampleConsortium:
        Organizations:
          - *AOP
          - *MoviChain
          - *BlockStar
          - *MasLedger
  OrgsChannel:
    Consortium: SampleConsortium
    <<: *ChannelDefaults
      Organizations:
        - *AOP
        - *MoviChain
        - *BlockStar
        - *MasLedger
```

Con el fichero configtx.yaml bien definido lanzamos los comandos necesarios para crear el bloque génesis primero.

```
configtxgen -profile OrgsOrdererGenesis -channelID $SYS_CHANNEL -outputBlock
./config/genesis.block
```

y el canal después

```
configtxgen -profile OrgsChannel -outputCreateChannelTx ./config/channel.tx -channelID
$CHANNEL_NAME
```

Las variables $SYS_CHANNEL y $CHANNEL_NAME indican el nombre del canal, y tienen los valores “portability-network-sys-channel” y “mychannel” respectivamente. Fabric indica en su documentación que han de ser obligatoriamente distintos para uno y otro comando.

Para terminar de generar los artefactos que se desplegarán en la red, generaremos las transacciones para los anchor peers de cada una de las organizaciones con los siguientes comandos:

```
# generate anchor peer transaction
configtxgen -profile OrgsChannel -outputAnchorPeersUpdate ./config/AOPanchors.tx -
channelID mychannel -asOrg AOP

# generate anchor peer transaction
configtxgen -profile OrgsChannel -outputAnchorPeersUpdate ./config/MoviChainAnchors.tx -
channelID mychannel -asOrg MoviChain

# generate anchor peer transaction
configtxgen -profile OrgsChannel -outputAnchorPeersUpdate ./config/BlockStarAnchors.tx -
channelID mychannel -asOrg BlockStar

# generate anchor peer transaction
configtxgen -profile OrgsChannel -outputAnchorPeersUpdate ./config/MasLedgerAnchors.tx -
channelID mychannel -asOrg MasLedger
```

En este caso, el comando configtxgen recibe el nombre del perfil como parámetro, el nombre de fichero de salida que se generará, el nombre del canal y la organización para la cual se está creando el fichero de transacción del anchor peer.

7.4. CONFIGURACIÓN DE LOS ELEMENTOS DE LA RED

En el apartado anterior hemos generado los certificados y ficheros necesarios para construir nuestra red portability-network según el diseño que habíamos determinado. Es el momento ahora de realizar la configuración más física de la red, entendiendo por "física" aquella que corresponde con las máquinas que ejecutarán los procesos necesarios para ejecutar nuestra red de Fabric. Para facilitar esta tarea, Fabric ha publicado en los repositorios de Docker distintas imágenes públicas que, configuradas correctamente, nos permitirán levantar peers, CAs, ordering service nodes, etc. Vamos a ir viendo en los siguientes apartados la configuración a realizar para cada componente físico de la red.

Peer

Para cada organización hemos definido que existirán dos peers, cuyos nombres coincidirán con los certificados generados por la herramienta cryptogen utilizada anteriormente:

- peer0.aop.com
- peer1.aop.com
- peer0.movichain.com
- peer1. movichain.com
- peer0.blockstar.com
- peer1. blockstar.com
- peer0.masledger.com
- peer1.masledger.com

La configuración de los peers está dividida en tres ficheros:

- En el fichero *peer-base.yaml* encontraremos la configuración común para todos los peers.
- En el fichero *docker-compose-base.yaml* se definirá, para cada peer, su configuración particular.
- En el fichero *docker-compose-cli.yaml* se declarará cada peer que será levantado para la red.

A continuación, se muestra un ejemplo de definición y declaración del peer0 de la organización AOP en cada fichero enumerado anteriormente:

```
peer-base:
  image: hyperledger/fabric-peer:$IMAGE_TAG
  environment:
    - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
    # the following setting starts chaincode containers on the same
    # bridge network as the peers
    # https://docs.docker.com/compose/networking/
    - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=${COMPOSE_PROJECT_NAME}_portability-
network
    - CORE_CHAINCODE_LOGGING_SHIM=DEBUG
    #- FABRIC_LOGGING_SPEC=DEBUG
    - CORE_PEER_TLS_ENABLED=true
    - CORE_PEER_GOSSIP_USELEADERELECTION=true
    - CORE_PEER_GOSSIP_ORGLEADER=false
    - CORE_PEER_PROFILE_ENABLED=true
    - CORE_PEER_TLS_CERT_FILE=/etc/hyperledger/fabric/tls/server.crt
    - CORE_PEER_TLS_KEY_FILE=/etc/hyperledger/fabric/tls/server.key
    - CORE_PEER_TLS_ROOTCERT_FILE=/etc/hyperledger/fabric/tls/ca.crt
  working_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer
  command: peer node start --peer-chaincodedev=true
```

Configuración de un peer en el fichero peer-base.yaml

En esta configuración genérica establecemos la configuración TLS, que hará referencia a los certificados generados por la herramienta cryptogen.

```
peer0.aop.com:
  container_name: peer0.aop.com
  extends:
    file: peer-base.yaml
    service: peer-base
  environment:
    - CORE_PEER_ID=peer0.aop.com
    - CORE_PEER_ADDRESS=peer0.aop.com:11051
    - CORE_PEER_LISTENADDRESS=0.0.0.0:11051
    - CORE_PEER_CHAINCODEADDRESS=peer0.aop.com:11052
    - CORE_PEER_CHAINCODELISTENADDRESS=0.0.0.0:11052
    - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer0.aop.com:11051
    - CORE_PEER_GOSSIP_BOOTSTRAP=peer1.aop.com:12051
    - CORE_PEER_LOCALMSPID=AOPMSP
  volumes:
     - /var/run/:/host/var/run/
     - ../crypto-
config/peerOrganizations/aop.com/peers/peer0.aop.com/msp:/etc/hyperledger/fabric/msp
     - ../crypto-
config/peerOrganizations/aop.com/peers/peer0.aop.com/tls:/etc/hyperledger/fabric/tls
     - peer0.aop.com:/var/hyperledger/production
  ports:
    - 11051:11051
```

Configuración de un peer en el fichero docker-compose-base.yaml

Ya en la configuración particular del peer (docker-compose-base.yaml) indicaremos el nombre del peer y su puerto (CORE_PEER_ID, CORE_PEER_ADDRESS), el puerto de escucha de chaincodes (CORE_PEER_CHAINCODELISTENADDRESS), y el MSP al que pertenece (CORE_PEER_LOCALMSPID). En el apartado “volumes”, mapeamos la carpeta generada por cryptogen para este nodo de los certificados MSP y TLS, para que el nodo pueda configurar y utilizar dichos certificados.

```
peer0.aop.com:
  container_name: peer0.aop.com
  extends:
    file:  base/docker-compose-base.yaml
    service: peer0.aop.com
  networks:
    - portability-network
```

Configuración de un peer en el fichero docker-compose-cli.yaml

En el fichero docker-compose-cli.yaml declaramos el peer e indicamos que pertenece a la red portability-network.

CA

Para cada una de las CAs, configuramos, en el fichero *docker-compose-ca.yaml*, TLS e indicamos como variables de entorno de la instancia de docker, la ruta de las claves pública y privada del servidor (FABRIC_CA_SERVER_TLS_CERTFILE y FABRIC_CA_SERVER_TLS_KEYFILE). Estas claves son las que hemos generado previamente con la herramienta cryptogen. También indicamos el puerto de escucha (FABRIC_CA_SERVER_PORT) y mapeamos en la sección "volumes" la carpeta correspondiente generada por cryptogen para la CA.

```
caAOP:
  image: hyperledger/fabric-ca:$IMAGE_TAG
  environment:
    - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
    - FABRIC_CA_SERVER_CA_NAME=ca-aop
    - FABRIC_CA_SERVER_TLS_ENABLED=true
    - FABRIC_CA_SERVER_TLS_CERTFILE=/etc/hyperledger/fabric-ca-server-config/ca.aop.com-
cert.pem
    - FABRIC_CA_SERVER_TLS_KEYFILE=/etc/hyperledger/fabric-ca-server-con-
fig/${PN_CAAOP_PRIVATE_KEY}
    - FABRIC_CA_SERVER_PORT=9054
  ports:
    - "9054:9054"
  command: sh -c 'fabric-ca-server start --ca.certfile /etc/hyperledger/fabric-ca-server-con-
fig/ca.aop.com-cert.pem --ca.keyfile /etc/hyperledger/fabric-ca-server-con-
fig/${PN_CAAOP_PRIVATE_KEY} -b admin:adminpw -d'
  volumes:
    - ./crypto-config/peerOrganizations/aop.com/ca/:/etc/hyperledger/fabric-ca-server-config
  container_name: ca_peerAOP
  networks:
    - portability-network
```

Configuración de una CA en el fichero docker-compose-ca.yaml

Orderer

Al igual que con los peers, definimos una configuración común en el fichero peer-base.yaml:

```
orderer-base:
  image: hyperledger/fabric-orderer:$IMAGE_TAG
  environment:
    - FABRIC_LOGGING_SPEC=INFO
    - ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
    - ORDERER_GENERAL_GENESISMETHOD=file
    - ORDERER_GENERAL_GENESISFILE=/var/hyperledger/orderer/orderer.genesis.block
    - ORDERER_GENERAL_LOCALMSPID=OrdererMSP
    - ORDERER_GENERAL_LOCALMSPDIR=/var/hyperledger/orderer/msp
    # enabled TLS
    - ORDERER_GENERAL_TLS_ENABLED=true
    - ORDERER_GENERAL_TLS_PRIVATEKEY=/var/hyperledger/orderer/tls/server.key
    - ORDERER_GENERAL_TLS_CERTIFICATE=/var/hyperledger/orderer/tls/server.crt
    - ORDERER_GENERAL_TLS_ROOTCAS=[/var/hyperledger/orderer/tls/ca.crt]
    - ORDERER_KAFKA_TOPIC_REPLICATIONFACTOR=1
    - ORDERER_KAFKA_VERBOSE=true
    - ORDERER_GENERAL_CLUSTER_CLIENTCERTIFICATE=/var/hyperledger/orderer/tls/ser-
ver.crt
    - ORDERER_GENERAL_CLUSTER_CLIENTPRIVATEKEY=/var/hyperledger/orderer/tls/ser-
ver.key
    - ORDERER_GENERAL_CLUSTER_ROOTCAS=[/var/hyperledger/orderer/tls/ca.crt]
  working_dir: /opt/gopath/src/github.com/hyperledger/fabric
  command: orderer
```

Configuración base del ordering service node en el fichero peer-base.yaml

En el fichero docker-compose-base.yaml establecemos la configuración particular para nuestro ordering service node, mapeando la carpeta donde se encuentra el bloque génesis, y la identidad digital creada con cryptogen.

```
orderer.example.com:
  container_name: orderer.example.com
  extends:
    file: peer-base.yaml
    service: orderer-base
  volumes:
      - ../channel-artifacts/genesis.block:/var/hyperledger/orderer/orderer.genesis.block
      - ../crypto-
config/ordererOrganizations/example.com/orderers/orderer.example.com/msp:/var/hyperledg
er/orderer/msp
      - ../crypto-
config/ordererOrganizations/example.com/orderers/orderer.example.com/tls/:/var/hyperledge
r/orderer/tls
      - orderer.example.com:/var/hyperledger/production/orderer
  ports:
    - 7050:7050
```

Configuración adicional del ordering service node en el fichero docker-compose-base.yaml

Por último, en el fichero docker-compse-cli.yaml, declaramos el ordering service node:

```
orderer.example.com:
  extends:
   file:   base/docker-compose-base.yaml
   service: orderer.example.com
  container_name: orderer.example.com
  networks:
   - portability-network
```

Configuración adicional del ordering service node en el fichero docker-compose-cli.yaml

Para configurar la instancia del ordering service node, estableceremos variables de entorno para indicar la ruta del bloque génesis, el id dentro de la red (MSPID) y la configuración TLS.

CouchDB

En el fichero docker-compose-couch.yaml definiremos una instancia de base de datos "CouchDB" para cada peer y los asociaremos.

```
couchdbAOP_0:
  container_name: couchdbAOP_0
  image: hyperledger/fabric-couchdb
  # Populate the COUCHDB_USER and COUCHDB_PASSWORD to set an admin user and
password
  # for CouchDB.  This will prevent CouchDB from operating in an "Admin Party" mode.
  environment:
   - COUCHDB_USER=
   - COUCHDB_PASSWORD=
  # Comment/Uncomment the port mapping if you want to hide/expose the CouchDB service,
  # for example map it to utilize Fauxton User Interface in dev environments.
  ports:
   - "9984:5984"
  networks:
   - portability-network

 peer0.aop.com:
  environment:
   - CORE_LEDGER_STATE_STATEDATABASE=CouchDB
   - CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=couchdbAOP_0:5984
   # The CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME and
CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD
   # provide the credentials for ledger to connect to CouchDB.  The username and password
must
   # match the username and password set for the associated CouchDB.
   - CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME=
   - CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD=
  depends_on:
   - couchdbAOP_0
```

Configuración de una de la base de datos de la organización AOP

Cli

Este contenedor no es estrictamente necesario, no participa en la red. "Cli" es una instancia de *fabric-tools* que nos permite realizar modificaciones en la red, ya que viene preconfigurada con las APIs y herramientas de Fabric. En nuestro caso, hemos configurado *cli* con la misma configuración que el nodo peer0.aop.com. Más adelante veremos cómo utilizar *cli* para realizar configuraciones de otros nodos.

```
cli:
  container_name: cli
  image: hyperledger/fabric-tools:$IMAGE_TAG
  tty: true
  stdin_open: true
  environment:
   - SYS_CHANNEL=$SYS_CHANNEL
   - GOPATH=/opt/gopath
   - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
   #- FABRIC_LOGGING_SPEC=DEBUG
   - FABRIC_LOGGING_SPEC=INFO
   - CORE_PEER_ID=cli
   - CORE_PEER_ADDRESS=peer0.aop.com:11051
   - CORE_PEER_LOCALMSPID=AOPMSP
   - CORE_PEER_TLS_ENABLED=true
   - CORE_PEER_TLS_CERT_FILE=/opt/gopath/src/github.com/hyperledger/fa-
bric/peer/crypto/peerOrganizations/aop.com/peers/peer0.aop.com/tls/server.crt
   - CORE_PEER_TLS_KEY_FILE=/opt/gopath/src/github.com/hyperledger/fa-
bric/peer/crypto/peerOrganizations/aop.com/peers/peer0.aop.com/tls/server.key
   - CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fa-
bric/peer/crypto/peerOrganizations/aop.com/peers/peer0.aop.com/tls/ca.crt
   - CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fa-
bric/peer/crypto/peerOrganizations/aop.com/users/Admin@aop.com/msp
  working_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer
  command: /bin/bash
  volumes:
    - /var/run/:/host/var/run/
    - ./chaincode/:/opt/gopath/src/github.com/chaincode
    - ./crypto-config:/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/
    - ./scripts:/opt/gopath/src/github.com/hyperledger/fabric/peer/scripts/
    - ./channel-artifacts:/opt/gopath/src/github.com/hyperledger/fabric/peer/channel-artifacts
  depends_on:
   - orderer.example.com
   - peer0.aop.com
   - peer1.aop.com
   - peer0.movichain.com
   - peer1.movichain.com
   - peer0.blockstar.com
   - peer1.blockstar.com
   - peer0.masledger.com
   - peer1.masledger.com
  networks:
   - portability-network
```

Configuración del nodo auxiliar *cli*

7.5. CONEXIÓN DE COMPONENTES A LA RED

Con todas las instancias de las máquinas de red configuradas, es el momento de levantar nuestra portability-network. Lo primero que haremos será arrancar todas las instancias configuradas en los ficheros de docker:

```
export COMPOSE_FILE=docker-compose-cli.yaml
export COMPOSE_FILE_CA=docker-compose-ca.yaml
export COMPOSE_FILE_COUCH=Docker-compose-couch.yaml
export COMPOSE_FILES=”-f ${COMPOSE_FILE -f COMPOSE_FILE_CA -f COMPOSE_FILE_COUCH}
docker-compose ${COMPOSE_FILES} up -d 2>&1
```

En este momento los peers están levantados, las CAs están escuchando y el ordering service node listo para funcionar. Así, lo que haremos ahora será crear el canal *mychannel* dentro de la red, lanzando el comando correspondiente desde la instancia *cli.*

```
docker exec \
   cli peer channel create \
      -o orderer.example.com:7050 \
      -c mychannel \
      -f /opt/gopath/src/github.com/hyperledger/fabric/peer/channel.tx \
      --tls \
      --
cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/exam
ple.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem \
      --
certfile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/exa
mple.com/orderers/orderer.example.com/tls/servert.crt
```

Con el canal ya creado ya podemos conectar a él cada uno de los peers. Para ello lanzaremos el siguiente comando por cada uno de los peers de cada organización participante en la red:

```
#join channel
docker exec \
   -e "CORE_PEER_LOCALMSPID=MoviChain" \
   -e "CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/
crypto/peerOrganizations/movichain.com/users/Admin@movichain.com/msp" \
   -e "CORE_PEER_ADDRESS=peer0.movichain.com:7051" \
   -e "CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/
peer/crypto/peerOrganizations/movichain.com/peers/peer0.movichain.com/tls/ca.crt" \
   cli peer channel join \
      -b mychannel.block \
      --tls \
      --
cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/exam
ple.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem \
      --
certfile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/movich
ain.com/users/Admin@movichain.com/tls/servert.crt
```

El comando **"peer channel join"** conecta a la red el nodo configurado en la instancia desde la que se lanza la orden. Como vimos en su configuración, la instancia cli está configurada con los valores del peer0 de AOP luego, para ejecutar el comando *"peer channel join"* como si lo hiciéramos desde el peer0 de MoviChain, ejecutamos *"docker exec"* estableciendo las variables de entorno CORE_PEER_LOCALMSPID, CORE_PEER_MSPCONFIGPATH, CORE_PEER_ADDRESS y CORE_PEER_TLS_ROOTCERT_FILE para que contengan los valores del peer deseado.

El siguiente comando, **"peer channel update",** actualiza en el canal el anchor peer indicado de cada organización. Nuevamente deberemos configurar las variables de entorno CORE_PEER_LOCALMSPID, CORE_PEER_MSPCONFIGPATH, CORE_PEER_ADDRESS y CORE_PEER_TLS_ROOTCERT_FILE para configurar cada uno de los anchor peer de las distintas organizaciones.

```
#update anchor peer
docker exec \
  -e "CORE_PEER_LOCALMSPID=MoviChain" \
  -
e "CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto
/peerOrganizations/movichain.com/users/Admin@movichain.com/msp" \
  -e "CORE_PEER_ADDRESS=peer0.movichain.com:7051" \
  -
e "CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/cry
pto/peerOrganizations/movichain.com/peers/peer0.movichain.com/tls/ca.crt" \
  cli peer channel update \
    -o orderer.example.com:7050 \
    -c mychannel \
    -f /opt/gopath/src/github.com/hyperledger/fabric/peer/MoviChainAnchors.tx \
    --tls \
    --
cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/exam
ple.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem \
    --
certfile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/movich
ain.com/users/Admin@movichain.com/tls/servert.crt
```

7.6. SMART CONTRACTS

El propósito principal de esta prueba de concepto es sustituir la lógica de negocio que alberga la ER en el proceso de portabilidad actual, eliminando a dicha entidad y a su actividad de todo el proceso. Así, las reglas y condiciones en las que se realizan los intercambios de información entre las operadoras hasta que una portabilidad se lleva cabo estarán incluidas en un smart contract desplegado en el canal mychannel de nuestra red. Dicho smart contract será invocado por las operadoras a través de sus aplicaciones, no pudiendo modificar las reglas de negocio definidas.

7.6.1. DEFINICIÓN E IMPLEMENTACIÓN

Como ya se ha indicado con anterioridad, los smart contracts no son más que software desarrollado en un lenguaje de programación determinado que se ejecuta dentro de la red de Fabric, tras invocarle previamente haciendo uso de alguna de las librerías ofrecidas por los desarrolladores de Hyperledger Fabric. El smart contract, a su vez, hará uso de las APIs o librerías correspondientes para obtener y/o modificar los objetos de negocio almacenados en el ledger.

Nuestro smart contract, llamado portabilityManagement, es una aplicación JavaScript con la estructura de un módulo de Nodejs, en el cual hemos incluido como dependencia una librería escrita en JavaScript llamada *fabric-contract-api* que Hyperledger Fabric pone a nuestra disposición para interactuar con el ledger. La estructura de directorios que alberga el smart contract es la siguiente:

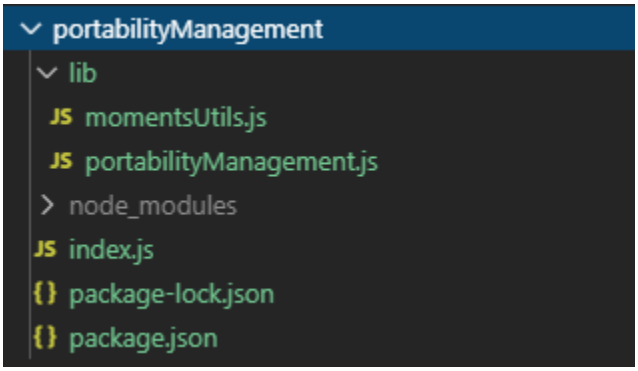

Figura 8. Estructura de directorios del smart contract *portabilityManagement*.

Como se ve, la estructura es la típica de un módulo npm (node package manager). La única particularidad que existe es en el fichero *index.js,* que debe exportar un módulo llamado "contracts" que representará un array con todos los smart contracts que se incluyan en el paquete a implantar.

```
'use strict';
const PortabilityManagement = require('./lib/portabilityManagement');
module.exports.PortabilityManagement = PortabilityManagement;
module.exports.contracts = [PortabilityManagement];
```

Para que las portabilidades puedan llevarse a cabo, en el smart contract hemos expuesto una serie de funciones que serán invocadas por las operadoras. Estas funciones contienen la lógica de negocio necesaria para coordinar las portabilidades. Las funciones son las siguientes:

initLedger(ctx)	Inicializa el ledger con valores de prueba
nuevaPortabilidad (ctx, idPorta, numComercial)	Da de alta una nueva portabilidad. Valida que el número pasado no

	pertenezca a la operadora receptora y/o esté en una portabilidad viva en el momento de la invocación.
aceptarPortabilidad(ctx,idPorta)	Pasa la portabilidad a estado "ACP". Valida que el estado de la solicitud sea "SP". Valida que no se ha superado el umbral de 40'' desde el inicio del minuto en el que se realizó la solicitud de portabilidad. En caso de haberse superado, el estado de la portabilidad pasará a "DSP".
denegarPortabilidad (ctx, idPorta, motivDeneg)	Pasa la portabilidad a estado "DSP" con el motivo indicado. Valida que la portabilidad indicada está en estado "SP". Valida que no se haya superado el umbral de 40'' desde el inicio del minuto en el que se realizó la solicitud. En caso de haberse superado, el estado de portabilidad pasará a "DSP" pero indicando como motivo el haber sobrepasado el umbral de los 40''.
cancelarPortabilidad (ctx, idPorta)	Establece el estado de la portabilidad a "CNP" siempre y cuando no se haya superado el umbral de 40''. Si se ha superado dejará la portabilidad en estado "DSP" por tiempo expirado.
notificarPortabilidadDonanteRealizada (ctx, idPorta)	Esta función deberá ser invocada por la operadora donante para confirmar que se han finalizado los trabajos de portabilidad, dejando constancia de la fecha y hora de la finalización de los trabajos.
notificarPortabilidadReceptoraRealizada (idPorta, idTitular, nombreTitular, NRN)	Esta función deberá ser invocada por la operadora receptora para confirmar que se han finalizado los trabajos de portabilidad. El estado de portabilidad pasará a "PR" y se incluirá el NRN en la línea para que las operadoras puedan enrutar correctamente las llamadas. Se asociarán a la línea los datos del titular, que solo serán consultables por las operadoras, no por la AOP.
getPortabilidadDonantes (ctx)	Obtiene las portabilidades donantes en vuelo de la operadora llamante.
getPortabilidadReceptoras (ctx)	Obtiene las portabilidades receptoras en vuelo de la operadora llamante.

getLinea (ctx, numComercial)	Obtiene los datos de la línea pasada como parámetro.
getOperadora (ctx)	Obtiene los datos de la operadora llamante.
getPortabilidad (ctx, idPortabilidad)	Obtiene los datos de la portabilidad pasada por parámetro.
getCliente (ctx,idCliente)	Obtiene los datos de un cliente a partir de su documento de identidad. Devolverá información siempre y cuando el invocante pertenezca a una de las tres operadoras, y devolverá un error en caso contrario. Hace uso del API de Fabric para obtener datos privados.

Para conocer quién realiza cada una de las invocaciones al smart contract, hacemos uso de la función *ClientIdentity.getMSPID(),* que nos devuelve el id del MSP que está realizando la invocación. Además, para poder realizar las verificaciones de tiempo, se ha diseñado un módulo llamado *momentsUtils.js* con varias funciones que facilitan el trabajo. Estas funciones son:

haSidoHoy (ahora, momento)	Comprueba si, dada la fecha del momento actual, la fecha pasada como segundo parámetro ha ocurrido en el "día de hoy", entendiendo como día de hoy el minuto en el que nos encontramos al invocar a la función.
fueAyer (ahora, momento)	Comprueba si, dada la fecha del momento actual, la fecha pasada como segundo parámetro ha ocurrido en el "día de ayer", entendiendo como día de ayer el minuto anterior al actual
superadoUmbralModificaciones (momento)	Indica si se ha superado la barrera del segundo 40, momento en el cual no se puede aceptar, cancelar o denegar una solicitud de portabilidad programada para el día de hoy

La fecha y hora de las peticiones realizadas por las organizaciones al smart contract se establecen según el momento en el que llegan a la red, i.e., las operadoras no envían la fecha como dato de las peticiones. Esto puede provocar que, puesto que los relojes de las operadoras y la portability-network no se sincronizan, se reciban peticiones fuera de plazo, pese a que las operadoras, según su reloj, las hayan enviado en un instante válido. No obstante, en un escenario real, las ventanas de portabilidad son bastante amplias y las operadoras no apuran hasta el último minuto para

lanzar sus peticiones, por lo que basta con que los servidores de las operadoras y la red estén configurados o tengan en cuenta el huso horario de la red.
En el caso de la PoC, puesto que todo se ejecuta dentro de la misma máquina, el reloj será compartido. Únicamente podremos apreciar algún inconveniente debido al reducido tamaño de la ventana de tiempo para procesar una portabilidad pues, en algunos casos y dependiendo del hardware en el que se ejecute la PoC, el procesamiento de las transacciones pueden llevar un tiempo lo suficientemente largo como para que una petición lanzada en un instante válido sea procesada en uno fuera de ventana.

El aspecto de nuestro smart contract puede verse en el siguiente extracto:

```
'use strict';
const { Contract } = require('fabric-contract-api');
const MUtils = require('./momentsUtils');
const MomentsUtils = new MUtils();
const DEBUG = false;
class PortabilityManagement extends Contract {

/*....*/

async notificarPortabilidadDonanteRealizada(ctx, idPortabilidad) {
      let idOperadora = this.getIdOperadoraInvocante(ctx);
      let operadoraDonante = await this.getOperadora(ctx, idOperadora);
      let portabilidad = await this.getPortabilidad(ctx, idPortabilidad);
      if (operadoraDonante == null || portabilidad == null || operadoraDonante.id !=
portabilidad.operadoraDonante) {
         return new Promise(function (resolve, reject) {
            reject("Datos de entrada incorrectos");
         });
      }
      this.configurarRealizacionPortabilidadDonante(portabilidad, operadoraDonante);
      await this.updateOperadora(ctx, operadoraDonante);
      await this.updatePortabilidad(ctx, portabilidad);
      return new Promise(function (resolve, reject) {
         resolve(portabilidad.id);
      });

   } /*fin notificarPortabilidadDonanteRealizada*/...

/*....*/

}
module.exports = PortabilityManagement
```

7.6.2. USO DE DATOS PRIVADOS

La AOP debe tener la posibilidad de obtener datos acerca de las portabilidades con el fin de sacar estadísticas, evitar actividades de las operadoras que dañen a la competencia, etc. Los datos relacionados con el titular de la línea {nombre, dirección, cuenta} son datos sensibles que hemos determinado que no sean consultados por la AOP, así que configuraremos estos datos como una colección privada y aplicaremos una política de seguridad para que solo las operadoras puedan consultarlos. Para llevar a cabo esto crearemos un fichero llamado *private-collections.json* que contendrá la configuración de la colección privada:

```
[
  {
    "name": "clientDataCollection",
    "policy":
"OutOf(1,'MoviChainMSP.member','BlockStarMSP.member','MasLedgerMSP.member')",
    "requiredPeerCount": 0,
    "maxPeerCount": 3,
    "blockToLive":1000000,
    "memberOnlyRead": true
  }
]
```

En el fichero indicamos el nombre de la colección y las políticas que le aplican, que indican que la consulta o modificación de estos datos será válida cuando la realice algún miembro de las operadoras. A la hora de instanciar el smart contract en el canal, indicaremos la ruta del fichero de colecciones y, ya en el código del smart contract, se hará referencia al nombre de la colección para recuperar o guardar los datos.

En el smart contract, por su parte, en las funciones que permiten obtener y grabar datos de cliente hemos hecho uso de los métodos API *getPrivateData(collection,key)* y *putPrivateData(collection, key, value)*, que permiten la manipulación de datos privados, validando con la política indicada en la colección *clientDataCollection.*

```
async insertCliente(ctx, cliente) {
    await ctx.stub.putPrivateData("clientDataCollection", cliente.id,
    Buffer.from(JSON.stringify(cliente)));
}

async getCliente(ctx, idCliente) {
    const lineaAsBytes = await ctx.stub.getPrivateData("clientDataCollection", idCliente);
    if (!lineaAsBytes || lineaAsBytes.length === 0) {
      return null;
    }
    const cliente = JSON.parse(lineaAsBytes.toString(), this.reviver);
    return cliente;
  }
```

Puede obtenerse información acerca de cómo trabajar con datos privados en la documentación oficial de Fabric a este respecto [10].

7.6.3. COMPILACIÓN Y DESPLIEGUE

Una vez programado el smart contract hemos de instalarlo en cada uno de los peers, para posteriormente instanciarlo en el canal para poder invocarlo. Lo primero que debemos hacer es compilar el paquete portabilityManagement. Para ello, invocaremos el comando *npm install* dentro del directorio, con el fin de que descargue todas las dependencias del módulo. Una vez compilado, haremos uso del nodo *cli* para instalar el paquete. Para ello, en la definición del contenedor cli en el fichero docker-compose.yml, hemos mapeado la ruta donde se encuentra nuestro smart contract para que esté disponible dentro del contenedor. Para instalarlo en un nodo ejecutaremos el siguiente comando:

```
#instalamos el smart contract en peer MoviChain
docker exec \
  -e "CORE_PEER_LOCALMSPID=MoviChain" \
  -e
"CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/p
eerOrganizations/movichain.com/users/Admin@movichain.com/msp" \
  -e "CORE_PEER_ADDRESS=peer0.movichain.com:7051" \
  -e
"CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypt
o/peerOrganizations/movichain.com/peers/peer0.movichain.com/tls/ca.crt" \
  cli peer chaincode install \
  -n portabilityManagement \
  -l node \
  -p /opt/gopath/src/github.com/chaincode \
  -v 1.0 \
  --tls \
  --cafile
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.co
m/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem \
          --certfile
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/movichain.com
/users/Admin@movichain.com/tls/servert.crt
```

*La ruta marcada en la opción *-p* es el path dentro del container cli donde hemos mapeado la carpeta portabilityManagement que contiene nuestro smart contract.

Para poder ejecutar el smart contract es necesario instanciarlo en el canal e indicar las políticas de ejecución de éste, así como el fichero de configuración de las colecciones privadas. Para la PoC vamos a indicar que cualquier miembro de cualquiera de las organizaciones que conforman la red puede ejecutar el smart contract.

```
#instanciamos el smartcontract en el canal
docker exec \
cli \
  peer chaincode instantiate \
    -o orderer.example.com:7050 \
    -C mychannel \
    -n portabilityManagement \
    --collections-config /opt/gopath/src/github.com/chaincode/portabilityManagement/private-
collections.json
    -l node \
    -v 1.0 \
    -c '{"Args":[]}' \
    -P "OutOf(1, 'AOP.member', 'MoviChain.member', 'BlockStar.member', 'MasLedger.member')"
\
    --tls \
    --cafile
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.co
m/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem \
    --tlsRootCertFiles
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/aop.com/peers
/peer0.aop.com/tls/ca.crt
```

Con la instanciación del smart contract finalizaría el despliegue de la portability-network, por lo que ya podríamos empezar a interaccionar con ella y con el smart contract para realizar portabilidades, utilizando para ello los datos de prueba que se incluyeron en el propio smart contract.

8. PRUEBAS

Para comprobar el funcionamiento de la red se han creado tres aplicaciones web, una para cada operadora de la PoC, que invocan al smart contract con el fin de solicitar nuevas portabilidades, rechazarlas, aceptarlas, listarlas y procesarlas. Estas aplicaciones se incluyen en los entregables del TFM y hacen uso de las APIs escritas en JavaScript provistas por Hyperledger Fabric [15].

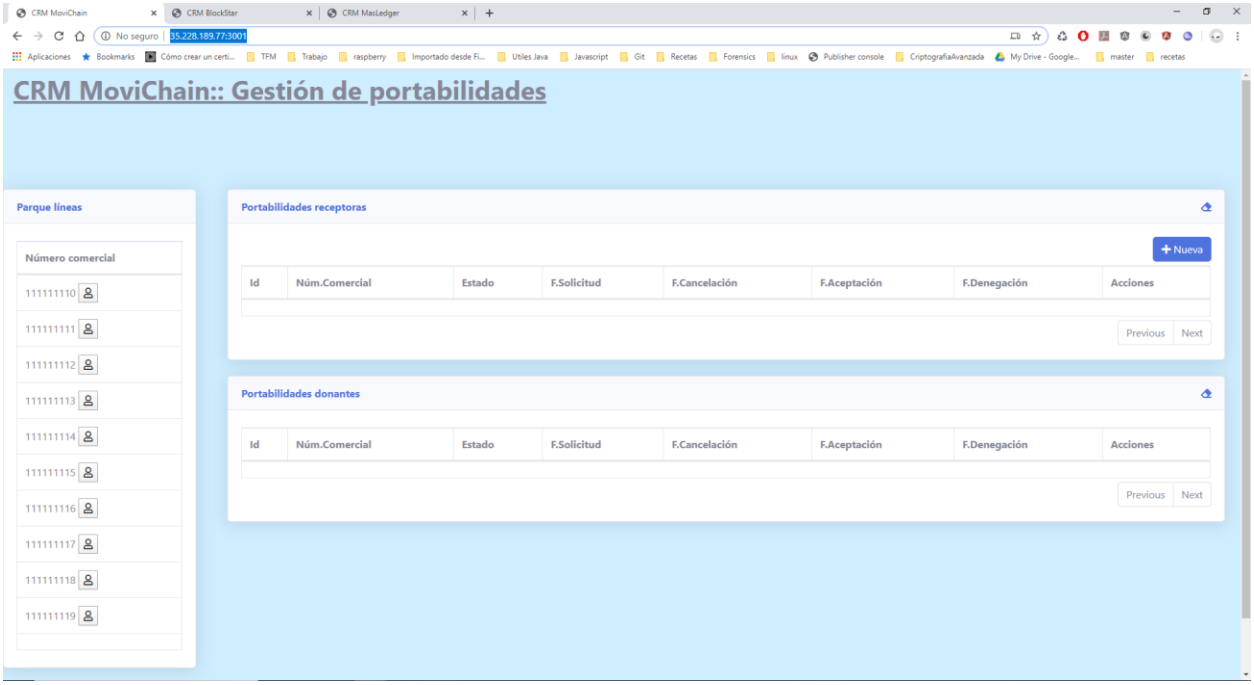

Figura 9. Aplicación web de la operadora MoviChain que interacciona con la portability-network.

La aplicación de cada operadora muestra en la parte izquierda un listado de número de teléfono a los que actualmente presta servicio junto con los datos del titular, mientras que en el centro de la aplicación podemos ver las portabilidades receptoras solicitadas y las donantes recibidas. Estas portabilidades son obtenidas invocando a las operaciones *getPortabilidadesReceptoras* y *getPortabilidadesDonantes* del smart contract, mientras que las líneas comerciales están almacenadas en la propia aplicación.

Realizamos algunas pruebas para comprobar el correcto funcionamiento de la red:

Aceptación portabilidad

Desde la operadora MoviChain se realiza una solicitud de portabilidad del número 333333337. La operadora donante, MasLedger, acepta la solicitud antes de que se sobrepase el segundo 40''. Posteriormente ambas operadoras realizan las acciones de procesado de la portabilidad y la línea se porta satisfactoriamente.

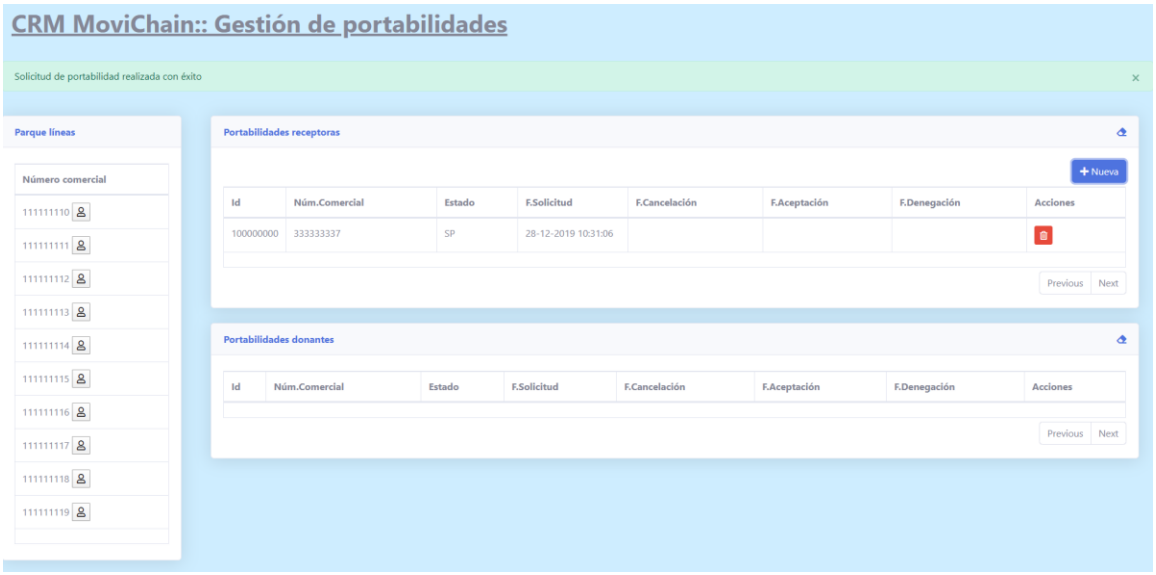

Figura 10. Solicitud de portabilidad receptora del número 333333337 desde MoviChain.

Figura 11. Recepción de la portabilidad donante del número 333333337 en la operadora MasLedger.

Figura 12. La operadora donante procesa la portabilidad, dándose de baja la línea del parque de líneas de la compañía.

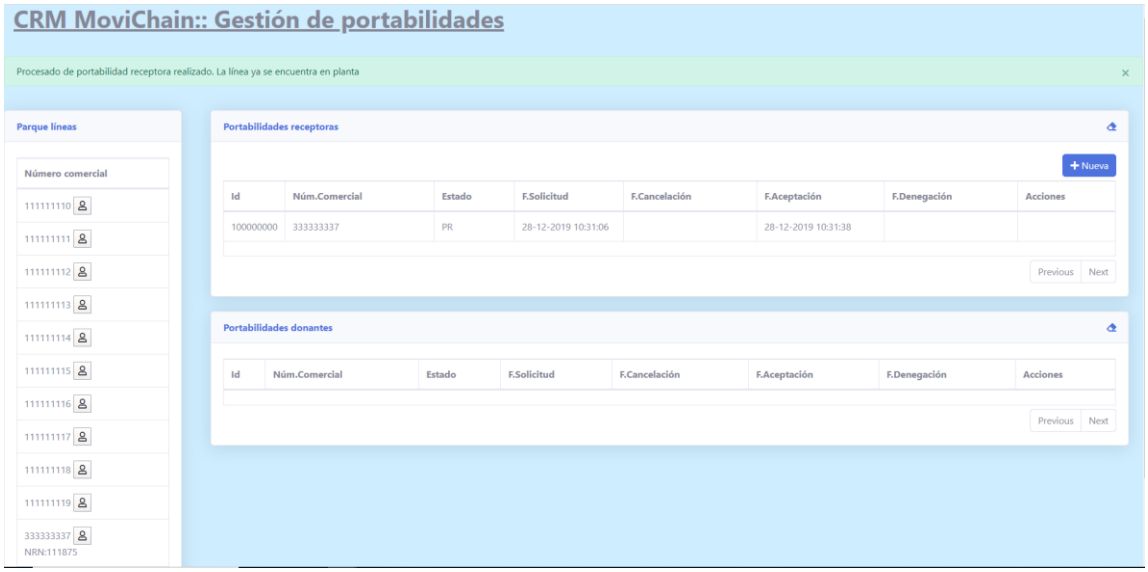

Figura 12. Se procesa la portabilidad en la operadora receptora. En el parque de líneas de la compañía aparece ahora la nueva línea portada, junto a su NRN.

Rechazo solicitud de portabilidad

Desde la operadora MasLedger realizamos una solicitud de portabilidad del número 222222223, perteneciente a la operadora BlockStar, que rechaza la solicitud simulando que el número no puede portarse en ese momento. En el listado de MasLedger aparecerá la portabilidad con el estado DSP debido al rechazo de BlockStar.

Figura 13. Solicitud de portabilidad desde MasLedger.

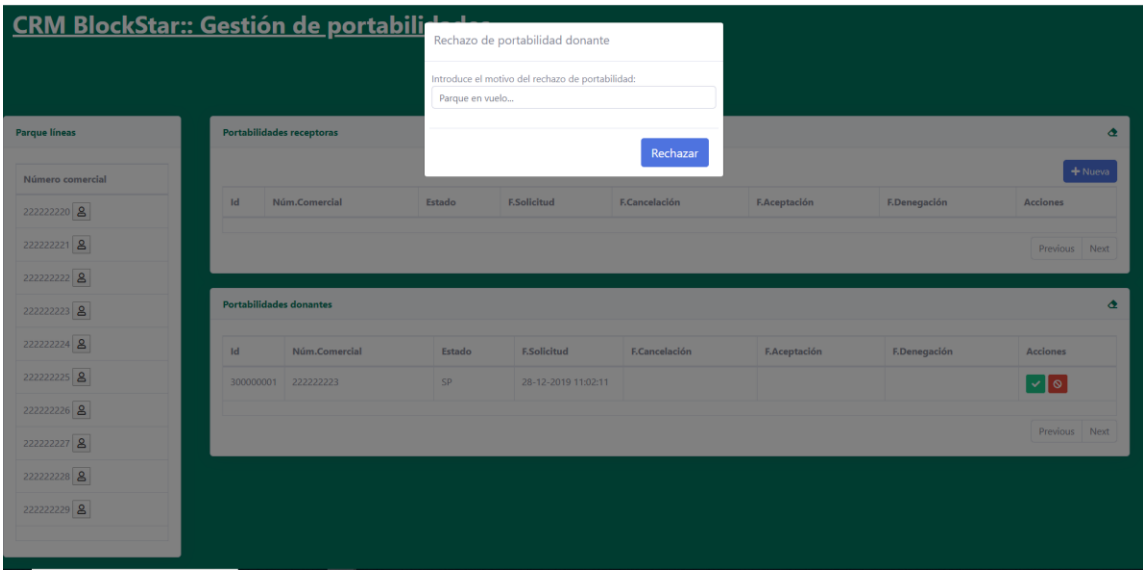

Figura 14. Rechazo de portabilidad desde operadora donante BlockStar.

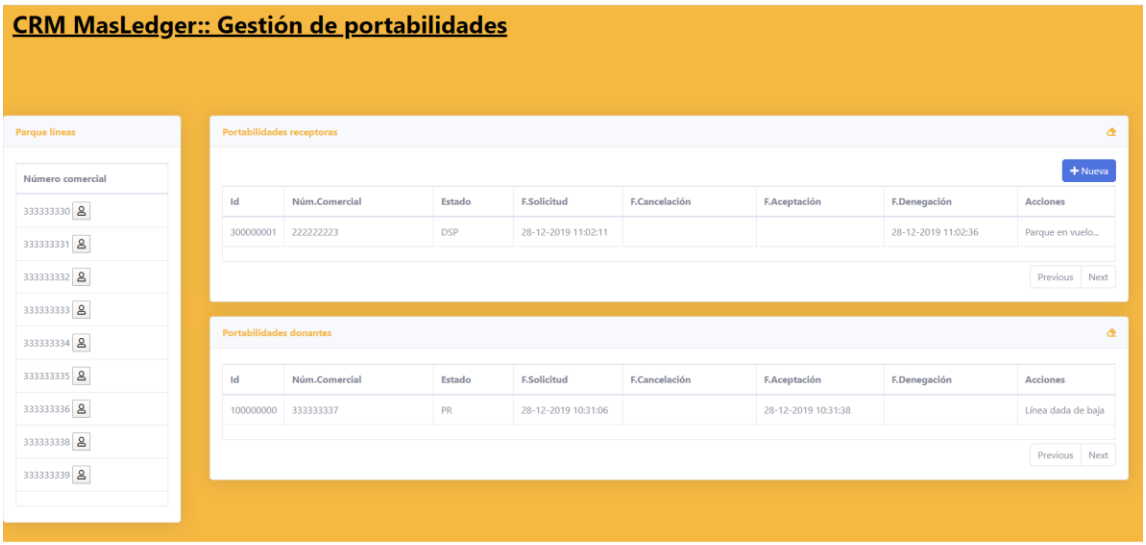

Figura 15. La portabilidad rechazada se muestra como tal en la operadora receptora.

Aceptación de una portabilidad donante fuera de tiempo

En esta ocasión, la operadora BlockStar va a solicitar una portabilidad del número 111111117, perteneciente a MoviChain. La operadora donante, MoviChain, va a aceptar la portabilidad después del segundo 40'', lo que provocará que el smart contract deniegue la portabilidad por tiempo expirado.

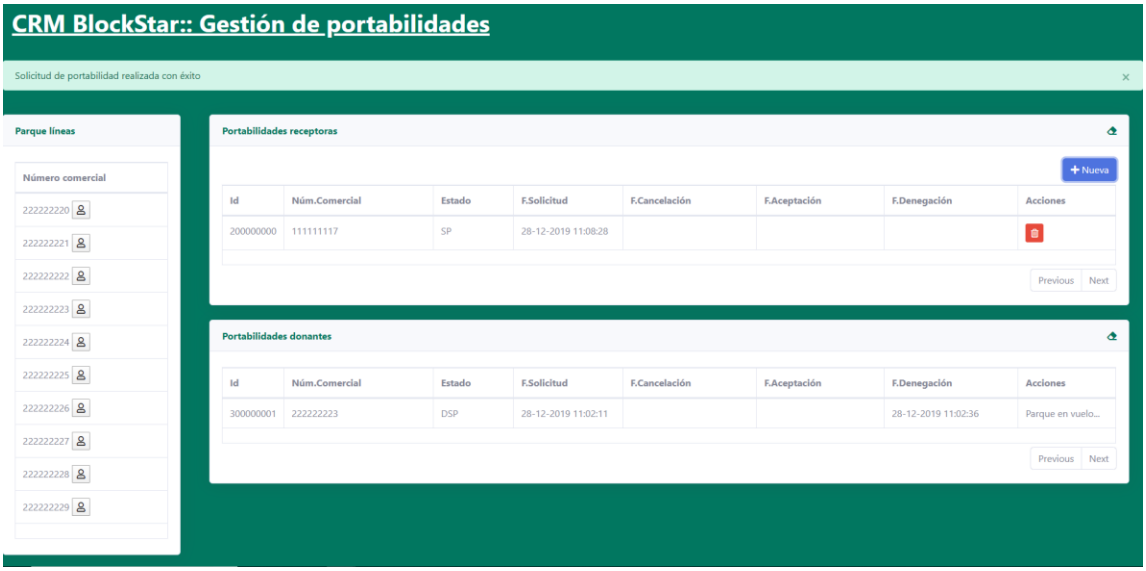

Figura 16. Solicitud de portabilidad por parte de BlockStar.

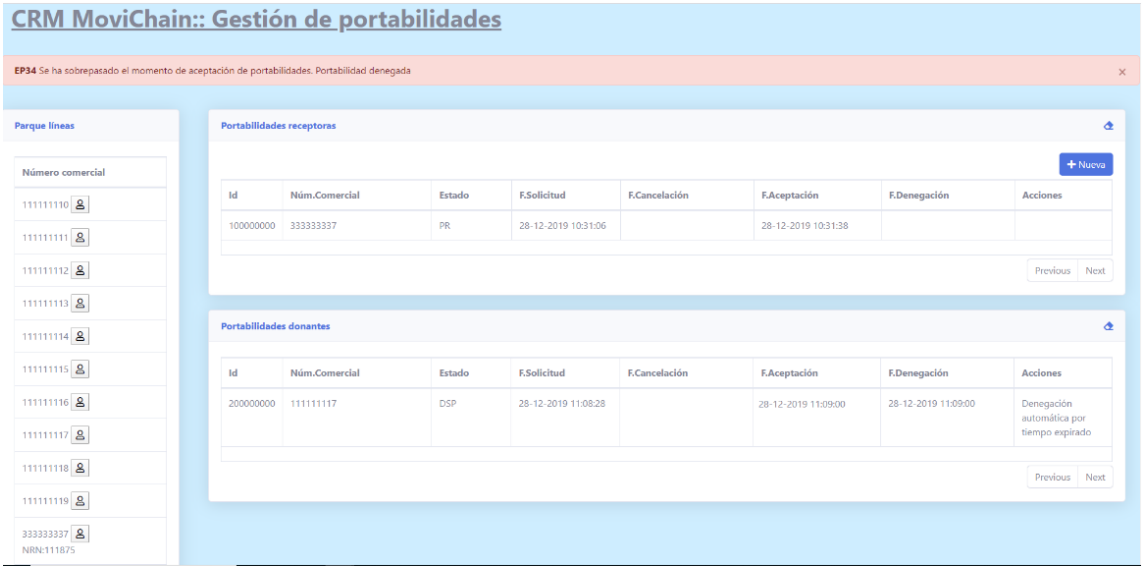

Figura 17. Aceptación de portabilidad fuera de plazo, provocando su rechazo automático.

Datos privados

Las operadoras pueden consultar en cualquier momento los datos de los titulares de las líneas pulsando sobre el icono  junto a cada número comercial, ya que las aplicaciones se conectan a la portability-network con usuarios pertenecientes a MSPs habilitados para consultar la colección privada de datos *clientDataCollection*, definida en el apartado **7.6.2.**

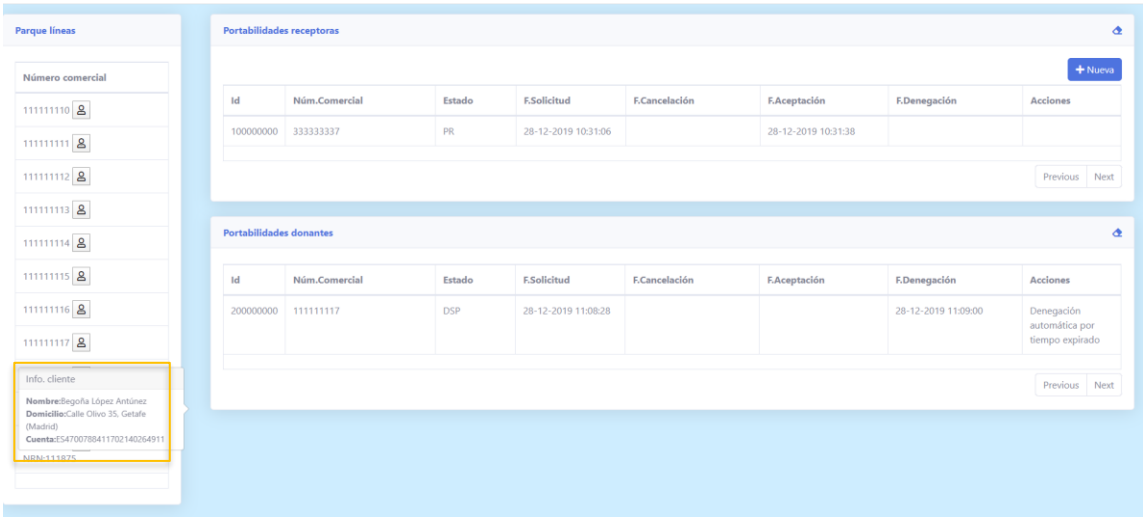

Figura 18. Información sensible del cliente obtenida invocando al método *getCliente*(id) del smart contract *portabilityManagement*.

Para confirmar que la entidad AOP no tiene acceso a los datos de los clientes diseñamos un script escrito en JavaScript que se conecta a la red e invoca al método *getCliente(id)*. Este script puede consultarse en la entrega de material de este TFM, cuyas instrucciones más interesantes son las indicadas en el siguiente extracto:

```
use strict';
const { FileSystemWallet, Gateway } = require('fabric-network');
const path = require('path');
const ccpPath = path.resolve(__dirname, '.', 'connection-aop.json');
async function main() {
  try {
    // Create a new file system based wallet for managing identities.
    const walletPath = path.join(process.cwd(), 'wallet');
    const wallet = new FileSystemWallet(walletPath);
    console.log(`Wallet path: ${walletPath}`);
    const gateway = new Gateway();
    await gateway.connect(ccpPath, { wallet, identity: 'Admin', discovery: { enabled: true,
asLocalhost: true } });
    const network = await gateway.getNetwork('mychannel');
    const contract = network.getContract('fabcar');
    const result = await contract.evaluateTransaction("getCliente","39154171Y");
    console.log(`Transaction has been evaluated, result is: ${result.toString()}`);
  } catch (error) {
    console.error(`Failed to evaluate transaction: ${error}`);
    process.exit(1);
  }
}
```

En este caso, el smart contract nos devuelve el siguiente error al no cumplirse las políticas de validación de la colección de datos privada *clientDataCollection.*

```
paco@CENTRAL:~/hyperledger/utilesPortabilityNetwork/pruebas/javascript/aopClient$ node query.js user444
Wallet path: /home/paco/hyperledger/utilesPortabilityNetwork/pruebas/javascript/aopClient/wallet
2019-12-27T11:05:18.795Z - warn: [DiscoveryEndorsementHandler]:  build endorse group member >> G2:0 - endorsement failed - Error: transaction returned with failure: Error: GE
T STATE failed: transaction ID: 08f588da42e846b448f6bd276361bd723d4007a5e554f0f0e68b86cbb0631a05: tx creator does not have read access permission on privatedata in chaincodeN
ame:portabilityManagement collectionName: clientDataCollection
2019-12-27T11:05:18.813Z - warn: [DiscoveryEndorsementHandler]:  build endorse group member >> G2:0 - endorsement failed - Error: transaction returned with failure: Error: GE
T STATE failed: transaction ID: 08f588da42e846b448f6bd276361bd723d4007a5e554f0f0e68b86cbb0631a05: tx creator does not have read access permission on privatedata in chaincodeN
ame:portabilityManagement collectionName: clientDataCollection
```

Figura 19. Acceso denegado a la colección de datos privada.

En cuanto a la implementación de la red y el smart contract habría que destacar que:

- Es complejo levantar una red de Fabric desde cero. Hay muchas configuraciones particulares no documentadas que, en caso de no establecerlas, producen errores no muy descriptivos, lo que obliga a buscar en Internet casos similares y soluciones al respecto. Así, a menudo se termina copiando una de las redes provistas en los casos de ejemplo que facilita Hyperledger Fabric y adaptándola a nuestras necesidades.
- La documentación y guías de Fabric se basa en el uso de contenedores de Docker. Se echa en falta quizás información acerca de cómo configurar los peers, ordering service nodes y demás elementos de red sin el uso de este tipo de tecnología.
- A la hora de desarrollar los smart contract y las aplicaciones que los consumen, hay que tener muy en cuenta que la red responde de manera asíncrona a cada una de las solicitudes de transacción que se le realizan, modificando la manera habitual de programar.

- Para realizar pruebas del smart contract es necesario levantar una pequeña red en la que desplegarlo. Esto ralentiza mucho el tiempo de desarrollo del smart contract, lo que obliga a desarrollarlo de manera aislada usando *mocks.*

Con respecto al diseño del proceso de portabilidad:

- **La descentralización del proceso hace que éste sea más seguro y más estable**, aunque la escalabilidad y rendimiento necesitan de un estudio más exhaustivo para confirmar que la implementación es viable en un entorno real. Las conclusiones a este respecto se encuentran definidas en el siguiente apartado.
- **Se han simplificado los flujos de comunicación y las limitaciones horarias del proceso.** En el proceso centralizado:
    - Las solicitudes de portabilidad deben llegar entre las 20:00 y las 21:00 del día anterior
    - Las solicitudes llegan a las operadoras donantes a las 07:00
    - Las aceptaciones/denegaciones deben llegar antes de las 14:00
    - Las cancelaciones deben producirse antes de las 19:00
    - La ER notifica las portabilidades no cancelables entre las 19:00 y las 20:00

  Estos horarios tan rígidos son debidos a la necesidad de la ER de disponer de tiempo suficiente para validar todas y cada una de las solicitudes diarias que se producen, lo que provoca que las operadoras donantes solo disponen de 7 horas teóricas para intentar retener al cliente (bastantes menos si se tiene en cuenta el horario habitual en el que las personas estamos disponibles para atender una llamada telefónica de nuestra operadora).

  Al descentralizar el proceso y automatizar las reglas de negocio se ha conseguido establecer un único umbral para realizar las solicitudes y sus modificaciones. En particular, y si lleváramos la PoC a un escenario real, las operadoras simplemente deberían realizar sus transacciones antes de las 19:00 para portabilidades que debieran realizarse ese mismo día. Esto permite a las operadoras donantes disponer de casi 24h para preparar una oferta de retención del cliente.
- **Datos siempre disponibles.** Los datos de las líneas, operadoras y portabilidades son consultables en cualquier momento por las operadoras, mientras que en el proceso centralizado debían esperar a que la ER enviara los ficheros de información. Aunque no se ha implementado en la PoC, sería relativamente sencillo que el smart contract permitiera la consulta de las portabilidades que se van a realizar cada día con el fin de que las operadoras terceras actualicen sus tablas de enrutamiento de llamadas.

9. CONCLUSIONES

Tras el estudio del proceso de portabilidad numérica, el análisis de Hyperledger Fabric, y el diseño e implementación de un PoC que soportara el proceso mencionado, volvemos a recuperar los problemas detectados al comienzo de este documento para valorar si el uso de redes permisionadas mitiga o salva cada uno de ellos, destacando además los problemas de rendimiento que podría tener la implementación en un entorno productivo.

Seguridad

Indicábamos, al analizar la arquitectura del proceso de portabilidad, que había un grave problema de seguridad al estar centralizada tanto la lógica de negocio como la base de datos en el nodo central o Entidad de Referencia. El acceso no permitido a esta entidad y la toma de su control comprometía todo el proceso de portabilidad. Por otro lado, mencionábamos también que las operadoras no podían comprobar de manera autónoma si la información ofrecida por la ER era veraz o no, ya que nadie podía garantizar su integridad, pues miembros de la propia ER podrían acceder a los registros de bases de datos, de ficheros, etc., y manipularlos a su antojo sin que ello quedara reflejado.

Con la implementación del proceso sobre la red de Hyperledger Fabric estos problemas desaparecen por completo: la naturaleza distribuida de la red impide que nadie pueda hacerse con el control total del proceso. Si alguna operadora empezara a tener un comportamiento anómalo, los administradores de la portability-network podrían excluir a sus peers de una manera relativamente rápida, modificando las políticas de los smart contracts. Con respecto a la integridad de las transacciones ocurridas, cualquier nodo, en cualquier momento, podría verificar que los datos reflejados son fruto de sucesivas transacciones.

Estabilidad

Relacionada con la respuesta anterior, el hecho de no existir en la red permisionada ningún nodo central garantiza que el proceso seguirá funcionando, aunque uno o varios nodos queden fuera de servicio. Además, que cada organización participe en la red con los recursos que considere, les permite, en la medida de lo posible, mejorar y garantizar la disponibilidad de la parte de la red que les corresponde, pues será a sus peers a los que pueda conectarse para obtener información del ledger.

Rendimiento

Aunque no ha sido objeto principal de este trabajo, en el estudio *"Performance, benchmarking & optimizing Hyperledger Fabric Blockchain Platform"* [11] los autores indican estrategias para mejorar el rendimiento de Hyperledger Fabric, de tal manera que consiguen pasar de 140tps hasta las 2250tps. Teniendo en cuenta que la media de portabilidades mensuales son unas 700.000, necesitaríamos un rendimiento de 0,27 tps, mucho menor que la cifra inicial dada por los autores del paper. No obstante, si revisamos el documento ***"Hyperledger Blockchain Performance Metrics"*** [12], se detallan distintos casos de

prueba en el que analizan no la velocidad de transacción del sistema, sino la capacidad de confirmar las transacciones, i.e., hacer commits o confirmaciones de los datos modificados. Desde este punto de vista, y según los datos oficiales de las pruebas de Hyperledger, el mejor resultado que obtuvieron fue de 1 commit cada nodo cada 11'', usando Practical Byzantine Fault Tolerance, PBFT, como algoritmo de consenso. Este resultado hace pensar que quizás fuera necesaria una optimización y mejora del rendimiento de la red para soportar las exigencias del proceso de portabilidad.

Escalabilidad

La red distribuida nos permite que nuevas organizaciones puedan añadirse a la portability-network y participar en ella sin necesidad de parar el proceso de portabilidad. Solamente será necesario actualizar las políticas de seguridad para añadir a la nueva organización (si es que ésta va a utilizar unos certificados propios y no unos que fueran expedidos por una CA intermedia válida para todas las organizaciones).

Sin embargo, mientras que en otros escenarios la escalabilidad también es sinónimo de mejora del rendimiento, en el caso de las redes permisionadas resulta todo lo contrario: a medida que vamos añadiendo nodos a la red necesitamos que los mensajes y comunicaciones alcancen a estos nuevos nodos, lo que provoca que los algoritmos de consenso vayan aumentando su tiempo de resolución. En el artículo *"The Blockchain Scalability Problem & the Race for Visa-Like Transaction Speed"* [13] el autor nos detalla los problemas de lentitud en el procesamiento de transacciones de la red Bitcoin (4,6tps) con respecto a las transacciones realizadas por VISA (1700tps), y como el aumento del tamaño de bloque o la disminución de la velocidad de generación de bloques no es suficiente para aumentar la velocidad de procesamiento de este tipo de redes.

10.BIBLIOGRAFÍA

[1] Domo. **Data Never Sleeps 6.0.** Recuperado el 28 de diciembre de 2019, desde https://www.domo.com/learn/data-never-sleeps-6

[2] CNMC. **Informe Mensual.** Recuperado el 28 de diciembre de 2019, desde http://data.cnmc.es/datagraph/jsp/inf_men.jsp

[3] Guerra, I. (2019). **Conceptos básicos de telecos: Portabilidad en redes fijas - CNMC blog.** Recuperado el 28 de diciembre de 2019, desde https://blog.cnmc.es/2010/02/26/conceptos-basicos-de-telecos-portabilidad-en-redes-fijas/

[4] **Circular 2/2004 CNMC** – CNMC (2004). Recuperado el 28 de diciembre de 2019, desde https://www.cnmc.es/sites/default/files/1519940_4.pdf

[5] **Circular 1/2008 CNMC** – CNMC (2008). Recuperado el 28 de diciembre de 2019, desde https://www.cnmc.es/sites/default/files/1519913_10.pdf

[6] IBM Research Editorial Staff (2018), **Behind the Architecture of Hyperledger Fabric**. Recuperado el 28 de diciembre de 2019, desde https://www.ibm.com/blogs/research/2018/02/architecture-hyperledger-fabric/

[7] **Hyperledger Fabric – Hyperledger.** Recuperado el 28 de diciembre de 2019, desde https://hyperledger-fabric.readthedocs.io/en/latest/whatis.html#hyperledger-fabric

[8] CNMC. (2015). **Especificación técnica de los procedimientos administrativos para la conservación de la numeración geográfica y de servicios de tarifas especiales y de numeración personal en caso de cambio de operador (Portabilidad Fija).** Recuperado el 28 de diciembre de 2019, desde https://www.cnmc.es/sites/default/files/ficheros/cnmc/AA_Telecomunicaciones/Portabilidad/POR-DTSA-2519-13%20Especificaci%C3%B3n%20t%C3%A9cnica.pdf

[9] **Policies in Hyperledger Fabric** — hyperledger-fabricdocs master documentation. (2019). Recuperado el 28 de diciembre de 2019, desde https://hyperledger-fabric.readthedocs.io/en/release-1.4/policies.html

[10] **Using Private Data in Fabric** — hyperledger-fabricdocs master documentation. (2019). Recuperado el 28 de diciembre de 2019, desde https://hyperledger-fabric.readthedocs.io/en/release-1.4/private_data_tutorial.html

[11] Thakkar, P., Nathan N, S., & Vishwanathan, B **Performance Benchmarking & Optimizing Hyperledger Fabric Blockchain Platform** Recuperado el 28 de diciembre de 2019, desde http://www.mscs.mu.edu/~mascots/Papers/blockchain.pdf

[12] (2019**). Hyperledger Blockchain Performance Metrics** Recuperado el 28 de diciembre de 2019, desde https://www.hyperledger.org/wp-content/uploads/2018/10/HL_Whitepaper_Metrics_PDF_V1.01.pdf.

[13] Li, K. (2019) **The Blockchain Scalability Problem & the Race for Visa-Like Transaction Speed.** Recuperado el 28 de diciembre de 2019, desde https://towardsdatascience.com/the-blockchain-scalability-problem-the-race-for-visa-like-transaction-speed-5cce48f9d44

[14] Sewan. 2018. **¿Cómo ha sido la evolución de las Telecomunicaciones?** Recuperado el 28 de diciembre de 2019, desde https://www.sewan.es/evolucion-de-las-telecomunicaciones/

[15] Hyperledger Fabric, **Hyperledger Fabric SDK for node.js**. Recuperado el 28 de diciembre de 2019, desde https://hyperledger.github.io/fabric-sdk-node/release-1.4/index.html