Abstraction & Application **29** (2020) 33 – 47

UADY

Método para el desarrollo de aplicaciones descentralizadas basadas en blockchain

Method for developing decentralized applications based on blockchain

José-Luis Balderas-Rosas[1], Ulises Juárez-Martínez[1],

Lisbeth Rodríguez-Mazahua[1], Beatriz-Alejandra Olivares-Zepahua[1], Adolfo Centeno Téllez[2]

[1]División de Estudios de Posgrado e Investigación, Instituto Tecnológico de Orizaba

Orizaba, Veracruz, México

[2]Universidad Veracruzana, Campus USBI

Ixtaczoquitlan, Veracruz, México

jl_balderas@outlook.com, ujuarez@orizaba.tecnm.mx,
{lrodriguezm, bolivares}@ito-depi.edu.mx, acenteno@uv.mx

Abstract

Blockchain technology has gained a massive boost in last years, due to the security it provides, because the blockchain is a shared peer-to-peer database; it is allowing immutable storage of information and ordered. Although this technology is not yet in full maturity, it has been integrating into the development of blockchain-based projects, so it is fundamental to represent properly it in software modeling. We carried out a research and analysis of related works, identifying artifact proposals for blockchain-oriented software modeling, as well as the advantages and disadvantages of the identified artifacts, which allowed selecting the artifacts that show better utility for BOS representation and modeling. In this work, we try to eliminate problems and conflicts during the design of blockchain-oriented software, making use of a method for the development of BOS. Use is UML (Unified Modeling Language) notations, considering the stereotypes or changes proposed in the analyzed works. It is worth mentioning that was used a case study, which is based on the development of a DApp (Decentralized Application) of an urban bus system. The platform used is Hyperledger Fabric, also provides the system architecture, architecture shows a complete view for implementation, at the same time that it allows to properly visualize the structure and organization of the software components.

Resumen

La tecnología *blockchain* ha ganado un impulso masivo en los últimos años gracias a la seguridad que proporciona, debido a que la cadena de bloques es una base de datos compartida P2P *(peer to*

Fecha de recepción: septiembre 1, 2020 / Fecha de aceptación: octubre 5, 2020

peer, igual a igual) que permite almacenar información de manera inmutable y ordenada. Aunque esta tecnología aún no está en una completa madurez, se ha integrado en el desarrollo de proyectos basados en *blockchain*, por lo que es fundamental representarla adecuadamente en el modelado de software. Se realizó una investigación y un análisis de trabajos relacionados, identificando propuestas de artefactos para el modelado de BOS (*Blockchain Oriented Software*, Software Orientado a Blokchain), asimismo se determinaron las ventajas y desventajas de los artefactos identificados, lo cual permitió seleccionar los artefactos que muestran mejor utilidad para la representación y modelación de BOS. En este trabajo se intentan eliminar problemas y conflictos durante el diseño de software orientado a *blockchain*, haciendo uso de un método para el desarrollo de BOS. Se hace uso de notaciones UML (*Unified Modeling Lenguage*, Lenguaje de modelado unificado), tomando en cuenta los estereotipos o cambios propuestos en los trabajos analizados. Cabe mencionar que lo anterior se aplicó mediante un caso de estudio, el cual se basa en el desarrollo de una DApp (*Decentralized Application*, Aplicación Descentralizada) de un sistema de autobuses urbano y se hace uso de la plataforma *Hyperledger Fabric*, asimismo se provee la arquitectura del sistema en donde se muestra una vista completa para implementación, al mismo tiempo que permite visualizar adecuadamente la estructura y organización de los componentes de software.

Keywords and phrases: Blockchain, BOS, DApp, Hyperledger Fabric, P2P, UML.

2020 Mathematics Subject Classification: 68N30.

---

1 Introducción

Blockchain se describió en el año 1991 bajo el nombre *Digital Timestamps*, sin embargo, pasó desapercibido hasta 2008 cuando se publicó el primer artículo [1] sobre la tecnología *blockchain* y posteriormente en 2009 apareció *Bitcoin*, llevando a una revolución no solo en el ámbito económico, sino también en el tecnológico con su protocolo *blockchain* donde se comprobó su seguridad y robustez. La tecnología *blockchain* es una de las tecnologías con mayor potencial disruptivo en los últimos años, permite implementar una base de datos distribuida, pública e inmutable basada en una secuencia creciente de bloques. Los nodos en la cadena de bloques cuentan con múltiples transacciones y réplicas de los datos, por lo que se considera como un registro de transacciones ordenado, eficaz y seguro.

En los últimos años la revolución tecnológica de *blockchain* ha provocado su expansión a ritmos sin precedentes, el desarrollo ha crecido de manera exponencial, sin embargo, el desarrollo rápido de DApps, conlleva a un desarrollo de software apresurado y sin control, lo cual resulta en un escenario que no garantiza un modelado de software adecuado por lo que la calidad del software que se obtiene es deficiente. Para la obtención de un modelado adecuado y de software de calidad, R. S. Pressman [2] habla de principios de ingeniería haciendo énfasis en el proceso y en los métodos, es decir, en los aspectos prácticos. Anthony I. Wasserman sugiere que existen ocho nociones fundamentales en la Ingeniería de Software que forman la base para una disciplina efectiva [3]:

abstracción, métodos y notaciones de análisis y diseño, prototipo de la interfaz de usuario, modularidad y arquitectura del software, proceso y ciclo de vida, reutilización, métricas, y herramientas y entornos integrados.

La ingeniería de software es una parte fundamental para el desarrollo de software por lo cual se hace uso de esta disciplina tomando en cuenta las técnicas, herramientas y artefactos reportados e identificados durante el estudio de trabajos relacionados con la ingeniería de software orientada a *blockchain*. El primer paso para desarrollar un sistema de software utilizando prácticas sólidas de ingeniería de software es tener un proceso de desarrollo claro y diseñar prácticas y notaciones útiles para el propósito [4]. Con base en esto, se utilizan prácticas específicas de desarrollo, prueba, implementación y evaluación de seguridad, asimismo realizar un modelado de aplicaciones descentralizadas adecuado para la obtención de un desarrollo de software completo.

El presente artículo está constituido por secciones, la sección 2 aborda una revisión de investigaciones y publicaciones enfocadas en el uso de la tecnología *blockchain* (Trabajos relacionados); en la sección 3 se muestran los antecedentes sobre artefactos utilizados para modelar BOS; en la sección 4, se hace mención de caso de uso y se muestra el modelado del sistema; en la sección 5, se muestra la propuesta del método para el desarrollo de aplicaciones descentralizadas; en la sección 6 se realiza una discusión de los resultados y por último en la sección VII se muestran las conclusiones del presente trabajo.

2 Trabajos relacionados

En [4] se hizo frente a los problemas que se tienen durante el desarrollo de software con la tecnología de *blockchain*, originados por un desarrollo apresurado y sin reglas, en el cual no se tienen en cuenta conceptos básicos de ingeniería de software. Para solucionar estos problemas se propuso un proceso de desarrollo de software, el proceso se basa en prácticas ágiles. Sin embargo, también se hace uso de diagramas UML para ayudar a modelar contratos inteligentes (*smart contracts*), donde se introdujeron algunos conceptos nuevos para modelarlos y especificarlos adecuadamente. En algunos casos se introdujeron estereotipos UML y en otros, se introdujo una notación específica.

En [5], se menciona que actualmente no se reporta un estándar de diseño para modelar BOS, debido a esto, los desarrolladores tienen dificultades para planificar su BOS, dado que el modelo es una parte importante del diseño para modelar BOS. La falta de una notación especializada complica la adopción o migración de este nuevo paradigma de software, ya que la interacción entre *blockchain* y la aplicación no se especifica correctamente. Por ello, se presentaron tres enfoques de modelo complementarios para BOS basados en estándares de modelo: Modelo E-R (*Entity Relation*, Entidad Relación), UML y BPMN (*Business Process Model and Notation*, Modelo y Notación de Procesos de Negocio). Se describió un escenario de aplicación donde se identifica que todos los modelos tienen sus ventajas y desventajas al especificar BOS.

En [6] se discutió el problema que sufrió *Ethereum*, un *hack* supuestamente involuntario, en el cual un desarrollador congeló varias cuentas administradas por la aplicación *Parity Wallet*. Lo anterior representa un ejemplo paradigmático de los problemas que actualmente surgen en el desarrollo de contratos inteligentes y software en general relacionado con *blockchain*. Dichos problemas están

asociados por la falta de prácticas estandarizadas para la ingeniería de software *blockchain*. Por ello se realizó un estudio y se presentó un enfoque que hubiese ayudado a mitigar los efectos del ataque, basándose en las mejores prácticas aceptadas en la ingeniería de software tradicional. Se necesitan nuevos roles profesionales, seguridad y confiabilidad, marcos de modelado y verificación, y métricas especializadas para llevar las aplicaciones *blockchain* a un nuevo nivel confiable. Al menos tres áreas se destacan para comenzar a abordar: mejores prácticas y metodología de desarrollo, patrones de diseño y pruebas.

En [7], se recopiló una serie de patrones para aplicaciones basadas en *blockchain*, debido a que la tecnología *blockchain* se encuentra en una etapa muy temprana y existe una falta de visión sistemática y holística sobre el diseño de sistemas de software que usan *blockchain*. *Blockchain* requiere patrones para usar en el diseño de su arquitectura de software, un patrón de diseño es una solución reutilizable a un problema que ocurre comúnmente dentro de un contexto dado durante el diseño de software. Un patrón de diseño define restricciones que limitan los roles de los elementos arquitectónicos (procesamiento, conectores y datos) y la interacción entre esos elementos. La adopción de un patrón de diseño provoca compensaciones entre los atributos de calidad. La colección de patrones recopilados se divide en cuatro categorías: interacción entre *blockchain* y el mundo externo, gestión de datos, seguridad y estructurales del contrato.

3 Antecedentes

La lógica de negocios en *blockchain* se implementa mediante el uso de contratos inteligentes, los contratos inteligentes son similares a las clases a las que llaman las aplicaciones cliente fuera de *blockchain*. Por lo tanto, es posible desarrollar BOS que implemente la lógica de negocios en *blockchain* mediante el uso de contratos inteligentes.

Mediante el estudio y análisis de los trabajos relacionados se identificaron algunos artefactos para el desarrollo aplicaciones descentralizadas. De la misma forma, se encontraron dos problemas específicos al introducir la tecnología de *blockchain* en el desarrollo de software:

- La tecnología *blockchain* se encuentran en una etapa temprana, por lo cual existe la falta de una visión sistemática y holística sobre el diseño de sistemas de software que usan *blockchain*.
- Actualmente no se reporta un estándar de diseño para modelar software orientado a *blockchain* y hay un desarrollo de software sin control y apresurado, en el cual, no se tienen en cuenta conceptos básicos de la ingeniería de software, por consecuencia no se asegura la calidad de software [5].

Debido a los problemas mencionados, se busca dar solución a dichos problemas tomando en cuenta los artefactos identificados en los trabajos antes mencionados para realizar un modelado adecuado. A continuación, se muestran los artefactos identificados en la Tabla 1, para ayudar a modelar BOS, se identificó el uso de diagramas UML, Modelo ER, y Notación BPMN. Sin embargo, para algunos diagramas se introducen nuevos conceptos como estereotipos UML.

Tabla 1. Artefactos identificados para modelar Software orientado a *blockchain* [4] [5] [6].

Artefacto	Descripción
Diagrama de casos de uso	No presenta cambios, ya que este diagrama representa la forma en que el cliente opera con el sistema.
Diagrama de clases	Representa la estructura y las relaciones de los contratos inteligentes, se introdujeron varios estereotipos en este tipo de diagrama. Tiene la ventaja de modelar atributos y funciones. De igual manera se propone el uso de un ícono llamado "*chain*" para la representación gráfica del contrato como una clase.
Diagrama de estado	Representa los diversos estados de un contrato inteligente; este diagrama no necesita ningún concepto nuevo.
Diagrama de secuencia	Representa los mensajes enviados a un contrato inteligente, y de un contrato inteligente a otro contrato inteligente. Si se hace uso de la plataforma *Ethereum*, el diagrama de secuencia necesita un nuevo tipo de mensaje: la transferencia de *ethers*.
Modelo ER	Es adecuado para capturar los datos entre *blockchain* y una base de datos privada relacional.
Notación BPMN	Es adecuada para especificar el proceso empresarial.

De igual manera, se identificaron una serie de patrones que se clasifican en cuatro tipos: Interacción de patrones y el mundo externo, gestión de datos, seguridad y patrones estructurales del contrato. Algunos patrones están diseñados teniendo en cuenta la naturaleza de *blockchain* y cómo se pueden introducir específicamente en aplicaciones del mundo real. Otros son variantes de patrones de diseño existentes aplicados en el contexto de aplicaciones basadas en *blockchain* y contratos inteligentes [8].

4 Caso de estudio y modelado

Se presenta a continuación el caso de estudio, el tipo y la plataforma *blockchain*, con ello se realiza el modelado correspondiente de la DApp.

Caso de estudio

Definición del caso de estudio: el sistema actual de transporte urbano presenta problemas con el monitoreo de autobuses y conductores, debido a que no se tiene un control total de los autobuses y conductores. Por ejemplo, no se tiene el control de que el conductor cumpla en tiempo y forma la ruta asignada durante la jornada laboral o que el conductor modifique su ubicación mediante algún

tipo de software o hardware. Debido a los problemas mencionados se tienen pérdidas económicas dentro del sector de transporte urbano.

Solución del caso de estudio: se plantea un sistema en donde se monitoreará a los conductores y autobuses mediante su ubicación. Los datos se envían a la red *blockchain*, es decir se guarda la ubicación del conductor y del autobús, las ubicaciones se guardan de manera constante para realizar un rastreo del recorrido del conductor durante su jornada laboral, además de esto deberán coincidir las ubicaciones del autobús con las del conductor. Debido a que los datos de la ubicación se guardan en la red *blockchain*, los datos no podrán ser alterados debido a la inmutabilidad de la tecnología *blockchain*.

Plataforma

Blockchain se clasifica en tres tipos [8]: *blockchain* público, privado y autorizado (consorcio). La clasificación se basa en la capacidad del nodo para acceder o agregar un nuevo bloque en la red. En *blockchain* público, cualquier nodo puede unirse a la red, participar en las transacciones y crear los bloques. Por otro lado, tanto los permisos de lectura y escritura como la participación en la red están restringidos en *blockchain* privado. El tercer tipo de *blockchain*, es una solución intermedia entre *blockchain* privado y público, se conoce como consorcio o *blockchain* autorizado. En este tipo de cadena de bloques, la red está controlada por un grupo de nodos.

Para la solución al caso de estudio se hace uso del tercer tipo de *blockchain*, mediante el uso de la plataforma *Hyperledger Fabric*. *Hyperledger Fabric* es una plataforma de código abierto de nivel empresarial mantenida por IBM y Linux *Foundation.* A diferencia de *Bitcoin* y *Ethereum*, *Hyperledger Fabric* no tiene ninguna moneda digital (criptomoneda), donde el acceso a la red está restringido solo a los miembros de la red, y nadie puede unirse a la red sin autorización. El mecanismo utilizado para validar las transacciones y crear bloques en *Hyperledger Fabric* es PBFT (*Practical Byzantine Fault Tolerance*, tolerancia práctica a fallas bizantinas). Las transacciones se controlan en *Hyperledger Fabric* utilizando *chaincode* (contrato inteligente), que es un código de programación que proporciona la capacidad de escribir y diseñar las aplicaciones para interactuar con la red. Una transacción es una invocación o una solicitud de instancia que el par envía para su pedido y validación. La solicitud de instancia inicializa un *chaincode* en un canal en particular, mientras que las transacciones de invocación ejecutan una operación de lectura / escritura en el libro mayor.

Modelado

Con base en la sección III se seleccionaron los artefactos que muestran mayor utilidad para llevar a cabo el modelado de la DApp dado el caso de estudio.

El modelado debe ser capaz de representar la información que el software transforma, la arquitectura y las funciones que permiten que esto ocurra, las características que desean los usuarios y el comportamiento del sistema. Los modelos deben cumplir estos objetivos en diferentes niveles de abstracción. En el trabajo de ingeniería de software hay dos tipos de modelos [9]:

1. Modelo de requerimientos: Representa los requerimientos del cliente.
2. Modelo de diseño: representa características del software que ayudan a la construcción del software, como: arquitectura, interfaz de usuario y detalle en el nivel de componente.

El modelado incluye el análisis, el diseño y describe una representación más detallada del software. El objetivo del modelado es afirmar el entendimiento del trabajo que se va a hacer y dar una guía a quienes lo implementarán.
Con base en los artefactos identificados y los tipos de modelos para ingeniería de software, se crean los siguientes modelos respecto al caso de estudio: modelo basado en escenarios (diagrama de casos de uso y descripción de casos de uso), modelo de comportamiento (diagrama de secuencia) y modelo de diseño (arquitectura). Se muestran los modelos creados para entender mejor la entidad real que se construye, sin embargo, antes de modelar se necesita la ingeniería de requerimientos, esta disciplina es importante dentro de la ingeniería de software. Para iniciar con el modelado se realiza una descripción completa del sistema, donde se identifican funciones del sistema, los actores que interactuaran con el sistema y las diversas acciones que hará dentro del sistema.

Modelo basado en escenarios

Diagrama de casos de uso

Alistair Cockburn [10] afirma que "un caso de uso conlleva a hacer un contrato" describe el comportamiento del sistema en distintas condiciones en las que el sistema responde a una petición de alguno de sus participantes. En esencia, un caso de uso narra una historia estilizada sobre cómo interactúa un usuario final con el sistema en circunstancias específicas.

Para el caso de estudio, el sistema consta de dos actores: administrador y conductores y de una serie de casos de uso, los cuales se muestran en la Figura 1.

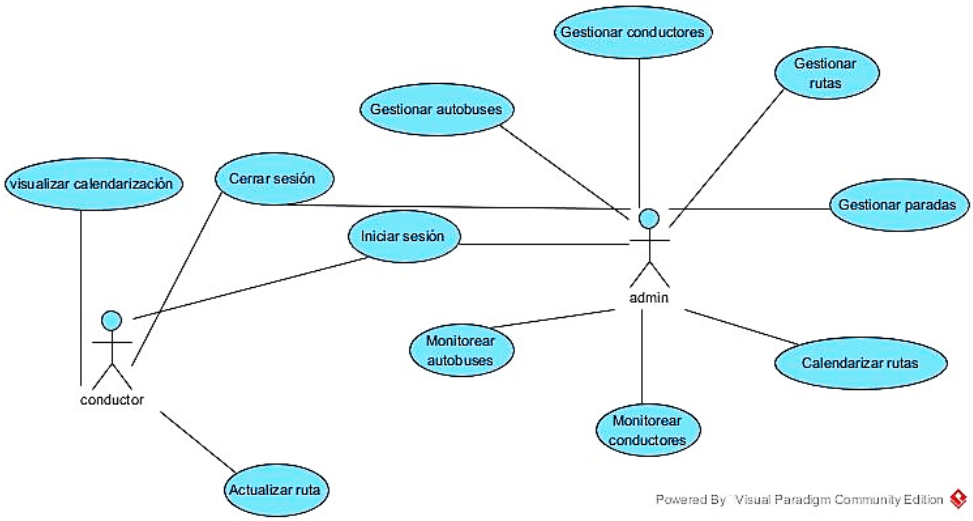

Figura 1. Diagrama de casos de uso para el sistema de transporte urbano.

Posteriormente se realiza una descripción de cada caso de uso, el cual describe de manera específica las actividades que realiza el actor para llevar a cabo un proceso dentro del sistema. En la Figura 3 se muestra el caso de uso monitoreo de autobús.

Modelo de comportamiento

Diagrama de secuencia

Indica la forma en la que los eventos provocan transiciones de un objeto a otro. Una vez identificados los objetos por medio del análisis del caso de uso, el modelador crea el diagrama de secuencia representando el modo en el que los eventos causan el flujo de uno a otro como función del tiempo. Con base en el caso de estudio se muestra como ejemplo el diagrama de secuencia, monitoreo del autobús en la Figura 2.

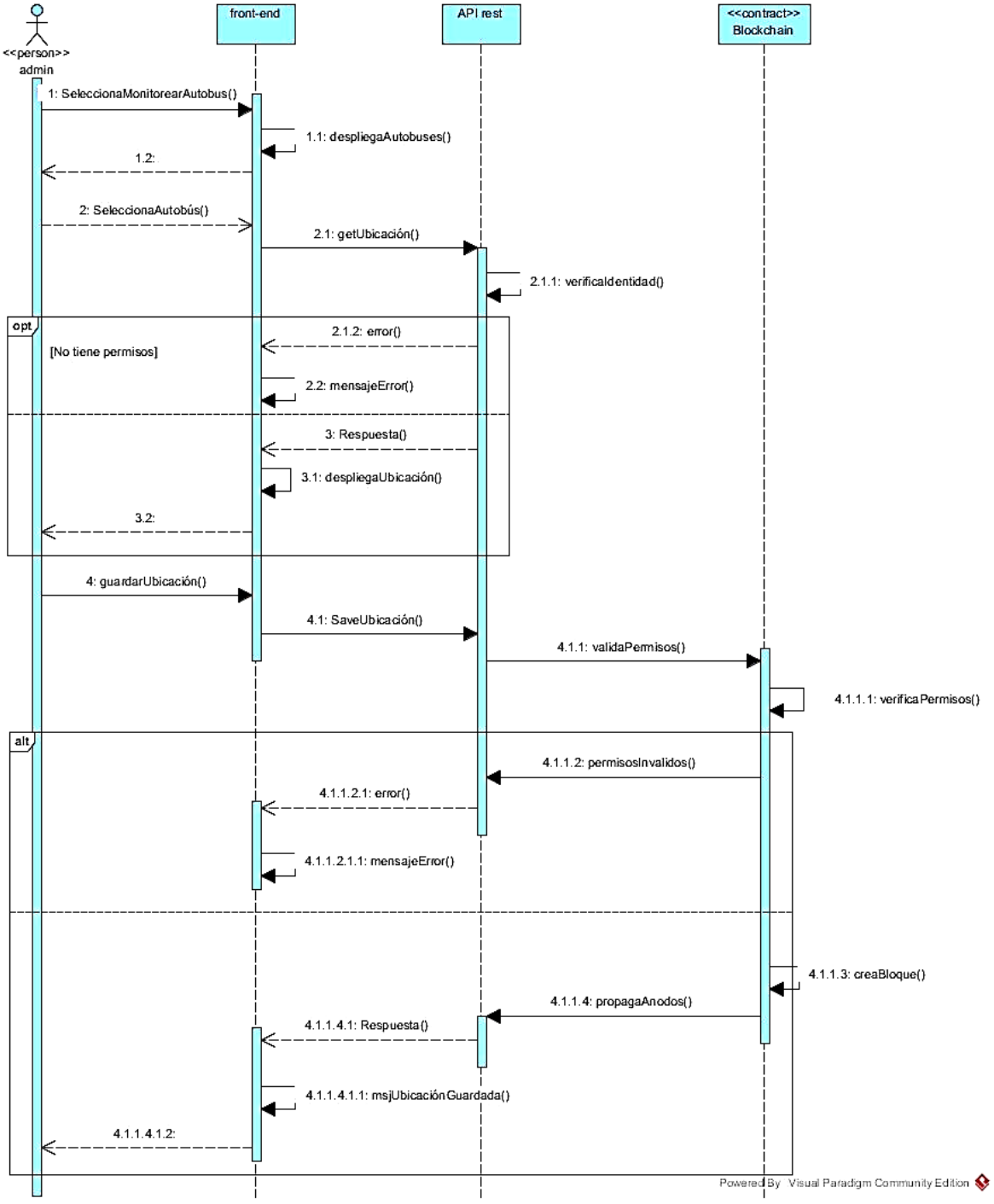

Figura 2. Diagrama de secuencia de monitoreo de autobús.

CDU-07	Monitorear autobús	
Actor	Administrador.	
Objetivo	Monitorear la ubicación del autobús en tiempo real y registrar la ubicación constante dentro de la red *blockchain*.	
Pre-condición	El usuario ha iniciado sesión. Ingresar al sistema como administrador. El usuario se encuentra en la pantalla de monitoreo.	
Curso básico	**Actor**	**Sistema**
	1.- El usuario selecciona un autobús.	
		2.- El sistema muestra los autobuses en ruta.
	3.- El usuario selecciona un autobús a monitorear.	
		4.- El sistema muestra mediante un mapa la ubicación actual del autobús.
	4.- El usuario selecciona guardar ubicación automáticamente del autobús.	
		5.- Se guarda la ubicación del autobús en la *blockchain* y en la base de datos.
Post-condiciones	El informe se mostrará en una tabla (hora y ubicación).	

Figura 3. Descripción de caso de uso monitoreo de autobús.

Modelo de diseño

Arquitectura

Pressman [2], menciona que un diagrama de arquitectura ayuda a plantear una vista completa del sistema que se construirá. Muestra la estructura y organización de los componentes de software.

Para el diseño de la arquitectura de la DApp se hace uso del patrón arquitectónico en capas, el cual está constituido por 5 capas: capa de aplicación, capa de contratos inteligentes (lógica de negocio), capa de libro mayor distribuido, capa de base de datos y capa de seguridad representadas en la Figura 4.

Descripción de la arquitectura:

Capa de aplicación

La capa de aplicación hace referencia a la interacción entre el usuario y el software.
La interfaz de usuario del lado del cliente se implementa utilizando Angular 7 y la API en Node.js. El paquete para el teléfono inteligente se construye desde el mismo software utilizando el marco de aplicación móvil llamado Ionic [11]. Esto permite utilizar la misma base de código para Web y móvil, lo cual reduce tiempo y costo.

Angular 7: componente que corresponde con el *front-end* del sistema. A través de este componente los clientes realizan el acceso al sistema, el *front-end* consultará datos a la API y a su vez permitirá hacer peticiones *post* para enviar transacciones a la cadena de bloques.

El SDK de *Fabric*: ayuda a establecer la comunicación entre el *front-end* y el *back-end,* mediante el SDK se ejecuta el *chaincode* del usuario y se realizan transacciones en la red. Para la ejecución de una transacción, el cliente se conecta a la red de *Hyperledger Fabric* a través del SDK, el cliente crea una transacción y la envía al par de aprobación. El SDK de *Hyperledger Fabric* se usará en los servicios **REST** desarrollados para interactuar con la red *blockchain*.

API REST: provee de una fachada de servicios REST que interactuaran con la *blockchain Hyperledger* a través de la ejecución de los *chaincode*. La API REST utiliza HTTP para peticiones de datos *get, put, post y delete.* Un escenario de petición HTTP funciona así:

1. Un cliente envía una petición HTTP a un servidor.
2. El servidor devuelve una respuesta HTTP.

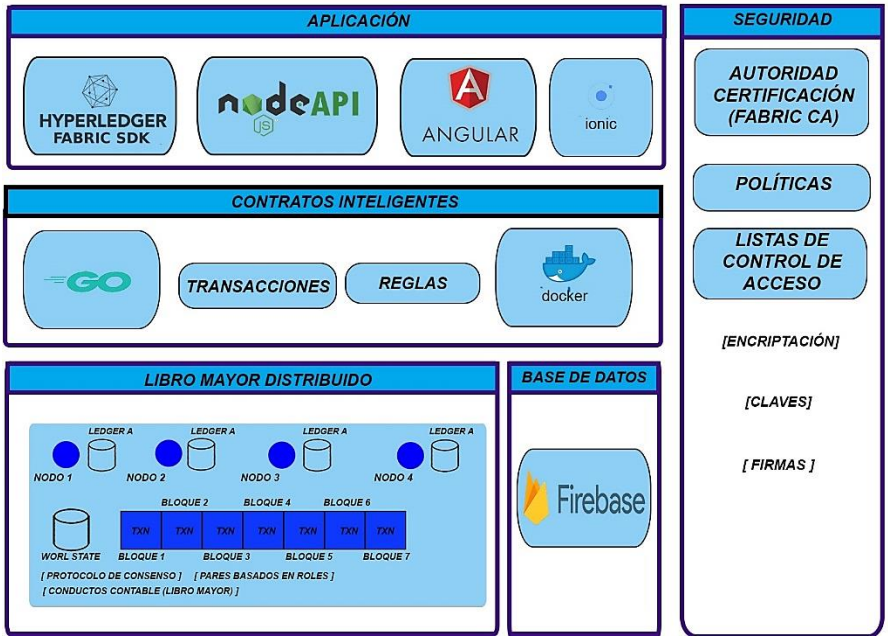

Figura 4. Arquitectura de DApp de autobuses urbanos.

Contrato inteligente

Un contrato inteligente en *Hyperledger Fabric* se llama *chaincode*. Es un programa, codificado en *Go*, para este caso. *Chaincode* se ejecuta en un contenedor *Docker* aislado. Esto brinda la capacidad de utilizar la API existente y facilitar la migración. El propósito de *chaincode* es ser la capa empresarial en el desarrollo de software. Al implementar un nuevo *chaincode*, es posible asignar una política, de la cual los *peers* deben firmar las transacciones que ejecutan este *chaincode*. *Chaincode* se puede implementar mediante el SDK.

Libro mayor

El libro mayor consta de dos componentes:

- Registro de transacciones

- *World state* (estado mundial)

Las transacciones cambian el estado mediante el uso de *chaincode*. La implementación de un nuevo *chaincode* se considera una transacción. El *chaincode* se firma y el sistema crea un paquete inmutable. Se crea una imagen de *Docker* separada y se ejecuta como una máquina separada.

Hyperledger Fabric depende de certificados, por lo que se proporciona un servicio separado llamado *Fabric* CA (*Certificate Authority*, Autoridad certificada) para generar dinámicamente certificados para los usuarios.

Seguridad

Una de las partes fundamentales que proporciona la tecnología *blockchain* es la seguridad. Para esto la aplicación necesita también lo que se denomina tarjetas o *business network cards*, que contienen las credenciales, las claves, los certificados y un perfil de conexión, necesario para conectar a las aplicaciones y componentes de la infraestructura de *Hyperledger*.

Los usuarios dispondrán de un certificado de clave pública que será empleado por la *blockchain* para validar las transacciones, el certificado de los usuarios se incluirá de modo que se realizará la validación del certificado cliente en el servidor.

Componente *Fabric* CA: asigna certificados, donde el usuario hace una petición para obtener el certificado, la autoridad de registro valida la identidad del participante y manda la petición a la autoridad de certificación, que asigna el certificado que es remitido al participante que lo había solicitado. Una vez que el participante inicia una transacción, un componente de la infraestructura de *Hyperledger Fabric* valida la autenticidad de la transacción comprobando la validez del certificado con la autoridad de validación.

Hyperledger Fabric también dispone de la opción de configurar ACL (*Access Control List*, Lista de Control de Acceso), permite controlar el acceso a los recursos de la *blockchain* mediante la asignación de políticas a las diferentes identidades. Estas políticas se asocian a nivel de recurso, donde un recurso es un *chaincode* de usuario, un *chaincode* de sistema o un *event resourse.*

Estructura de Base de Datos

Para obtener escalabilidad, integridad y confiabilidad de los datos, se hace uso de una base de datos No SQL (*Structured Query Language*, lenguaje de consulta estructurada). Esta base de datos aporta mayor flexibilidad, velocidad y manejabilidad, características necesarias para la creación del *blockchain*. El motor de base de datos seleccionado es Firebase.

Posteriormente se realizó el diseño de la interfaz de usuario, tomando en cuenta todo lo anterior, la interfaz de usuario permitió crear y ver el medio de comunicación entre el usuario y la máquina; posteriormente la interfaz de usuario ayudara a tener una idea para la construcción de la misma. Actualmente el caso de estudio se encuentra en la realización de pruebas con las API´s de Hyperledger Fabric y el chaincode, es decir se encuentra en el desarrollo de la lógica de negocio.

5 Método para el desarrollo de DApp

Con base en la sección anterior, se realizó el método propuesto para el desarrollo de la aplicación descentralizada, que se muestran en la Figura 5.

Se descarta el uso del modelo ER debido a que en el caso de estudio se utiliza una base de datos NoSQL (no relacional) y el modelo ER es bueno para utilizarlo con una base de datos relacional.

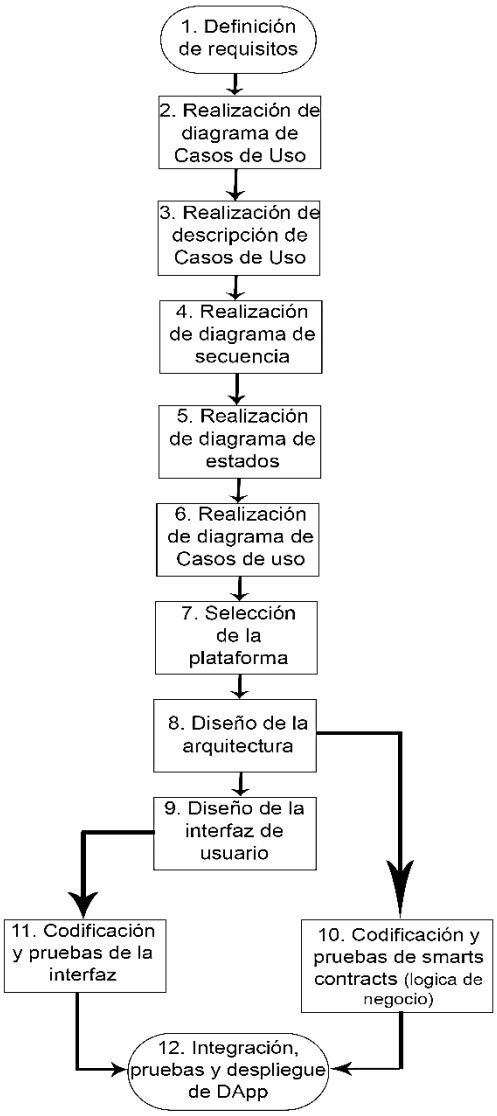

Figura 5. Método para el desarrollo de DApps.

Pasos para el desarrollo de DApps

1. Definición de los requisitos: descripción completa del comportamiento del sistema que se desarrollará.
2. Realización de casos de uso: con base a la definición de los requisitos se identifican los actores que interactuaran con el sistema y la forma en que estos operan el sistema.
3. Realización de la descripción de casos de uso: definición y descripción de la forma en que los actores interactuaran con el sistema, para cada caso de uso se realiza una descripción.

4. Realización de diagramas de secuencia: define la forma en que interactuaran o comunicaran los objetos con *blockchain*.
5. Realización del diagrama de estados: representación de los diversos estados del sistema y definición del comportamiento del sistema: descripción de la estructura del sistema y comunicación con los contratos inteligentes.
6. Selección de la plataforma *blockchain*: realización de una investigación y evaluación para hacer la selección del tipo de plataforma (privada, pública o consorcio) para este caso se hace uso de una plataforma *blockchain* consorcio (*Hyperledger Fabric*).
7. Diseño de la arquitectura: con base en el modelado que se realizó del sistema y la plataforma seleccionada, se diseña la arquitectura para plantear una vista clara y concisa del sistema a construir.
8. Diseño de la interfaz de usuario: Tomando en cuenta todo lo anterior se realiza el diseño de la interfaz de usuario, la cual permite crear y ver el medio de comunicación entre el usuario y la máquina.
9. Codificación y pruebas de contratos inteligentes: El contrato inteligente hace referencia a la lógica de negocio, los contratos inteligentes se codifican en el lenguaje de programación *Go*.
10. Codificación y pruebas de UI: Hace referencia al *front-end*, es decir a la interacción entre el usuario y el sistema, éste será codificado mediante el marco de trabajo Angular 7.
11. Integración, pruebas y despliegue de la DApp: para la integración se hace uso del SDK de *Fabric*, el cual ayuda a establecer la comunicación entre el *front-end* y el *back-end,* mediante el SDK se ejecuta el *chaincode* del usuario y se realizan transacciones en la red, la actividad de despliegue incluye: entrega, apoyo y retroalimentación.

6 Discusión y resultados

El método para el desarrollo de DApps presenta grandes ventajas debido a que está basado en la ingeniería de requerimientos, identificando los requerimientos y realizando el modelado correspondiente. Para el modelado de requerimientos se tomaron en cuenta los artefactos identificados y los elementos del modelo de requerimientos, llamado modelo de análisis. El modelo de requerimientos consta de cuatro capas o modelos, donde se incluyen los artefactos identificados para el modelado de DApps, dichas capas y artefactos se mencionan a continuación:

1. Modelo basado en el escenario: Diagrama de casos de uso y descripción de caso de uso.
2. Modelo de comportamiento: Diagrama de secuencia y diagrama de estados.
3. Modelo de clase: Diagramas de clase.
4. Modelo de flujo: Diagrama de flujo de datos.

Sin embargo, este último modelo no se consideró debido a que este artefacto no se encontró útil para el modelado de DApps. Para el modelo de clase es útil el diagrama de clases, se encontraron una serie de estereotipos para realizar este diagrama, sin embargo, se encontró una desventaja. Para el diagrama de clases cambia la forma de modelar contratos inteligentes, con base en la plataforma y lenguaje de programación, además de ello, se necesita conocer plenamente cómo trabaja, funciona y se relaciona con el mundo exterior.

También se contempló el modelo de diseño, donde se realizó el diseño de la arquitectura, la cual permite plantear una vista completa del sistema que se construirá. Mostrando y comprendiendo mejor la estructura y organización de los componentes de la DApp.

Con lo anterior se permite al desarrollador comprender los elementos y componentes que se requieren para construir el sistema de manera adecuada, asimismo comprende mejor la estructura del sistema antes del diseño de interfaz de usuario y de la codificación, eliminando problemas durante el desarrollo del sistema y generando software de calidad.

7 Conclusiones

El método muestra ser flexible y agiliza el desarrollo de software orientado a *blockchain*, además ayuda a tener un panorama más amplio del sistema a construir debido a que en parte de este método se lleva a cabo el modelado de requerimientos y el modelado de diseño, los artefactos identificados muestran una gran utilidad en el modelado de requerimientos ya que estos se adaptan al modelo de requerimientos que R. S. Pressman presenta.

Los artefactos identificados dentro del análisis de los trabajos relacionados resultaron ser útiles para el modelado de la aplicación descentralizada, respecto al caso de estudio, dichos artefactos fueron incorporados dentro el método propuesto. De los estereotipos que se encontraron para modelar correctamente solo se hace uso de los estereotipos *<<contract>>* y *<<person>>* en el diagrama de secuencia, el diagrama que más cambios presenta es el diagrama de clases, sin embargo se encontró una desventaja para el modelo de clases, debido a que el modelado cambia respecto al tipo de *blockchain* a utilizar (público, privado o consorcio). Teniendo definido lo anterior mencionado se observa que los pasos posteriores del método, enfocados hacia la construcción del software se entienden mejor debido a que ya se tiene un panorama más amplio y se ha planteado una vista completa del software, lo cual conlleva a disminuir los problemas durante el desarrollo, obteniendo el sistema en tiempo y forma, y obteniendo software de calidad.

Respecto a los trabajos relacionados, se hizo uso y se realizaron pruebas de los artefactos propuestos para el modelado de aplicaciones descentralizadas, identificando ventajas y desventajas estos artefactos, en estos trabajos se presenta una descripción de los artefactos propuestos y en otros casos se ejemplifican. A diferencia del presente proyecto en el cual se tomaron en cuenta todos los artefactos de los diversos trabajos relacionados y se realizaron las pruebas mediante un caso de estudio y se hace uso de un tipo de *blockchain* especifico (consorcio), específicamente *Hyperledger Fabric*, además de ello, se muestra un método funcional para el desarrollo de aplicaciones descentralizadas con *Hyperledger Fabric* incorporando los artefactos que presentan mayores ventajas.

Agradecimientos

Los autores agradecen al Consejo Nacional de Ciencia y Tecnología (Conacyt), así como al Tecnológico Nacional de México (TecNM) por el apoyo otorgado para la realización de la presente investigación.

Referencias

[1] S. Nakamoto, «Bitcoin: A Peer-to-Peer Electronic Cash System», p. 9.

[2] R. S. Pressman, *Software engineering: a practitioner's approach*, 7th ed. New York: McGraw-Hill Higher Education, 2010.

[3] F. Zambonelli, «Towards a Discipline of IoT-Oriented Software Engineering», p. 7.

[4] M. Marchesi, L. Marchesi, y R. Tonelli, «An Agile Software Engineering Method to Design Blockchain Applications», en *Proceedings of the 14th Central and Eastern European Software Engineering Conference Russia on ZZZ - CEE-SECR '18*, Moscow, Russian Federation, 2018, pp. 1-8, doi: 10.1145/3290621.3290627.

[5] H. Rocha y S. Ducasse, «Preliminary steps towards modeling blockchain oriented software», en *Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain - WETSEB '18*, Gothenburg, Sweden, 2018, pp. 52-57, doi: 10.1145/3194113.3194123.

[6] G. Destefanis, M. Marchesi, M. Ortu, R. Tonelli, A. Bracciali, y R. Hierons, «Smart contracts vulnerabilities: a call for blockchain software engineering?», en *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, Campobasso, mar. 2018, pp. 19-25, doi: 10.1109/IWBOSE.2018.8327567.

[7] X. Xu, C. Pautasso, L. Zhu, Q. Lu, y I. Weber, «A Pattern Collection for Blockchain-based Applications», en *Proceedings of the 23rd European Conference on Pattern Languages of Programs - EuroPLoP '18*, Irsee, Germany, 2018, pp. 1-20, doi: 10.1145/3282308.3282312.

[8] Q. Nasir, I. A. Qasse, M. Abu Talib, y A. B. Nassif, «Performance Analysis of Hyperledger Fabric Platforms», *Security and Communication Networks*, vol. 2018, pp. 1-14, sep. 2018, doi: 10.1155/2018/3976093.

[9] R. S. Pressman, *Ingeniería del software: un enfoque práctico*. 2013.

[10] I. Jacobson, «1 Un poco de historia en la progra- mación», p. 4.

[11] Ionic, «Ionic Framework - Ionic Documentation», *Ionic Docs*. https://ionicframework.com/docs/ (accedido may 17, 2020).