Nicolás Sánchez Gómez

Aplicación de los principios de las pruebas tempranas durante el ciclo de vida de desarrollo de los smart contracts en la tecnología blockchain

- Ph. D. Tesis -

Directores
Ph. D. Jesús Torres Valderrama
Ph. D. Manuel Mejías Risoto

Tutora
Ph. D. María José Escalona Cuaresma

Departamento de Lenguajes y Sistemas de Información
E.T.S. de Ingeniería Informática. Universidad de Sevilla

Sevilla, octubre de 2023

A Merchi, mi compañera de vida, si no te hubiera conocido te habría soñado, y a nuestros retoños Nicolás y Manuel, nuestros pilares, la luz que ilumina nuestros días, con todo mi cariño.

Índice de Contenido

Índice de Figuras

Índice de Tablas

Reconocimientos

Son muchas las personas que han participado directa o indirectamente para hacer posible la presente Tesis Doctoral. Estas breves, pero sentidas, líneas están dedicadas a todas ellas.

Mi tutora, María José Escalona, por su vitalidad, esfuerzo, entusiasmo, dinamismo, … y, sobre todo, su apoyo sin fin, que me ha permitido encontrar y, sobre todo, aprovechar las oportunidades para realizar la presente Tesis Doctoral.

Mis directores, Jesús Torres y Manuel Mejías, por ayudarme con sus valiosos consejos y sus numerosas recomendaciones y aportaciones hasta el último momento, que me han permitido orientar, organizar, desarrollar y armonizar de forma fructífera este trabajo de investigación.

El Grupo de investigación Engineering and Science for Software Systems (ES3) de la Universidad de Sevilla, por los momentos vividos y compartidos - y los que quedan - y, por todo lo que he aprendido sobre la investigación, ese trabajo creativo y sistemático que implica la recopilación, organización y análisis de información, que me ha permitido la comprensión y análisis de un problema previamente identificado. Os quiero agradecer a todos/as, los/las que estáis y a los/las que habéis estado, vuestra amistad, compañerismo, empuje y, sobre todo, haber podido compartir juntos esta experiencia - que no habría tenido el mismo resultado sin vosotros -.

David Lizcano por la ayuda prestada en todo momento y por promover y, sobre todo, facilitar mi estancia de investigación en la Universidad a Distancia de Madrid, UDIMA (España).

Alejandra Garrido por el apoyo decidido y constante que he recibido de ella y de todos sus compañeros/as, durante los meses de estancia de investigación en el LIFIA, Centro de Investigación perteneciente a la Facultad de Informática de la Universidad Nacional de La Plata, y vinculado a la Comisión de Investigaciones Científicas de la Provincia de Buenos Aires (Argentina).

Mis amigos/as, todos/as, y mi familia, por tener paciencia conmigo en lo bueno y, sobre todo, en lo malo, aguantando mi carácter y mis impertinencias, a veces fuera de lugar.

Y, por último, no por ello menos importante para mí, a mis padres, Nicolás y Puri, por la enseñanza que me han dado y, sobre todo, por el respaldo y cariño que siempre he recibido y sigo recibiendo de ellos.

Prólogo

Esta investigación se ha desarrollado en el seno del grupo de investigación Engineering and Science for Software Systems (ES3) del Departamento de Lenguajes y Sistemas Informáticos (LSI) de la Escuela Técnica Superior de Ingeniería Informática (ETSII) de la Universidad de Sevilla, grupo de trabajo al que me incorporé hace ya más de una década.

Mi incorporación, realmente, fue al grupo de investigación Ingeniería Web y Testing Temprano (IWT2), pero decidimos cambiar su nombre al objeto de adecuarlo a los últimos trabajos de investigación y de transferencia de resultados de investigación que se estaban abordando en su seno. Desde 2022, pasó a llamarse grupo de investigación ES3. No obstante, desde sus orígenes, las principales líneas de trabajo de este grupo de investigación han sido la ingeniería de requisitos, la ingeniería de pruebas, el testing software y el aseguramiento de la calidad de los proyectos durante el ciclo de vida del desarrollo de software, a través de la puesta en marcha y establecimiento de Oficinas de Gestión de Proyectos (OGP), también conocida por sus siglas en inglés, PMO (*Project Management Office*).

En el ámbito del grupo ES3, la presente Tesis Doctoral se mueve en el contexto de la ingeniería del software y, más en concreto, en el aseguramiento de la calidad del software durante el ciclo de vida del desarrollo de software, mediante la aplicación de los principios generales del testing temprano. Todo ello, aplicado en una de las tendencias tecnológicas más disruptivas actualmente - dado su gran potencial de aplicación en diversos sectores -, como es la tecnología blockchain y, su pilar, los smart contracts (contratos inteligentes).

Resumen

Los sistemas software son cada vez más multidisciplinares y complejos. Su implementación de forma satisfactoria se ha convertido en un desafío continuo para cualquier tipo de empresa u organismo. Llevar a cabo la "transformación digital" de una empresa u organización implica, entre otras cosas, la adopción de tecnologías digitales avanzadas al objeto de mejorar su funcionamiento, eficiencia, productos y servicios. Es decir, para las empresas y organizaciones supone la aceptación e integración de tecnologías como internet, la nube y los servicios en línea, el internet de las cosas, la automatización de procesos, el análisis masivo de datos, la inteligencia artificial, y otros avances tecnológicos.

En este contexto, hace ya unos años, apareció la tecnología blockchain. Ésta constituye también una de esas tecnologías que están impulsando la comentada "transformación digital", gracias a sus características únicas y su potencial para abordar ciertas limitaciones de los sistemas tradicionales. Además, dado su carácter transversal[1], esta tecnología está permitiendo la disrupción en la economía y en la empresa más allá de las conocidas criptomonedas. Solo habría que realizar una búsqueda rápida por internet para evidenciar que estamos asistiendo a una importante apuesta del mercado hacia los desarrollos basados en esta tecnología disruptiva. Esto es debido, fundamentalmente, a su capacidad para transformar la forma en que se registran las transacciones y la manera en que se almacenan y recuperan los datos.

[1] Se adapta a numerosos sectores tales como agricultura, industria, finanzas, administración pública, seguros, legal, logística, automoción, aeroespacial, etc. (prácticamente a todos).

En la tecnología blockchain, los llamados smart contracts (contratos inteligentes) podrían actuar como complemento o sustituto de los contratos legales, ya que pueden automatizar y ejecutar acuerdos de manera eficiente y transparente. Estos "contratos digitales" se registran en un lenguaje informático que es desplegado y ejecutado en una plataforma blockchain. Estos *scripts* contienen una serie de reglas y condiciones preestablecidas y, cuando se cumplen las mismas, el smart contract permite la ejecución de las acciones programadas, sin necesidad de intervención humana o de terceros.

Una de las características intrínsecas de la blockchain es su inmutabilidad, es decir, la capacidad de un "libro mayor" de blockchain para permanecer como un historial permanente, indeleble e inalterable de transacciones. Por tanto, es necesario, por no decir fundamental, que antes de desplegar un smart contract en una red empresarial, estos *scripts* pasen por unos minuciosos procesos de verificación al objeto de validar que su funcionamiento sea el esperado. Un error o defecto en el código de estos programas, por su naturaleza inmutable, puede conducir a resultados inesperados o no deseados y, lo más grave, podría causar un efecto no reparable.

Además, desde un punto de vista ingenieril, todo cambio tecnológico debería ir acompañado de un adecuado aseguramiento de la calidad, tanto de los productos software como del proceso productivo para su desarrollo y puesta en marcha. Pero, hoy en día, existen pocas utilidades, técnicas o métodos que proporcionen una solución, de forma global, para ello.

Repasando la literatura existente, la tecnología blockchain está aún en sus inicios desde el punto de vista de la calidad del software. Aunque ya empiezan a ver la luz propuestas interesantes de desarrollo y enfoques metodológicos concretos, las propuestas para el aseguramiento de la calidad de los smart contracts y, sobre todo, las propuestas para abordar el testing temprano, son aún escasas o muy ambiguas. Todo esto se ha hecho evidente tras el trabajo previo de investigación realizado como parte de la presente Tesis

Doctoral, donde se ha identificado que todavía son muy escasos los estudios primarios enfocados a dar una respuesta, aunque sea parcial, a esta línea de trabajo.

Una vez identificado el problema y los objetivos perseguidos en relación al aseguramiento de la calidad de los smart contract, a lo largo de esta Tesis Doctoral se ha intentado dar respuesta a la siguiente pregunta de investigación:

> *"¿Los principios que rigen las pruebas tempranas en el ciclo de vida del desarrollo de software son aplicables, en un contexto blockchain, para garantizar la calidad funcional de los smart contracts?".*

Para ello, se ha analizado de forma detallada los componentes y el funcionamiento de la tecnología blockchain y, más en concreto, de los smart contract. Además, se han analizado posibles soluciones, hasta identificar una solución factible para resolver el problema identificado. En concreto, en este trabajo se ha definido un mecanismo que nos permite generar pruebas funcionales de los smart contracts a partir de las especificaciones facilitadas por el área usuario o cliente y, se ha implementado una utilidad que de soporte a la ejecución de este mecanismo de generación de pruebas funcionales de forma sistemática. Es más, gracias a un proyecto de I+D+i como es el Proyecto SmartAuditor, entre otros, se ha podido dar una respuesta práctica a estos trabajos y en un contexto industrial, siendo este proyecto pionero en resolver de forma satisfactoria esta problemática, mediante nuestra propuesta de solución basada en la aplicación de los principios del testing temprano durante el ciclo de vida de desarrollo de los smart contracts en la tecnología blockchain.

Summary

The software systems are becoming increasingly multidisciplinary and complex. Their successful implementation has become a continuous challenge for any type of company or organisation. Carrying out the "digital transformation" of a company or organization involves, among other things, the adoption of advanced digital technologies to improve its operations, efficiency, products and services. In other words, for companies and organizations, it involves the acceptance and integration of technologies such as the internet, cloud computing and online services, the internet of things, process automation, big data analytics, artificial intelligence, and others.

In this context, blockchain technology appeared a few years ago. It is also one of those technologies that are driving the aforementioned “digital transformation”, thanks to its unique characteristics and its potential to address certain limitations of traditional systems. Moreover, given its transversality, this technology is enabling disruption in the economy and in business beyond the well-known cryptocurrencies. A quick search on the Internet is enough to see that we are witnessing a significant market commitment to developments based on this disruptive technology. This is mainly due to its ability to transform the way transactions are recorded and the manner in which data is stored and retrieved.

In blockchain technology, so-called smart contracts could act as a complement or substitute for legal contracts, as they can automate and execute agreements efficiently and transparently. These "digital contracts" are recorded in a computer language that is deployed and executed on a blockchain platform. These scripts contain a series of pre-established rules and conditions and, when these are satisfied, the smart contract allows the execution of the programmed actions, without the need for human or third-party intervention.

One of the intrinsic characteristics of the blockchain is its immutability, i.e., the ability of a blockchain "ledger" to remain a permanent, indelible and unalterable history of transactions. Therefore, it is necessary, if not essential, that before deploying a smart contract on an enterprise network, these scripts go through an exhaustive verification process to validate that their operation is as expected. An error or bug in the code of these programs, due to their immutable nature, can lead to unexpected or undesired results and, more seriously, could cause an unrepairable effect.

Moreover, from an engineering point of view, any technological change should be accompanied by adequate quality assurance, both of the software products and of the production process for their development and implementation. However, nowadays, there are very few utilities, techniques or methods that provide an adequate solution for this.

Reviewing the existing literature, blockchain technology is still in its infancy from the point of view of software quality. Although interesting development proposals and concrete methodological approaches are beginning to emerge, proposals for quality assurance of smart contracts and, above all, proposals to address early testing, are still scarce or very ambiguous. All this has become evident after the previous research work carried out as part of this Doctoral Thesis, where it has been identified that there are still very few primary studies focused on providing an answer, even if only partial, to this line of work.

Having identified the problem and the objectives pursued in relation to the quality assurance of smart contracts, throughout this Doctoral Thesis, we have tried to answer the following research question:

> *"Are the principles governing early testing in the software development lifecycle applicable, in a blockchain context, to ensure the functional quality of smart contracts?".*

For this purpose, the components and operation of blockchain technology and, more specifically, of smart contracts have been analysed in detail. In addition, possible solutions have been analysed, until a feasible solution to solve the identified problem has been identified. Specifically, in this work we have defined a mechanism that allows us to generate functional tests of smart contracts from the specifications provided by the user area or client and, we have implemented a utility that supports the execution of this mechanism for generating functional tests, in a systematic way. Moreover, thanks to a R&D&i project such as SmartAuditor Project, among others, it has been possible to give a practical response to these tasks in an industrial context, being this project a pioneer in solving this problem satisfactorily, through our proposed solution based on the application of the principles of early testing during the development life cycle of smart contracts in blockchain technology.

Capítulo I.
Introducción

"The last thing you know, is where to start"
- Blaise Pascal -

Este Capítulo inicia la Tesis Doctoral presentando, por un lado, los antecedentes de la investigación realizada, el problema existente y la estrategia de investigación y, por otro, la hipótesis de partida y los objetivos y retos de la presente Tesis.

Además, en este mismo Capítulo se expone cómo se ha estructurado el presente trabajo de investigación y los Capítulos que la componen, con la intención de realizar una adecuada presentación global de la misma.

I.1. Antecedentes

I.1.1. Grupo Engineering and Science for Software Systems

El grupo de investigación y de trabajo Engineering and Science for Software Systems (ES3) del Departamento de Lenguajes y Sistemas Informáticos (LSI) de la Escuela Técnica Superior de Ingeniería Informática (ETSII) de la

Universidad de Sevilla (US), es un grupo reconocido en el Plan Andaluz de Investigación como grupo PAIDI[2] TIC-021 y está adscrito como equipo de trabajo a la Fundación FIDETIA (Fundación para la Investigación y el Desarrollo de las tecnologías de la información en Andalucía) de la ETSII y a la Fundación FIUS (Fundación para la Investigación en la Universidad de Sevilla).

Durante más de 15 años, este grupo ha participado y participa en diversos proyectos de investigación de ámbito autonómico, nacional e internacional, y en numerosos proyectos de transferencia de resultados de investigación y en proyectos de I+D+i en cooperación nacional o internacional. El grupo ES3, aparte de participar en diferentes proyectos de investigación, destaca por su dilatada colaboración con el tejido empresarial.

Las actuales líneas de trabajo del grupo se podría encuadrar, aunque no limitar, dentro de la ingeniería de requisitos, el testing software y el aseguramiento de la calidad durante el desarrollo de sistemas de información (orientados o no a la web), dando un rol especial a la ingeniería de pruebas, el desarrollo de técnicas y protocolos que mejore la competitividad de las empresas de cualquier ámbito de negocio en base a la gestión eficaz y eficiente de sus procesos de negocio y en la definición, generación y transformación de modelos bajo el paradigma de la ingeniería dirigida por modelos.

El grupo ES3 también contempla soluciones enmarcadas en tecnologías referentes como RPA (*Robotic Process Automation*), AI (*Artificial Intelligence*), blockchain y es experto en dar soporte a empresas y organismos en sus procesos de Transformación Digital.

[2] PAIDI. Plan Andaluz de Investigación, Desarrollo e Innovación

I.1.2. Propuesta metodológica SofIA

NDT (Navigational Development Techniques) es una propuesta metodológica (Escalona, 2008) incluida dentro del paradigma de la ingeniería dirigida por modelos, que se definió inicialmente para cubrir las necesidades en el desarrollo Web.

Esta propuesta, basada en el uso de herramientas CASE (*Computer Aided Software Engineering*), partía de la definición de metamodelos formales para la fase de requisitos y de la definición de un conjunto de transformaciones que permiten generar los modelos de la fase análisis (Escalona, 2008). La etapa de ingeniería de requisitos comienza con la definición de los objetivos del sistema, para posteriormente ir capturando y definiendo los actores y los requisitos del mismo: requisitos funcionales, requisitos de almacenamiento de información, requisitos de interacción y requisitos no funcionales.

En los últimos años, esta propuesta metodológica enfocada al aseguramiento de la calidad de los resultados durante el desarrollo software, además de cambiar su nombre por SofIA, ha evolucionado para ofrecer una suite de herramientas y un soporte completo para todo el ciclo de vida de desarrollo de software.

Actualmente, SofIA (Escalona, 2022) (Escalona, 2023) cubre tres fases relevantes: (1) El análisis de requisitos, considerando los diferentes tipos de requisitos, así como la elaboración de mockups (prototipos); (2) El diseño, que incluye la arquitectura y el modelo de comportamiento de la solución, así como el modelo de persistencia; (3) La fase de pruebas, que incluye los distintos tipos de pruebas durante todo el desarrollo de software.

Además, SofIA también permite gestionar la trazabilidad de los artefactos (Escalona, 2023). Por ejemplo, si la ejecución de un caso de prueba no se ha podido completar, se podría rastrear qué requisito funcional generó este caso de prueba. Empresas como Airbus, Fujitsu, Lantia, NTT Data o la Administración Pública andaluza han utilizado con éxito esta metodología y sus herramientas.

De forma ilustrativa, en la Figura 1, se muestra un ejemplo de un caso de prueba funcional obtenida a partir de un caso de uso modelado mediante este marco metodológico.

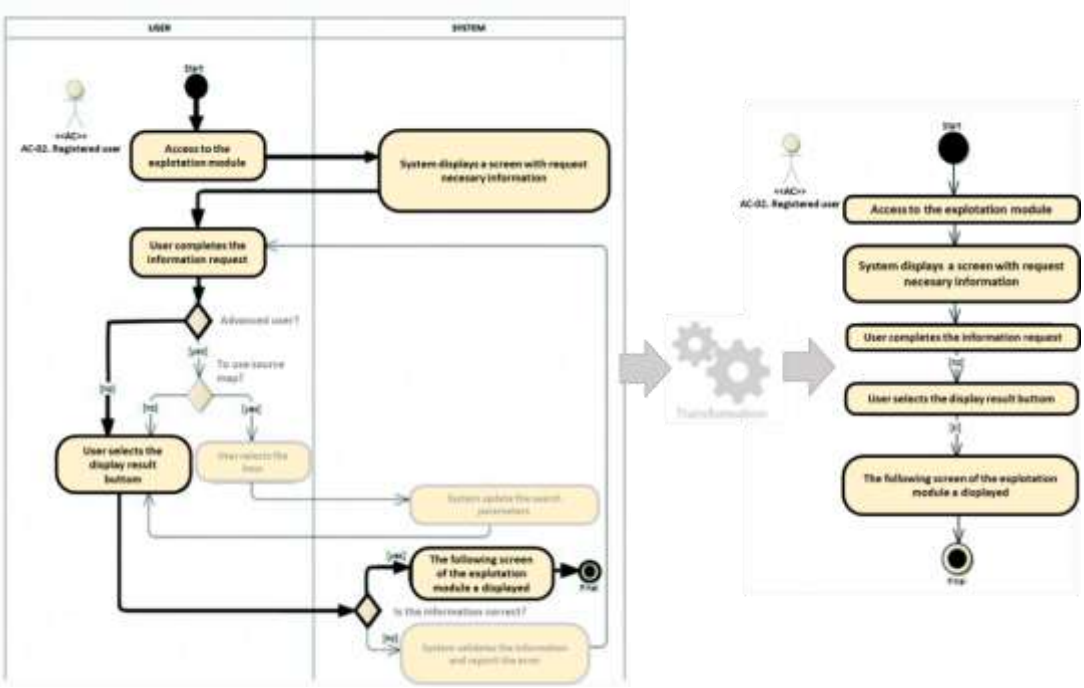


Figura 1. Ejemplo de un caso de prueba funcional generado mediante SofIA

Gracias a la suite de herramientas que proporciona SofIA, se puede definir tanto la fase de requisitos como la fase de análisis y diseño de un proyecto, así como generar las pruebas funcionales del sistema de forma temprana y sistemática, a partir de los casos de uso previamente elaborados. En concreto, para generar las pruebas funcionales, este marco metodológico realiza una serie de actividades, tal y como muestra la Figura 2.

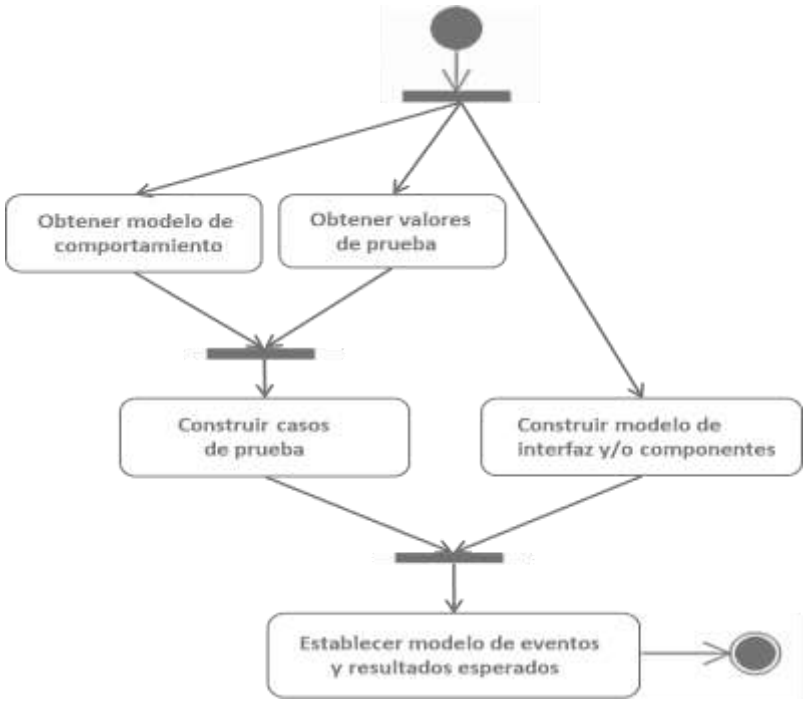


Figura 2. Pautas de SofIA para la generación de pruebas funcionales tempranas

Como se puede apreciar en la Figura anterior, el proceso para generar pruebas tempranas comienza con la actividad "Obtener modelo de comportamiento". Para ello, SofIA propone utilizar los diagramas de actividades que, a diferencia de los diagramas de secuencia, permiten expresar caminos alternativos fácilmente y, a diferencia de los diagramas de estado, permiten expresar la interacción entre el sistema y los actores externos, identificando claramente a cada uno de estos actores y obteniendo los objetivos de prueba a partir de él.

La actividad siguiente, denominada "Obtener valores de prueba", consiste en la identificación de las variables del caso de uso y establecer los posibles datos de prueba a tener en cuenta en la siguiente actividad, denominada "Construir casos de prueba", es decir, se establecen los casos de prueba funcionales, combinando el modelo de comportamiento con los valores de prueba.

Las siguientes actividades son "Construir modelo de interfaz y/o componentes", donde se establecen las pantallas y/o componentes que el sistema debe ofrecer al área usuaria y, finalmente, la actividad "Establecer modelo de eventos y resultados esperados", donde se determinarán los 'árbitros' que comprobarán si el resultado de la prueba es el esperado.

I.1.3. Proyecto SmartISBN

En los últimos años, uno de los proyectos donde el grupo ES3 ha colaborado estrechamente con el tejido empresarial, ha sido el proyecto de investigación y desarrollo para la obtención de un nuevo modelo de metadatos para el sector editorial basado en ONIX[3] y la tecnología blockchain, Proyecto SmartISBN (P049-19/E09). Este proyecto fue en colaboración con la empresa andaluza LANTIA Publishing S.L., en el periodo 2019 – 2020.

El objetivo principal del Proyecto SmartISBN era obtener un nuevo modelo de metadatos aplicado al sector editorial que permita el tratamiento de los diferentes datos asociados a una obra literaria de forma unificada. Este modelo se implementaría a través de una plataforma tecnológica (ver Figura 3) que, además, incorporaría otras extensiones funcionalidades necesarias para alcanzar todos los objetivos perseguidos.

[3] ONIX es un estándar para la codificación y el intercambio de información bibliográfica del sector editorial.

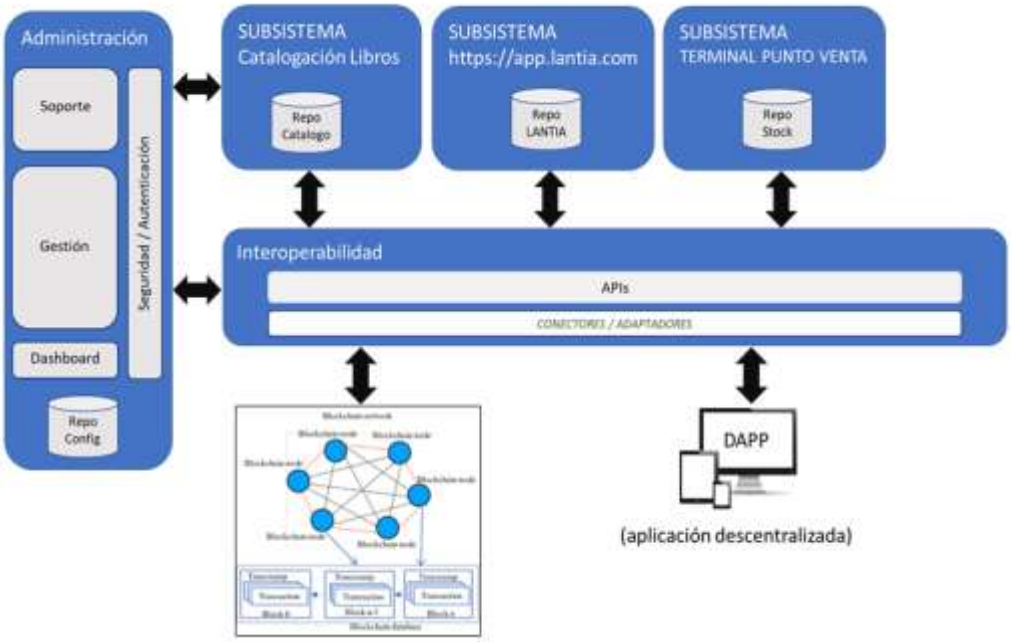


Figura 3. Arquitectura tecnológica de la plataforma SmartISBN

Las principales funcionalidades que debería incorporar esta plataforma son las siguientes:

- Catalogación de obras literarias mediante la obtención de sus metadatos. Para ello, se utilizaron herramientas como web crawling y web scraping que permiten navegar sistemáticamente a través de internet explorando webs y siguiendo links con el propósito de recoger el contenido web disponible.

- Tratamiento de datos, de forma masiva, para los diferentes actores del sector: editoriales, distribuidoras, autores, librerías, etc.

- Incorporación de la tecnología blockchain para almacenar las transacciones de obras literarias. Además, de la inmutabilidad de la información almacena en la blockchain - característica intrínseca de esta tecnología -, se tendría una mayor transparencia y trazabilidad en la cadena de suministro.

Entre los objetivos técnicos del proyecto destacaba la incorporación de la tecnología blockchain y, sobre todo, el uso de smart contracts para registrar las transacciones de las obras literarias. Las características intrínsecas de esta tecnología permitiría, a los diferentes actores de la plataforma, asegurar que la información del canal de distribución es correcta o, en caso de no serlo, detectar en tiempo real posibles anomalías o fraudes en las transacciones realizadas.

En el proyecto SmartISBN, para abordar todo el proceso de especificación de los requisitos, se acordó la utilización del marco metodológico SofIA lo que permitiría no solo especificar dichos requisitos, sino obtener los diagramas del proceso empresarial, así como los casos de uso y los diagramas de secuencia y/o actividad del sistema, incluida toda la lógica funcional de la blockchain como parte del sistema y también, a priori, el comportamiento esperado por los smart contracts.

En 2020, tras finalizar este proyecto de investigación, los resultados obtenidos no fueron del todo satisfactorio a la hora de establecer la lógica de funcionamiento del proceso del negocio empresarial, junto con el funcionamiento específico de los smart contracts. En concreto, los diagramas UML contemplados por SofIA permitieron identificar los actores, visualizar la forma en la que se debía comportar el sistema y cómo éste interactuaría dentro de sí mismo, así como con los usuarios y otros sistemas, pero no facilitaba el modelado de los requisitos de decisión[4] de los smart contracts, ni se disponía de ninguna utilidad para visualizar de forma amigable los

[4] El modelado de requisitos de decisión es un enfoque para analizar y capturar los requisitos relacionados con la toma de decisiones en un sistema o proceso. Consiste en identificar, especificar y representar las decisiones que se deben tomar, así como los factores o criterios que influyen en esas decisiones.

acuerdos establecidos entre las partes. En concreto, los artefactos utilizados hasta el momento por SofIA no permitieron modelar de forma visual, clara y precisa el comportamiento esperado de los smart contracts, lo que implicó que, ciertas pruebas funcionales críticas del sistema no pudieran obtenerse de forma sistematizada.

Para subsanar esta carencia, se tuvo que recurrir a determinados procesos manuales a la hora de elaborar casos de prueba funcionales que permitieran verificar el correcto comportamiento de los smart contracts desarrollados, siendo al final un proceso poco riguroso, que provocaron ciertos problemas de funcionamiento y, sobre todo, errores cuando estos *scripts* fueron ejecutados en la red blockchain.

I.2. Identificación del problema

Como se ha comentado en el apartado anterior, una característica intrínseca de la tecnología blockchain es la inmutabilidad. Ésta, además de garantizar la detección de fallos en la integridad de la información almacenada en una red blockchain, permite asegurar que la información almacenada en la red permanece como un historial permanente, indeleble e inalterable de transacciones.

Dada la característica de inmutabilidad, como se verá de forma detallada en el siguiente Capítulo, es primordial que antes de desplegar un smart contract (programa informático) en una red empresarial, éstos pasen por unos rigurosos procesos de verificación al objeto de validar su correcto funcionamiento, ya que un error o defecto en el código que lo forma podría causar un efecto no reparable (Legerén-Molina, 2018). Es más, desde una perspectiva ingenieril, no se puede avanzar de manera seria y competitiva en la implantación de una tecnología como es la blockchain y los smart contracts, si no se avanza en métodos, herramientas y técnicas que permiten

gestionar la calidad en el desarrollo de este producto software y, sobre todo, asegurar la calidad del mismo.

Como indican algunos autores (Legerén-Molina, 2018), un error en un smart contract desplegado en una red blockchain, podría dar lugar a un resultado no deseado por las partes involucradas, generando una serie de consecuencias jurídicas y económicas nefastas muy relevantes. En este sentido, dadas las características de esta tecnología, es importante resaltar que la única forma de corregir un error en un smart contract ya desplegado, es volver a desplegar una nueva versión del contrato, pero la versión anterior con el error, así como las consecuencias del error, permanecerán en la red y permanecerán allí para siempre.

Es más, actualmente existen numerosas plataformas blockchain que admiten implementar, desplegar y ejecutar smart contracts sin muchas restricciones. Es el caso de Ethereum, Hyperledger o EOS, entre otros (Zheng Z., 2020), plataformas que permiten realizar despliegues de estos scripts sin pasar por ningún proceso concreto de verificación y validación de lo que se ha implementado. Desafortunadamente, la seguridad de los smart contracts no ha recibido mucha atención y, a día de hoy, cualquier red blockchain puede estar ejecutando estos programas informáticos con un comportamiento no esperado, con graves deficiencias, errores e incluso con vulnerabilidades de seguridad (Luu, 2016). A diferencia de las aplicaciones clásicas, que pueden parchearse cuando se detectan errores o bugs, los smart contracts son irreversibles e inmutables, dadas las características intrínsecas de la tecnología que lo sustenta.

En este contexto, dada nuestra experiencia como grupo de investigación en el aseguramiento de la calidad software y nuestro bagaje en el proyecto SmartISBN, las cuestiones que nos planteamos como punto de partida en la presente Tesis Doctoral son:

RQ1. ¿Se podrían evitar o minimizar los posibles defectos o errores de los smart contracts, si se hiciera un descubrimiento, refinamiento, modelización y especificación funcional mediante requisitos?.

RQ2. ¿Se podrían evitar o minimizar esos posibles defectos o errores si, además, se inicia la actividad de pruebas del software en fases tempranas del desarrollo de estos programas informáticos? (como ya se hace en otro tipo de sistemas).

Aparentemente, desde el punto de vista ingenieril y dado el know-how del grupo de investigación en esta temática, todas estas actividades (elicitación, refinamiento, modelización y especificación funcional mediante requisitos) permiten mejorar la calidad de cualquier software y, por ende, todo apunta a que también deberían permitir mejorar la calidad de estos programas informáticos, llamados smart contracts.

Por tanto, en la presente Tesis Doctoral nos centraremos en detallar por qué es necesario mejorar la calidad de los smart contracts, en estudiar cómo se puede mejorar la calidad de éstos y, sobre todo, en verificar si mejoraría su calidad si se realizaran pruebas funcionales desde las etapas tempranas del desarrollo del software, es decir, aplicando los principios de lo que se conoce como “testing temprano”, “pruebas tempranas” o “early testing”.

Estos principios se basan en la idea de incorporar pruebas de software lo antes posible en el ciclo de vida de desarrollo, con el objetivo de identificar y solucionar problemas de manera temprana, siguiendo las directrices marcadas por el estándar ISO 29119 Ingeniería de software y sistemas - Pruebas de software -. Estos principios se podrían resumir de la siguiente forma (Abrahäo, 2022):

- Realización de pruebas desde el inicio. Las pruebas deben comenzar en las etapas tempranas del ciclo de vida de desarrollo, antes de que se empiece a escribir o generar el código. Esto implica contemplar las pruebas durante las especificaciones, los requisitos, etc. para asegurar su comprensión y sea realizable su implementación.

- Automatización de pruebas. Se debe buscar la automatización de las pruebas, lo que implica establecer los escenarios y los casos de prueba automatizados que puedan ejecutarse de manera rápida y repetible. Esta automatización ayuda a identificar rápidamente problemas y reduce el tiempo y esfuerzo requerido para realizar pruebas manuales repetitivas.

- Ejecución de pruebas continuas. Las pruebas deben realizarse de manera continua a lo largo del ciclo de vida de desarrollo y no esperar hasta el final del proceso de desarrollo para probar el software. Se deben realizar pruebas de forma regular y frecuente, lo que facilita la detección de problemas de manera temprana y abordar las posibles soluciones antes de que se conviertan en costosos errores de difícil corrección.

- Prevención y realimentación: El enfoque principal del testing temprano es prevenir problemas en lugar de encontrarlos. Se busca identificar y corregir posibles errores, deficiencias o inconsistencias antes de que se conviertan en defectos más graves. Un *feedback* rápido permite, a los equipos de desarrollado, corregir errores de manera oportuna y mejorar el software de manera iterativa.

- Colaboración multidisciplinaria. El testing temprano requiere la colaboración de diferentes roles y disciplinas dentro del equipo de desarrollo de software. Éstos deben trabajar desde el principio para garantizar que se realicen las pruebas adecuadas y se aborden los problemas de manera efectiva.

Estos principios promueven una cultura de calidad y mejora continua en el desarrollo de software, permitiendo la detección temprana de problemas y la entrega de software más confiable y robusto.

I.3. Estrategia de investigación

Es evidente que un soporte metodológico ayuda a cualquier investigador a garantizar la calidad de sus resultados y, sobre todo, a organizar el contenido de sus conclusiones. También ayudaría a estructurar y presentar el contenido de su investigación de una forma más comprensible. Por tanto, durante la realización de esta Tesis Doctoral se ha tenido como marco metodológico de referencia el método *Design Science* (Johannesson, 2014) (Wieringa, 2014) (Dresch, 2015). Este método contiene directrices generales que facilita: (i) Llevar a cabo las actividades, (ii) Elegir las estrategias y métodos de investigación que se utilizarán en las actividades y (iii) Relacionar la investigación con una base de conocimientos ya existente.

Mediante la aplicación de este método, los resultados obtenidos deben estar relacionados con una base de conocimientos existente para garantizar que la investigación producida exprese conocimientos válidos y fiables. Para ello, este marco de trabajo describe cinco actividades relacionadas que definen las preguntas que el investigador debe responder y los resultados que se deben producir. Estas actividades comienzan definiendo el problema y continúan hasta demostrar y evaluar los artefactos producidos que resuelven un problema identificado. Cada actividad está asociada a directrices que ofrecen consejos prácticos que apoyan y facilitan el trabajo de investigación, ayudando a garantizar el rigor científico.

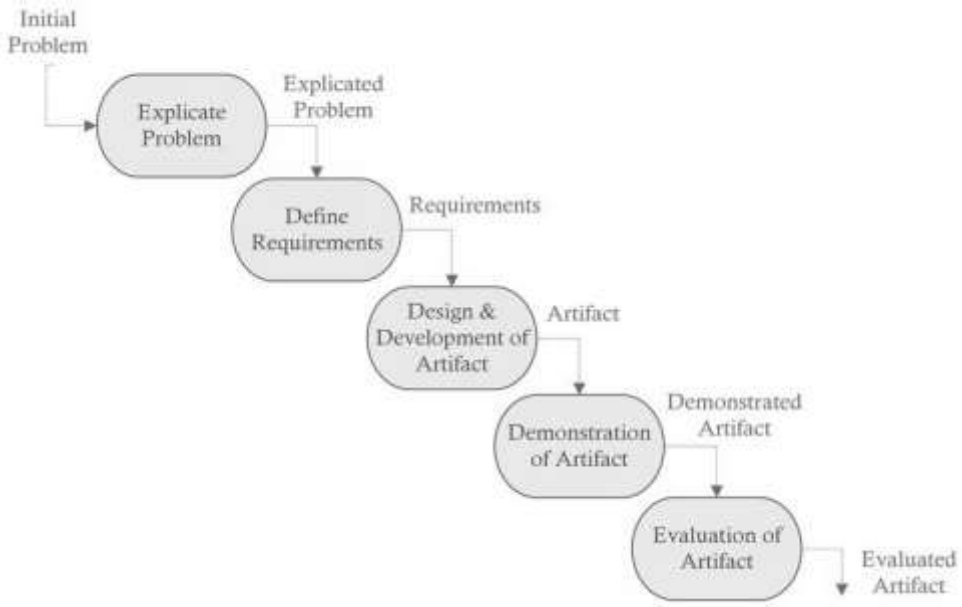


Figura 4. Marco metodológico de la investigación en *Design Science* (Johannesson, 2014)

En concreto, como se puede apreciar en la Figura anterior, las actividades que componen este marco de trabajo son:

- Explicar el problema. Durante esta actividad se estudia las causas fundamentales del problema y su objetivo es proporcionar una comprensión esencial de un problema.
- Definir los requisitos. Esta actividad define los requisitos y propone un esbozo de solución.
- Diseñar/desarrollar el artefacto. Esta actividad es el desarrollo teórico del enfoque de la solución, que incluye el artefacto en cuestión.
- Demostrar el artefacto. Se utiliza el artefacto, planteado en la etapa anterior, en uno o varios casos para demostrar que puede resolver el problema.

- Evaluar el artefacto. La última actividad determina la eficacia del artefacto para resolver el problema para el que fue diseñado.

En esta metodología, un problema se define como una brecha entre el estado actual y el deseable, siendo necesario generalizar el problema al que se hace frente y contar con los conocimientos de la bibliografía de investigación existente. Teniendo en cuenta este método, explicar el problema se compondría de varias subactividades:

- Definir el problema con precisión para garantizar que diferentes personas lo entiendan de la misma manera. Un entendimiento común ayuda a las personas a compartir la misma visión de un problema y ayuda a limitar el alcance del proyecto de investigación,

- Justificar el problema, describiendo su propósito, el entorno y los retos, de forma que se esté de acuerdo en que merece la pena abordarlo y,

- Encontrar las causas del problema, lo que ayuda a definir una solución para mitigarlo.

En la presente Tesis Doctoral, explicar el problema se ha abordado de la siguiente manera:

- En el Capítulo II se describe y justifica el problema, determinando el contexto de éste de forma precisa, concisa y comprensible.

- En el Capítulo III se destaca la importancia del problema y se señala la generalización del problema abordado. Para ello, se describen estudios primarios disponibles en la literatura que han identificado o experimentado previamente el problema descrito.

La segunda actividad, relativa a definir los requisitos, abordaría la pregunta: *"¿Qué solución resolvería el problema identificado y qué requisitos a cubrir son los más importantes?".* En el Capítulo IV se aborda esta pregunta y se describe una solución que permitiría resolver el problema identificado.

La siguiente actividad, relativa a diseñar/desarrollar el artefacto, debe proporcionar los mecanismos o utilidades que permitan resolver el problema identificado. En el Capítulo V se identifica y detalla el artefacto desarrollado, a partir de la solución planteada en el Capítulo anterior.

La última actividad, relativa a evaluar el artefacto, se centra en cómo usar el artefacto desarrollado y, sobre todo, en su evaluación práctica. Con esta actividad se intenta determinar hasta qué punto lo producido hasta el momento cumple con lo esperado (de acuerdo con el problema que motivó la investigación). En el Capítulo VI, de la presente Tesis, se recogen dichos casos prácticos y los resultados obtenidos tras su validación.

I.4. Hipótesis preliminar

La pregunta general de investigación (RQ - Research Question) que nos planteamos en este trabajo de investigación sería la siguiente:

PREGUNTA DE INVESTIGACIÓN (RQ)
¿Los principios que rigen las pruebas tempranas en el ciclo de vida del desarrollo de software son aplicables, en un contexto blockchain, para garantizar la calidad funcional de los smart contracts?.

Tabla 1. Pregunta general de investigación de la Tesis

Por tanto, esta Tesis se enfrenta al reto de aplicar los principios de las pruebas tempranas en el contexto de la blockchain y, más en concreto, en el aseguramiento de la calidad funcional de los smart contracts, su pilar fundamental. Desde nuestro punto de vista, al adaptar y aplicar estos principios de manera adecuada, se puede mejorar la confiabilidad y seguridad de los smart contracts, antes de que estos programas informáticos sean compilados, desplegados y ejecutados en una plataforma blockchain.

Partiendo de esta RQ y el contexto general del presente trabajo, el reto de este trabajo de investigación sería demostrar que la RESPUESTA a esta pregunta es AFIRMATIVA.

Luego, teniendo estas premisas en cuenta, la hipótesis de partida de esta Tesis sería la siguiente.

HIPÓTESIS DE PARTIDA
La aplicación de los principios y técnicas del testing temprano sobre el ciclo de vida de desarrollo de software, podría ayudar a generar pruebas funcionales de los smart contracts, de forma independiente de la plataforma blockchain, al objeto de garantizar que las transacciones realizadas a través de estos programas informáticos se ejecutan conforme a las especificaciones, condiciones y reglas preestablecidas.

Tabla 2. Hipótesis de partida de la Tesis

I.5. Objetivos de la Tesis

I.5.1. Objetivo general

Partiendo de la hipótesis de partida y teniendo en cuenta nuestro objetivo de demostrar que podemos resolver de forma afirmativa la RQ recogida en la Tabla 1, en este apartado se hace mención al objetivo general de este trabajo de Tesis.

Este objetivo queda plasmado en la Tabla 3.

OBJETIVO GENERAL
Desarrollar una solución, basada en los principios del testing temprano, para generar pruebas funcionales a partir de las especificaciones de los smart contract, lo que permitirá validar la calidad funcional de estos programas informáticos, independientemente de la plataforma blockchain que se vaya a emplear.

Tabla 3. Objetivo general de la Tesis

La Figura 5 muestra un primer esbozo del *Graphical Abstract* de la presente Tesis Doctoral. Esta ilustración recoge el flujo de trabajo o mecanismo previsto que, desde nuestro punto de vista, permitirá aplicar los principios de pruebas tempranas durante el ciclo de vida del desarrollo de software y, más en concreto, del desarrollo de los smart contracts. Como se puede apreciar en dicha Figura, el proceso se inicia con la definición de las especificaciones, condiciones y reglas a tener en cuenta en los smart contracts, detalles que normalmente son facilitados por el área usuaria o clientes. Toda esta información será modelada por el equipo de analistas mediante un lenguaje estructurado, a partir de unos metamodelos previamente establecidos y, posteriormente, estos modelos podrán ser

implementados por el equipo de desarrollo. De manera paralela a este proceso, aplicando los resultados de este trabajo de Tesis, se podrán ir generando de forma sistemática pruebas funcionales asociadas a la solución global y, de forma temprana, el equipo de trabajo podrá ir verificando que la funcionalidad de la solución, que incluye los smart contracts, es la esperada.

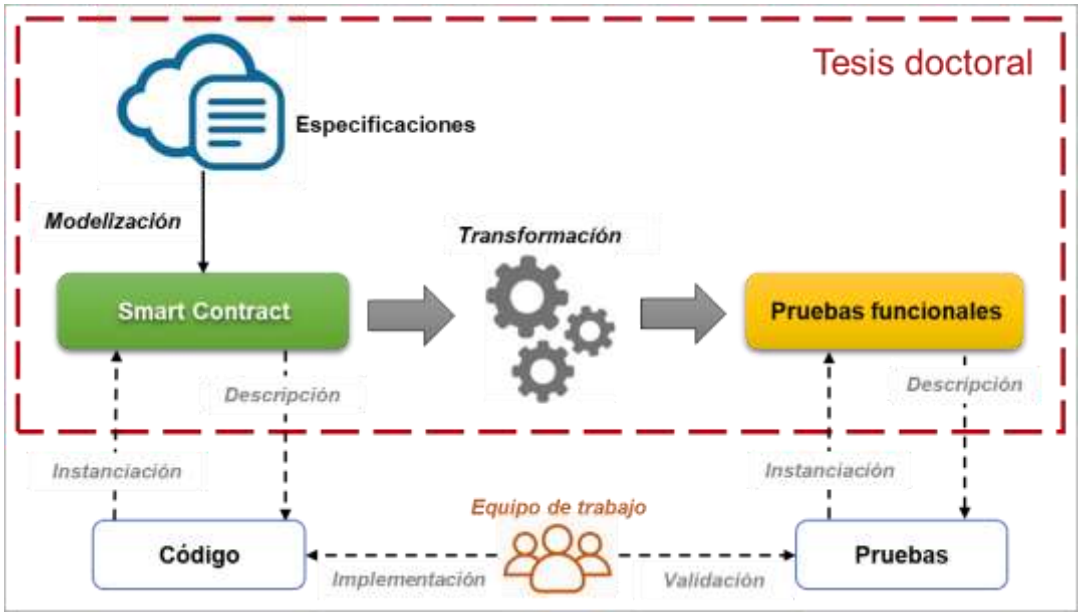


Figura 5. Esbozo del *Graphical Abstract* de la Tesis Doctoral

Como muestra la Figura anterior, las pruebas funcionales obtenidas a partir del modelado de la solución global – que incluye los smart contracts -, podrán ser ejecutadas, por el equipo de pruebas, una vez implementado el código, al objeto de asegurar que la solución global o las específicas, como son los smart contract, generan los resultados esperados. Es decir, el equipo de trabajo podrá validar que todas las características y funcionalidades de la solución global, se comportan según las especificaciones, condiciones, reglas, etc. facilitadas y sin ningún problema.

Este modo de trabajo, además, garantizaría que la obtención de las pruebas funcionales a ejecutar, se hacen de manera independiente a la implementación del código, siguiendo así los principios establecidos y las buenas prácticas recogidas en la norma UNE EN ISO 29119 (Tuya, 2009).

I.5.2. Objetivos específicos

Una vez establecido el objetivo general de la presente Tesis, en la tabla siguiente se exponen los objetivos específicos que nos ayudarán a alcanzarlo.

Objetivo	Descripción
OBJ-00	Valorar la importancia del problema, describiendo estudios primarios disponibles en la literatura que han identificado, experimentado o analizado previamente el problema descrito.
OBJ-01	Identificar qué artefactos podrían aportar una solución para el problema identificado, planteando alguna estructura que permita modelar los smart contracts y las pruebas funcionales, a partir de las especificaciones, condiciones, reglas, etc.
OBJ-02	Definir un mecanismo que nos permita generar pruebas funcionales de los smart contracts, en fases tempranas del ciclo de vida de desarrollo, a partir de la solución propuesta y de la manera más automática posible.
OBJ-03	Implementar una utilidad o herramienta que facilite la concreción de la solución propuesta y que, a su vez, permita ejecutar el mecanismo de generación de pruebas funcionales de los smart contracts de forma sistemática.
OBJ-04	Evaluar la solución propuesta, los mecanismos propuestos y la herramienta implementada, mediante verificaciones y validaciones en un contexto industrial.

Tabla 4. Objetivos específicos de la Tesis Doctoral

I.6. Estructura de la Tesis

Aunque ya se ha indicado brevemente en el apartado de estrategia de investigación, los estudios realizados durante el desarrollo de esta Tesis Doctoral se han organizado en siete Capítulos:

El Capítulo II se centra en el contexto de la investigación y, sobre todo, en la identificación y justificación de la problemática existente. En el Capítulo III se identifican y analizan estudios primarios relacionados con la temática de la presente Tesis Doctoral, trabajos que se han identificado a través de una revisión sistemática de la literatura existente en distintas bibliotecas digitales que cubren una amplia gama de temas, lo suficiente como para ser razonablemente considerados exhaustivos para el campo de investigación en el que se centra esta Tesis.

En el Capítulo IV se recoge nuestra propuesta para alcanzar uno de los objetivos de la presente Tesis. En concreto, se identifican que artefactos podrían aportar una solución para el problema identificado. Posteriormente, en el Capítulo V, se detallan los mecanismos y utilidades que podrían generar pruebas funcionales de los smart contracts, en fases tempranas del ciclo de vida de desarrollo.

En el Capítulo VI se detalla cómo se han validado estos trabajos de investigación en la práctica mediante la aplicación de la solución propuesta. Concluyendo la presente Tesis Doctoral, con el Capítulo VII, donde se realiza una exposición de los resultados obtenidos y de los posibles trabajos futuros, asi como de las conclusiones.

I.7. Resumen del Capítulo

Nuestro trabajo de investigación trata de dar solución o, más bien, minimizar un riesgo identificado: *¿cómo mejorar la calidad de los smart contracts antes de su despliegue en una red blockchain?*. Y, más en concreto, se realiza una investigación sobre si sería posible mejorar la calidad de los "contratos digitales" de la tecnología blockchain, mediante la consideración de las pruebas funcionales desde las primeras etapas del ciclo de vida del proceso de desarrollo de software.

Este primer Capítulo nos ha ayudado a identificar inicialmente el problema, indicar la estrategia que se seguirá en este trabajo e indicar la hipótesis de partida, así como los objetivos de la presente Tesis Doctoral. Además, se ha presentado la estructura que seguirá el presente documento.

Capítulo II.
Contexto de la investigación

True science teaches, above all, to doubt and to be ignorant.
- Ernest Rutherford -

En este Capítulo se identifica el área de investigación en la que se desarrolla esta Tesis Doctoral, detallando y delimitando el alcance y enfoque de la investigación.

Además, el contexto proporcionado en este Capítulo intenta ofrecer una amplia perspectiva de esta investigación, aportando conceptos, motivaciones y, sobre todo, discutiendo los retos relevantes a los que nos enfrentamos.

Actualmente, con el aumento paulatino de las prestaciones de las computadoras y la permanente evolución tecnológica que vivimos, el desarrollo de software se está haciendo cada vez más complejo, conduciendo a problemas, retrasos, errores y, en definitiva, parece que van siempre encaminados a un mal resultado final. Además, el desarrollo de software ha pasado de ser una tarea realizada por un grupo normalmente reducido de

personas, a convertirse en un conjunto de actividades interrelacionadas que se realizan con grandes equipos de trabajo multidisciplinares.

Dada esta problemática, los procedimientos de desarrollo, conocidos como SDLC, siglas en inglés de *Software Development Life Cycle* (ciclo de vida de desarrollo de software), son cada vez más necesarios, por no decir fundamentales, para poner orden en ese conjunto de actividades y, por ende, reducir el nivel de dificultad, organizar mejor las tareas, agilizar el proceso de desarrollo y mejorar la calidad de los productos software resultantes.

En definitiva, dada esta situación, se ha pasado a una industrialización del software, donde empresas e individuos persiguen activamente el desarrollo de productos software de calidad. Esta industrialización, que involucra la investigación, desarrollo, distribución y comercialización de software, persigue que estos productos software cumplan con unas determinadas características para que puedan funcionar de manera eficiente y satisfactoria, teniendo como base la ingeniería del software y la reutilización[5] como piezas claves en las tácticas abordadas (Götz, 2020).

Es más, el uso de patrones de diseño, la aplicación de la ingeniería basada en modelos y/o frameworks de desarrollo son técnicas cada vez más empleadas en el entorno industrial, con el objetivo de lograr que la reutilización esté totalmente integrada, de forma sistémica, en las diferentes fases del SDLC.

En este contexto, uno de los retos que motiva a la comunidad de investigadores de nuestro sector, es proponer esquemas de desarrollo en los cuales los modelos, antes que el código, sean los pilares del proceso de desarrollo. Es decir, se buscan mecanismos sistemáticos y soluciones que asistan al ingeniero informático en la confección y transformación paulatina de modelos hasta obtener la solución final. Esta tendencia, denominada *Model-Driven Engineering* (MDE) (OMG, 2006) (Bézivin, 2012), ya está

[5] La reutilización en la ingeniería informática se refiere a la práctica de aprovechar el conocimiento, código fuente, componentes, diseños, algoritmos u otros recursos previamente desarrollados para utilizarlos en nuevos proyectos o aplicaciones, en lugar de crearlos desde cero.

liderada por el *Object Management Group* (OMG) que ya ofrece una plataforma arquitectónica ampliamente aceptada como estándar para su uso, denominada MDA (Model-Driven Architecture) (OMG, 2006).

Entre otros objetivos, MDE tiene como propósito mejorar la calidad del software que se construye y, bajo este paradigma, las herramientas de modelado o CASE (Computer Aided Software Engineering) se han convertido en esenciales para este trabajo (Bézivin, 2012). Estas herramientas permitir sistematizar y automatizar el proceso de generación de software a partir de modelos previamente definidos.

A continuación, como parte del presente trabajo de investigación, se va a analizar, por un lado, el estado del arte de la ingeniería del software para, posteriormente, poner el foco sobre el paradigma MDE y la reutilización - que según la literatura existente permite mejorar aún más la calidad del software que se construye - y, por otro lado, se examinará con más detalle la tecnología blockchain y, sobre todo, los smart contracts. Por último, nos centraremos en la problemática existente, y la visión general de la solución propuesta.

II.1. Ingeniería del software

La ingeniería del software es una disciplina de la ingeniería que se preocupa por todos los aspectos de la producción del software, en todas sus fases, desde su especificación hasta la fase de mantenimiento. El estándar IEEE 610.12-1990 define la ingeniería del software como "*la aplicación de un método sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software".* Es una de las ramas de las ciencias de la computación que estudia la creación de software confiable y de calidad, basándose en métodos y técnicas de ingeniería, y proponiendo soporte operacional y de mantenimiento (Morales, 2021).

Esta ingeniería dispone metodologías, modelos y herramientas pensadas para el desarrollo de proyectos de software, es decir, se vale de una serie de procedimientos que establecen y muestran las distintas fases y estados por los que pasa un producto software, desde su concepción inicial, pasando por su desarrollo, puesta en marcha y posterior mantenimiento, hasta incluso la retirada del producto.

A estos modelos, se les denomina "Modelos de Ciclo de vida de desarrollo del software". Por ejemplo, uno de los grandes ciclos de vida concebidos fue el modelo en "Cascada" (Royce, 1987), que estableció que las actividades que se van realizando al desarrollar software se suceden de forma lineal.

Por tanto, el SDLC es la estructura que contendría los procesos, actividades y tareas relacionadas con el desarrollo y mantenimiento de un producto software, abarcando la vida completa del mismo, desde su concepción inicial hasta la finalización de su uso. Con este enfoque se pretende evitar los costes de rectificar errores de implementación, aplicando un método sistemático que permita a los desarrolladores adelantarse para mejorar los resultados finales.

En la actualidad, dada la permanente evolución tecnológica y la actual tendencia hacía la industrialización del software, existen una gran variedad de ciclo de vida del software, cada uno con sus características, ventajas e inconvenientes. Éstos suelen coincidir en dos fases esenciales en el desarrollo del producto software como es el análisis y las pruebas.

Además, con esta industrialización, las empresas persiguen que los productos software cumplan con unas características determinadas para que puedan funcionar de manera eficiente, efectiva y satisfactoria, siendo la ingeniería del software y la reutilización de software piezas clave de las tácticas abordadas. En concreto, la reutilización se ha convertido es una de las estrategias más prometedora para poder afrontar el reto de desarrollar software con niveles de calidad adecuados, sobre todo en escenarios de negocio complejos o donde se producen continuos cambios tecnológicos. La ingeniería basada en modelos, sobre todo, se ha convertido en una técnica

cada vez más utilizadas con el objetivo de lograr que la reutilización esté integrada, de forma sistémica, en las diferentes fases del SDLC.

Numerosos autores (Elallaoui, 2016) (García-García, 2014) (Whittle, 2013) (Escalona, 2011) (Escalona, 2008) (Baker, 2007) indican los beneficios de usar el paradigma MDE. Estos autores indican que se podrían usar diversas técnicas para obtener las especificaciones del software y, a partir de éstas, realizar una automatización de las pruebas funcionales. En concreto, estos autores proponen el uso sistemático de UML *(Unified Modeling Language)*, lo que permitiría la transformación de modelados de requisitos en modelados de prueba e incluso obtener el código fuente del software.

II.2. Model-Driven Engineering

Model-Driven Engineering (MDE), como se ha comentado, es una disciplina dentro de la ingeniería del software que tiene por objetivo dar soporte a las actividades de producción de software, utilizando los modelos como principal artefacto.

Un concepto clave para trabajar con modelos es el metamodelo. Por ejemplo, podemos crear un modelo de un vehículo mediante diferentes técnicas como puede ser un prototipo o un plano, donde dicho modelo representaría un elemento del mundo real. Estas dos ideas normalmente serían suficientes, pero en esta disciplina se discute sobre el concepto de metamodelo, es decir, ¿cómo construir modelos?.

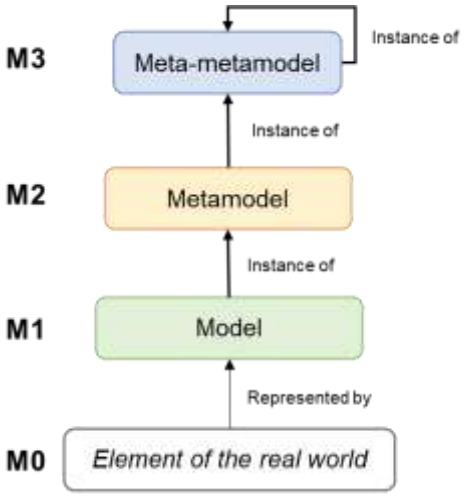


Figura 6. Arquitectura de capas de MDE

En concreto, el Object Management Group (OMG) (OMG, 2006) propone una arquitectura MDE de varios niveles, tal y como se muestra en la Figura 6:

- M3 (Meta-metamodelo). En este nivel se definen los conceptos, los atributos y las relaciones para los elementos del nivel M2, mientras que los elementos situados en el nivel M2 son las instancias de los elementos del nivel M3.

- M2 (Metamodelo). En este nivel, teniendo en cuentas las especificaciones del nivel M3, se definen los elementos válidos de un modelo específico del nivel M1, mientras que los elementos situados en el nivel M1 serían las instancias de los elementos del nivel M2.

  En el caso de un metamodelo elaborado según UML (OMG, 2011), es posible citar ejemplos de conceptos que se encuentran en este nivel como es 'Clase', 'Atributo' o 'Asociación'.

- M1 (Modelo). El nivel M1 define las clasificaciones de los elementos del nivel M0, mientras que los elementos situados en el nivel M0 son las instancias de los elementos del nivel M1.

  Un ejemplo en este nivel, según UML, sería un modelo de caso de uso o una clase como, por ejemplo, los conceptos "Cliente", "Venta" o "Libro" y sus atributos "Nombre", "Autor", etc.

- M0 (Realidad). En este nivel hay instancias de modelos de M1, como por ejemplo objetos "Libro" instanciados.

El elemento clave en todo este proceso es la transformación entre modelos, que puede definirse como el proceso por el cual un modelo se convierte en otro modelo del mismo sistema. Estas transformaciones se realizan mediante la asociación de constructores del metamodelo origen con los constructores correspondientes del metamodelo destino. Existen dos tipos de transformaciones: Model-to-Model (M2M) y Model-to-Text (M2T), según el modelo destino corresponda a otro modelo abstracto del sistema o a un modelo de código (texto).

En MDE, desde hace años, las herramientas de modelado o CASE están ayudando a documentar la funcionalidad de los procesos de negocio y, a través de las transformaciones de los modelos, a automatizar la generación del código fuente del software (Bézivin, 2012). Es más, dado que los modelos suelen ser más fáciles de entender que el código fuente del software (Forward, 2008), el uso del modelado UML u otros estándares se han extendido bastante, ya que su uso está permitiendo mejorar la productividad y la calidad del desarrollo (Forward, 2008). Según algunos autores (Lu, 2018), es más fácil comprobar la corrección de un modelo y las herramientas de modelado pueden garantizar, además, que el código desplegado no ha sido modificado después de su generación a partir del modelo.

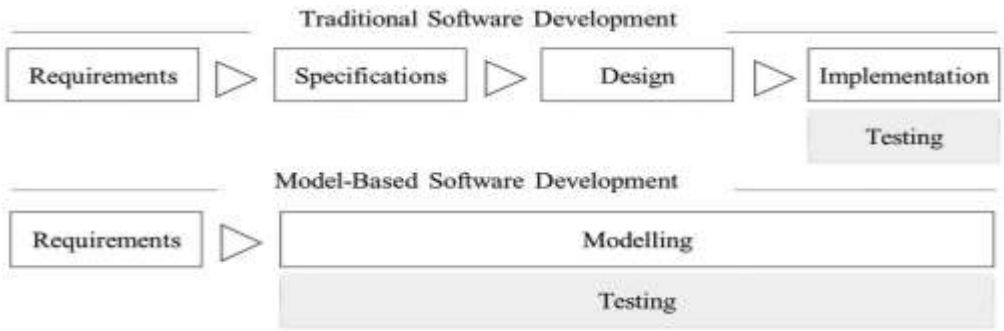


Figura 7. Ciclo de vida basado en modelos, en comparación con el tradicional (Conrad, 2005)

Como se puede apreciar en la Figura 7, en comparación con el SDLC tradicional donde las fases están claramente separadas, los desarrollos que han seguido un SDLC basado en el paradigma MDE, donde las fases de especificaciones de requisitos, diseño e implementación han evolucionado juntas con mucha más fuerza y de forma paralela a la fase de pruebas (Conrad, 2005).

Además, algunos autores (Seebache, 2018) indican que el SDLC basado en el paradigma MDE contribuye a describir y comprender un producto software de varias maneras: ayuda al equipo de trabajo a comunicar la visión del sistema que se está construyendo; permite especificar la estructura y conducta del producto software, facilitando la visualización del mismo para usuarios o clientes.

Otros (PivotPoint Technology™, 2019) sostienen que las principales ventajas de este enfoque, guiado por modelos, son que: (1) los requisitos son una parte integral del modelo y partes del modelo pueden remontarse a los requisitos, lo que se conoce como trazabilidad de los requisitos; (2) los modelos pueden utilizarse para representar el comportamiento esperado de un producto software bajo pruebas, reduciendo los errores durante el SDLC; (3) la automatización de los casos de prueba garantizan que la implementación del producto software sea más fiable e, incluso, se facilita la generación de un código fuente de calidad.

En este contexto, la verificación del modelado es llevado a cabo para evaluar la exactitud entre lo modelado y el producto software que se desea obtener. Si un modelado no cumpliera con las especificaciones funcionales indicadas, se debería introducir modificaciones en el mismo, al objeto de rectificar los errores de análisis y/o diseño, siendo este enfoque la columna vertebral sobre la que se sustentan las "pruebas tempranas".

Las "Pruebas tempranas" es uno de los 7 principios de las pruebas de calidad de software (Homès, 2013) y establece que "*las pruebas deben comenzar lo antes posible en el ciclo de vida del desarrollo de software, de modo que cualquier defecto en los requisitos o la fase de diseño se capture en las primeras etapas*". Este razonamiento es de vital importancia para la correcta implementación de los procesos de calidad de software en cualquier proyecto, ya que detectar defectos lo más temprano posible tiene impacto directo sobre distintos aspectos del proyecto, tales como: calidad, tiempo, presupuesto e incluso la reputación del proyecto de cara al usuario final o cliente.

Es importante destacar que los avances tecnológicos requieren de arquitecturas cada vez más flexibles y sostenibles. Esto ha provocado que, en los últimos años, exista un mayor interés en este nuevo paradigma basado en modelos, ya que ha demostrado ser enormemente robusto, flexible y escalable en comparación con las estrategias tradicionales. Además, cuenta con conceptos como la independencia de la plataforma y la reutilización, ya que los diseños basados en modelos pueden reutilizarse para añadir más funcionalidad y/o cambiar la arquitectura de destino (Rodrigues, 2012).

II.3. Tecnología blockchain

Blockchain es una tecnología software disruptiva que avanza a pasos agigantados (Olea, 2019). Constituye una de esas tecnologías fundamentales que impulsan la "transformación digital" de una empresa u organización y,

dado su carácter transversal[6], está permitiendo la disrupción en la economía y en la empresa más allá de las criptomonedas.

Este potencial, principalmente, se basa en su capacidad para ofrecer a personas, empresas u organizaciones un canal de comunicación que permite transferir derechos, valores o activos reales (tokenización), a través de Internet, de forma segura y confiable. Aunque a menudo esta tecnología se confunde con la criptomoneda Bitcoin, blockchain solo es la tecnología subyacente utilizada para operar por esa u otras criptomonedas.

Esta tecnología software, de forma resumida, se trataría de una base de datos distribuida[7], conocida como libro mayor, anexa a un software. En esta base de datos solo se registran transacciones validadas por la red. Estas transacciones son almacenadas en unidades elementales llamadas bloques. Estos bloques de datos, físicamente independientes, una vez validados por el "mecanismo de consenso" establecido en la red, se agregan a una cadena secuencial de bloques vinculados mediante funciones hash criptográficas.

Esta forma de organización de la información confiere a la tecnología blockchain una de sus principales propiedades: la detección de la integridad de los datos registrados. No es posible añadir, eliminar o modificar la información almacenada en un bloque sin que se produzca un cambio en el valor del hash utilizado como vínculo en el bloque siguiente.

Esta propiedad, conocida también como inmutabilidad, garantiza la detección de fallos en la integridad de los bloques almacenados. Además, algunos autores destacan que esta característica intrínseca de la blockchain es un beneficio clave ya que:

[6] Se adapta a numerosos sectores tales como agricultura, industria, finanzas, administración pública, seguros, legal, logística, automoción, aeroespacial, etc. (prácticamente a todos).

[7] La información no se almacena en un único ordenador, sino en múltiples terminales conectados entre sí a través de Internet.

- Permitiría que un libro mayor de la blockchain permanezca como un historial permanente, indeleble e inalterable de transacciones (Crosby, 2016)

- Transformaría el proceso de auditoría en un procedimiento rápido y eficiente, dando más confianza e integridad a los datos que las empresas u organizaciones usan y comparten todos los días (Argañaraz, 2019).

- Facilitaría la realización de auditorías externas e internas, siendo la trazabilidad una de las funcionalidades más destacadas de esta tecnología sobre otras soluciones (Westerkamp, 2020).

Profundizando un poco en esta característica intrínseca, en una red blockchain es extremadamente difícil cambiar las transacciones registradas, ya que cada bloque está vinculado al bloque anterior al incluir el hash de ese bloque anterior. Este hash incluye el hash de raíz del árbol de Merkle (Merkle, 1988) de todas las transacciones en el bloque anterior y, si una sola transacción fuera a cambiar, no solo cambiaría el hash raíz del árbol de Merkle, sino también el hash contenido en el bloque modificado. Además, cada bloque subsiguiente debería actualizarse para reflejar este cambio. Por tanto, cualquier intento de manipulación de una transacción de un bloque validado provocaría un cambio en los hashes propagados, hasta llegar al hash raíz. Por tanto, si se detectara un intento de cambio, éste se invalida automáticamente y, lo mismo sucedería, si se intentan añadir transacciones.

Esta tecnología, además, permite a cualquier persona, empresa u organización realizar transacciones de manera fiable y precisa, mediante el uso de los llamados smart contracts (Tapscott, 2017). Estos smart contracts, de forma muy resumida, no son más que programas informáticos que se ejecutan automáticamente, sin intermediarios, en redes blockchain (Sheikh, 2019).

Desde una perspectiva jurídica, el término "contrato" es un "*acuerdo de voluntades por el que se exige el cumplimiento de una cosa determinada"*. Este acuerdo está escrito en términos legales, términos entendibles por los humanos, y crea o transfiere derechos y obligaciones entre dos o más partes. En esencia, es un documento dónde se identifican las partes contratantes, se define lo que se puede hacer, cómo se puede hacer, qué pasa si algo no se hace, etc. (Legerén-Molina, 2018).

Desde una perspectiva ingenieril, las especificaciones funcionales que los smart contracts deben satisfacer serían análogos a los términos, condiciones y reglas en un contrato legal convencional.

Figura 8. Marco de referencia de los smart contracts en la blockchain

Por tanto, como se muestra en la Figura anterior, un smart contract podría dar respuesta a un acuerdo entre partes, pero escrito como un *script* (programa informático). Este *script* sería entendible por las maquinas, y las operaciones realizadas a través de él quedarían automáticamente registradas en la red blockchain donde éste se despliegue. Los términos de este "contrato digital" no son más que comandos, sentencias en el código que lo forma, independientemente del lenguaje de programación utilizado.

Es importante indicar que los smart contracts puede codificar la ejecución automatizada de acuerdos, como se ha comentado, pero no todos los smart contracts tiene que ser acuerdos y no siempre codifican necesariamente acciones entre más de una parte. Por ejemplo, en las aplicaciones

descentralizadas (DApps), los smart contracts se utilizan para ejecutar lógica empresarial y transacciones sin necesidad de una autoridad central. Otro ejemplo, sería los tokens no fungibles (NFTs) que son representaciones digitales únicas de activos. En este caso, los smart contracts se utilizan para respaldar la creación, venta y transferencia de estos tokens únicos. Aunque no son contratos en el sentido tradicional, se basan en la tecnología de smart contracts para funcionar. Éstos son solo algunos ejemplos en los que los smart contracts se utilizan en aplicaciones más allá de los contratos tradicionales y la interacción entre múltiples partes. La tecnología blockchain y los smart contracts tienen un amplio potencial y pueden ser aplicados de diversas formas según las necesidades específicas de cada caso.

Si nos centramos en un caso de uso de un smart contract en transacciones comerciales, se puede apreciar con más detalle la importancia y potencial de estos programas informáticos. Por ejemplo, imaginemos que por un lado tenemos a una empresa que vende yogures y, por el otro, a una cadena de hipermercados que los compra. Están en diferentes países y es la primera vez que van a hacer negocios juntos. Acudir a un smart contract les facilitaría la transacción ya que estos programas operan de forma automática, esto significa que no es necesaria la verificación de una entidad supervisora y, además, garantizaría que cada una de las partes cumple con lo acordado en dicho contrato. Para ello, es imprescindible que cada una de las partes que intervienen en dicho contrato conozca, acepten y firmen con anterioridad las condiciones y los pasos que se van a permitir ejecutar durante su vigencia. Una vez desplegado el smart contract en una red blockchain, no se puede alterar la funcionalidad programada, es decir, tanto el smart contract, como cada paso o sentencia conformada en el programa informático es registrada en la blockchain y no se podrían modificar – dada la característica intrínseca de inmutabilidad de esta tecnología -.

A continuación, para disponer de un mayor conocimiento sobre esta tecnología, se exponen sus antecedentes y las características generales de su funcionamiento para, posteriormente, detallar los aspectos clave de los smart contracts, su pieza angular.

II.3.1. Antecedentes

La blockchain no apareció de la nada, ya que fue el producto de la evolución de las Fintech y las monedas virtuales en las últimas décadas. El uso generalizado de Internet, a finales del siglo pasado, favoreció el surgimiento de las monedas digitales como una extensión de los sistemas electrónicos de efectivo. Se desarrollaron numerosos proyectos para crear nuevas monedas digitales (Amsyar, 2020): DigiCash y CyberCash, por nombrar solo algunos. A pesar del enorme éxito alcanzado en la década de los noventa, muchos de estos proyectos dejaron de existir a principios de este siglo, ya sea por quiebra o clausurados por las autoridades, dado que era difícil generar confianza en tales entornos sin una autoridad central, y la creación de una moneda digital confiable pasó a ser todo un hándicap. El avance de la criptografía y la aparición de algunas soluciones inteligentes, afortunadamente, trajeron la esperanza de romper ese punto muerto.

En 2008, Satoshi Nakamoto (Nakamoto, 2008) aceptó el desafío y presentó una moneda digital llamada bitcoin. Para administrar la propiedad y asegurar el sistema, esta nueva moneda aprovechó eficazmente los métodos criptográficos, de ahí el nombre de criptomoneda. Nakamoto resolvió los problemas antes indicados al introducir lo que inicialmente llamó cadena de bloques. En su documento técnico publicado (Nakamoto, 2008), presentó su visión de un nuevo sistema de efectivo electrónico peer-to-peer — bitcoin — y describió en detalle su maquinaria subyacente — blockchain.

Desde su aparición, la blockchain ha revolucionado el panorama financiero, al permitir la transferencia segura de valor monetario en entornos sin "confianza", es decir, sin depender de intermediarios externos como bancos

y plataformas de pago. No obstante, esta maquinaria que sustenta el funcionamiento de las criptomonedas puede aplicarse a muchos otros procesos de intercambio de valor seguro, a través de Internet.

Si se hace un repaso de su rápida evolución y el estado actual de esta tecnología, se hace evidente que se trata de una tecnología disruptiva que sigue evolucionando y que tiene la capacidad de reconfigurar todos los aspectos de la sociedad actual, como se puede apreciar por las etapas por las que ha ido pasando (Tapscott, 2017).

Estas etapas, según se describe en el libro "blockchain: Blueprint for a new economy (Swan, 2015) son las siguientes:

blockchain 1.0 *(moneda)*	Invención de la criptomoneda Bitcoin. Como Bitcoin fue la primera implementación de criptomoneda, tiene sentido categorizarlo como primera generación de la tecnología blockchain para incluir solo las monedas criptográficas. Todas las monedas alternativas, incluida Bitcoins entrarían en esta categoría. Igualmente entrarían en esta categoría las aplicaciones relacionadas con el efectivo, como los sistemas de transferencia de criptomoneda, remesas y pagos digitales.
blockchain 2.0 *(contratos)*	Se introduce en esta generación los servicios financieros y los contratos. Las aplicaciones económicas y financieras que utilizan la blockchain de esta generación van más allá de las simples transacciones en efectivo, dando paso a acciones, derivados, bonos, swaps, préstamos, etc. La plataforma de código abierto Ethereum fue la encargada de introducirnos en esta nueva etapa con su concepto de contratos inteligentes (smart contracts).

blockchain 3.0 *(industria)*	De esta etapa forman parte la industria y otros ámbitos como gobierno, salud, ciencia, justicia, turismo, cultura, etc. Gracias a la aplicación de esta tecnología en numerosas entidades financieras y bancarias, la industria ha vuelto su mirada a esta tecnología con la finalidad de incrementar su competitividad.

Tabla 5. Etapas de la blockchain
(Swan, 2015)

Ya se empieza a hablar de la blockchain 4.0, evolución que incluye la inteligencia artificial como parte de la plataforma (Angelis, 2019), reduciendo la necesidad de gestión humana al permitir que las funciones tomen decisiones y actúen sobre los sistemas. Otros autores (Ratanasopitkul, 2018) indican que esta evolución se centrará en mejorar la escalabilidad, la flexibilidad y la eficiencia energética, ..., adaptando así la blockchain a entornos reales, contemporáneos y futuros.

Todo apunta a que esta tecnología ha llegado para quedarse y que ésta alcanzará, paso a paso, un nivel de adopción masiva. Su adopción e incorporación parece una estrategia inteligente, que permitirá a las empresas u organizaciones estar a la vanguardia de la economía digital y del futuro del mundo empresarial.

II.3.2. Aspectos generales

Desde que apareció la primera blockchain, todas se basan en la misma operativa (Preukschat A., 2017), siendo la criptografía, ese mecanismo que subyace, lo que les convierte en plataformas confiables e inviolables para el intercambio de información.

De forma resumida, la tecnología blockchain sería como una base de datos distribuida, en forma p2p (*peer-to-peer)*, cuyos registros (bloques) son inmutables al estar enlazados entre sí de forma criptográfica. Además, en el caso de la criptomoneda bitcoin, la información añadida a la red es pública, es decir, puede ser consultada por cualquier usuario de la red blockchain, en cualquier momento y de formar transparente. No obstante, actualmente existen redes públicas, privadas y semipúblicas, pero todas tienen un importante requisito, deben existir varios ordenadores (nodos) que son los encargados de verificar y validar las transacciones antes de ser registradas, de forma inmutable, en la base de datos descentralizada.

Por definición, la blockchain es una tecnología que permite que partes, no confiables plenamente, puedan mantener un "consenso" sobre la existencia, evolución y estado de los llamados activos digitales compartidos.

Una forma simple de explicar esta tecnología, entrando un poco en materia, sería compararla con un libro y los bloques con las páginas del libro, donde cada lector (ordenador) tiene una copia actualizada de ese libro. Ésta, en el caso de la blockchain, es un registro con todos los acontecimientos (transacciones) ocurridos dentro de la red blockchain. Cada transacción añade un nuevo bloque a la red, de forma cronológica o secuencial, cómo el número de páginas de un libro y, cada página está basada en la anterior, es decir, el contenido de una página se basa en lo que se venía contando en la página anterior.

Si se aterriza más este concepto, la blockchain sería un libro de contabilidad o mayor digital, que se distribuye en varias ubicaciones (ordenadores). Cada bloque hace referencia al bloque anterior, pero no lo hace a través de simples números, sino por una huella digital, llamada hash.

De forma ilustrativa, sin entrar en muchos detalles, esta tecnología funcionaría de la siguiente forma:

1. Un ordenador inicia una transacción y la propaga mediante el uso de un protocolo que valida dicha transacción en base a criterios previamente establecidos.

2. Tras la validación de la transacción, ésta se incluye en un bloque y se propaga por una red de ordenadores distribuidos en diversas ubicaciones. Es decir, no la posee solo una persona u organización, sino que está replicada en muchos sitios a la vez, sitios que están conectados entre sí y donde un cambio en uno de ellos afecta a todos los demás.

3. Este bloque, recién creado, se convierte en parte del libro mayor (entendido como una base de datos, un sitio en el que se escriben cosas como alfanuméricos, números, letras, … y programas informáticos). El bloque anterior se enlaza de forma criptográfica a este bloque, mediante su *hash*, donde las transacciones existentes se vuelven a confirmar cada vez que se crea un nuevo bloque.

Un ejemplo típico del funcionamiento de la blockchain para realizar pagos podría ser el siguiente (ver Figura 9)

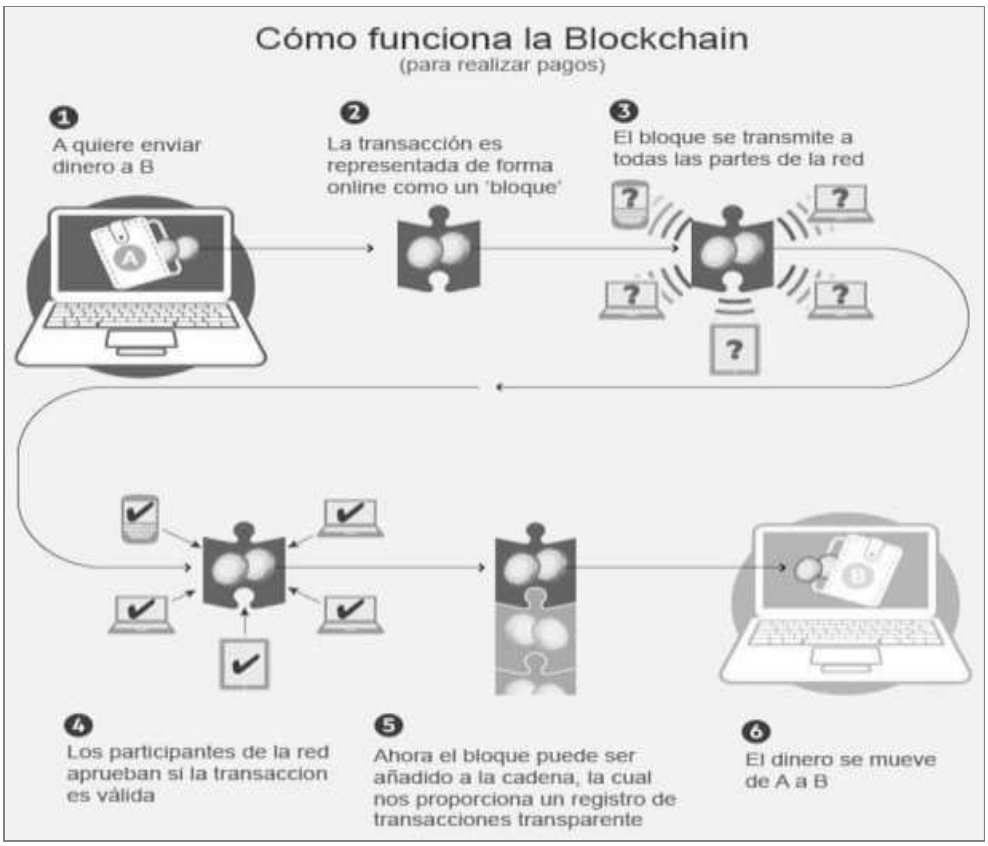


Figura 9. Cómo funciona blockchain
(Fuente: Methereum.com)

Las plataformas blockchain, normalmente, se clasifican en función de cómo se accede a los datos y si éstas gestionan o no los permisos de acceso a los mismos. Por tanto, si tenemos en cuenta cómo se accede a los datos, tendríamos la siguiente clasificación (ver Figura 10):

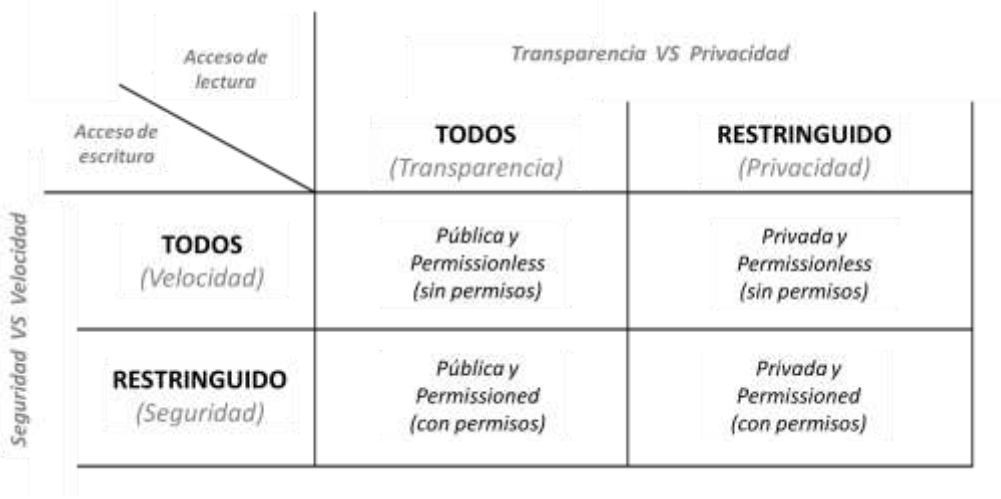


Figura 10. Tipos de plataformas blockchain

Concretamente, como se muestra en la Figura anterior, las plataformas blockchain pueden clasificarse según el acceso a ellas:

- ✓ Públicas. Son redes abiertas, cualquiera podría unirse a ella y participar en la misma, así como en el proceso de obtención de consenso, ya que la toma de decisiones sucede con mecanismos de consenso descentralizados como son PoW *(Proof of Work)*, PoS *(Proof of Stake)*, etc.

  Una de las debilidades de estas blockchains es que es necesaria una gran cantidad de proceso computacional para mantener un libro mayor distribuido a gran escala. Específicamente, para alcanzar el consenso, cada nodo de la red debe resolver un complejo, intensivo en recursos, problema criptográfico para asegurar que todos los nodos están sincronizados.

- ✓ Privadas, son redes que requieren una invitación, que debe ser validada por el creador de la red o por un grupo de restricciones puestas en marcha cuando se creó la red.

Normalmente, las organizaciones que abren una blockchain privada lo hacen aplicando permisos, al objeto de poner restricciones sobre acceso y uso de la red. El mecanismo de control de acceso puede variar, ya que pueden ser los participantes quienes decidan futuros entrantes o puede ser una autoridad reguladora quien otorgue las licencias para participar.

Los permisos de escritura se mantienen centralizados a una organización y los permisos de lectura pueden ser públicos o restringidos.

También existe la blockchain federada o de consorcio, éstas intentan eliminar la completa autonomía de una sola entidad sobre una blockchain. En este tipo de redes, básicamente, hay un grupo de organizaciones que se unen para tomar decisiones para toda la red (estos grupos son llamados consorcios o federaciones).

Al contrario que en las blockchain públicas, en estas redes no se permite a ninguna persona con acceso a internet participar en el proceso de verificar transacciones y el proceso de consenso es controlado por un grupo de nodos preseleccionados.

Otra categorización se puede establecer en función de quien puede leer o escribir en la blockchain, es decir, son redes con permisos (Permissioned) o sin permisos (Permissionless). Por tanto, teniendo en cuenta estas categorizaciones, una blockchain puede ser Publica, con o sin permiso o Privada, con o sin permisos.

Para mayor compresión de esta tecnología, en el Anexo de la presente Tesis, se ha incluido información detallada sobre las características de esta tecnología, trabajo realizado para identificar claramente y analizar como esta tecnología gestiona la información de manera segura y trazable, proporcionando transparencia y privacidad a los usuarios.

II.3.3. Smart contracts

Como ya se ha comentado, un smart contract es un programa informático diseñado para ejecutarse automáticamente en una red blockchain. Estos programas amplían la funcionalidad de la red blockchain y permiten a partes no confiables establecer confianza en la ejecución veraz de un acuerdo (Shailak, 2020).

En un contexto empresarial, es habitual realizar una integración entre los procesos de negocio y la red blockchain (Viriyasitavat, 2019), siendo los smart contracts la pieza clave para realizar esta integración ya que, desde un punto de vista externo, estos programas informáticos constituyen los puntos de integración con la red blockchain. Ocurre lo mismo cuando la blockchain requieren datos externos, datos que están fuera de la red, siendo los smart contracts ese punto de unión con el mundo exterior, a través del Oráculo (plataforma externa a la blockchain).

Aunque Ethereum blockchain fue una de las soluciones pioneras en usar los smart contracts, actualmente el mundo empresarial tiene a su disposición una extensa lista de plataformas que los usan (Nanayakkara, 2021)(Kuo, 2019). En estas plataformas, los smart contracts se invocan utilizando diferentes protocolos, técnicas y formatos de datos, pero la forma de implementarlos parece que es como la de un tipo Class en cualquier lenguaje de programación orientado a objetos (OOPL, siglas de *Object-Oriented Programming Language*). Esta característica de programación, junto con el factor humano, hacen que el proceso de desarrollo de estos programas sean propensos a errores.

En un contexto de multiplataformas y, sobre todo, teniendo en cuenta las peculiares características de la tecnología blockchain, los smart contracts deberían ser diseñados e implementados utilizando las mejores prácticas que permitan reducir el número de errores de codificación y los errores operativos (Huang, 2017).

A continuación, al objeto de seguir profundizando en las características de los smart contracts, se analiza su finalidad, su aplicación desde un punto de vista legal, su anatomía y como se estructuran.

I.3.3.1. Finalidad

Como ya se ha comentado, uno de los pilares fundamentales de la tecnología blockchain es el smart contracts. La primera conceptualización de este término fue publicada por Nick Szabo (Szabo, 1994) que lo definió como un conjunto de cláusulas, datos y protocolos. Estos protocolos implementarían algoritmos para verificar automáticamente el cumplimiento de las condiciones establecidas y serían más "inteligentes", que los contratos en papel ya que hacen cumplir automáticamente las obligaciones de las partes.

La primera plataforma blockchain en implementar los smart contracts fue Ethereum (Buterin, 2014), creando un lenguaje de programación para dichos contratos llamado Solidity, el cual es Turing completo (Ethereum, 2022).

Por tanto, un smart contract es un *script* escrito en algún lenguaje "maquina", que permite a un dispositivo ejecutar de forma automatizada las instrucciones previamente programadas, prescindiendo de cualquier tipo de intervención humana.

Las instrucciones que contiene ese *script* no son más que sentencias y comandos en el código que lo forma. Se podría decir que, cada una de las cláusulas que contiene ese *script*, se ha redactado siguiendo el paradigma "if X then Y, else Z". Luego la máquina que ejecute ese código, lo hará inexorablemente en la forma programada. Con esta ejecución, las partes asegurarían que ciertas acciones o eventos ocurran en el marco de ese conjunto de cláusulas y condiciones.

De forma resumida, el objetivo general de un smart contract sería satisfacer o validar las condiciones acordadas, sin necesidad de intermediarios y

almacenar el resultado de dicha ejecución en la blockchain. De forma resumida, la siguiente ilustración auto-explicativa muestra el uso de los smart contracts.

Figura 11. Ejemplo de uso de los smart contracts

Con estas mimbres, cualquier bien (dinero, propiedades, etc.) podría ser compartido entre partes de manera transparente sin la necesidad de un tercero de confianza, eliminando los posibles conflictos de intereses. Obviamente, el hecho de eliminar la autoridad central o un tercero de confianza en un proceso de negocio y delegar esa tarea en un smart contract hace que su diseño y programación deba ser rigurosa, ya que no debe contener errores de ningún tipo que puedan provocar situaciones no previstas, ni deseables.

Por último, aunque ya se ha mencionado, es importante recalcar que los smart contracts pueden codificar la ejecución automatizada de acuerdos, tal y como ya se ha expuesto, pero no todos los smart contracts tiene que ser acuerdos y no siempre codifican necesariamente acciones entre más de una parte.

II.3.3.2. Ámbito legal

La cuestión de si los smart contracts son legalmente vinculantes puede variar según la jurisdicción y la interpretación legal específica de cada país. En general, los smart contracts se consideran como una forma de contrato que utiliza la tecnología blockchain y la ejecución automatizada para hacer cumplir los términos y condiciones establecidos en dicho acuerdo.

En concreto, la legalidad de éstos depende de varios factores, entre ellos (Legerén-Molina, 2018):

- Jurisdicción: Cada país puede tener leyes y regulaciones diferentes con respecto a la validez y el reconocimiento de los contratos, incluidos los smart contracts. Algunos países pueden tener leyes específicas relacionadas con los contratos electrónicos y la tecnología blockchain.

- Intención y consentimiento: Como en cualquier contrato, la legalidad de un smart contract depende de que todas las partes involucradas tengan la intención de cumplir con sus términos y hayan otorgado su consentimiento de manera voluntaria y consciente.

- Ejecución y cumplimiento: Si un smart contract se implementa correctamente y las condiciones se cumplen según lo previsto, puede considerarse legalmente vinculante.

- Efectos legales: Los smart contracts deben ser capaces de producir efectos legales en un sistema legal determinado para ser reconocidos como vinculantes. Esto implica que las acciones o transacciones realizadas a través del smart contract deben tener relevancia legal en el contexto en el que se utilizan.

En algunas jurisdicciones, los smart contracts ya han sido reconocidos como legalmente vinculantes y válidos para ciertos tipos de transacciones. Sin embargo, en otros lugares, todavía se están estudiando y desarrollando

marcos legales para abordar adecuadamente los aspectos legales de esta tecnología emergente.

No obstante, una de las cuestiones principales que se debaten en el ámbito legal es si estos programas informáticos podrán algún día llegar a ser considerados, o no, como un sustituto de los contratos legales. Esto se refiere tanto a su legitimidad como a los requisitos para formalizar un contrato legal (Werbach, 2017) (Legerén-Molina, 2018).

En este sentido, existen algunas iniciativas muy interesantes como es el contrato Ricardiano (Grigg, 2004). Este es un método para registrar un documento como un contrato en derecho y, vincularlo de forma segura a otros sistemas. En concreto, este autor introduce el concepto de contrato Ricardiano en un primer sistema de pago electrónico, donde un documento contractual se podía firmar electrónicamente y se intercambiaba digitalmente de forma encriptada. Por tanto, un este tipo de contrato no es más que un archivo de texto que está firmado criptográficamente por un emisor legal y que expresa algún valor para sus titulares (Grigg, 2004).

Si se analiza en detalle la estructura de un contrato legal, éste contiene aspectos operativos que pueden ser automatizados mediante software. Por ejemplo, los pagos periódicos de un préstamo hipotecario podrían ser susceptibles de ello, pero también existen otros aspectos de "naturaleza interpretativa".

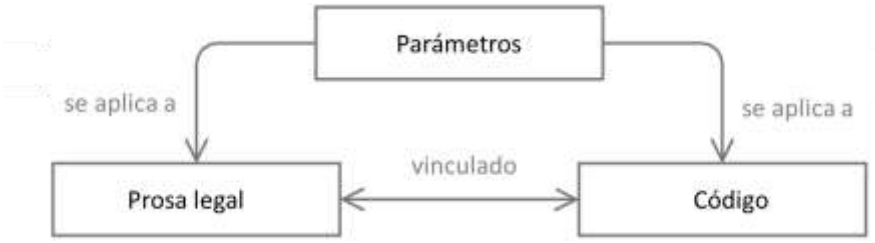


Figura 12. Componentes básicos del Contracto Triple Ricardiano

Grigg, siendo consciente de este problema, perfeccionó el Contrato Ricardiano dando lugar al Contrato Triple Ricardiano (véase Figura *12*), indicando tres componentes básicos:

- Prosa legal. Los contratos legales, algunas de sus cláusulas, no contienen descripciones de comportamiento procesables por una máquina, ya que estos aspectos "no operativos" son intangibles, ambiguos o de naturaleza interpretativa y se enmarcaría en la prosa legal o jurídica.

- Código. Este componente, escrito en un lenguaje de programación ejecutable por máquinas, permite a los sistemas de software automatizar ciertas partes del contrato legal subyacente, es decir, este componente recoge los aspectos "operativos" de los contratos legales.

- Parámetros. Representan los resultados de la negociación a nivel de un acuerdo concreto y vinculante entre partes. El código y los componentes de la prosa legal los considera como "plantillas", que se concretan para un fin determinado y es donde se fijan los valores concretos de estos parámetros, por ejemplo, el plazo para pagar una deuda o la subida anual de un arrendamiento.

Teniendo en cuenta el Contrato Triple Ricardiano y la división propuesta: datos (Parámetros); aspectos operativos (Código) y no operativos (Prosa), la estructura de un smart contract debería guardar cierta similitud con las asociaciones mostradas en la Figura 13.

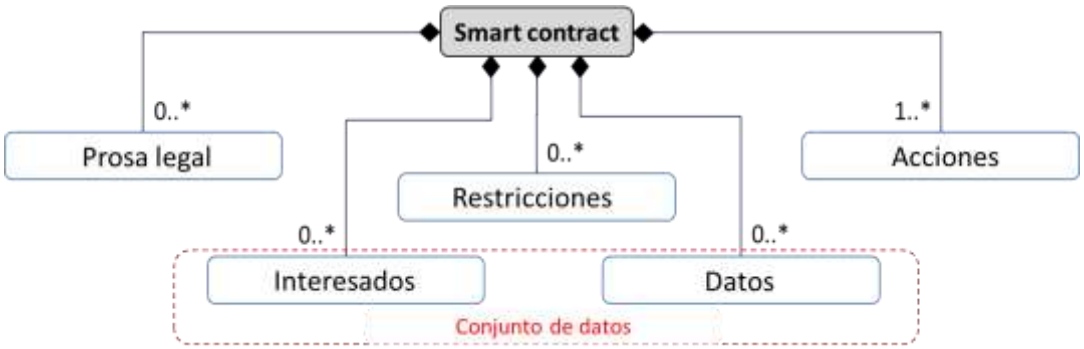


Figura 13. Asociaciones de un Contrato Triple Ricardiano

Esta estructuración del Contrato Triple Ricardiano es como una tupla que puede tener o no un conjunto de relaciones jurídicas entre partes (Prosa legal) y donde Datos e Interesados (Conjunto de datos) son parte del componente Parámetros - pudiendo no existir partes interesadas, ni datos – y Acciones (especificaciones funcionales) y Restricciones (especificaciones de comportamiento) son partes del componente Código.

No obstante, aunque todo el tema legal o jurídico es una temática bastante interesante, todo este debate queda fuera del alcance de la presente Tesis Doctoral, es decir, en ningún momento se ha evaluado o evalúa cualquier conflicto filosófico relativo a la ubicación exacta de los smart contracts dentro del ámbito legal. A día de hoy, los expertos jurídicos todavía difieren - desde un punto de vista conceptual - sobre si los smart contracts realmente se ajustan a la definición de contrato legal, aludiendo también a si éstos se ajustan adecuadamente al ciclo de vida del contrato y si cumplen los pasos legalmente requeridos para celebrar un acuerdo vinculante (Werbach, 2017) (Legerén-Molina, 2018).

Luego, centrándonos en la Figura 13 y dejando al margen los aspectos legales, un smart contract debería permitir un acuerdo entre partes, que serían los interesados[8] y, existe una automatización del smart contract, gracias a las acciones y restricciones establecidas, ya que éstas se aplican automáticamente, de forma autónoma, mediante la ejecución del Código. En concreto:

- Las acciones se componen de actividades u operaciones a realizar, y es atómica desde el punto de vista de su aplicación. Además, las acciones pueden recibir datos y también puede leer y escribir datos. Estas acciones pueden contener pasos y el hecho de que una acción se pueda realizar, o no, puede estar sujeto a restricciones de comportamiento.

- Las restricciones modelan los términos y condiciones del acuerdo, es decir, imponen restricciones en cuanto a cuándo se puede realizar una acción, si las circunstancias permiten o no ejecutar la acción, etc. Por tanto, en un smart contract, una restricción vincula la ejecución de una acción al cumplimiento de las condiciones y reglas. Una restricción puede leer datos y podría afectar a una o más acciones.

II.3.3.3. Anatomía

Un smart contract es una colección de código (sus funciones) y datos (su estado) que residen, una vez desplegado, en una dirección específica en la red blockchain (Ethereum, 2022).

[8] Estos, también conocidos como firmantes, pueden ser una persona, una organización o cualquier otra entidad capaz de celebrar un acuerdo legal.

En concreto, los componentes básicos de un smart contract son las propiedades y la lógica y, el libro mayor (ledger) donde reside el smart contract una vez desplegado y los datos. Cada uno de estos componentes puede ser mapeado directamente en conceptos técnicos, donde las propiedades representan un esquema de datos y la lógica representa el código y, la base de datos sería el libro mayor.

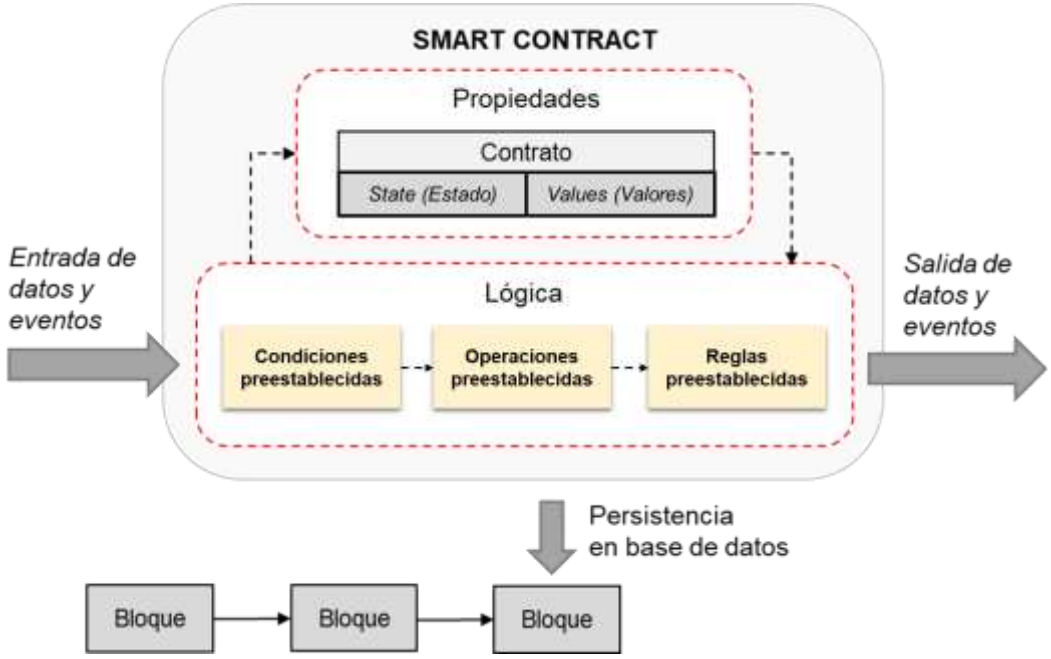


Figura 14. Anatomía de smart contract en la blockchain

Como muestra la Figura 14, los smart contracts se ejecutan al recibir una transacción (entrada de datos y eventos) y, durante su ejecución, pueden enviar mensajes (salida de datos y eventos) a otros contratos o sistemas externos (Ethereum, 2022). Estos mensajes contienen la dirección del remitente y la del destinatario, así como los parámetros de entrada del smart contract del destinatario y los datos a transferir al destinatario, si procede.

Es importante indicar que hay dos tipos de cuenta, donde una cuenta es un elemento fundamental que permite a los usuarios interactuar con la red blockchain y realizar operaciones dentro del ecosistema de estas plataformas (Ethereum, 2022). Estas cuentas son esenciales para el funcionamiento de estas plataformas y para interactuar con los smart contracts. En concreto, tenemos:

- La cuenta externa (EOA, siglas de *External Owned* Account). Estas cuentas están controladas por claves privadas, lo que significa que los usuarios tienen control directo sobre ellas. Una cuenta externa tiene una dirección única, que es análogo a una cadena de caracteres hexadecimal.

- La cuenta del smart contract (una vez desplegado en la red blockchain). Cada smart contract tiene también una dirección única y son utilizadas para poder automatizar la ejecución de las transacciones.

Ambos tipos de cuenta tienen la capacidad de recibir, mantener y enviar tokens e interactuar con otros smart contracts desplegados. Pero solo la cuenta externa puede iniciar una transacción, mientras que la otra cuenta (la del smart contract) sólo puede enviar respuestas a la recepción de una transacción.

Entrando en un mayor nivel de detalle, siguiendo la Figura 14, la composición de los smart contract sería la siguiente:

- Propiedades (o datos). El estado del smart contract son sus datos y éste sólo puede ser modificado por el propio smart contract. Los datos de un smart contract se asignan a una ubicación, ya sea en memoria o almacenamiento, de la plataforma blockchain en cuestión. Existen dos tipos de datos:

- Persistentes. Datos que están representados por variables de Estado (State) y que estarán almacenados de forma permanente en la red blockchain.

- Temporales. Datos que están representados por valores de variables (Values). Son variables ubicadas en memoria, que serán usadas solamente durante la ejecución de una función del contrato. Estos valores se denominan variables de memoria y pueden ser locales o globales.

Además de estas variables (datos), también existen algunas variables globales específicas e intrínsecas de la plataforma blockchain en cuestión, que se utilizan principalmente para proporcionar información sobre la red blockchain o la transacción actual.

- Lógica (o funciones). Condiciones, operaciones y reglas preestablecidas en el código del smart contract, que obtienen información o establecen información en respuesta a las transacciones recibidas.

  Existen unas operaciones específicas, llamadas constructores, que se ejecutan solo una vez cuando el contrato se invoca por primera vez. Como ocurre con los constructores de cualquier OOPL, estos métodos o funciones permiten inicializar datos a unos valores preestablecidos.

Tanto los datos como las funciones pueden ser públicas o privadas, como ocurre en cualquier OOPL. En el caso de las funciones, las públicas pueden llamarse internamente, desde dentro del contrato, o externamente a través de mensajes y, las funciones privadas, solo son visibles para el smart contract en el que se definen.

II.3.3.4. Estructura

Actualmente, las plataformas blockchain soportan diferentes lenguajes de programación para implementar los smart contracts (Valerievitch, 2019). Si se realiza una búsqueda por internet, se puede ver que los lenguajes de programación más utilizados son JavaScript, Java, Python, C#, C++ y Ruby. No obstante, también existen lenguajes de programación específicamente diseñados para escribir ethereum smart contract como, por ejemplo, Solidity.

Hay una idea, comúnmente aceptada, que insinúa que los contratos de Ethereum necesariamente deben estar escritos en Solidity, pero esto no es cierto (Ethereum, 2022). Es más, uno de los atractivos de la plataforma ethereum, es que se puede usar casi cualquier lenguaje de programación, aunque se potencie unos lenguajes específicos como Solidity o Vyper (Ethereum, 2022).

En cualquier caso, como ya se ha mencionado, los smart contract tienen unos elementos clave: los Datos (estado) y la Lógica (comportamiento).

Figura 15. Estructura básica de smart contract en la blockchain

Los elementos más importantes de esta estructura, que muestra esquemáticamente la Figura 15, se pueden resumir de la siguiente manera:

- Smart contract. El nombre del smart contract debe ser único, aunque una vez desplegado el código, éste tenga una dirección específica que corresponde a una ubicación, es decir, un smart contract es un programa que se ejecuta en una determinada dirección de la red blockchain.

- Datos (estado). Son las variables persistentes, es decir, los datos del smart contract que se almacenan de forma permanente. Estas variables serán guardadas en una dirección de la plataforma blockchain y podrán ser leídas o escritas por los usuarios y/p por otros smart contracts. Como ya se ha comentado, dada la característica de inmutabilidad de la blockchain, el estado no se puede alterar después de la inicialización, lo único que se puede hacer es escribir modificaciones de las variables de estado durante la ejecución del smart contract.

- Lógica (comportamiento). Son métodos con operaciones, eventos, condiciones, reglas, etc.:

    - Funciones. Son unidades de código ejecutable dentro de un contrato, es decir, son operaciones que un contrato puede realizar y, por lo tanto, constituyen su funcionalidad. Éstas pueden ser privadas o públicas y, cuando se ejecuta una función, las variables de estado del contrato pueden cambiar dependiendo de la lógica implementada en la función.

      Todas las funciones tienen un nombre único, parámetros de entrada y, opcionalmente, parámetros de salida. Algunas plataformas blockchain permiten la invocación directa de funciones usando su nombre, mientras que otras fuerzan el uso

de una sola función "despachador", para reenviar valores de entrada a las funciones de destino.

- Constructores. Esta funcionalidad se utiliza para inicializar las variables de estado de un contrato. Si no se define ningún constructor, existe un constructor predeterminado en el contrato.

- Modificadores. Esta funcionalidad se usa para añadir o cambiar el comportamiento de las funciones. Éstos están asociados a una función y permiten cambiar el comportamiento de ejecución de la misma, ya que se ejecuta justo antes de ejecutar la función a la cual está asociado. Por ejemplo, se utilizan para comprobar de forma automática que una condición se cumple antes de ejecutar la función en cuestión.

- Eventos. Esta funcionalidad permite disparar eventos cuando ocurre algo significativo y tiene que comunicarlo, por ejemplo, a una DApp (aplicaciones descentralizadas) o a otro smart contract.

  Algunas plataformas blockchain pueden generar eventos del sistema, otras admiten eventos personalizados definidos por el desarrollador y, dependiendo de la plataforma, se pueden lanzar uno o varios eventos.

Como se puede apreciar, viendo todos estos elementos, un smart contract no es más que un tipo Class, que es la base de cualquier OOPL y los constructores, modificadores y eventos serían como una "generalización" de la función. Para una mejor compresión, en la Figura 16, se ilustran todos estos elementos con un ejemplo de una estructura básica de smart contract de Ethereum.

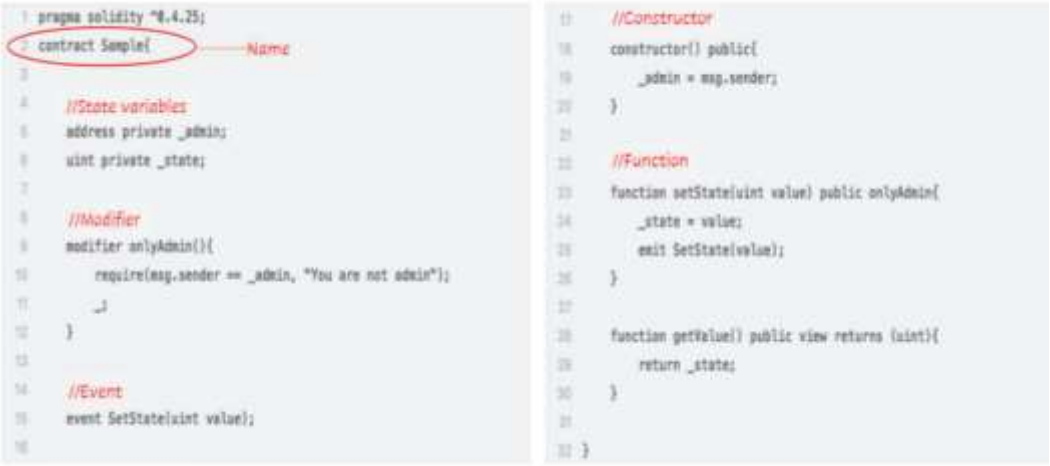


Figura 16. Ejemplo estructura básica de smart contract de Ethereum

Como se puede apreciar en la Figura *16*, el código de un smart contract, en este caso de Ethereum, se divide en los elementos indicados anteriormente:

- Datos o variables de estado. Ésta es la columna vertebral del smart contract. Registra la información del contrato. Estos datos se almacenan de forma permanente y pueden ser solo modificados por las funciones. Estas modificaciones entrarán en vigencia después de que la transacción haya sido confirmada por la red blockchain.

- Modificadores. Éstos se utilizan para cambiar el comportamiento de las funciones de una manera ágil. Éstos son capaces de comprobar una condición, por ejemplo, OnlyAdmin, antes de ejecutar una función.

- Eventos. Al efectuar una transacción sobre un smart contract, pueden generarse eventos. Algunos de estos eventos, por ejemplo, pueden dejar un registro con otra información adicional a la propia transacción en la blockchain (a modo de logs), que es accesible y puede ser consultada y aprovechada por aplicaciones externas.

- Constructores. Utilizados para inicializar el smart contract, lo que permite que los datos pasados al instanciar el smart contract, puedan ser inicializados y almacenados como variables de estado. A diferencia de otros OOPL, Solidity (Ethereum) solo puede especificar un constructor.

- Funciones. Son las operaciones que un contrato puede realizar y, por lo tanto, constituyen su funcionalidad.

La representación UML del tipo Class, del smart contract "Sample" de la Figura 16, sería la siguiente (ver Figura 17):

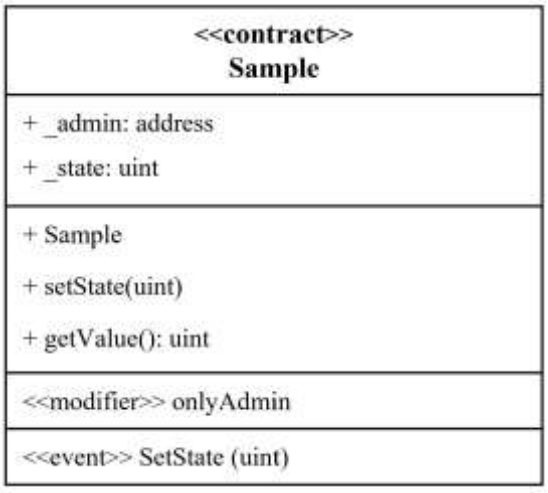


Figura 17. Tipo Class "Sample"

II.5. Identificación de la problemática existente

La blockchain es una tecnología inmutable, transparente y segura para registrar el estado y la propiedad de un activo (Sultan, 2018). Este activo puede ser algo tangible y diverso como dispositivos IoT, una obra de arte, etc. e incluso podría ser algo intangible como, por ejemplo, la propiedad intelectual, los derechos de autor, etc.

Solo hay que hacer una búsqueda por internet para ver que ya existen numerosas plataformas blockchain que usan smart contracts (Ethereum, Hyperledger, Stellar o EOS, entre otros) (Zheng Z., 2020). Estas plataformas permiten construir y ejecutar estos programas informáticos sin seguir las buenas prácticas de la ingeniería software y, lo más preocupante, sin pasar por rigurosos procesos de prueba. Esto significa que el software desplegado en estas redes puede estar ejecutando código con errores, bugs y vulnerabilidades.

Viendo esta problemática, ya han aparecido empresas auditoras de smart contracts. Por ejemplo, OpenZepelin (www.openzeppelin.com) cuya oferta de servicios plantea desde la auditoría del código hasta la construcción de smart contracts. Otras empresas especializadas en este tipo de servicios son ConsenSys (consensys.net), Quantstamp (quantstamp.com) o Certik (www.certik.com) que ofrecen servicios de detección de vulnerabilidades o violaciones de seguridad en el código de los smart contract.

Dada la característica intrínseca de inmutabilidad de la blockchain, es fundamental que antes de desplegar código en una red blockchain empresarial, este software pase por rigurosos procesos de evaluación y validación. Un error o defecto en el código de un smart contract puede tener consecuencias graves y potencialmente irreparables en una red blockchain. Algunos escenarios que pueden surgir si un smart contract tuviera un error o defecto:

- Pérdida de fondos. Si un smart contract tiene un error de programación que permite que los fondos almacenados en él sean manipulados o robados, esos fondos podrían perderse permanentemente. Una vez que se ejecuta el smart contract en la blockchain, sus operaciones son inmutables y no se podría volver atrás.

- Paralización de contratos. Un defecto en un smart contract podría provocar un fallo en su funcionamiento, lo que puede llevar a que el contrato quede inaccesible o bloqueado. Si el contrato afectado es esencial para el funcionamiento de otras aplicaciones o contratos, podría generar problemas en cadena.

- Comportamientos inesperados. Los errores en el código pueden hacer que el smart contract se comporte de manera inesperada o impredecible. Esto podría conducir a resultados no deseados o incluso perjudiciales para las partes involucradas en las transacciones.

- Congestión de la red. Un smart contract con un defecto puede entrar en un bucle infinito o realizar operaciones costosas en tiempo de computación. Esto puede provocar congestión y afectar el rendimiento general de la red blockchain.

Es importante destacar que, debido a la naturaleza inmutable de la blockchain, una vez que un smart contract ha sido desplegado, no se puede cambiar su código. Si se descubre un error después de su despliegue, no se puede simplemente corregir el código existente. La única opción sería crear un nuevo contrato con el código corregido y, en la medida de lo posible, migrar los fondos y la lógica del contrato antiguo al nuevo. Sin embargo, esto dependerá de la complejidad y el impacto del error y puede ser difícil o incluso imposible de realizar en ciertos casos.

Para mitigar estos riesgos, es fundamental realizar pruebas exhaustivas, revisiones de código y auditorías de seguridad antes de desplegar un smart contract en la red blockchain. Además, se recomienda seguir las mejores prácticas de desarrollo de smart contract y mantenerse actualizado con las últimas investigaciones y avances en seguridad para minimizar la posibilidad de errores costosos.

Además, desde nuestro punto de vista, sería una buena práctica que las especificaciones del producto software, así como las especificaciones, condiciones y reglas que deben cumplir los smart contracts fueran una parte integral de un marco de trabajo único. Esto supone que, en el caso de las pruebas funcionales, todos los requisitos del negocio han sido recopilados previamente y éstos serán verificados y validados a través de los casos de prueba, con argumentos válidos y no válidos, con valores límite o realizando combinaciones de argumentos.

Probar manualmente todas las extensiones funcionales de cualquier software no solo es difícil, sino también ineficiente. Es más, probar el funcionamiento de una solución no es posible sin la generación automática de pruebas funcionales, debido a las numerosas combinatorias existentes en las condiciones previas del caso de prueba. Por tanto, desde nuestro punto de vista, es más eficiente que este marco de trabajo se sustente en una plataforma y que ésta, además, permita organizar la documentación elaborada por vistas o puntos de vista, al objeto de que ésta sea entendible por todas las partes interesadas.

Teniendo en cuenta estas premisas y sin olvidar que se requieren pruebas funcionales y no funcionales para validar y corregir el comportamiento general del software, podemos ya entender la hipótesis planteada en el presente trabajo de Tesis Doctoral (Tabla *2*):

> *"La aplicación de los principios y técnicas del testing temprano sobre el ciclo de vida de desarrollo de software, podría ayudar a generar pruebas funcionales de los smart contracts, de forma independiente de la plataforma blockchain, al objeto de garantizar que las transacciones realizadas a través de estos programas informáticos se ejecutan conforme a las especificaciones, condiciones y reglas preestablecidas".*

Es importante indicar que el enfoque de las pruebas del software siempre debe estar centrado en la funcionalidad, la seguridad y el rendimiento, pero esta Tesis Doctoral se centrará solo en la funcionalidad. Es decir, se centrará en la posibilidad de aplicar mecanismos sistemáticos para asegurar la calidad funcional de los smart contracts, en un contexto blockchain, desde etapas tempranas del ciclo de vida del desarrollo.

Todo lo indicado hasta ahora parece que podría ayudar a reducir el número de defectos de estos *scripts* (programas informáticos) y, con ello, minimizar el posible impacto de transacciones no reversibles en una red blockchain.

Por tanto, como ya se ha comentado, este trabajo pretende dar respuesta a la pregunta de investigación (Tabla 1):

> *¿Los principios que rigen las pruebas tempranas en el ciclo de vida del desarrollo de software son aplicables, en un contexto blockchain, para garantizar la calidad funcional de los smart contracts?.*

Es decir, se podrían obtener casos de prueba funcionales en fases tempranas del ciclo de vida de desarrollo de los smart contracts, de forma sistemática y automatizada, que permitan mejorar la verificación y validación del comportamiento definido y esperado de estos programas informáticos, antes de ser desplegados en una red blockchain.

Para ello se propone, como se irá viendo a continuación, aplicar técnicas de ingeniería software y técnicas de modelado y transformaciones propias del paradigma MDE, que permitan instanciar casos de prueba funcionales a partir de una especificación desarrollada mediante requisitos funcionales y, sobre todo, a partir de los acuerdos, condiciones y reglas de negocio, establecidos entre las partes.

II.6. Visión general de la solución propuesta

Como se ha venido analizando en la presente Tesis, la aplicación del paradigma de la ingeniería dirigida por modelos ha ofrecido soluciones adecuadas para la generación de pruebas tempranas en otros contextos y, sobre todo, ya ha sido aplicado con éxito en numerosos proyectos de investigación y de transferencia por el grupo ES3.

Por tanto, el reto que nos planteamos en la presente Tesis para resolver la problemática existente es abordar una solución encaminada al uso de metamodelos, modelos y a transformaciones entre modelos, aprovechando los trabajos previos ya implementados en la solución SofIA e incluir aquellos mecanismos que sean necesarios para alcanzar nuestros objetivos.

En concreto, el punto de partida de nuestra visión general de la solución es el esquema que se muestra en la Figura 18. Este diseño basado en el paradigma de la ingeniería dirigida por modelos (MDE) tiene como objetivo obtener uno de los pilares de la solución global, la modelización (descripción) del smart contract.

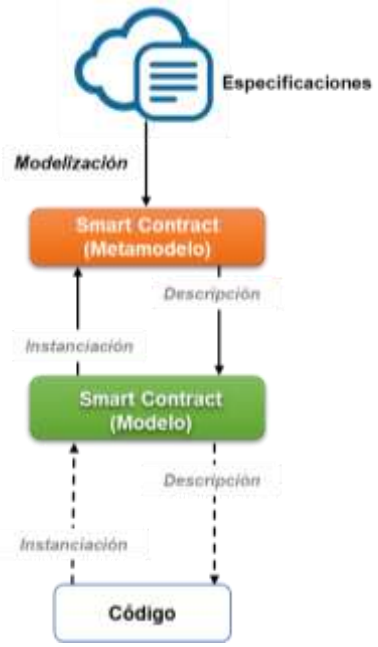


Figura 18. Enfoque MDE para modelar smart contracts

Este enfoque, como se puede apreciar, es intrínsecamente jerárquico donde el código del smart contract podría llegar a ser una instancia de un modelo de smart contract concreto (que resuelve un problema concreto). El modelo del smart contract proporcionaría un plano general de los acuerdos, operaciones, condiciones y reglas que fueron negociados por las partes interesadas. En concreto, este modelo sería una instanciación del metamodelo de smart contract. Este metamodelo es el artefacto encargado de describir e interrelacionar, de forma abstracta, los elementos que estructuran cualquier smart contract, así como las pautas a tener en cuenta sobre su lógica.

Con este enfoque, ya se podría plantear una visión más detallada del esbozo de *Graphical Abstract* (recogido en la Figura 5). En concreto, en la Figura 19 se muestra este detalle, donde se propone definir el modelo de smart contract y los modelos de pruebas funcionales, ambos, a partir de sus metamodelos respectivos que estarán interrelacionados.

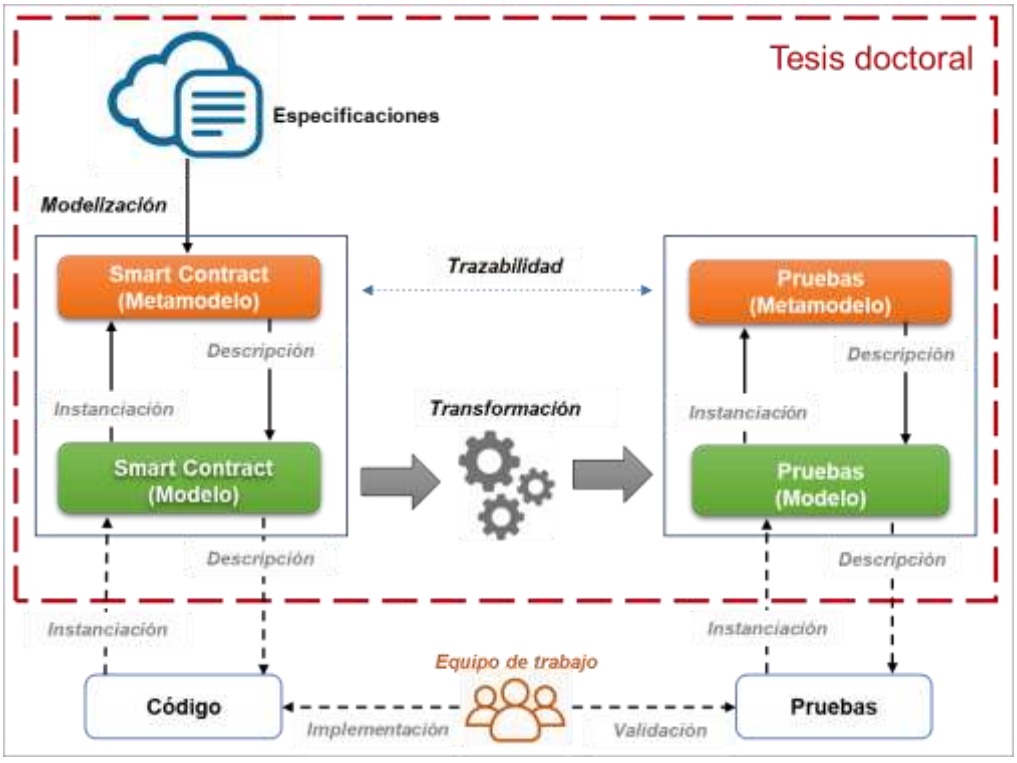


Figura 19. Graphical Abstract de la Tesis - versión detallada -

Como se puede apreciar en la Figura anterior, lo que se propone es generar el modelo de pruebas funcionales desde el modelo de smart contract, usando para ello un motor de transformaciones que instanciará un conjunto de conversiones que se definen. La trazabilidad entre metamodelos, que aparece en la Figura, se refiere a la necesidad de establecer relaciones y conexiones entre los metamodelos, lo que permitirá rastrear y entender cómo los diferentes metamodelos se relacionan entre sí y es útil para: garantizar que los modelos derivados de diferentes metamodelos son consistentes entre sí, evitando conflictos o discrepancias en la información representada; comprender cómo los cambios en un metamodelo afectan a otros modelos que dependen de él, lo que facilita el análisis de impacto de los cambios propuestos y facilita la reutilización de elementos de modelos entre diferentes metamodelos, lo que puede ahorrar tiempo y esfuerzo en el proceso de modelado.

La trazabilidad entre metamodelos puede lograrse mediante la definición de relaciones formales entre los elementos de los diferentes metamodelos, estableciendo mapeos y enlaces entre ellos. Estos mapeos pueden ser unidireccionales o bidireccionales, dependiendo de las necesidades y la complejidad de los mismos.

Por tanto, en posteriores Capítulos, se estudiará como obtener estos modelos a partir de la instanciación de un metamodelo de smart contract y de un metamodelo de pruebas, respectivamente. Metamodelos que también habrá que definir.

En el siguiente Capítulo, antes de seguir avanzando, se presenta el estudio del estado del arte en el contexto del ciclo de vida de desarrollo de los smart contracts, al objeto de analizar si hay estudios primarios que hayan abordado la ingeniería dirigida por modelos u otras buenas prácticas para analizar, diseñar, probar y/o implementar los smart contracts.

II.7. Resumen del Capítulo

En este Capítulo se ha concretado el ámbito de investigación en la que se desarrolla la presente Tesis Doctoral. En concreto, se ha realizado, por un lado, una introducción del concepto de ingeniería del software, del paradigma de la ingeniería dirigida por modelos y del concepto de reutilización que, según la literatura existente, permite mejorar la calidad del software que se construye y facilita la reutilización de elementos de modelos entre diferentes metamodelos. Por otro lado se ha realizado un análisis pormenorizado de la tecnología blockchain y, sobre todo, de los smart contracts, detallando su finalidad, su anatomía y su estructura.

Una vez contextualizada la investigación, se ha identificado el problema existente, el cual es consecuencia de las características intrínsecas de la tecnología blockchain, su inmutabilidad - característica que la tecnología blockchain destaca como un beneficio clave -. Esta característica obliga a que todo software que se despliegue en una red blockchain, deba pasar por rigurosos procesos de evaluación y de validación, ya que un defecto o error en cualquier smart contract desplegado y ejecutado en una red blockchain podría causar un efecto no reparable.

Por último, este Capítulo finaliza aportando una visión general de la solución objetivo y los retos a los que nos enfrentamos al objeto de mitigar dicho problema.

Capítulo III.
Trabajos relacionados

"If you spend too much time thinking about a thing,
you'll never get it done".
– Bruce Lee –

Una revisión sistemática de la literatura nos ha permitido reunir diversos trabajos de investigación donde se estudian algunos de los problemas a los que se enfrenta esta Tesis Doctoral.

En concreto, en este Capítulo se analizan diferentes estudios al objeto de explorar posibles lagunas en la literatura existente y, sobre todo, motivar la producción de esta Tesis Doctoral basada en la hipótesis de investigación descrita anteriormente. Como se podrá apreciar posteriormente, los resultados obtenidos y expuestos en este Capítulo muestran que hay margen de acción para aportar nuevos enfoques en esta línea de trabajo.

El desarrollo de software basada en modelos, en los últimos años, se ha convertido en un enfoque práctico, eficiente y exitoso para describir diferentes aspectos del software durante todo el ciclo de vida del desarrollo y es cada vez más popular su uso en la industria y en la comunidad de ingeniería de software (Van Der Straeten, 2008). Muchas propuestas (Baudry, 2007) (Escalona, 2011) (Enríquez, 2016) (Blanco, 2018), utilizan este enfoque para definir los requisitos del software con modelos estructurados, comprensibles y formales. Esta formalización permite establecer mecanismos para verificar y validar estos requisitos desde etapas tempranas (Gutiérrez, 2008) (Gutiérrez, 2015). Además, la definición de estos modelos formales facilita la definición de las pautas de transformación para generar casos de prueba de forma sistemática y automática, e incluso código fuente (Drave, 2019).

En este contexto, el presente Capítulo presenta el estudio que se realizó sobre el estado del arte de los trabajos de investigación en el contexto del ciclo de vida de desarrollo de los smart contracts en la blockchain (Sánchez-Gómez, 2020b). En concreto, este trabajo se centró en el análisis de estudios primarios que abordasen algunas de las fases del ciclo de vida de desarrollo y/o la ingeniería dirigida por modelos u otras buenas prácticas para analizar, diseñar, implementar o probar los smart contracts.

Para este propósito, se utilizó el método SLR *(Systematic Literature Review),* que permite a los investigadores identificar esas propuestas, detallar posibles brechas y ofrecer futuras propuestas de investigación.

El método SLR fue inicialmente propuesta por Kitchenham y otros autores como Khalid y Petticrew. Este método tuvo sus raíces en revisiones bibliográficas realizadas para Ciencias Humanas y Medicina (Khalid, 2003) (Petticrew, 2005), pero en los últimos años se han propuesto adaptaciones para otras disciplinas como la ingeniería (Kitchenham, 2007).

En nuestro caso, este estudio se llevó a cabo utilizando uno de los métodos más ampliamente aplicados en el campo de la ingeniería del software, el método de Kitchenham (Kitchenham, 2010) y, más en concreto, se ha seguido la última revisión de este método (Kitchenham, 2013) que describe tres fases principales para ejecutar una revisión sistemática. En concreto, de forma resumida estas fases son las siguientes:

- Planificación de la revisión sistemática (planning), que define aspectos como la necesidad de la investigación, protocolo de revisión y preguntas de investigación.
- Ejecución del protocolo de revisión (conducting), donde se lleva a cabo el protocolo establecido.
- Presentación de los resultados obtenidos (reporting), que presenta el análisis final para responder a cada pregunta de investigación.

A continuación, se exponen los pasos dados, así como los resultados obtenidos tras finalizar este estudio.

III.1. Planificación de la revisión sistemática (planning)

Durante esta fase del proceso, aparte de identificar la necesidad de realizar esta revisión de la literatura, se formulan las preguntas de investigación y se valida el protocolo de revisión.

III.1.1. Necesidad de realizar la revisión

En los últimos años, se han publicado muchos estudios para evaluar e identificar desafíos y obstáculos actuales de la tecnología blockchain y también, aunque menos, la aplicación de los smart contracts.

Algunas de estas investigaciones tenían como objetivo evaluar el uso de la blockchain en múltiples sectores como, por ejemplo, la cadena de suministros (Pranto, 2019), (Hidayanto et al, 2019), el sector educativo (Steiu, 2020), el sector agrícola (Yadav, 2019) o el sector sanitario (Yaqoob, 2021). Otras han puesto el foco en la identificación de los retos y barreras de esta tecnología en diferentes escenarios como, por ejemplo, Internet de las cosas (Lo, 2019), Big Data (Karafiloski, 2017) o Administración electrónica (Batubara, 2018).

Otros autores, incluso, han publicado estudios parcialmente relacionados con nuestra propuesta de SLR como, por ejemplo:

- Alharby et al. (Alharby, 2018), que presentaron un mapeo sistemático sobre todas las investigaciones orientadas a la tecnología de los smart contracts. En concreto, seleccionaron 188 artículos relevantes y los clasificaron en seis categorías como seguridad, privacidad, ingeniería del software, aplicación, rendimiento y escalabilidad y otros temas relacionados con smart contracts. Los autores siguieron el método presentado por Petersen (Petersen, 2008) para este estudio y en comparación con una encuesta que realizaron previamente (Alharby, 2017), estos autores observaron que el número de artículos relevantes se había multiplicado por ocho aproximadamente y se ha había desplazado considerablemente hacia las aplicaciones de los smart contracts.

- Macrinici et al. (Macrinici, 2018) realizaron también un mapeo sistemático, pero, en este caso, para identificar la aplicación de smart contracts y ofrecer una perspectiva sobre la problemática actual. En concreto, los autores presentan las tendencias de investigación dentro de este contexto y recopilan 64 artículos. El trabajo de estos autores concluye indicando que, desde 2016, ha habido una tendencia creciente hacia la publicación de artículos relacionados con los smart contracts y que, los problemas y soluciones más discutidos en la literatura, están relacionados con la seguridad, privacidad y

escalabilidad de la blockchain y la "programabilidad" de los smart contracts.

- Leka et al. (Leka, 2019) realizaron una revisión sistemática para identificar dónde se han centrado los últimos estudios y ofrecer una perspectiva sobre el uso de las aplicaciones de la blockchain y los smart contracts. En concreto, el estudio seleccionó cerca de 300 artículos y, posteriormente, realizó un proceso de revisión detallado sobre 28 publicaciones siguiendo los criterios de inclusión y exclusión definidos. Este estudio, desde nuestro punto de vista, no ofrece ninguna explicación detallada sobre el proceso seguido, no sería una SLR al uso, y los resultados obtenidos son difícilmente reproducibles.

- Dhaiouir et al. (Dhaiouir, 2020) presentaron una revisión sistemática sobre los smart contracts, centrándose en plataformas, lenguajes o aplicaciones y criterios de selección. En concreto, este estudio indica que los smart contracts se están adoptando en diferentes tipos de proyectos, pero que éstos aún se enfrentan a muchos desafíos y problemas técnicos. Según los autores esta brecha se debe principalmente a la falta de estándares y proponen como trabajo un marco de referencia muy preliminar.

Por tanto, a diferencia de lo visto en las principales bibliotecas digitales, nuestro trabajo se enfocó a realizar una revisión sistemática de la literatura poniendo el foco en la ingeniería del software y el SDLC y, en particular, sobre el desarrollo de smart contracts. Más en concreto, este estudio puso el foco en el desarrollo de software basado en modelos al realizar una revisión del estado del modelado formal de smart contracts, la generación automática de pruebas y/o código a partir de dicho modelado, con el fin de caracterizar y presentar el estado del arte en este campo e identificar las posibles brechas existentes y las oportunidades para seguir investigando.

III.1.2. Preguntas de investigación

Siguiendo el método de Kitchenham, las preguntas de investigación (RQ, siglas de *Research Questions)* que se establecieron para enfocar claramente la investigación y mejorar la comprensión de la propuesta que se estaba llevando a cabo fueron las siguientes:

La pregunta general de investigación propuesta, que es la que ha guiado toda nuestra revisión es: *¿Cuál es el estado del arte de los smart contracts utilizados en la blockchain dentro de la ingeniería del software?.*

Dado que la ingeniería del software es una disciplina que implica el uso de estructuras, herramientas y técnicas para construir programas informáticos, esta pregunta podría considerarse demasiado general, por lo que se reformuló en varias preguntas más específicas para proporcionar una visión más clara de los aspectos que se deseaban revisar en un contexto de los smart contracts. En concreto, las preguntas de investigación que finalmente se definieron se muestra en la Tabla 6, donde también se recoge la motivación y las respuestas esperadas.

RQ1	Are there approaches in the literature that promote the application of a SDLC? What do phases of the life cycle promote the different studies?
	This RQ aims to find proposals that have been published and to identify their general contexts and the objectives they achieved using SDLC, all in the context of blockchain smart contracts
	Sub-questions
RQ1.1	Is any type of smart contract development life cycle promoted? (single or multiple selection)
	A. The proposal does not describe or apply a life cycle
	B. The development process consists of several phases and there is a common vocabulary for each step
	C. These phases are arranged in order of precedence and are clearly defined I/O from one step to the next
	D. There is a deterministic "definition of done" (*) that can be used to confirm whether a step is truly complete
RQ1.2	Does the proposal promote any of the following phases? (single or multiple selection)
	A. Requirements gathering, analysis and/or design phases
	B. Software coding phase
	C. Software testing phase
	D. The proposal promotes other phases (or does not mention anything about phases)
RQ2	Do they promote model-based software engineering, early starting of the testing phase or automatic source code generation?
	The purpose of this RQ is to identify the techniques and guidelines applied in the different proposals, all in the context of blockchain smart contract
RQ2.1	Does the study apply model-based software engineering? (Boolean:Y/N)
RQ2.2	Does the proposal promote early starting of the testing phase? (known as early testing) (Boolean-Y/N)
RQ2.3	Does the study propose automatic smart contract code generation? (single or multiple selection)
	A. The proposal does not contemplate automatic code generation
	B. The proposal contemplates generation using domain-specific ontologies, glossaries and/or semantic rules
	C. The proposal contemplates generation through the application of model-based software engineering
	D. The proposal contemplates generation through templates and other utilities
RQ3	What scientific or empirical validation methods were used in the different proposals?
	The aim of this RQ is to determine the type of validation methods used in the different studies
RQ3.1	Does the study use a scientific validation method? (Boolean:Y/N)
RQ3.2	Does the study use an empirical validation method? (Boolean:Y/N)

(*) The "definition of done" is usually clear when a product delivered to the client is complete. The definition is that the software engineer, following a methodological approach or previous guidelines, has 100% confidence that no more work is needed before the product can be handed to the client.

Tabla 6. Preguntas de investigación

NOTA: *Estas preguntas están en inglés, ya que solo se tuvieron en cuenta estudios preliminares publicados en inglés y los términos empleados en las preguntas de investigación son los utilizados normalmente en nuestro sector.*

III.1.3. Protocolo de revisión

Después de formular el juego de preguntas de investigación, siguiendo el método de Kitchenham, se especificó el protocolo de revisión al objeto de definir las estrategias de búsqueda, los criterios de inclusión/exclusión para los estudios primarios que se identifiquen, los criterios de calidad, etc.

A continuación, se indica la estrategia y criterios que se siguieron:

Estrategia de búsqueda

Esta estrategia es el procedimiento que se sigue para localizar los estudios primarios más relevantes indexados en diferentes bibliotecas digitales. En nuestro caso, esta estrategia se centró en localizar artículos publicados en revistas de revisión por pares, presentados en congresos relevantes, y se hizo en dos pasos:

1. Se definieron las palabras clave a utilizar en el protocolo de búsqueda. Éstas se fueron concretando después de observar los n resultados obtenidos en búsquedas previas.

   Estas búsquedas preliminares nos permitieron refinar el conjunto de palabras clave y seleccionar las más adecuadas al objeto de mejorar la calidad de los resultados. En la Tabla 7 se muestran estas palabras clave que finalmente fueron utilizadas, donde cada columna es un conjunto acotado de términos relacionados, utilizados para realizar la búsqueda en el título (*Title*), en el resumen (*Abstract*) o en las palabras clave (*Keywords*) de los artículos.

A	B	C
Engineering	Requirement	Blockchain
Model-based	Analysis	Smart Contract
Semantic	Design	
Ontology	Test	
	Debug	
	Check	
	Validation	
	Verification	

Tabla 7. Palabras clave

2. Una vez establecidas las palabras claves, éstas fueron utilizadas de forma sistemática para realizar búsquedas en diferentes bibliotecas digitales.

   Para este propósito, se utilizaron combinaciones de las palabras claves como indica la siguiente ecuación.

$$Equation_1 = [(V_{i=1}^{4} A_i) \wedge (V_{j=1}^{8} B_j)] \wedge (V_{k=1}^{2} C_k)$$

(esta fórmula formaliza la expresión booleana de las palabras clave utilizadas en las búsquedas)

Respecto a las bibliotecas digitales a utilizar, algunos autores establecen criterios para seleccionar las bibliotecas más adecuadas. Por ejemplo, (Ngai, 2011) propone más de diez bibliotecas digitales como Academic Search Premier, ACM Digital Library, Emerald Full Text o IEEE Xplore Digital, entre otras. Sin embargo, durante el proceso de realización de búsquedas preliminares, se observó que muchos estudios fueron devueltos repetidamente por diferentes bibliotecas digitales, por lo que se decidió utilizar solo las siguientes bases de datos: ACM Digital Library, IEEE Xplore Digital Library, ScienceDirect, Elsevier's Scopus y Springer Link. Las referencias obtenidas se gestionaron utilizando la herramienta JabRef (Feyer, 2017) y hojas de cálculo.

Una vez quedaron establecidas las palabras clave y las bibliotecas digitales a emplear, las expresiones de búsqueda se formalizaron utilizando la $Equation_1$ (E_1) indicada anteriormente y se ejecutaron en cada biblioteca digital. En concreto, se seleccionaron metadatos de cada artículo como fuentes de información, formalizándose el uso de estos metadatos según la siguiente ecuación:

$$Equation_2 = keyword(E_1) \wedge abstract(E_1) \wedge Title(E_1)$$

(esta fórmula formaliza la expresión booleana para la búsqueda en los metadatos de un artículo)

Dado que cada biblioteca digital tiene su propia sintaxis para indicar expresiones de búsqueda personalizadas y éstas, además, tienen limitaciones como el número máximo de cláusulas lógicas que se pueden emplear, se tuvieron que construir sentencias específicas para cada biblioteca digital empleada. En la Tabla 8 se muestra cada una de las consultas que se tuvieron que utilizar para este cometido.

ACM Digital Library	
Q1	"query":keywords.author.keyword:(Engin* Model* Semantic Ontology) AND record Abstract:(Requirement Analysis Design Test* Debug* Check* Validation Verification) AND acmdlTitle:(Blockchain "Smart Contract")
Q2	"query":recordAbstract:(Engin* Model* Semantic Ontology) AND keywords.author.keyword:(Requirement Analysis Design Test* Debug* Check* Validation Verification) AND acmdlTitle:(Blockchain "Smart Contract")
IEEE Xplore	
Q3	((((("Author Keywords":engin*) OR "Author Keywords":model*) OR "Author Keywords":semantic) OR "Author Keywords":ontology) AND (((((((("Abstract":requirement) OR "Abstract":analysis) OR "Abstract":design) OR "Abstract":test*) OR "Abstract":debug*) OR "Abstract":check*) OR "Abstract":validation) OR "Abstract":verification) AND (("Document Title":blockchain) OR "DocumentTitle":smart contract)
Q4	(((((((("Author Keywords":requirement) OR "Author Keywords":analysis) OR "Author Keywords":design) OR "Author Keywords":test*) OR "Author Keywords":debug*) OR "Author Keywords":check*) OR "Author Keywords":validation) OR "Author Keywords":verification) AND ((((("Abstract":engin*) OR "Abstract":model*) OR "Abstract":semantic) OR "Abstract":ontology) AND (("Document Title":blockchain) OR "Document Title":smart contract)
Science Direct	
Q5	Find articles with these terms: ("Smart Contract") Title, abstract, keywords: ("Engin*" OR "Model*" OR "Semantic" OR "Ontology" OR "Requirement" OR "Analysis" OR "Design" OR "Test*" OR "Debug*" OR "Check*" OR "Validation" OR "Verification" OR "Blockchain" OR "Smart Contract")
Elsevier's Scopus	
Q6	(ABS("Requirement" OR "Analysis" OR "Design" OR "Test*" OR "Debug*" OR "Check*" OR "Validation" OR "Verification") AND KEY("Engin*" OR "Model*" OR "Semantic" OR "Ontology") AND TITLE("Blockchain" OR "Smart Contract"))
Q7	(KEY("Requirement" OR "Analysis" OR "Design" OR "Test*" OR "Debug*" OR "Check*" OR "Validation" OR "Verification") AND ABS("Engin*" OR "Model*" OR "Semantic" OR "Ontology") AND TITLE("Blockchain" OR "Smart Contract"))
Springer Link	
Q8	("Engin*" OR "Model*" OR "Semantic" OR "Ontology") AND ("Requirement" OR "Analysis" OR "Design" OR "Test*" OR "Debug*" OR "Check*" OR "Validation" OR "Verification") AND "Blockchain" AND "Smart Contract"

Tabla 8. Consultas de búsqueda por biblioteca digital

También fue necesario modificar algunas consultas de búsqueda utilizando determinados filtros, sobre todo, debido al considerable número de resultados obtenidos tras las búsquedas. Estos filtros, normalmente, son proporcionados por cada biblioteca digital. En concreto, se realizaron consultas por áreas científicas, por temas específicos y con año de publicación mayor o igual a 2016, ya que a partir de este año se comenzaron a obtener artículos relevantes, según los criterios de búsqueda predefinidos.

Por último, es importante indicar que el protocolo de búsqueda definido fue ampliado utilizando la técnica *Snowball* (Wohlin, 2013), "bola de nieve" en español. Esta técnica, cuyo objetivo es identificar otros posibles estudios relevantes, nos permitió ampliar el proceso de búsqueda para abarcar tanto las listas de referencias como las citas en cada artículo en estudio.

Proceso de selección

En este momento es cuando se ejecuta, de forma iterativa, el proceso de selección para obtener los estudios relevantes. Trabajos que, posteriormente, deben ser analizados teniendo en cuenta los objetivos de esta SLR.

A continuación, en la Tabla 9, se reflejan las fases seguidas en el proceso de selección de estudios primarios para su análisis.

Fase	Descripción	Investigadores participantes
Ph1	Lectura de Título, Abstract y Keywords de Estudios Primarios para identificar el objetivo de la investigación	Investigador principal
Ph2	Cribado: exclusión de estudios primarios que tratan otros temas	Investigador principal
Ph3	1ª reunión de consenso	Todos
Ph4	Análisis de las exclusiones tras leer al texto completo de los estudios primarios e inclusión de estudios tras aplicar la técnica de "bola de nieve".	Todos
Ph5	2ª reunión de consenso	Todos

Tabla 9. Fases seguidas en el proceso de selección de estudios primarios

Como se puede apreciar, el proceso de selección que se siguió incluye reuniones presenciales. En concreto, la primera reunión (Ph3) tuvo como objetivo resolver cualquier duda cuando se aplican los criterios de inclusión / exclusión. En estos casos, se hace necesario realizar una revisión completa de los estudios, llamémosle, dudosos. Después de esta lectura, todos los investigadores deciden, de forma conjunta, incluir o excluir finalmente el estudio primario identificado. La segunda reunión presencial (Ph5) se realizó tras aplicar la técnica ''bola de nieve'' indicada anteriormente. Su objetivo es también resolver cualquier duda cuando se evalúen los estudios obtenidos por esta técnica o cuando se apliquen criterios de exclusión/inclusión sobre los mismos.

Criterios de exclusión/inclusión

Los criterios de exclusión/inclusión establecidos fueron acordados de forma objetiva y fueron agrupados en cada fase del proceso de selección seguido, tal y como muestra la

Tabla *10*.

Fase	Criterio de inclusión o exclusión
Ph1	Se realizó una búsqueda automática en cada biblioteca digital. En esta fase se tiene en cuenta los estudios primarios devueltos por las consultas de búsqueda.
Ph2	Sólo estudios primarios en inglés, se obtiene el texto completo y, con año de publicación mayor o igual a 2016 (ya que tras analizar numerosos trabajos de años anteriores, sólo a partir de este año empezamos a identificar artículos que cumplían los criterios de búsqueda definidos). Se excluyen trabajos no relacionados con el tema de investigación. Esta fase de exclusión incluyó la eliminación de trabajos duplicados tras la lectura del título (Title) y el resumen (Abstract) del trabajo. En caso de duda sobre algún documento, éste se incluiría de forma preliminar. La decisión final se consideraría y evaluaría en la siguiente fase.
Ph3	No hay nuevos criterios de exclusión al ser la 1ª reunión, pero es posible incluir trabajos relevantes que se hayan identificado. En esta fase, también se analizaron detalladamente todos los artículos "dudosos", teniendo en cuenta todo su contenido.
Ph4	En esta fase se aplicó la técnica de "bola de nieve", por lo que fue necesario volver a aplicar los criterios de la fase 2 (Ph2).
Ph5	En esta 2ª reunión no se consideran nuevos criterios de exclusión/inclusión, pero todos los investigadores analizaron detalladamente todos los estudios "dudosos", teniendo en cuenta todo su contenido.

Tabla 10. Criterio de exclusión/inclusión por fase.

Como se puede apreciar en la tabla anterior, se tuvieron en cuenta solo artículos escritos en inglés y publicados en revistas de buena reputación, es decir, revistas indexadas en Journal Citation Reports; JCR) o conferencias prestigiosas (es decir, nivel de conferencia A*, A, B y C categorizado en CORE Conference Rank e incluidas en el ranking GII-GRIN-SCIE). Además, se decidió excluir encuestas, discusiones, revisiones o estudios de opinión relacionados con la temática buscada.

Criterios de Calidad

Se definieron estos criterios de calidad para obtener los mejores resultados para futuras investigaciones. En concreto, la Tabla 11 muestra los criterios de calidad (QC*, siglas de Quality Criteria)* aplicados en esta SLR, así como el peso que tendría cada criterio.

Id.	Preguntas (y puntuación)	Peso
QC1	En cuanto al SDLC, ¿el estudio presenta de forma clara los objetivos previstos, el alcance de la solución aportada y sus conclusiones?. Posibles respuestas (puntuación): - Sí (+1) - No (+0)	10%
QC2	¿El estudio detalla o contextualiza alguna fase del SDLC? Posibles respuestas (puntuación): - Todas las fases (+2) / Algunas fases (+1) - Ninguna (+0)	10%
QC3	¿El estudio aplica la ingeniería dirigida por modelos? ¿Promueve un inicio temprano de la fase de pruebas? Posibles respuestas (puntuación): - Ambas preguntas (+2) / Una de las preguntas (+1) - Ninguna (+0)	40%
QC4	¿El estudio hace referencia a la generación automática de código utilizando ontologías específicas de dominio, plantillas y/o aplicando la ingeniería dirigida por modelos?	30%

	Posibles respuestas (puntuación): - Todas las indicadas (+2) / Al menos una (+1) - Ninguna (+0)	
QC5	¿El Estudio está validado científicamente? Posibles respuestas (puntuación): - Estudio de caso o experimento (+1) - No validado (+0)	10%

Tabla 11. Tabla con criterios de calidad

La puntuación acumulada, según peso, para cada criterio constituiría la calificación de calidad (CC) del estudio primario, siendo la calificación máxima un 10. Por ejemplo, si un estudio tiene como puntuación en QC1 un 1 y en QC4 un 1, su calificación sería 2,5. Este valor sale en función del peso indicado, es decir, si la QC1 solo tiene dos posibles respuestas la calificación solo podrá ser 0 o 1 y, si la QC4 tiene 3 posibles respuestas, la calificación de calidad solo podrá un 0, 1,5 o 3.

Es importante tener en cuenta que estos QC no se utilizan para excluir artículos, sino que se utilizan para determinar qué artículo es más relevante y representativo para futuras investigaciones.

Esquema de datos y validación del protocolo de revisión

Dado que el análisis de cada estudio primario podría convertirse en una tarea ardua, debido a la información heterogénea existente y las diferentes estructuras de cada estudio. Se propuso un esquema de caracterización (ver Tabla 12) para homogeneizar este análisis y reducir así la complejidad de esta tarea.

Característica	Descripción
Tipo de publicación	Revista (*journal*) o conferencia en la que se publica el estudio primario
Área de negocio	Área donde se aplicó el enfoque del estudio primario
Descripción y motivación	Esta característica representa una breve descripción de cada estudio primario y su motivación
Fase del ciclo de vida	Se refiere a la fase del SDLC en la que se centra la propuesta del estudio primario

Tabla 12. Esquema de caracterización

Y, por último, siguiendo las recomendaciones dadas en el método de Kitchenham, el protocolo SLR fue revisado para obtener un proceso de revisión integral.

En concreto, se aplicaron algunas búsquedas aleatorias para refinar las palabras clave definitivas y los criterios de exclusión/inclusión, entre otros aspectos. Sin embargo, finalmente decidimos buscar el asesoramiento de expertos en la realización de SLR para definir un proceso sistemático, completo e integral. En este sentido, participó como externo un Catedrático de Ingeniería del Software de la Universidad de Sevilla (España) experto para validar nuestro protocolo de revisión.

III.2. Ejecución del protocolo de revisión (conducting)

El objetivo de esta fase del proceso es presentar los estudios primarios obtenidos después de ejecutar el protocolo de revisión definido. También se detalla la puntuación de calidad de cada trabajo de investigación analizado, teniendo en cuenta los criterios de calidad indicados anteriormente.

III.2.1. Identificación, selección y análisis de estudios primarios

La Tabla 13 muestra la distribución de estudios primarios obtenidos tras aplicar los criterios de inclusión/exclusión establecidos en cada fase del protocolo de selección.

Fuente	Ph1	Ph2	Ph3	Ph4	Ph5
ACM Digital Library	27	6	2	-	-
IEEE Xplore	39	7	3	-	-
ScienceDirect	372	31	7	-	-
Elsevier's Scopus	352	42	6	-	-
SpringerLink	243	24	4	-	-
Snowball technique	-	-	-	10	3
Subtotales	1,033	110	**22**	10	**3**
TOTAL	**25**				

Tabla 13. Distribución de estudios primarios

Como se puede observar en la Tabla anterior, para cada biblioteca digital, se muestra el número de artículos obtenidos en cada fase del protocolo de revisión, por ejemplo, en la fase 3 (Ph3) quedaron 22 estudios a tener en cuenta. Además, esta tabla muestra los estudios obtenidos después de aplicar la técnica de "bola de nieve" (3 estudios). Para agilizar el manejo de los resultados, estos últimos artículos no fueron clasificados por biblioteca digital.

Los estudios primarios obtenidos durante el protocolo de búsqueda, en comparación con los estudios finalmente seleccionados quedaron recogidos de forma ilustrativa en la Figura 20.

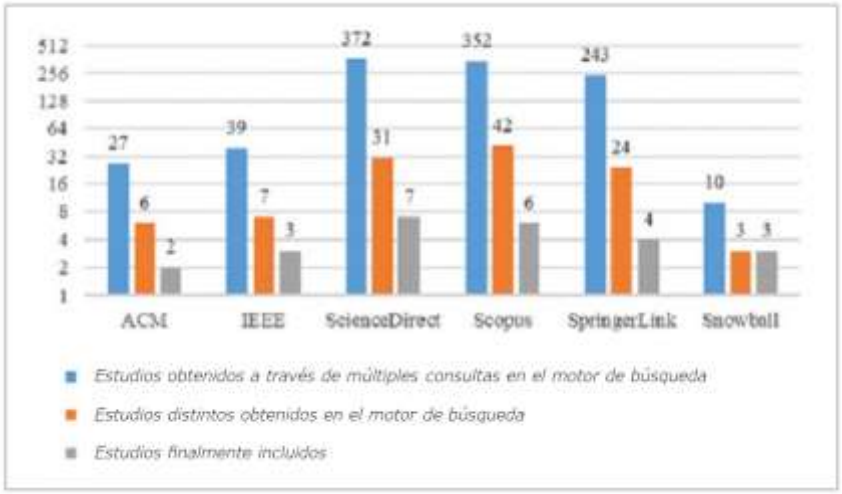


Figura 20. Estudios obtenidos de las bibliotecas digitales

Por otro lado, el porcentaje (%) de estudios primarios obtenidos de cada biblioteca digital se muestran en la Figura 21. Este gráfico visualiza el porcentaje de estudios primarios que finalmente fueron incluidos en el análisis en comparación con el porcentaje de artículos preseleccionados inicialmente por biblioteca digital tras las búsquedas realizadas y la aplicación de los criterios de exclusión/inclusión.

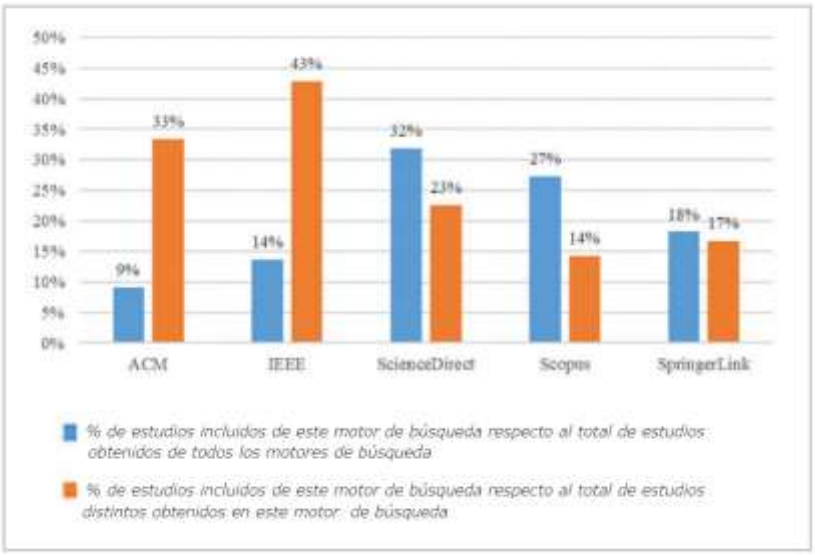


Figura 21. Análisis de los resultados obtenidos de cada biblioteca digital.

Por ejemplo, para la biblioteca ACM, la barra de la izquierda muestra un 9%, número que se obtiene a partir de la división de los 2 estudios primarios incluidos entre los 22 estudios primarios identificados inicialmente. Y el porcentaje de la barra de la derecha, se obtiene a partir de la división de los 2 estudios primarios incluidos entre los 6 estudios primarios seleccionados durante la fase 2, lo que supone un 33%.

III.2.2. Riesgos identificados

El protocolo de revisión y validación presentado en las secciones anteriores puede implicar debilidades o amenazas en el protocolo seguido, ya que las tareas han sido realizadas por personas. La selección de los estudios de investigación, debido a este factor humano, podría verse afectada por errores durante el proceso de clasificación y selección de los mismos. Estos riesgos se intentaron mitigar con la ejecución de varias iteraciones en el proceso de revisión y con reuniones entre los investigadores cuando había dudas sobre la categorización de cualquier estudio primario.

No obstante, aunque el proceso de revisión se ha definido y ejecutado exhaustivamente, no es posible garantizar una cobertura completa de la literatura científica sobre un tema. Por ejemplo, los artículos no indexados o la literatura gris no se consideran en esta SLR.

Schmucker (Schmucker,2017) afirma que ese tipo de publicaciones rara vez son relevantes en los resultados de una SLR. Es por eso por lo que, en nuestro estudio, los términos de búsqueda se utilizaron en cinco bibliotecas digitales que cubren una amplia gama de temas, lo suficiente como para ser razonablemente considerados exhaustivos para el campo de investigación en el que se centró este estudio. Además, es importante tener en cuenta que todos los autores del estudio han participado en la definición del protocolo de búsqueda, en las preguntas de investigación y en los términos de búsqueda.

Esta decisión nos ha permitido aumentar la objetividad del proceso de revisión.

III.3. Presentación de los resultados obtenidos (reporting)

Esta fase tiene como objetivo dar respuesta, de forma argumentada, a cada una de las preguntas de investigación planteadas, al objeto de identificar posibles debilidades de acuerdo con el objetivo de esta SLR.

Pero, antes de dar respuestas a las preguntas planteadas, se indican todos los estudios primarios identificados y analizados (ver Tabla 14). En concreto, esta Tabla muestra los estudios que finalmente se incluyeron en esta SLR, siguiendo todos los criterios previamente establecidos, junto con la calificación de calidad obtenida tras aplicar las puntuaciones y pesos recogidos en la Tabla *11*, que solo determina qué artículo es más relevante y representativo para futuras líneas de investigaciones.

Estudio	Autores	Titulo	Año	CC
PS01	Marchesi, M. et al.	An Agile Software Engineering Method to Design Blockchain Applications [43]	2018	5
PS02	Liu, Y. et al.	Applying Design Patterns in Smart Contracts [38]	2018	1.5
PS03	Choudhury, O. et al.	Auto-Generation of Smart Contracts from Domain-Specific Ontologies and Semantic Rules [9]	2018	6
PS04	Tateishi, T. et al.	Automatic smart contract generation using controlled natural language and template [67]	2019	6
PS05	Tsai, W. et al.	Beagle: A New Framework for Smart Contracts Taking Account of Law [69]	2019	4
PS06	Koul, R.	Blockchain Oriented Software Testing - Challenges and Approaches [33]	2018	2.5
PS07	Dolgui, A. et al.	Blockchain-oriented dynamic modelling of smart contract design and execution in the supply chain [18]	2019	1.5
PS08	Porra, S. et al.	Blockchain-Oriented Software Engineering: Challenges and New Directions [54]	2017	3.5
PS09	Shishkin, E.	Debugging Smart Contract's Business Logic Using Symbolic Model-Checking [62]	2018	1.5
PS10	Mavridou, A. et al.	Designing Secure Ethereum Smart Contracts: A Finite State Machine Based Approach [44]	2018	5
PS11	Parizi, R. M. et al.	Empirical vulnerability analysis of automated smart contracts security testing on blockchains [50]	2018	1.5
PS12	Lee, S. et al.	Formal Specification Technique in Smart Contract Verification [34]	2019	0.5
PS13	Mavridou, A. et al.	FSolidM for Designing Secure Ethereum Smart Contracts: Tool Demonstration [45]	2018	6
PS14	Sillaber, C. et al.	Life Cycle of Smart Contracts in Blockchain Ecosystems [63]	2017	0
PS15	Grigg, I.	On the intersection of Ricardian and Smart Contracts [25]	2015	0.5
PS16	Kruijff, J. et al.	Ontologies for Commitment-Based Smart Contracts [16]	2017	2
PS17	Clack, C. D.	Smart Contract Templates: Legal semantics and code validation [10]	2018	1.5
PS18	Clack, C. D. et al.	Smart Contract Templates: Foundations, design landscape and research directions [12]	2016	2
PS19	Syahputra, H. et al.	The Development of Smart Contracts for Heterogeneous Blockchains [65]	2019	6
PS20	Liao C. et al.	Toward A Service Platform for Developing Smart Contracts on Blockchain in BDD and TDD Styles [37]	2017	2.5
PS21	Al Khalil, F. et al.	Trust in Smart Contracts is a Process, As Well [1]	2017	1.5
PS22	Mavridou, A. et al.	VeriSolid: Correct-by-Design Smart Contracts for Ethereum [46]	2019	6
PS23	Permenev, A. et al.	VerX: Safety Verification of Smart Contracts [51]	2019	2.5
PS24	Mao, D. et al.	Visual and User-Defined Smart Contract Designing System Based on Automatic Coding [42]	2019	1.5
PS25	Clack, C. D. et al.	Smart Contract Templates: essential requirements and design options [11]	2016	2

Tabla 14. Estudios primarios finalmente incluidos y calificación de calidad (CC) obtenida

Es importante insistir en que el número de estudios primarios seleccionados no se ha visto afectado después de aplicar los criterios de calidad, ya que este dato solo nos ha permitido puntuar el estudio como más o menos relevante y representativo, teniendo en cuenta los pesos, para su consideración en futuras investigaciones. En este sentido, la última columna de la Tabla 14 muestra la calificación (CC) para los 25 estudios primarios finalmente incluidos. Por ejemplo, el articulo PS15 (Grigg, 2004), propone utilizar los contratos ricardianos (concepto que combina elementos del derecho tradicional con la tecnología blockchain), por lo que obtuvo un CC de 0,5, es decir, tenía un 1 como puntuación en QC2, ya que este trabajo puede tener cierta relevancia en futuras investigaciones, pero no trata las cuestiones planteadas en esta SLR.

Teniendo en cuenta las cuestiones planteadas en la SLR, el enfoque, así como el nivel de detalle aportado por los autores de la propuesta, en esta lista de estudios primarios destacan varios trabajos, desde nuestro punto de vista:

- Marchesi et al. (PS01), ya que proponen un proceso de desarrollo de software que permita recopilar los requisitos, analizar, diseñar, desarrollar, probar e implementar aplicaciones blockchain. El proceso se basa en prácticas *Agile,* utilizando historias de usuario y un desarrollo iterativo e incremental basado en ellas.

- Choudhury et al. (PS03), pues proporcionan un marco de trabajo para la generación automática de smart contracts. Este marco utiliza ontologías y reglas semánticas para codificar el conocimiento específico del dominio y luego aprovecha la estructura de los árboles de sintaxis abstracta para incorporar las restricciones requeridas.

- Tateishi et al. (PS04), ya que proponen una técnica para generar automáticamente un smart contract a partir de un documento de contrato comprensible para el ser humano. En concreto, éste se crea utilizando una plantilla y un lenguaje natural controlado. La automatización se basa en una asignación de la plantilla y ese lenguaje natural a un modelo formal que puede definir los términos y condiciones de un contrato, incluidas las restricciones y procedimientos temporales.

- Mavridou et al. (PS13) argumentan que, en la práctica, los smart contracts están plagados de vulnerabilidades. Para facilitar el desarrollo de smart contracts seguros, estos investigadores han creado un marco que permite definir contratos como máquinas de estado finito (FSM, siglas de *Finite State Machines*) con una semántica rigurosa y clara, según indican.

- Syahputra et al. (PS19), pues aborda una discusión sobre cómo debe ser el proceso de desarrollo de una plataforma de smart contract que tenga como objetivo generar smart contracts para tecnologías blockchain heterogéneas.

- Mavrodou et al. (PS22), ya que presentan un marco de trabajo para la verificación formal de smart contracts. Éstos se especifican utilizando un modelo basado en un sistema de transición con semántica operativa y permite la generación de código Solidity a partir de los modelos verificados, lo que permitiría el desarrollo a partir del diseño de los smart contracts.

A continuación, se exponen las preguntas planteadas y se argumentan, desde nuestro punto de vista, los aspectos destacables identificados en los artículos seleccionados.

RQ1. ¿Existen propuestas en la literatura que promuevan la aplicación de un ciclo de vida de desarrollo para los smart contracts? y, sobre todo, ¿qué fases del ciclo de vida tratan o promueven?.

Las fases más comunes del ciclo de vida de desarrollo de software, como se ha comentado, son la especificación de los requisitos, el análisis, el diseño, la codificación, las pruebas y el mantenimiento. En este sentido, la Tabla 15 muestra la distribución de estudios primarios (PS, siglas de Primary Studies) seleccionados y las fases que contemplan los autores en sus estudios.

PS	A1	A2	A3	A4
[PS01]	X	X	X	X
[PS02]	X			
[PS03]	X	X		
[PS04]	X	X		
[PS05]	X	X	X	X
[PS06]			X	
[PS07]				X
[PS08]	X	X	X	
[PS09]			X	
[PS10]	X	X		
[PS11]			X	
[PS12]	X			
[PS13]	X	X		
[PS14]				
[PS15]	X			
[PS16]				
[PS17]				
[PS18]	X			
[PS19]	X	X		
[PS20]	X		X	
[PS21]				
[PS22]	X	X		
[PS23]	X	X		
[PS24]	X			
[PS25]	X			
TOTAL	**17**	**10**	**7**	**3**

A1: Requirements gathering, analysis, and/or design phases
A2: Coding phase
A3: Testing phase
A4: Other phases

Tabla 15. Distribución de estudios primarios en relación con las fases del ciclo de vida de desarrollo de los smart contracts.

Como muestra la Tabla 15, las fases más abordadas por los autores fueron: (A1) Requisitos, análisis o diseño (68%), (A2) Codificación (40%), (A3) Pruebas (28%) y (A4) Otras fases (12%).

Analizando los resultados obtenidos, tenemos que un 12% de los estudios primarios no están referenciados a ninguna fase concreta (son muy generalista), pero hay otros trabajos como el de Marchesi et at (PS01) y el de Tsai et al. (PS05) que destacan sobre el resto. En concreto, el estudio PS01 propone un proceso de desarrollo de software para especificar los requisitos, analizar, diseñar, desarrollar, probar e implementar aplicaciones blockchain y el estudio PS05 propone un marco de trabajo con cinco etapas: desarrollo de plantillas de smart contracts, a partir del análisis de dominios, modelo formal de smart contracts, desarrollo de código a partir de plantillas, verificación y validación.

Parece, por tanto, que algunos esfuerzos de la comunidad científica van encaminados actualmente a implementar algún tipo de ciclo de vida de desarrollo. No obstante, en el contexto de la blockchain, los procesos analizados consisten solo en un número determinado de fases no vinculadas, al no estar dispuestas en un orden claro de precedencia y las entradas/salidas de cada etapa tampoco están claramente definidas.

Es importante destacar, por el peso que debe tener en la tecnología blockchain, el hecho de que la fase con menor impacto en la literatura identificada es la de pruebas de software. Como ya se ha comentado, desde nuestro punto de vista, las aplicaciones en un contexto blockchain difieren bastante de otras aplicaciones tradicionales, ya que una vez que se implementa un smart contract, su ejecución no se puede revertir. Por ello, es fundamental realizar pruebas robustas, poniendo énfasis en la elicitación de requisitos, en su verificación y validación y en la depuración de código tras la ejecución de las pruebas. Es más, las pruebas deberían implicar la simulación de todas las posibles variables esperadas e inesperadas para cada smart contract y para los desencadenantes que ejecutan las transacciones.

RQ2. ¿Promueven la ingeniería de software basada en modelos, el inicio temprano de la fase de prueba o la generación automática de smart contracts?.

Desde hace unos años, como ya se ha comentado, el uso de herramientas de modelado o herramientas CASE, así como el uso del estándar UML han ayudado a documentar la funcionalidad de los procesos empresariales y a utilizar transformaciones entre modelos para automatizar, incluso, la generación de código. Por ejemplo, entre los estudios primarios seleccionados, Marchesi et al. (PS01) y Syahputra et al. (PS19) destacan este aspecto al proponer la utilización de diagramas UML para describir los requisitos de las aplicaciones.

En este contexto, la aparición del lenguaje de modelado para la especificación de casos de prueba (U2TP) que cerró la brecha entre diseñadores y probadores al proporcionar una buena razón para usar UML, ha permitido que los diagramas UML de requisitos y análisis se puedan reutilizar para las pruebas (Gutiérrez, 2008) (Gutiérrez, 2015) y, lo que es más importante, permiten que se puedan comenzar las pruebas en una fase temprana del desarrollo del sistema. Además, como ya se ha indicado anteriormente, realizar una ingeniería del software basada en modelos es importante, ya que aporta las siguientes ventajas (Ramos, 2011) (Pohl, 2012):

- Se pueden implementar las mejores prácticas y generar códigos bien probados, lo que reduce la aparición de código vulnerable. Las herramientas basadas en modelos pueden mejorar la verificación / validación del código mediante la aplicación de técnicas de prueba desde las primeras etapas del ciclo de vida de desarrollo.

- El código de software es más difícil de entender que los modelos. Por lo tanto, es más fácil comprobar la exactitud de un modelo. El código del software también se puede generar automáticamente a partir de herramientas basadas en modelos, lo que garantiza que el código generado no se haya modificado después de obtenerlo.

- La ingeniería basada en modelos se puede aplicar en múltiples plataformas.

En este sentido, la Tabla 16 muestra la distribución de los estudios primarios (PS, siglas de Primary Studies) en relación con las propuestas de los diferentes autores.

PS	B1	B2	B3
[PS01]	X		
[PS02]			
[PS03]	X		X
[PS04]	X		X
[PS05]	X		
[PS06]		X	
[PS07]			
[PS08]	X		
[PS09]			
[PS10]	X		X
[PS11]			
[PS12]			
[PS13]	X		X
[PS14]			
[PS15]			
[PS16]	X		
[PS17]			X
[PS18]			X
[PS19]	X		X
[PS20]			
[PS21]			X
[PS22]	X		X
[PS23]			
[PS24]			
[PS25]			X
TOTAL	**10**	**1**	**10**

B1. Application of model-based software engineering
B2: Promotion of early testing
B3: Proposal of automatic smart contract generation

Tabla 16. Distribución de los estudios primarios según el alcance de la propuesta

Las propuestas abordadas por los autores, como se muestra en la Tabla *16*, están relacionadas con: (B1) Aplicación de la ingeniería del software basada en modelos (48%), (B2) Promover el testing temprano (5%), y (B3) Propuesta de generación automática de código (48%).

Viendo los resultados obtenidos, aunque como se ha indicado el testing temprano ayuda a reducir el número de errores y asegurar la calidad del software, parece que los esfuerzos de la comunidad científica no están dirigidos hacía este enfoque.

La realidad es que muchos proyectos siguen realizando las pruebas al final del ciclo de vida de desarrollo, es decir, después de la fase de codificación. Esto implica que muchas veces no tienen suficiente tiempo para detallar, ni hacer pruebas rigurosas y de forma adecuada, dando lugar a posibles reajustes que podrían afectar a la calidad del software (Jain, 2016).

Las pruebas tempranas, desde nuestro punto de vista, proporcionaría tiempo suficiente para identificar la ausencia o inadecuación de cualquier requisito funcional. Además, los casos de prueba obtenidos a partir de las especificaciones de estos requisitos funcionales podrían ayudar a identificar o revelar posibles fallos (Jain, 2016).

Está claro que para realizar las pruebas funcionales del software, éste debe estar construido, luego evidentemente que las pruebas funcionales, como tal, hay que realizarlas después de su codificación. Otro asunto diferente, desde nuestro punto de vista, es que desde las etapas iniciales del ciclo de vida del desarrollo, estas pruebas deben ser tenidas en cuenta, e incluso es deseable obtener los casos de prueba de forma más o menos automática al objeto de tener una amplia cobertura funcional. El testing temprano o el early testing va en esta línea.

Por suerte, algunos autores como Koul et al. (PS06) destacan la necesidad de asegurar la calidad del software desde etapas tempranas, indicando los desafíos a los que se enfrentan actualmente las pruebas de este tipo de aplicaciones. Estos autores, además, reconocen la necesidad de diseñar herramientas y técnicas específicas para las pruebas de este tipo software, al objeto de garantizar altos estándares de calidad.

Está claro que el papel de las pruebas no es solo verificar la "corrección" del software, sino más bien descubrir defectos a tiempo y poder rectificarlo lo antes posible. Realizar pruebas exhaustivas, que cubren el 100% de la funcionalidad de un software no es factible en ningún caso, pero es importante eliminar el mayor número de errores o defectos, lo antes posible. Luego, si se aplicaran las técnicas de testing temprano, los programas informáticos y, por ende, los smart contracts, podrían implementarse con mayor fiabilidad y menores costes de desarrollo.

Respecto a la generación automática de smart contracts, un aspecto importante a considerar es la técnica que los autores proponen emplear. En la Tabla 17 se muestra la distribución de estudios primarios (PS, siglas de Primary Studies) con respecto a las diferentes técnicas propuestas.

PS	C1	C2	C3
[PS01]			
[PS02]			
[PS03]	X		X
[PS04]			X
[PS05]			
[PS06]			
[PS07]			
[PS08]			
[PS09]			
[PS10]			X
[PS11]			
[PS12]			
[PS13]			X
[PS14]			
[PS15]			
[PS16]			
[PS17]			
[PS18]			X
[PS19]		X	
[PS20]			
[PS21]			
[PS22]			X
[PS23]			
[PS24]			
[PS25]			X
TOTAL	**1**	**1**	**7**

Automatic code generation:
C1. Using domain-specific ontologies and/or semantic rules
C2. Using model-based software engineering
C3. Through templates and other utilities

Tabla 17. Distribución de los estudios primarios según las técnicas de generación de código (smart contract) propuestas

La generación automática del código del smart contract utilizando un proceso de ingeniería del software basado en modelos, eliminaría el esfuerzo manual requerido en la codificación desde el diseño y, por lo tanto, aceleraría el proceso, al tiempo que disminuye la posibilidad de errores en comparación con la codificación manual de los requisitos o modelos. No obstante, como muestra la tabla anterior, las técnicas más usadas o propuestas por los autores son: (C1) Generación utilizando ontologías y/o reglas semánticas específicas del dominio (4%), (C2) Generación utilizando ingeniería basada en modelos (4%) y (C3) Generación a través de plantillas u otras utilidades (28%).

La ingeniería de software basada en modelos ha ido ganando terreno en el desarrollo de software, especialmente en dominios críticos (Bialy, 2017). Es más, la generación automática de código a partir de dicha ingeniería es un paso importante, ya que ofrece la posibilidad de implementar, verificar y validar el software a partir de los requisitos.

En este sentido, resulta interesante el estudio de Syahputra et al. (PS19) que proponen el uso de una plataforma de smart contracts que permita generar smart contracts para tecnologías blockchain heterogéneas, haciendo uso de UML y OCL *(Object Constraint Language)*.

Está claro que una buena práctica es establecer un ciclo de vida de desarrollo que aproveche las tecnologías de generación de código, pero éste debe adherirse bien a los principios de la ingeniería del software, buscando la reducción de la complejidad y la trazabilidad de los requisitos.

RQ3. ¿Qué métodos de validación científica o empírica se utilizaron en las diferentes propuestas?

Después de analizar los estudios primarios seleccionados, solo el 12% de los artículos (3 de 25 estudios) utilizaron una validación científica, tal y como se muestra en la Tabla 18. El mayor porcentaje, un 44% (11 de 25 estudios) de los estudios primarios, realizaron su evaluación mediante estudios de casos experimentales o pruebas de concepto y con éste mismo porcentaje, hay estudios que ni siquiera cuentan con un plan de validación, lo que dificulta enormemente la verificación de sus afirmaciones.

Método de validación científica	Estudio Primario	Total	%
Experiment	[PS07][PS09][PS10]	3	12%
Case studies / Proof-of-concept	[PS01][PS02][PS03][PS04] [PS11][PS13][PS19][PS20] [PS22][PS23][PS24]	11	44%
No evaluation	[PS05][PS06][PS08] [PS12][PS14][PS15][PS16] [PS17][PS18][PS21][PS25]	11	44%
	TOTAL	**25**	

Tabla 18. Distribución de estudios primarios según el modelo de validación científica utilizado

III.4. Conclusiones tras finalizar este trabajo

Los resultados obtenidos en la SLR realizada (Sánchez-Gómez, 2020b) nos ha permitido identificar y, sobre todo, analizar el estado del arte de las publicaciones científicas relacionadas con la temática de la presente Tesis.

En dicha SLR se siguió el método más reciente de Kitchenham (Kitchenham, 2013) y, tras realizar el protocolo planteado y los criterios establecidos, se identificó un total de 25 estudios primarios que tenían en cuenta algún aspecto del ciclo de vida de desarrollo de los smart contracts.

Desde nuestro punto de vista, la elicitación de requisitos y las pruebas de software se encuentran entre los aspectos más importantes del ciclo de vida de desarrollo. Pero estos estudios primarios, casi siempre, han pasado de puntillas por estas fases. No obstante, destaca Marchesi et al. (PS01), que proponen un proceso de desarrollo de software teniendo en cuenta las típicas fases del ciclo de vida del mismo, pero se focalizan en la aplicación de metodologías Agiles. En su estudio proponen el uso de diagramas UML para describir el diseño de las aplicaciones e, incluso, proporcionan un modelado de las interacciones entre el software tradicional y el entorno blockchain. Otros autores como Syahputra et al. (PS05) discuten sobre el proceso de desarrollo a partir de una plataforma de smart contract. Esta plataforma tiene

como objetivo crear un smart contract para tecnologías blockchain heterogéneas y proponen el uso de UML, además de OCL, para el diseño.

Todos los estudios primarios, de alguna manera u otra, indican la necesidad de obtener un software que funciona correctamente. Pero, en el caso de los smart contracts, creemos que se debe hacer mayor hincapié en las pruebas funcionales, de seguridad y de rendimiento.

Como se ha repetido anteriormente, dada la característica de inmutabilidad de la red blockchain, es esencial que antes de desplegar estos *scripts* en un entorno empresarial, los smart contracts pasen rigurosos procesos de verificación y validación, ya que un error o defecto en él podría causar un efecto no reparable. Por tanto, las pruebas son esenciales para validar y corregir el comportamiento de un smart contract antes de su publicación.

Si nos centramos en las pruebas funcionales, objeto de la presente Tesis, probar de manera eficiente un smart contract antes de implementarlo, asegurará que este funcione como se espera, es decir, según las especificaciones funcionales establecidas. En las pruebas funcionales, todas las reglas o requisitos empresariales, incluidos los argumentos válidos y no válidos, los valores de límite y las combinaciones de argumentos, deben verificarse con casos de prueba. En este sentido, algunos autores como Koul et al. (PS06) destacan la necesidad de asegurar la calidad del software desde etapas tempranas. Por tanto, este estudio coincide parcialmente con nuestro planteamiento de obtener casos de prueba en fases tempranas del ciclo de vida de desarrollo de smart contracts. Además, estos autores reconocen la necesidad de diseñar herramientas y técnicas específicas para las pruebas de este tipo de software, al objeto de garantizar altos estándares de calidad.

Después de dar por finalizada la SLR, se vio de forma clara la necesidad de explorar una nueva línea de investigación, potencialmente muy interesante, como es la aplicación del paradigma de la ingeniería dirigida por modelos ya que, como se ha expuesto en Capítulos anteriores, parece que este paradigma puede facilitar los mecanismos necesarios para verificar y validar

los smart contracts, ya que allana el camino para aplicar técnicas de prueba tempranas antes de que los smart contracts se implementen y desplieguen en la red blockchain. El uso de este enfoque ha dado resultados muy satisfactorios en otras tecnologías, luego su utilización en este contexto parece muy prometedora.

Este estudio concluyó indicando que los modelos tradicionales de desarrollo no son los más adecuados dadas las necesidades identificadas para el desarrollo de smart contracts. Es más, después de terminar la SLR, han seguido apareciendo nuevos estudios relevantes, como los indicados a continuación.

Nuevos estudios relevantes

- Vandenbogaerde et al. (Vandenbogaerde, 2019) proponen un marco de trabajo basado en gráficos para calcular las métricas de diseño de software para el lenguaje Solidity (Ethereum). Concluyen indicando que la mayoría de los smart contract analizados son bastante sencillos desde el punto de vista de la orientación a objetos y que se deben desarrollar nuevas métricas de diseño específicas para los smart contracts.

- Ladleif et al. (Ladleif, 2019) pretenden allanar el camino hacia un enfoque basado en modelos para el desarrollo de smart contracts legales. Para ello, combinan los conocimientos de la literatura sobre derecho e informática jurídica con las capacidades de los enfoques de modelado existentes y ofrecen un modelo que encapsula los componentes esenciales de los smart contracts legales. Los autores indican que ese modelo podría utilizarse como referencia para los diseñadores de lenguajes que pretendan una representación holística de los smart contracts legales en una arquitectura basada en modelos. Además, puede servir de base para comparar los marcos de modelado

existentes, lo que demuestran aplicando el modelo a un conjunto de ocho lenguajes distintos.

- Pankov et al. (Pankov, 2020) enumeran una serie de herramientas existentes para probar y verificar los sistemas de la blockchain y los smart contracts, y también identifican el problema de la falta de una base normativa adecuada y de estándares en este ámbito.

- Miraz et al. (Miraz, 2020) exploran seis modelos de ciclo de vida de desarrollo tradicionales, identificando los conflictos existentes con la aplicación de smart contracts y defienden que hay una necesidad urgente de desarrollar nuevos modelos estándar para abordar los problemas que surgen. Los autores concluyen su trabajo indicando que los modelos tradicionales de desarrollo son inadecuados para las aplicaciones basadas en los smart contracts.

- Hu (Hu, 2020) propone el concepto de *smart contract engineering* (SCE) para facilitar la generación de smart contracts legales, que sería una combinación de ingeniería de software, métodos formales y derecho computacional. El objetivo de SCE es reducir los posibles errores y mejorar la eficiencia durante el proceso de desarrollo de los contratos, promoviendo al mismo tiempo la estandarización de las metodologías de diseño de contratos. Este artículo presenta la hoja de ruta de una metodología de diseño formal basada en el refinamiento iterativo y dirigida por modelos, no sólo para validar los contratos inteligentes, sino también para apoyar todo el ciclo de vida de su ingeniería.

- Lu et al. (Lu, 2021) indican que la MDE ayuda a reducir el riesgo de que los desarrolladores introduzcan vulnerabilidades en los smart contracts. Indican que la combinación de fragmentos de código probados según la especificación del modelo es más fácil de entender que el código fuente. Por ello, en este estudio presentan un enfoque MDE integrado en los procesos de negocio y la gestión de activos. Su

enfoque incluye métodos para el registro de activos fungibles/no fungibles, la custodia para el pago condicional y el intercambio de activos. El enfoque propuesto es implementado en Lorikeet - herramienta de generación de smart contracts - y lo evalúan en términos de viabilidad, corrección funcional y rentabilidad.

- Skotnica et al. (Skotnica, 2020) proponen un enfoque basado en modelos para crear smart contracts basados en un lenguaje visual específico del dominio, llamado DasContract. Se presenta un diseño mejorado de este lenguaje y se describe un proceso de generación de código en un smart contract.

- Hsain et al. (Hsain, 2021) destacan cómo se podría explotar la ingeniería basada en modelos para generar smart contracts. Para ello, revisan las investigaciones sobre generación de smart contracts en la blockchain Ethereum y basándose en los enfoques estudiados, indican ventajas y desventajas de cada enfoque. Este resultado de investigación se podría usar como base de la selección de herramientas para aspectos específicos del desarrollo de smart contracts.

- Vacca et al. (Vacca, 2021) analizan 96 artículos (escritos entre 2016 y 2020) que presentan soluciones para abordar los desafíos específicos de la ingeniería de software relacionados con el desarrollo, la prueba y la evaluación de la seguridad del software orientado a blockchain. En concreto, revisan artículos (que aparecieron en revistas y conferencias internacionales) relacionados con seis temas específicos: pruebas de smart contracts, análisis de código de smart contracts, métricas de smart contracts, seguridad de smart contracts, rendimiento de DApp y aplicaciones de blockchain.

- Jurgelaitis et al (2022) profundizan, en primer lugar, en los detalles sobre cómo las transformaciones de modelo a modelo a partir del Modelo Independiente de la Plataforma Blockchain (PIM) especificado, con el comportamiento de estado especificado, se pueden utilizar para producir un Modelo Específico de la Plataforma Solidity (PSM). En segundo lugar, estos autores explican cómo se utiliza el PSM de Solidity para la generación de código de smart contract de Solidity empleando transformaciones de modelo a texto (M2T).

Todos estos nuevos estudios apoyan la inquietud ya expuesta en la SLR, de alguna manera u otra. Pero, aunque todos son muy interesantes, en algunos casos su contenido o el enfoque abordado no permite abordar un testing temprano ni la generación automática de casos de prueba (Vandenbogaerde, 2019) (Ladleif,2019) (Hu, 2020) (Skotnica, 2020) (Lu, 2021) (Hsain, 2021) (Vacca, 2021) (Jurgelaitis, 2022), o son de difícil aplicación en la industria ya que se mantienen a un nivel teórico (Vandenbogaerde, 2019) (Ladleif,2019) (Hu, 2020) (Lu, 2021) (Jurgelaitis, 2022)), o se limitan a detallar, a alto nivel, las fuentes de datos o de información (Ladleif, 2019) (Skotnica, 2020) (Hsain, 2021) (Vacca, 2021). No obstante, estos estudios no hacen más que corroborar la necesidad ya identificada en la presente Tesis e indican que lo recorrido hasta ahora, en nuestras investigaciones, no está nada desencaminado.

III.5. Resumen del Capítulo

En este Capítulo se ha presentado el estado del arte en el contexto del ciclo de vida de desarrollo de los smart contracts, al objeto de analizar si hay estudios primarios que aborden la ingeniería dirigida por modelos u otras buenas prácticas para analizar, diseñar, implementar o probar los smart contracts.

En concreto, los resultados obtenidos mediante una SLR (Sánchez-Gómez, 2020b), nos ha permitido identificar y, sobre todo, analizar el estado del arte de las publicaciones científicas relacionadas con la temática de la presente Tesis.

Tras finalizar la SLR, hemos identificado que todos los estudios primarios, de alguna manera u otra, indican la necesidad de obtener un software que funciona de forma correcta. Si nos centramos en las pruebas funcionales, objeto de la presente Tesis, se vio de forma clara la necesidad de explorar una nueva línea de investigación como es la aplicación del paradigma de la ingeniería dirigida por modelos ya que, como se ha expuesto en Capítulos anteriores, parece que éste puede facilitar los mecanismos necesarios para verificar y validar los smart contracts, y nos allana el camino para aplicar los principios y técnicas del testing temprano, antes de que estos scripts, los smart contracts, se desplieguen en la red blockchain.

Este estudio concluyó exponiendo que los modelos tradicionales de desarrollo de software no serían los más adecuados, dadas las necesidades identificadas para el análisis, diseño, desarrollo y pruebas de los smart contracts.

Capítulo IV.
Metamodelos

"Make everything as simple as possible, but not simpler"
- Albert Einstein –

En este Capítulo se presenta una propuesta de metamodelo de smart contract, así como una propuesta de metamodelo de pruebas funcionales de un smart contract, ambos serán la base conceptual de nuestra propuesta y nos permitirán alcanzar los dos primeros objetivos específicos indicados anteriormente.

En concreto, la definición de estos dos metamodelos nos ayudará, por un lado, a generalizar de forma abstracta los componentes claves de nuestro trabajo: el smart contract y las pruebas funcionales de los smart contract y, por otro, a definir cada uno de los conceptos involucrados, junto con las relaciones y restricciones más relevantes que se producen entre ellos y que sentarán las bases para las transformaciones que podremos generar entre ellos y que serán la base para los automatismos que introduciremos en los siguientes Capítulos.

En el Capítulo también se ofrece un perfil UML para cada uno de los metamodelos definidos, que nos aportan un lenguaje específico de dominio de los mismos. Esto nos va a permitir que nuestros metamodelos sean transferibles, como se verá en el Capítulo VI al contexto industrial.

Este Capítulo recoge parte de nuestra propuesta de solución para el proceso sistemático de generación de pruebas funcionales, a partir de las especificaciones de los requisitos de los smart contract.

Lo que se pretende, como se ha venido introduciendo, es ofrecer una manera de especificar los requisitos de la solución y las especificaciones de los acuerdos, condiciones y reglas de negocio establecidos entre las partes y, a partir de todas estas especificaciones, generar de la manera más sistemática posible las pruebas funcionales que garanticen un correcto funcionamiento de la solución global y, de forma específica, de los smart contract. Para ello, vamos a utilizar la ingeniería guiada por modelos, paradigma que ya nos ha servido al grupo ES3 en otras ocasiones, para resolver problemáticas análogas.

De esta manera, a lo largo del presente Capítulo se define un metamodelo para representar los smart contracts, otro metamodelo para representar las pruebas funcionales de un smart contract, las relaciones existentes entre ambos metamodelos y un conjunto de transformaciones, del tipo M2M (Model to Model o Modelo a Modelo), para definir un proceso normalizado que permita generar las pruebas funcionales a partir de los smart contracts. Y, tras esto, se establecerá un lenguaje específico de dominio para que se puedan representar los elementos de ambos metamodelos de una forma cercana al contexto industrial.

Este Capítulo se completará con la información aportada en los siguientes Capítulos de la presente Tesis en el que, este contexto teórico, se sistematiza y se instancia para ser llevado a la práctica en un contexto industrial.

Para ello, por un lado, habría que definir los metamodelos que permitan describir los conceptos y las relaciones que entre ellos se producen y, por otro, diseñar las transformaciones necesarias entre modelos para poner en pie todo el proceso, teniendo en cuenta que estas transformaciones se sustentan en las relaciones que se establecen entre los conceptos del metamodelo.

En la Tabla 19, se indica a modo de resumen, tanto los metamodelos previstos, como las transformaciones necesarias en el Capítulo:

Metamodelos	- Metamodelo de smart contract. - Metamodelo de pruebas funcionales de smart contract.
Transformaciones	- Transformación (T1) para obtener modelo de escenarios de Caso de Uso. - Transformación (T2) para obtener modelo de Casos de Prueba.

Tabla 19. Metamodelos y transformaciones de la solución.

Partiendo de estos metamodelos y transformaciones, podemos definir un procedimiento sistemático de generación de las pruebas funcionales a partir de los smart contracts. En la Figura 22, se muestra el procedimiento propuesto.

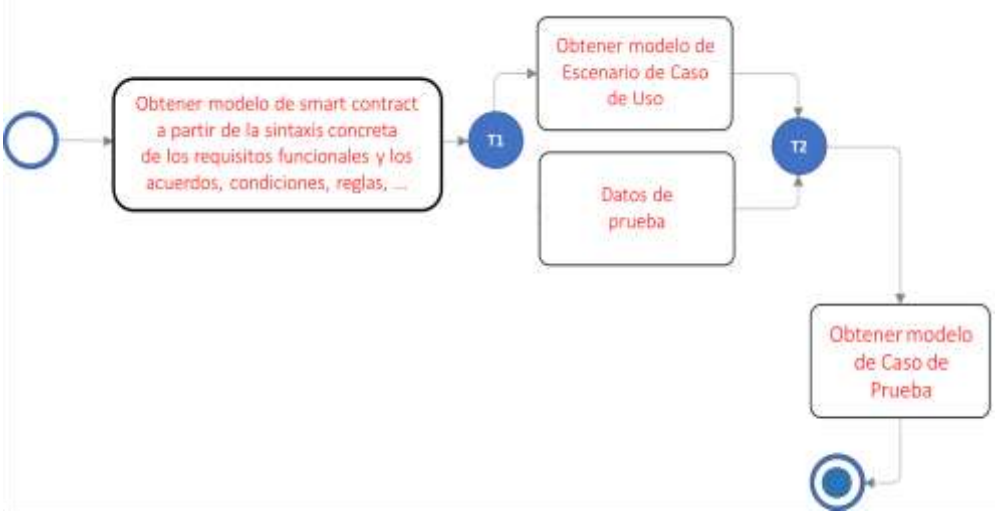


Figura 22. Proceso de generación de pruebas funcionales

Gracias al **metamodelo de smart contract** se podrá precisar la información necesaria para definir un modelo concreto de smart contract a partir de la sintaxis concreta de los requisitos funcionales y de los acuerdos, condiciones y reglas. El metamodelo va a marcar las reglas para que podamos instanciar un modelo acorde a todas las restricciones que en dicho metamodelo se han definido.

La transformación T1 nos va a permitir obtener un modelo de escenarios de Casos de uso. Este modelo debe ser acorde al **metamodelo de pruebas funcionales.** Enriqueciendo y combinado el modelo de *escenarios de Caso de Uso* con los *datos de prueba*, se podrá obtener el *modelo de Casos de Prueba* mediante la transformación T2.

Por ser más detallados, respecto a las transformaciones indicadas en la Tabla 19, habría que indicar que:

- La Transformación T1 es la transformación de un modelo de smart contract a un modelo de escenarios de Caso de Uso (transformación M2M). El modelo de smart contract proporciona una descripción general de lo que éste debe hacer, mientras que el modelo de escenario de Caso de Uso se centra en cómo se llevarán a cabo las interacciones entre los usuarios y el sistema para cumplir con esos requisitos.

  El objetivo de esta transformación es generar dicho modelo de escenarios conforme al metamodelo de pruebas funcionales de un smart contract, partiendo para ello del modelo de smart contract (modelo que se obtuvo conforme al metamodelo de smart contract).

- La Transformación T2 es la transformación a un modelo de Casos de Prueba (transformación M2M) a partir de un modelo de escenario de Casos de Uso. Esto implica convertir las interacciones de los usuarios y el sistema, descritas en los escenarios, en pasos detallados y específicos que puedan ser ejecutados y validados para verificar que el software funciona correctamente.

  Teniendo en cuenta los datos de prueba, el objetivo de esta transformación es generar el modelo de Casos de Prueba conforme al metamodelo de pruebas funcionales, quedando descrita cada acción que el actor realiza y cómo el sistema debe responder a esa acción.

Todas estas pautas, partiendo del *Graphical Abstract* de la Tesis, quedarían ilustradas de la siguiente forma.

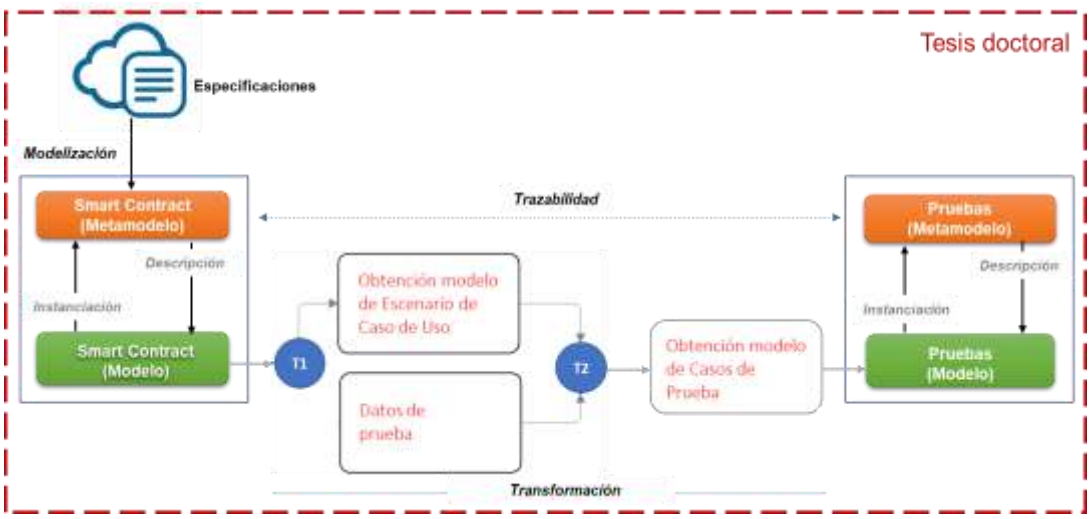


Figura 23. Pautas seguidas para la generación de pruebas funcionales

A continuación, en línea con todo lo expuesto, se define nuestra propuesta de metamodelo de smart contract y, posteriormente, la del metamodelo de pruebas funcionales.

IV.1. Metamodelo de smart contract

A continuación se define este metamodelo poniendo el foco, sobre todo, en los términos y las condiciones operativas - partes del contrato que se puede automatizar – del smart contract, dejando al margen todo lo relativo a los aspectos legales o jurídicos que, como ya ha indicado, quedan fuera del ámbito de esta Tesis Doctoral.

IV.1.1. Metamodelo de smart contract

En la Figura 24 se muestra el **metamodelo de smart contract** propuesto y, como se puede apreciar, este modelo se ha definido siguiendo la propuesta MOF (Meta-Object Facility) y se ha representado mediante un diagrama UML de clases, siguiendo las directrices del OMG.

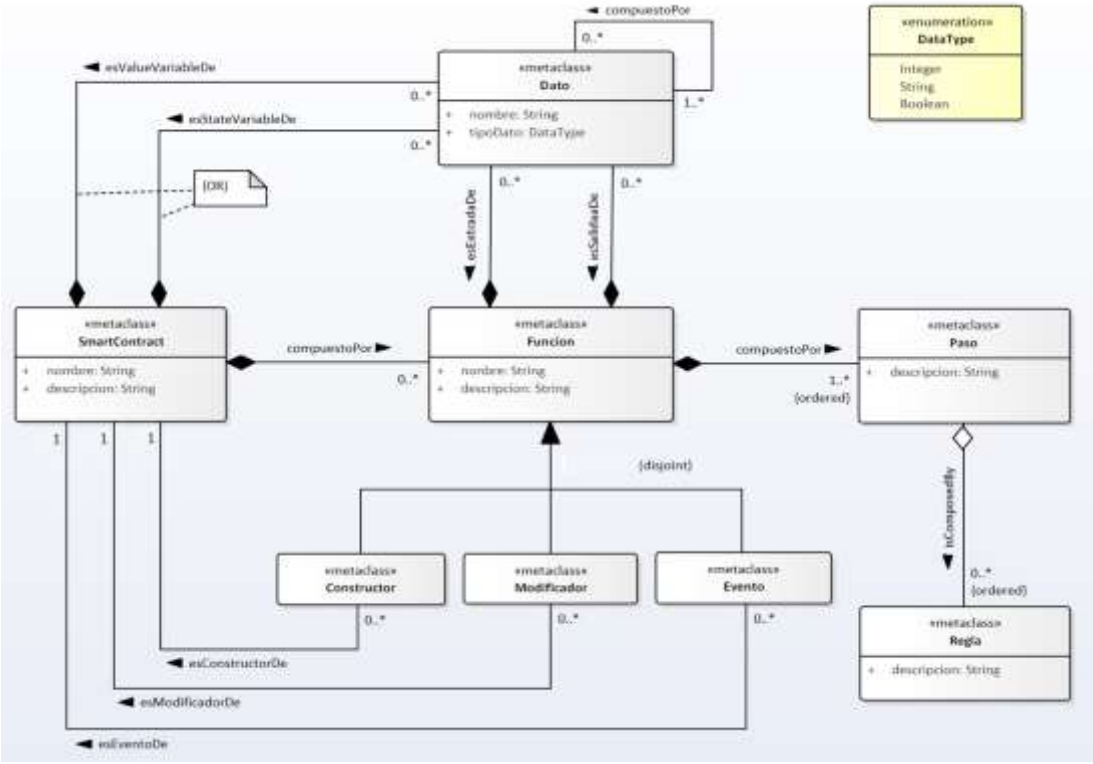


Figura 24. Propuesta de metamodelo de smart contract

Teniendo en cuenta los componentes del Contrato Triple Ricardiano (ver Figura 13), la anatomía y estructura de los smart contract, la información relativa a *Interesados* y *Datos* (Conjunto de Datos) estarían representados por la **Metaclass Dato**, las *Acciones* por la **Metaclass Funcion** y los pasos de estas acciones por la **Metaclass Paso** y, por último, las *Restricciones* estarían representadas por la **Metaclass Regla**. Además, como ya se ha indicado, los constructores, modificadores y eventos son metaclases más específicas de la Metaclass Funcion.

Para describir con más detalle nuestra propuesta, se seguirá el mismo esquema que utiliza UML (OMG, 2016), pero las relaciones entre clases, atributos y métodos no se indicarán en esta descripción, ya que son específicas de UML y se pueden obtener del estándar. En concreto, en las siguientes tablas se describirán cada una de estas metaclases que conforman el metamodelo propuesto, utilizando para ello el esquema de definición que el propio UML v2.5.1. utiliza para definir las metaclases de sus propios metamodelos (OMG, 2016).

- Metaclass SmartContract

Descripción	Representa a la entidad smart contract. Debe tener un nombre único al objeto de poder identificarlo de manera unívoca.
Generalización	Ninguna
Atributos	+ nombre: String [1] Descripción corta y concisa con la que los usuarios podrán identificar de manera unívoca al smart contract. + descripcion: String [1] Descripción amplia sobre cuál es el cometido del smart contract.
Operaciones	Ninguna

Asociaciones	+ esValueVariableDe: Dato [0..*] Conjunto de datos (variables) de memoria del smart contract (Value variable). + esStateVariableDe: Dato [0..*] Conjunto de datos (variables) de estado del smart contract (State variable). + compuestoPor: Funcion [0..*] Conjunto de funciones (métodos) que componen el smart contract. + esConstructorDe: Constructor [0..*] Conjunto de constructores que componen el smart contract. + esModificadorDe: Modificador [0..*] Conjunto de modificadores que componen el smart contract. + esEventoDe: Evento [0..*] Conjunto de eventos que componen el smart contract.
Restricciones	Ninguna

Tabla 20. Metaclass SmartContract

- Metaclass Dato

Descripción	Los datos son fundamentales para establecer cuál es la funcionalidad a ejecutar durante la instanciación de un smart contract ya que permiten establecer caminos alternativos en el flujo de datos y la ejecución del proceso y, además, son elementos de construcción para definir las restricciones en el proceso. Éste es el propósito de la Metaclass Dato.
Generalización	Ninguna

Atributos	+ name: String [1] Descripción corta y concisa con la que identificar de manera unívoca a la variable. + tipoDato: DataType [1] Establece cuál es el tipo de dato. Los posibles valores de este tipo han sido modelados por medio del enumerado DataType, cuya lista de valores (Integer, String, Boolean) puede ser ampliada en cualquier momento.
Operaciones	Ninguna
Asociaciones	+ compuestoPor: Dato [1..*] Conjunto o estructura en el que datos, de diversos tipos, pueden agruparse en un dato personalizado. Una vez que los tipos de datos se agrupan en una estructura, el nombre de la estructura representa los subconjuntos de variables que contiene. + esValueVariableDe: SmartContract [1] (Asociación de composición) Establece el smart contract en el cual se está definiendo este dato. + esStateVariableDe: SmartContract [1] (Asociación de composición) Establece el smart contract en el cual se está definiendo este dato. + esEntradaDe: Funcion [1] (Asociación de composición) Establece la función (método) que hace uso de este dato. + esSalidaDe: Funcion [1] (Asociación de composición) Establece la función (método) que devuelve este dato.
Restricciones	Respecto a la asociación «compuestaPor», es necesario definir una restricción en el modelo para controlar que una estructura no se contenga a sí misma, con el propósito de evitar que un usuario pueda definir de manera indeterminada y recursiva una estructura.

Tabla 21. Metaclass Dato

- Metaclass Funcion

Descripción	Esta Metaclass es el epicentro del metamodelo y es el elemento alrededor del cual orbitan el resto de los elementos del metamodelo. Representa cualquier acción que pueda contener el smart contract.

Generalización	Ninguna
Atributos	+ nombre: String [1] Descripción corta y concisa con la que identificar de manera unívoca a la función (o método). + descripcion: String [1] Descripción amplia sobre cuál es la funcionalidad que se cubre en esta función (o método).
Operaciones	Ninguna
Asociaciones	+ compuestoPor: SmartContract [1] (Asociación de composición) Establece el smart contract en el cual se está definiendo esta función. + esEntradaDe: Dato [0..*] Conjunto de variables de entrada de la función del smart contract. + esSalidadDe: Dato [0..*] Conjunto de variables de salida de la función del smart contract. + compuestoPor: Paso [1..*] Conjunto de pasos (sentencias), ordenados, con la funcionalidad de esta función (método).
Restricciones	Ninguna

Tabla 22. Metaclass Funcion

- Metaclass Constructor

Descripción	Esta Metaclass representa a los constructores que pueda contener el smart contract. Un constructor no deja de ser una función dentro de un smart contract, pero es la encargada de ejecutar cierto código de inicialización cuando se crea una instancia del contrato.
Generalización	Metaclass Función
Atributos	Ninguno

Operaciones	Ninguna
Asociaciones	+ esConstructorDe: SmartContract [1] Establece el smart contract en el cual se está definiendo este constructor.
Restricciones	Ninguna

Tabla 23. Metaclass Constructor

- Metaclass Modificador

Descripción	Esta Metaclass representa a los modificadores que pueda contener el smart contract. Los modificadores son utilizados para modificar el comportamiento de las funciones del smart contract.
Generalización	Metaclass Funcion
Atributos	Ninguno
Operaciones	Ninguna
Asociaciones	+ esModificadorDe: SmartContract [1] Establece el smart contract en el cual se está definiendo este modificador.
Restricciones	Ninguna

Tabla 24. Metaclass Modificador

- Metaclass Evento

Descripción	Esta Metaclass representa a los eventos que pueda tener el smart contract. Los eventos se utilizan para notificar los cambios realizados en un contrato. Por ejemplo, cuando se quiere notificar la recepción de criptomonedas, se puede emplear este mecanismo para que una aplicación externa reciba el mensaje e informe del hecho.
Generalización	Metaclass Funcion
Atributos	Ninguno
Operaciones	Ninguna
Asociaciones	+ esEventoDe: SmartContract [0..*] Establece el smart contract en el cual se está definiendo este Evento.
Restricciones	Ninguna

Tabla 25. Metaclass Evento

- Metaclass Paso

Descripción	La estructura de la función se construye a través de un conjunto ordenado de pasos (sentencias). Éstos son representados mediante la Metaclass Paso.
Generalización	Ninguna
Atributos	+ descripcion: String [1] Descripción detallada del paso (sentencia) de la función del smart contract.
Operaciones	Ninguna

Asociaciones	+ compuestoDe: Funcion [1] (Asociación de composición) Establece la función (método) en el cual se está definiendo este paso. + compuestoDe: Regla [0..*] Conjunto de reglas, ordenadas, con el comportamiento del paso de la función.
Restricciones	Ninguna

Tabla 26. Metaclass Paso

- Metaclass Regla

Descripción	El comportamiento de un paso (sentencia) de una función se construye a través de un conjunto ordenado de reglas. Éstas son representadas mediante la Metaclass Regla.
Generalización	Ninguna
Atributos	+ descripcion: String [1] Descripción detallada sobre las reglas a tener en cuenta.
Operaciones	Ninguna
Asociaciones	+ compuestaDe: Paso [1] (Asociación de Agregación) Establece el paso en el cual se está definiendo esta Regla.
Restricciones	Ninguna

Tabla 27. Metaclass Regla

IV.1.2. Perfil UML

Nuestra propuesta de metamodelo de smart contract, como ya se ha comentado, sigue los principios MOF y el estándar UML, pero, para que pueda hacerse uso de metamodelo en un contexto industrial, es necesario otorgar

un adecuado lenguaje específico de dominio, pensando en los equipos de trabajo que hagan uso de nuestra propuesta.

Para representar las necesidades funcionales y de comportamiento de los smart contracts, es decir, como lenguaje específico de dominio de nuestro metamodelo, se ha optado por hacer uso de modelos ya definidos por UML. En concreto, por un lado, se propone hacer uso de los diagramas de Casos de Uso y de los diagramas de Actividades para los requisitos funcionales y, por otro lado, los diagramas DMN (Decision Model and Notation) para las especificaciones de comportamiento. Estas elecciones no son la única posible, se podrían usar BPMN (Business Process Model and Notation) y los diagramas de máquina de estados para describir los comportamientos, pero en el contexto en el que se desarrolla esta Tesis es la opción que mejor se adecua, por los siguientes motivos:

- El grupo de investigación seno de esta Tesis, como ya se ha mencionado, tiene amplia experiencia con la metodología SofIA generando pruebas a partir de los requisitos funcionales - todo ello basado siempre en el estándar UML -.

- UML es un estándar mundialmente aceptado y conocido, lo que facilita la diseminación de nuestros trabajos.
- Las organizaciones en las que se realizan las validaciones de esta Tesis, así como la mayoría de las organizaciones empresariales, hacen uso profuso de UML como lenguaje de modelado. Por tanto, usarlo es una garantía a la hora de futuros trabajos con estas organizaciones en relación a posible transferencia de los resultados que obtengamos.
- Dado que, además, estas empresas se focalizan principalmente en UML como notación de modelos, garantizamos que en nuestras futuras transferencias, la curva de aprendizaje se reduce al usar esta notación.

La Figura *25* muestra la propuesta de un Perfil UML (Profile) que se aporta en este trabajo de Tesis.

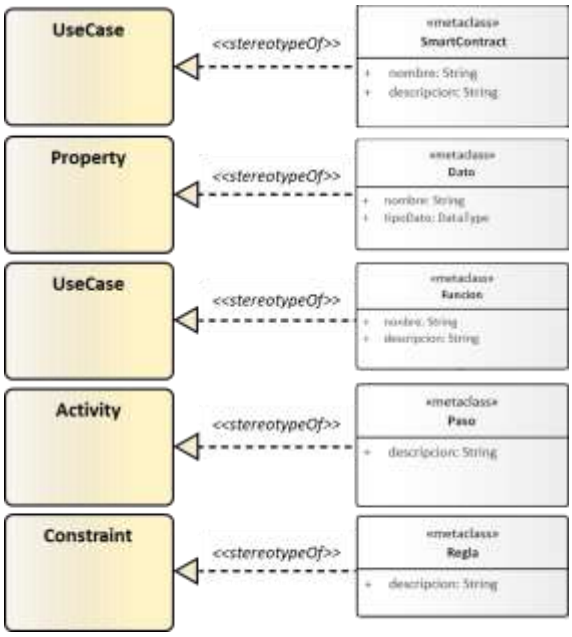


Figura 25. Perfil UML para el metamodelo de smart contract

Dado que en el metamodelo propuesto (ver Figura 24) tenemos un conjunto de clases que son herencias de la Metaclass Funcion (Metaclass Constructor, Metaclass Modificador y Metaclass Evento), se ha podido obtener un UML Profile muy compacto con sólo cinco definiciones. En concreto, como se puede apreciar en la Figura 25, tenemos: Metaclass SmartContract y Metaclass Funcion que son estereotipos de UseCase [Class], Metaclass Dato es un estereotipo de Property [Class], Metaclass Paso es un estereotipo de Activity [Class] y, por último, Metaclass Regla es un estereotipo de Constraint [Class]. Gracias a estos estereotipos - mecanismos de extensión del lenguaje UML -, se puede cambiar o complementar la semántica de cualquier elemento.

Es importante indicar que la simplicidad del UML Profile, que se muestra en la Figura 25, no implica que no se haya obtenido una herramienta con un enorme potencial, todo lo contrario, ya que la definición del metamodelo de

smart contract y el UML Profile van a permitir definir claramente cualquier concepto involucrado en un smart contract y emparejar cada concepto con conceptos similares en UML. Es decir, se podrán utilizar todos los mecanismos de extensión que UML ofrece para obtener una definición formal de un smart contract, siendo todo ello la base para la generación automática de pruebas funcionales.

IV.2. Metamodelo de pruebas funcionales

Las pruebas funcionales garantizan que las características y funcionalidades de un sistema de software se comportan según lo esperado. Para hacer pruebas funcionales se requiere una planificación que consiste en definir los aspectos a chequear y la forma de verificar su correcto funcionamiento.

Antes de seguir avanzando, vamos a recapitular algunos conceptos generales relacionados con los casos de uso, los diagramas de actividad, los diagramas DMN *(Decision Model and Notation)* y las pruebas funcionales - compuestas por escenarios de Caso de Uso y casos de prueba -:

- Un Caso de Uso es la descripción de una acción o actividad. Un diagrama de Casos de Uso es una descripción de todas las actividades que puede realizar un sistema de software. La siguiente Figura muestra un típico diagrama UML de Casos de Uso, que contiene cuatro casos de uso (sacar dinero, transferencias, depositar dinero y administración).

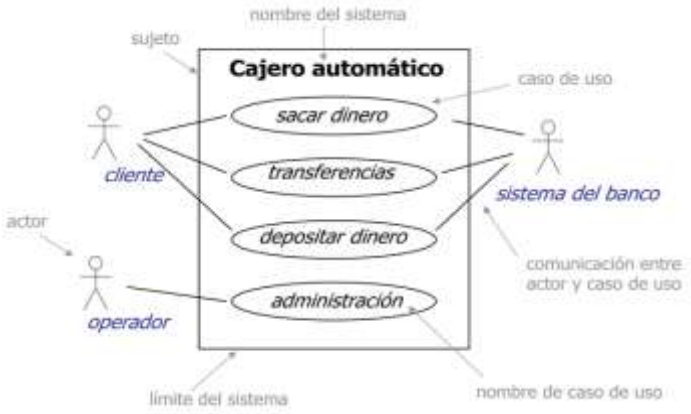


Figura 26. Ejemplo de Diagrama UML de Casos de Uso

- El comportamiento de un Caso de Uso se puede representar mediante un diagrama de comportamiento como es el diagrama de Actividad. La Figura *27* muestra un diagrama UML de Actividad y, como se puede apreciar, su estructura es muy parecido a un diagrama de flujo, pero permite mostrar un procesado paralelo. Este hecho es importante si se desea modelar varios hilos en los programas concurrentes o para flujos de procesos de negocio, algunos de los cuales pueden actuar en paralelo.

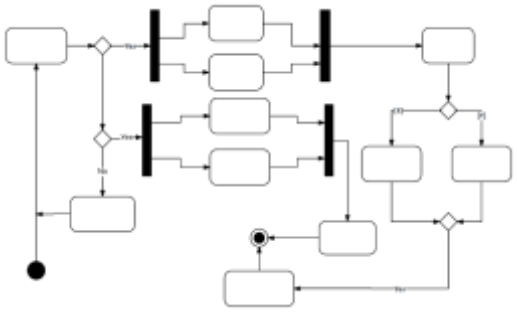


Figura 27. Ejemplo de Diagrama UML de Actividad

- En cualquier flujo de trabajo de un proceso de negocio hay una red de tomas de decisiones y los diagramas DMN proporcionan una notación de modelado que respalda la gestión de decisiones y las reglas de negocio.

  En la Figura 28, se muestra un modelado de procesos de negocio (nivel 1), donde es necesario tomar una decisión.

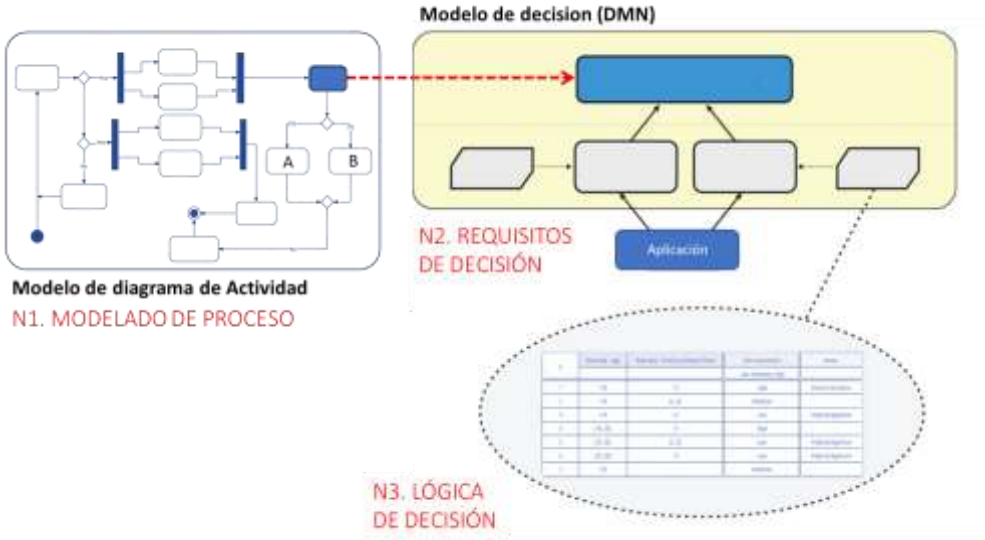


Figura 28. Ejemplo de modelado de decisión mediante DMN

Como se puede apreciar en la Figura 28, de una decisión depende el camino a seguir, camino A o camino B, luego cobra bastante importancia la capacidad que nos da DMN para automatizar dicha toma de decisión o de al menos describirla cubriendo todos los casos posibles y de forma exacta.

En la Figura 29, se muestra un ejemplo de diagrama de Requisitos de Decisión (nivel 2).

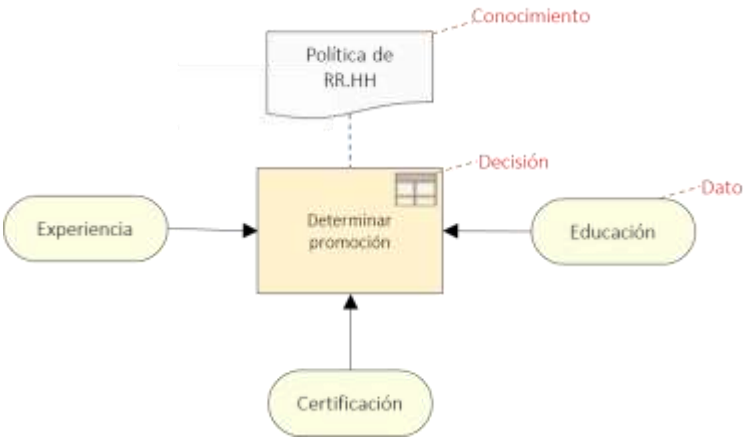


Figura 29. Ejemplo de diagrama de Requisitos de Decisión

Como se puede apreciar en la Figura 29, las notaciones básicas de este diagrama son:

- Conocimiento. Representada por una forma de documento con un nombre y ligada al rectángulo de la decisión con una línea punteada (esta línea puede terminar con un círculo). Las decisiones se derivan o soportan en el conocimiento involucrado en: políticas, regulaciones, mejores prácticas.

- Datos. Representada por un ovalo con el nombre del o los datos y va ligado al rectángulo de la decisión con una línea solida con una cabeza de flecha apuntándola. Datos relacionados con las preguntas y respuestas de la decisión y

- Decisiones. Representadas por un rectángulo con el nombre de la decisión. Permitiendo preguntas y respuestas validas en el contexto del negocio, la organización y el proceso. Donde la lógica de decisión (nivel 3) requiere de datos de entradas para determinar una salida (decisión), tal y como se muestra en la figura siguiente.

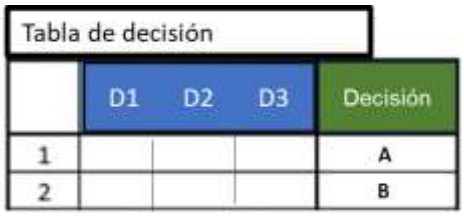


Figura 30. Ejemplo de Tabla de Decisión

- Un escenario es una instancia de un Caso de Uso, es decir es una descripción de una ruta específica a través del flujo del diagrama de comportamiento. En la Figura 31 se muestran posibles rutas (en amarillo) de un diagrama de Actividad, siendo cada una un escenario de Caso de Uso.

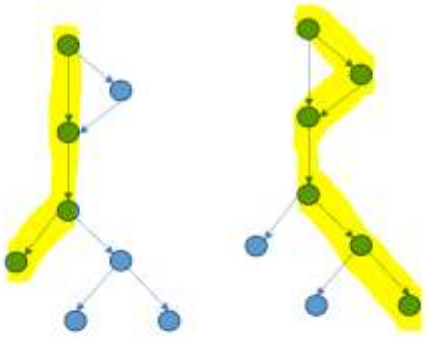


Figura 31. Ejemplo de escenarios de Caso de Uso

- Un Caso de Prueba representa un conjunto de datos de prueba de entrada que ejecutan un escenario (secuencia de pasos) para obtener unos datos de salida esperados (resultados).

En línea con lo expuesto, a continuación se define nuestra propuesta de metamodelo de pruebas funcionales de un smart contract.

IV.2.1. Metamodelo de pruebas funcionales de smart contract

En la Figura 32, se muestra nuestra propuesta de metamodelo de pruebas funcionales, así como los elementos y relaciones que componen nuestra propuesta.

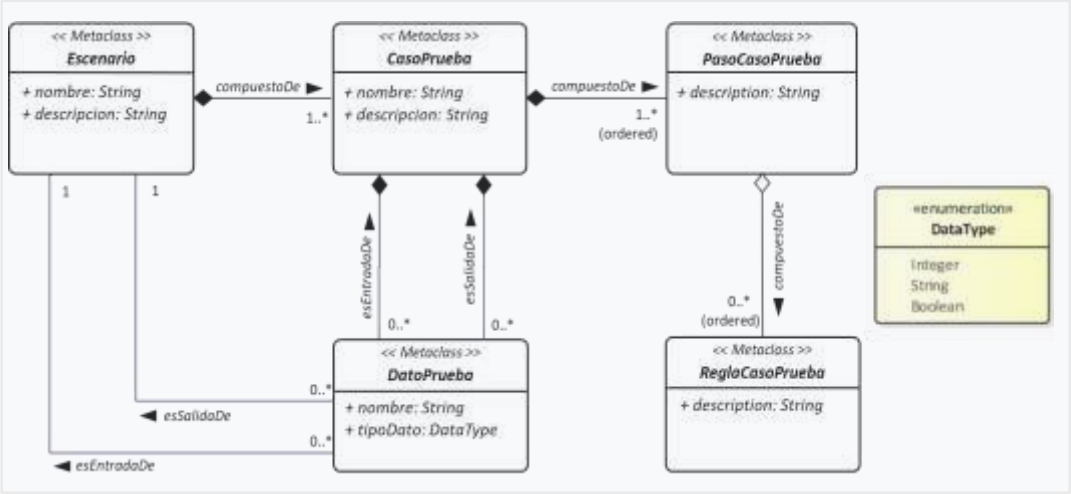


Figura 32. Metamodelo de pruebas funcionales

La Figura 32 muestra los elementos que se han definido para normalizar la información que recoge una prueba funcional. El elemento central de este metamodelo es la Metaclass CasoPrueba, la cual está relacionado con un requisito funcional y las especificaciones de comportamiento y está compuesto por pasos de Caso de Prueba.

En las siguientes tablas se describe con mayor nivel de detalle los elementos que aparecen en la Figura 32. Para ello, se seguirá haciendo uso de la estructura de OMG para la documentación de UML y especificaciones relacionadas, es decir, se seguirá la misma estructura utilizada durante la definición del metamodelo de smart contract.

- Metaclass Escenario

Descripción	Ésta representa al escenario de Caso de Uso, es decir, es una instancia del Caso de Uso.
Generalización	Ninguna
Atributos	+ descripcion: String [1] Descripción detallada del escenario que se pretende cubrir.
Operaciones	Ninguna
Asociaciones	+ compuestoDe: CasoPrueba [1..*] Conjunto de Casos de Prueba que componen el escenario. + esEntradaDe: DatoPrueba [0..*] Conjunto de variables de entrada a tener en cuenta en el escenario. + esSalidaDe: DatoPrueba [0..*] Conjunto de variables de salida a tener en cuenta en el escenario.
Restricciones	Ninguna

Tabla 28. Metaclass Escenario

- Metaclass CasoPrueba

Descripción	Esta es el elemento alrededor del cual orbitan el resto de los elementos del metamodelo. Este Metamodelo representa las funcionalidades a probar (en nuestro caso es una "Funcion").
Generalización	Ninguna
Atributos	+ description: String [1] Descripción detallada sobre cuál es la funcionalidad a probar.
Operaciones	Ninguna
Asociaciones	+ compuestoDe: Escenario [1] (Asociación de composición) Escenario del Caso de Prueba. + esEntradaDe: DatoPrueba [0..*] Conjunto de variables de entrada del Caso de Prueba. + esSalidaDe: DatoPrueba [0..*] Conjunto de variables de salida del Caso de Prueba. + compuestoDe: PasoCasoPrueba [1..*] Conjunto de pasos, ordenados, con la funcionalidad o comportamiento del Caso de Prueba.
Restricciones	Ninguna

Tabla 29. Metaclass CasoPrueba

- Metaclass DatoPrueba

Descripción	El concepto de datos de prueba es fundamental para establecer cuál es la funcionalidad / comportamiento concreto a probar ya que permite establecer caminos alternativos en el flujo de datos y en la ejecución del proceso. Éste es el propósito de la Metaclass DatoPrueba.
Generalización	Ninguna

Atributos	+ nombre: String [1] Descripción corta y concisa con la que identificar de manera unívoca el dato. + tipoDato: DataType [1] Establece cuál es el tipo de dato. Los posibles valores de este tipo han sido modelados por medio del enumerado DataType, cuya lista de valores (Integer, String, Boolean) puede ser ampliada en cualquier momento.
Operaciones	Ninguna
Asociaciones	+ esEntradaDe: Escenario [1] Escenario del conjunto de variables de entrada. + esSalidaDe: Escenario [1] Escenario del conjunto de variables de salida. + esEntradaDe: CasoPrueba [1] (Asociación de composición) Caso de Prueba del conjunto de variables de entrada. + esSalidaDe: CasoPrueba [1] (Asociación de composición) Caso de Prueba del conjunto de variables de salida.
Restricciones	Ninguna

Tabla 30. Metaclass DatoPrueba

- Metaclass PasoCasoPrueba

Descripción	La estructura de un Caso de Prueba se construye a través de un conjunto ordenado de pasos de prueba. Éstos son representados mediante la Metaclass PasoCasoPrueba (en nuestro caso son los pasos de la "Funcion").
Generalización	Ninguna
Atributos	+ descripcion: String [1] Descripción detallada sobre cuál es la funcionalidad o comportamiento que cubre el paso de Caso de Prueba.
Operaciones	Ninguna

Asociaciones	+ compuestoDe: CasoPrueba [1] (Asociación de composición) Caso se Prueba del conjunto de reglas, ordenadas, con el comportamiento. + compuestoDe: ReglaCasoPrueba [0..*] Conjunto de reglas con el comportamiento del paso de Caso de Prueba.
Restricciones	Ninguna

Tabla 31. Metaclass PasoCasoPrueba

- Metaclass ReglaCasoPrueba

Descripción	El comportamiento de un paso del Caso de Prueba se puede construir a través de un conjunto ordenado de reglas. Éstas son representadas mediante la Metaclass ReglaCasoPrueba (en nuestro caso son las reglas de un paso de "Funcion").
Generalización	Ninguna
Atributos	+ descripcion: String [1] Descripción detallada sobre cuáles son las reglas de comportamiento a tener en cuenta en el paso de Caso de Prueba.
Operaciones	Ninguna
Asociaciones	+ compuestoDe: PasoCasoPrueba [1] (Asociación de Agregación) Establece el paso en el cual se está definiendo esta Regla.
Restricciones	Ninguna

Tabla 32. Metaclass ReglaCasoPrueba

IV.2.2. Perfil UML

De igual manera que el metamodelo de smart contract, nuestra propuesta de metamodelo de pruebas funcionales de un smart contract sigue los principios MOF y el estándar UML. Luego, para hacer este metamodelo totalmente compatible con UML, en este apartado se propone el siguiente UML Profile que, como ya se indicó, no es más que un mecanismo de extensión genérico para personalizar el modelo UML propuesto.

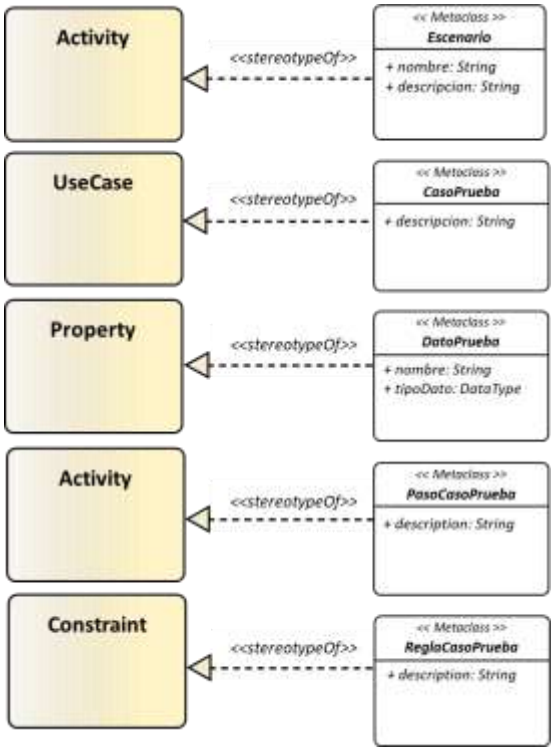


Figura 33. Perfil UML para el metamodelo de pruebas funcionales

Como se puede apreciar en la Figura 33, tenemos: Metaclass Escenario que es un estereotipo de Activity [Class] al igual que la Metaclass PasoCasoPrueba, la Metaclass CasoPrueba son estereotipos de UseCase [Class], la Metaclass DatoPrueba es un estereotipo de Property [Class], y, por último, la Metaclass ReglaCasoPrueba es un estereotipo de Constraint [Class]. Como ya se ha indicado, gracias a estos estereotipos se puede cambiar o complementar la semántica de cualquier elemento.

Es importante recalcar, de nuevo, que la simplicidad de este UML Profile no implica que no se haya obtenido una herramienta con un enorme potencial, todo lo contrario, ya que la definición del metamodelo de pruebas funcionales y el UML Profile nos van a permitir definir claramente cualquier concepto involucrado en una prueba funcional de un smart contract y emparejar cada concepto con conceptos similares en UML. Es decir, se podrán utilizar todos los mecanismos de extensión que UML ofrece para obtener una definición formal de una prueba funcional de un smart contract.

IV.3. Relaciones entre los metamodelos

Como se comentó al inicio del Capítulo, las relaciones que se establecen de manera teórica, entre los conceptos definidos en el metamodelo de smart contracts y el metamodelo de pruebas funcionales, son la base que nos van a permitir definir y generar las transformaciones entre ellos. En la Figura 32, se presentan las relaciones que se producen entre estos conceptos y que usaremos en el siguiente Capítulo para la definición de las transformaciones.

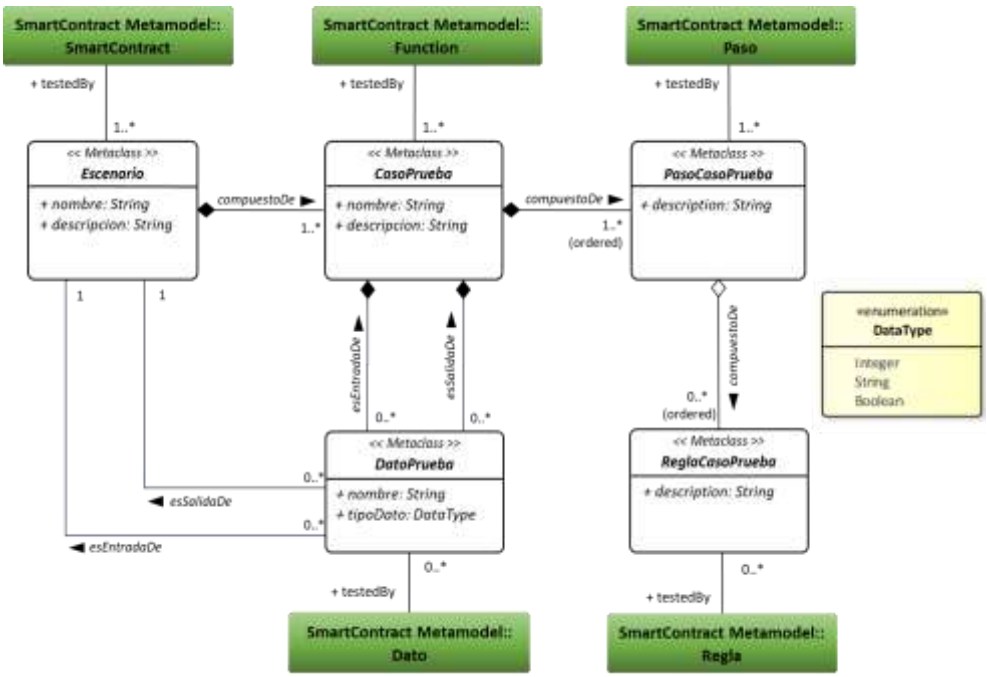


Figura 34. Relaciones de trazabilidad entre metamodelos

En la figura, los conceptos relativos al metamodelo de smart contracts se han representado de color verde. El uso de este tipo de decoraciones está autorizado por UML y aumentan la legibilidad de nuestra representación. Estos conceptos, se relacionan, como se pueden ver, con los conceptos definidos en el metamodelo de pruebas. Al objeto de definir las relaciones de trazabilidad entre metamodelos, tenemos las siguientes interrelaciones:

- La Metaclass Escenario está relacionada con la Metaclass SmartContract del metamodelo de smart contract.
- La Metaclass CasoPrueba está relacionada con la Metaclass Function del metamodelo de smart contract.
- La Metaclass DatoCasoPrueba está relacionada con la Metaclass Dato del metamodelo de smart contract.
- La Metaclass PasoCasoPrueba está relacionada con la Metaclass Paso del metamodelo de smart contract.
- La Metaclass ReglaCasoPrueba está relacionada con la Metaclass Regla del metamodelo de smart contract.

IV.4. Resumen del Capítulo

Este Capítulo recoge la parte conceptual de nuestra propuesta de solución para el proceso sistemático de generación de pruebas funcionales, a partir de las especificaciones de los requisitos y de comportamiento de los smart contract.

En concreto, a lo largo del presente Capítulo, se ha definido un metamodelo para representar los smart contracts, un metamodelo para representar las pruebas funcionales. Y, tras esto, se ha establecido un lenguaje específico

de dominio para que se puedan representar los elementos de ambos metamodelos mediante perfiles UML. Además, se ha presentado el proceso teórico que nos permitirá generar un modelo a partir de otro y las relaciones que entre los conceptos de ambos modelos se producen y que sustentaran dichas transformaciones.

Capítulo V.
Mecanismos y herramienta de soporte a la generación de pruebas

"You don't really understand something unless you are able to explain it to your grandmother."
- Albert Einstein -

En este Capítulo se expone, por un lado, el proceso de generación de pruebas funcionales, es decir, se especifican las transformaciones que permitirán a partir del metamodelo de smart contract y del metamodelo de pruebas funcionales y, de las relaciones que entre ellos existen, generar el modelo de escenarios de Caso de Uso y el modelo de Casos de Pruebas que permitirán verificar y validar la funcionalidad y comportamiento de los smart contract. Por otro lado, se detalla la utilidad o herramienta implementada para la sistematización de este proceso.

Como ya se ha comentado, esta Tesis Doctoral y su solución se enfocan dentro del contexto de trabajo del grupo ES3 y se pretenden instanciar dentro de su herramienta metodológica SofIA. Actualmente, el marco metodológico SofIA permite generar pruebas funcionales, de forma sistemática, a partir de

un requisito funcional. Un ejemplo de aplicación para la generación de los casos de pruebas funcionales a partir de un requisito funcional sería el siguiente.

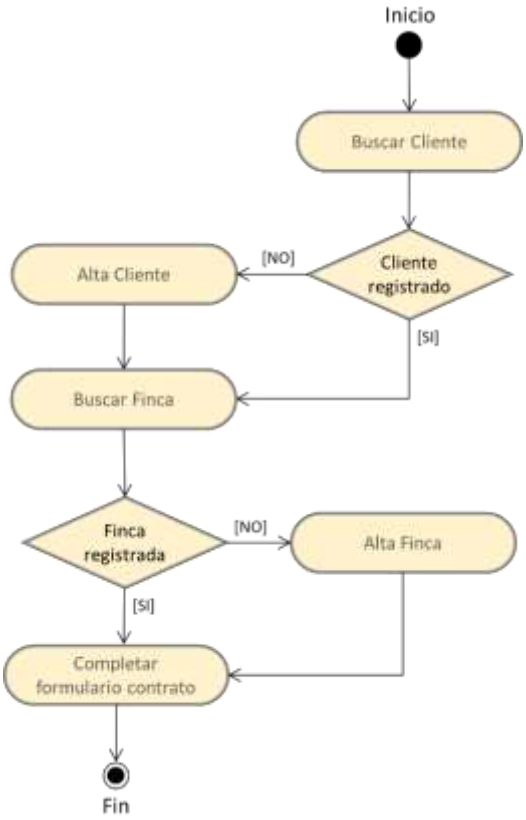


Figura 35. Ejemplo de un diagrama de Actividad de un Caso de Uso

En la Figura *35* se muestra el diagrama de Actividad de un requisito funcional recogido en la fase de requisitos de un sistema de software, en concreto, este ejemplo describe cómo se asocia una Finca a un Cliente a través de su Contrato de suministro de agua.

Las pautas de generación de los escenarios que sigue SofIA, depende de cómo esté definido cada requisito funcional. En nuestro caso de ejemplo, los requisitos funcionales están definidos mediante diagramas UML de Actividad, luego la suite de herramientas de SofIA generará tantos casos de pruebas como caminos (flujos) existan entre la actividad inicial y final del diagrama.

En concreto, a partir de este requisito funcional, SofIA ha permitido generar varios escenarios

- Escenario 1, cuando Cliente y Finca están registrados en el sistema.
- Escenario 2, si el Cliente está registrado, pero la Finca no está registrada.
- Escenario 3, cuando no está registrado el Cliente, pero si lo está la Finca.
- Escenario 4, si no está registrado ni Cliente ni Finca. En este caso hay que dar de alta a ambos y completar el formulario de contrato.

A modo de ejemplo, en la Figura siguiente se muestra como quedaría estos escenarios.

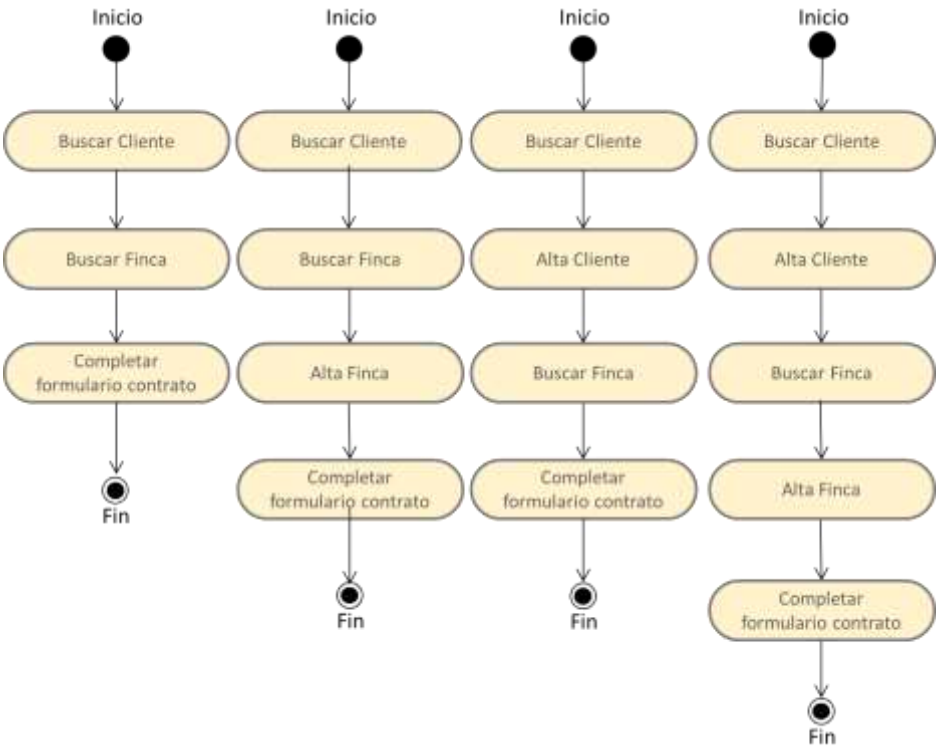


Figura 36. Escenarios obtenidos por SofIA

Dado que la aplicación de MDE y, en particular, la aplicación de las transformaciones de modelos es una tarea muy monótona y supone un coste elevado, es necesario disponer de una herramienta software que automaticen el proceso. El marco metodológico SofIA incluye una suite de herramientas desarrolladas bajo Enterprise Architect que permiten dar soporte al ciclo de vida de un proyecto software, como ya se ha indicado.

Dado que esta generación sistemática de pruebas funcionales de un sistema genérico de software ya lo soporta SofIA, a continuación se presenta nuestra propuesta para obtener las pruebas funcionales a partir de nuestra propuesta, es decir, se detallarán cómo se realizan las transformaciones especificadas anteriormente para generar las pruebas funcionales de un smart contract. Para esta propuesta, usaremos siempre como referencia el contexto tecnológico y la herramienta CASE donde reside SofIA.

V.1. Obtención del modelo de smart contract

En base al lenguaje específico de dominio de smart contract que se definió en el Capítulo anterior, el modelo de requisito funcional que instanciará nuestro metamodelo de smart contract será representado mediante un diagrama UML de Actividad a partir de este metamodelo. En la Figura 37 se muestra un ejemplo, donde se presenta la funcionalidad del smart contract. Este pequeño ejemplo nos va a permitir introducir de una manera natural el proceso que vamos a desarrollar, con las transformaciones que se detallarán más adelante en este mismo Capítulo.

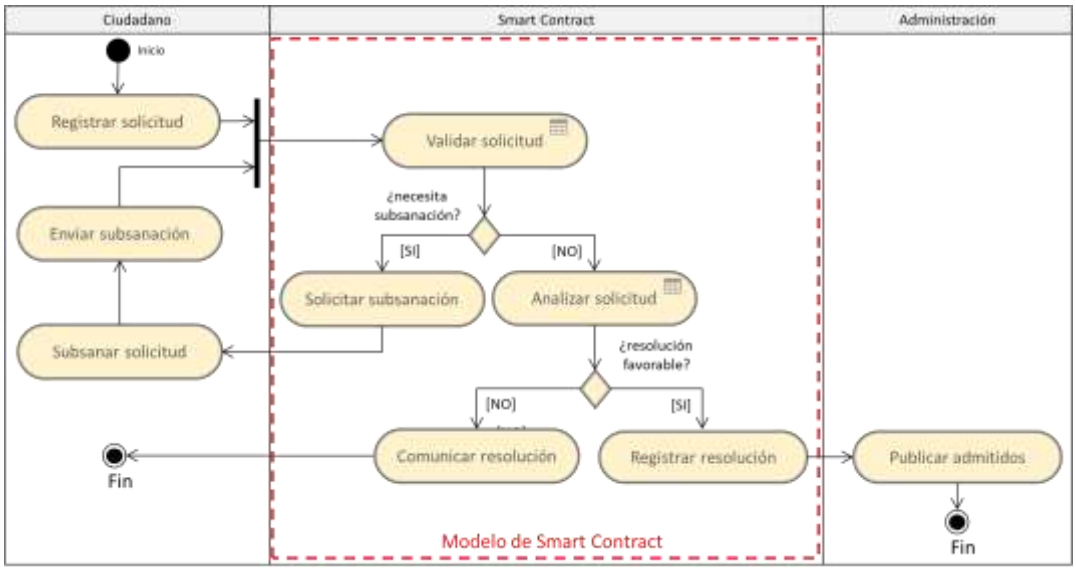


Figura 37. Modelo de smart contract

A partir de este diagrama, que ya tiene establecidas las posibles operaciones, condiciones y reglas del smart contract, se podría obtener el tipo Class del smart contract (ver ejemplo mostrado en la Figura *17*). Como se puede apreciar en la Figura *37*, en este diagrama se pueden visualizar parte de los diferentes elementos que componen el smart contract como, por ejemplo, "Solicitar subsanación" que sería un evento del smart contract. También hay elementos específicos para la toma de decisiones como es "Analizar solicitud". En concreto, este elemento como se puede apreciar en la Figura *38*, contiene una tabla de decisión (icono en forma de tabla en la parte superior derecha).

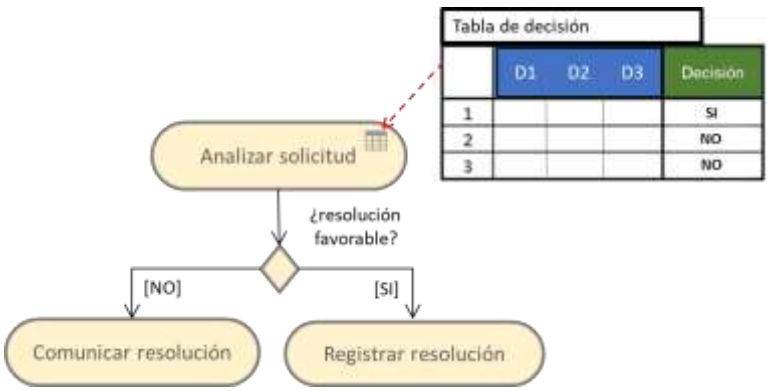


Figura 38. Tabla de decisión de un diagrama DMN

Gracias a esta tabla se puede definir la lógica de una decisión, es decir, esta tabla incluida en el diagrama DMN permitiría especificar las entradas / salidas de datos, las condiciones, reglas, así como una "política de aciertos" del smart contract de una forma totalmente visual (ver Tabla *33*).

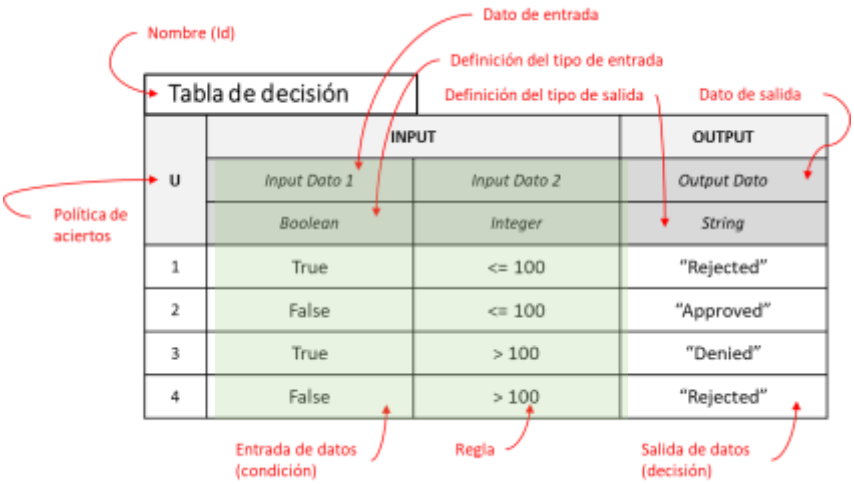


Tabla 33. Ejemplo de tabla de decisión

Analizando la Tabla *33*, vemos que en la esquina superior izquierda aparece el nombre de esta tabla de decisión y debajo hay una "U", que significa único y es la política de aciertos definida para esta tabla de decisión. Indica que, cuando hay que tomar una decisión, sólo una de las filas de abajo puede ser verdadera.

Las dos columnas en verde claro se refieren a los posibles datos de entrada: Input Dato 1 e Input Dato 2. En DMN, éstos son las etiquetas de una expresión de entrada. Las celdas de más abajo (llamadas entradas) se refieren a las posibles condiciones de la entrada. Si estas condiciones estuvieran entre comillas, como por ejemplo "SI", es porque estamos comparando valores de cadena. En nuestro caso, tenemos un dato tipo *Boolean* (condición) y otro de tipo *Integer* (regla). Para cada entrada posible

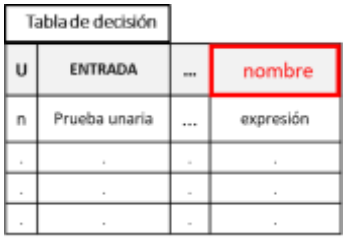
(éstas van enumeradas de 1 a n), definimos la salida de datos (decisión), correspondiente con la celda de más a la derecha de la tabla.

La definición de que, según una entrada de datos que sea verdadera (o una combinación de entrada de datos verdaderas), debe aplicarse una salida de datos específica, es una regla. Por tanto, cada regla se define en una fila de la tabla, debajo de la cabecera de la tabla, y tiene un número, que puede encontrar en las celdas de la izquierda.

Además, estas tablas de decisión, como se muestra en la siguiente figura, nos permitirían incluso estructurar perfectamente muchos de los componentes de un smart contract. En concreto, haciendo una traslación de los elementos de la tabla de decisión a un smart contract se podrían obtener el siguiente pseudocódigo (ver Tabla *34*).

	Tabla de decisión	Smart Contract (pseudocódigo)

Representar datos de entrada		<< DECLARACIÓN DE VARIABLES >>
		Contrato XXX {
		[...]
		Funcion xxx () {
		Declara variable nombre
		[...]
		}
		}

Representar datos de salida		<< ASOCIADOS A EVENTOS >>
		Contrato XXX {
		[...]
		Declara Evento ([...], nombre);
		[...]
		Funcion xxx() {
		[...]
		Llamar al evento (nombre);
		}
		}

<table>
<tr>
<td rowspan="2">Representar condiciones y reglas</td>
<td> 
</td>
<td><< ESTABLECER REGLAS Y CONDICIONES >>

Contrato XXX {
[...]
Funcion xxx () {
[...]
si (true [...]) {
[...]
}
[...]
}
}</td>
</tr>
<tr>
<td> 
</td>
<td><< PRUEBAS UNARIAS[9] >>

Contrato XXX {
[...]
Funcion xxx () {
[...]
si (true [...] and nombre comp* valor) {
[...]
}
[...]
}
}

* comp (comparador)</td>
</tr>
</table>

Tabla 34. Traslación elementos de la tabla de decisión a un smart contract

A continuación, teniendo en cuenta estas pautas del modelado del smart contract, se especificarán formalmente las transformaciones indicadas en el Capítulo anterior al objeto de generar las pruebas funcionales de un smart contract.

[9] Las pruebas unarias son sentencias que se evalúan como verdadero o falso para un valor específico.

V.2. Transformaciones a realizar

Una vez modeladas la funcionalidad, pasos y reglas que conforman el modelo de smart contract, las transformaciones a realizar serían las siguientes:

- Transformación T1 para obtener el modelo de Escenarios de Caso de Uso.
- Transformación T2 para obtener el modelo de Casos de Prueba.

Como ya se indicó en el Capítulo anterior, se van a utilizar transformaciones de modelo a modelo (M2M y, en concreto, las transformaciones que se muestran en la Figura siguiente.

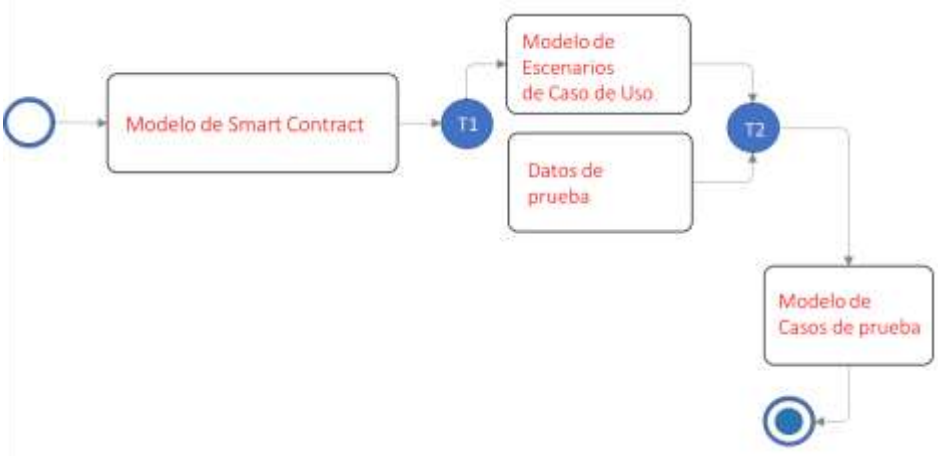


Figura 39. Transformaciones para la obtención de pruebas funcionales

Para facilitar su comprensión, se presentan ambas transformaciones usando "pseudocódigo" para su representación. No obstante, a la hora de describir una transformación existen otras opciones como el lenguaje QVT (Query-View-Transformation), pero hemos optado por el pseudocódigo ya que nuestro fin último es implementar dichas transformaciones en el entorno de SofIA mediante el lenguaje de programación Java y, la translación del pseudocódigo al entorno de SofIA, es mucho más natural que si usamos otros lenguajes de representación de transformaciones.

A continuación, en la Figura 40 se muestra la transformación principal (main) cuya funcionalidad es recuperar la información de los smart contracts modelados y, por cada uno, hacer una llamada a las transformaciones indicadas en la Figura 39.

```
Transformacion () {
    main() {
        obtenerSmartContracts()
        for each smartContract {
            call Transformacion1 ()  \\ Llamar a la trasformación T1
            call Transformacion2 ()  \\ Llamar a la trasformación T2
        }
    }
}
```

Figura 40. Pseudocódigo de las transformaciones

V.2.1. Transformación para la obtención del modelo de escenario de Caso de Uso

En el caso de la transformación T1, se obtienen los escenarios de Caso de Uso, es decir, por cada camino del flujo descrito en el modelo de smart contract se obtendrá un escenario, tal y como muestra el siguiente pseudocódigo:

```
Transformacion1 () {
        obtenerEscenarios()    \\ Obtener todos los escenarios del modelo de Smart Contract
        for each escenario {
            create new escenario    \\ Crear escenario de Caso de Uso
        }
    }

    obtenerEscenarios(){
        \\ Obtener todos los caminos posibles del modelo, siendo cada camino un escenario
        \\ Se deben identificar el conjunto de datos, sí como todas las funciones por donde
        \\ pasa el escenario, además de los pasos y restricciones, si existen
        (...)
}
```

Figura 41. Pseudocódigo de la transformación T1

V.2.2. Transformación para la obtención del modelo de Casos de Prueba

Mediante la transformación T2 se crearán los casos de prueba de forma combinada a los datos, en concreto, se sigue el siguiente pseudocódigo:

```
Transformacion2() {
    for each escenario {
      obtenerCasos()                  \\ Obtener Casos de Prueba
      for each caso {
        obtenerDatosCaso              \\ Identificar el  conjunto de datos de prueba
        create new caso               \\ Crear Caso de Prueba
        obtenerPasos ()               \\ Obtener pasos del Caso de Prueba
        for each paso {
          create new paso             \\ Crear paso del Caso de Prueba
          obtenerReglas ()            \\ Obtener reglas del paso
          for each regla {
            create new regla          \\ Crear regla del paso del Caso de Prueba
          }
        }
     }
  }

  obtenerCasos(){
    \\ Recuperar los Caso de Uso
    (...)
  }
  obtenerDatosCaso (){
  {
    \\ Establecer el conjunto de datos específicos
   (...)
  }
  obtenerPasos(){
    \\ Recuperar los pasos del Caso de Uso
    (...)
  }
  obtenerReglas(){
    \\ Recuperar las reglas del paso de Caso de Uso
    (...)
   }
}
```

Figura 42. Pseudocódigo transformación T2

V.3. Herramienta de soporte

Para hacer posible la utilización práctica de este entorno teórico, es necesario desarrollar una herramienta CASE que dé soporte durante esta construcción y que permita la automatización de las reglas de transformación definidas. Este apartado tiene como propósito la descripción de los fundamentos de la herramienta CASE que hará posible la aplicación práctica y efectiva de todo el marco de trabajo teórico presentado hasta ahora.

Para desarrollar esta herramienta se podría optar por una de las dos alternativas siguientes: (i) diseñar y desarrollar una herramienta ad-hoc con capacidad para modelar diagramas de procesos y ejecutar reglas de derivación; o (ii) utilizar como base una herramienta de modelado que proporcione mecanismos de extensión – basados, por ejemplo, en plugins – y cuyo uso sea extendido en el ámbito empresarial, facilitando de esta manera una posible transferencia tecnológica del conocimiento teórico desarrollado en esta Tesis hacia el tejido empresarial.

Al objeto de aprovechar los trabajos previos del grupo de investigación ES3, en este trabajo de Tesis se opta por la segunda de estas opciones y para ello, se opta por la herramienta de modelado Enterprise Architect (Sparx Systems, 2023), en adelante EA, como herramienta base sobre la que desplegar y desarrollar un módulo funcional específico. Es más, esta herramienta será una extensión de las herramientas del marco metodológico SofIA, ya que con ella se podrá gestionar el ciclo de vida del proceso software, como hasta ahora, e incluir nueva extensión para la generación sistemática de pruebas funcionales de un smart contract.

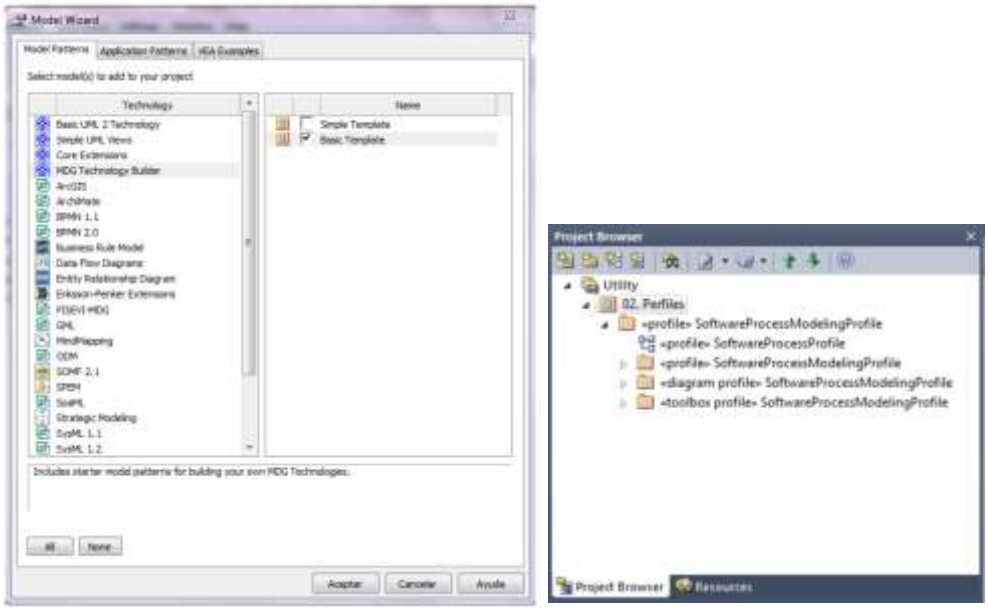
A continuación, se describen con algo más de detalle algunas evidencias sobre cómo se han definido en EA los perfiles UML descritos y cómo se han implementado las transformaciones indicadas anteriormente. Aunque este proceso no se describe en su totalidad, se presenta de la manera más rigurosa y completa posible como para hacer ver al lector el trabajo realizado.

V.3.1. Sintaxis definidas en Enterprise Architect

EA ofrece la posibilidad de definir sintaxis concretas, entre otros recursos de modelado, utilizando para ello su herramienta MDG Technologies, Model Driven Generation Technologies. Esta herramienta permite extender las capacidades de modelado que ofrece por defecto esta herramienta CASE, de tal manera que es posible definir diferentes recursos para modelar y, entre dichos recursos, se encuentra la posibilidad de definir perfiles UML.

En concreto, se han seguido los siguientes pasos para implementar un perfil UML del metamodelo. Lo primero es crear un proyecto "MDG Technology" en EA para, posteriormente, crear e instalar un fichero "MDG Technology" desde EA.

Para este primer paso es necesario crear un proyecto «MDG Technology» utilizando el asistente de EA (ver Model Wizard en Figura *43*).



(Model Wizard) *(Project Browser)*

Figura 43. Proyecto «MDG Technology» en Enterprise Architect

Una vez seleccionado este tipo de proyecto e indicado su nombre, EA creará una estructura de paquetes, tal y como aparece en el *Project Browser* de la Figura anterior. El objetivo de cada uno de estos paquetes es el siguiente:

- Profile. Contiene el conjunto de estereotipos, etiquetados con la palabra clave "stereotype", junto con sus valores etiquetados, que componen el perfil UML. Cada uno de estos estereotipos estará enlazado con la metaclase UML oportuna a través de una relación "extend".

- Diagram profile. Este contiene todos aquellos artefactos de EA que son necesarios para definir "cómo crear diagramas", según determinados estereotipos de los definidos en el paquete anterior, y seleccionados previamente. A partir de este conjunto de artefactos, se le otorga al usuario la capacidad para modelar siguiendo un determinado perfil UML.

- Toolbox profile. Este paquete contiene los artefactos de EA necesarios para la creación de cajas de herramientas personalizadas, conocidas en EA como "toolbox". Esta toolbox proporciona acceso rápido a los constructores de los elementos contenidos en un determinado perfil UML. Además, es posible organizar visualmente estos constructores en compartimentos o páginas según la terminología EA.

Una vez creados los perfiles UML necesarios, los tipos de diagramas y las cajas de herramientas, se crea el fichero "MDG Technology" para la definición de procesos software. Para ello, basta con utilizar el asistente[10] que proporciona esta herramienta CASE e introducir determinados parámetros de configuración, tal y como se muestra en la Figura *44*.

[10] Este asistente está disponible en el menú principal de Enterprise Architect: Tools | Generate MDG Technology File.

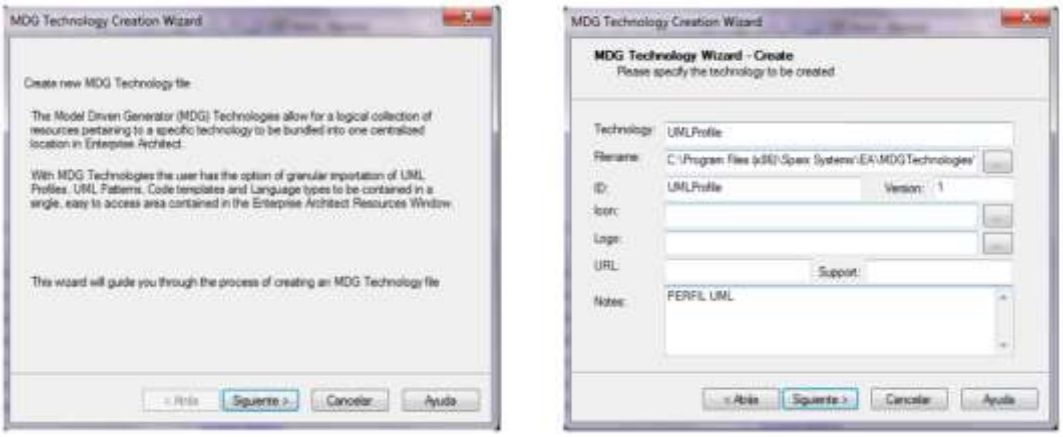


Figura 44. Asistente de Enterprise Architect para generación del fichero "MDG Technology"

Una vez finalizado el proceso que nos ha guiado el asistente, se genera un fichero, con formato XML, que contiene la configuración indicada. Como último paso para poder utilizar el perfil UML, basta con desplegar el fichero generado dentro de la carpeta "MDGTechnologies" situada en el directorio de instalación de esta herramienta CASE.

V.3.2. Nuevo «Add-in» para Enterprise Architect

Como ya se ha comentado, en nuestro caso la traslación práctica de todo el entramado teórico de metamodelos y reglas de transformación definidos en Capítulos anteriores se traduce en el desarrollo de un nuevo módulo funcional o plugin de la herramienta CASE. Esto es lo que se conoce como "Add-in" en la terminología de la herramienta EA.

Un "Add-in" es un mecanismo para extender la funcionalidad que ofrece EA en base a la utilización de objetos Windows OLE Automation (ActiveX) que exponen métodos de uso público que responde a la interfaz de EA y a determinados eventos. Gracias a este mecanismo no se requiere una configuración por parte del usuario, ya que simplemente basta con instalar el "Add-in" desarrollado y se podrán definir interfaces gráficas de usuario integradas dentro del entorno EA, así como recibir notificaciones sobre

eventos de la interfaz de usuario del propio EA como, por ejemplo, eventos de manipulación de los modelos UML o asociados a la gestión del repositorio de datos.

En el caso que nos ocupa, el "Add-in" incluye los perfiles descritos en Capítulos anteriores y, además, implementa todas las transformaciones vistas anteriormente, permitiendo la automatización del proceso de generación de pruebas funcionales. Estas transformaciones se han implementado en la herramienta en lenguaje de programación Java. No obstante, la elección de este lenguaje no es determinante ya que otros de propósito general como Python, C#, Ruby, etc., podrían haber sido utilizado de la misma manera.

A continuación, a modo de evidencia, se muestra el programa principal implementado en Java (ver Figura 45).

```
public class Transformacion {
    public static void main(String[] args) throws Exception {
        List<String> result;
        String resultado;
        //Leer archivos de la carpeta indicada
        String directorio=args[0]+File.separator+"entrada"+File.separator+"modelos";
        File modelos = new File(directorio);
        String[] lista = modelos.list();
        //Lee los ficheros que se encuentran en la carpeta indicada
        for (int i = 0; i < lista.length; i++) {
            File archivoInterno = new File(directorio + File.separator + lista[i]);
            directorio=args[0]+File.separator+"entrada"+File.separator+"modelos"+
                                File.separator+archivoInterno.getName();
        String clase = "";
        List<String> atributos = new ArrayList<>();
        try (Stream<String> lines = Files.lines(Paths.get(directorio))) {
            result = lines.collect(Collectors.toList());
            for(String line : result){
                if Transformacion1(line){
                    Transformacion2(line);
                }
            }
        }
    }
}
```

Figura 45. Extracto del código de la clase Transformación

En la Figura *46*, a modo ilustrativo, se muestra otro extracto de código de la clase Transformacion2, donde se ha implementado la declaración de las reglas establecidas a partir de las anotaciones del diagrama DMN.

```
//Declaración de reglas
resultado+="\n\tcheckRules(proceso=null,etapa=null,evento=null){\n";
String directorioReglas=args[0]+File.separator+"entrada"+File.separator+"reglas";
File reglas = new File(directorioReglas);
String[] listaReglas = reglas.list();
String resultadoReglas="";
for (int i = 0; i < listaReglas.length; i++) {
    File archivoInterno = new File(directorio + File.separator + listaReglas[i]);
    directorio=args[0]+File.separator+"entrada"+File.separator+"reglas"+
                    File.separator+archivoInterno.getName();
    try (Stream<String> lines = Files.lines(Paths.get(directorio))) {
        result=lines.filter(x->!x.isEmpty()).collect(Collectors.toList());
        int count=0;
        while(count<result.size()){
            while(!result.get(count).contains("when")){
                if(result.get(count).contains("rule")){
                    String line=result.get(count).replace("rule ", "");
                    resultadoReglas+="\t\t//Regla "+line+"\n";
                }
                count++;
            }
            String existAttributes="";
            String evalResult="";
            while(!result.get(count).contains("then")){
                //Comprueba si existen los valores
                if(result.get(count).contains(":")){
                    String line=result.get(count);
                    existAttributes+=line.trim().split(":")[0]+"!= null && ";
                }
                //Crea la condición eval
                if(result.get(count).contains("eval")){
                    //Múltiples condiciones de eval
                    if(result.get(count).contains("&&")){
                        String[] evals = result.get(count).split(" && ");
                        evalResult+="\t\tif(";
                        for(int e=0; evals.length-1>=e; e++){
                            String rule=evals[e].trim().substring(5, evals[e].trim().
                                                                    length()-1);
                            rule=formatString(rule).replace("size", "lenght");
                            if(e<evals.length-1){
                                evalResult+=rule+ " && ";
                            } else {
                                evalResult+=rule;
                            }
                        }
                        evalResult+="){\n";
                    } else {
                        //Una condición de eval
                        String rule=result.get(count).trim().substring(5,
                                        result.get(count).trim().length()-1);
                        rule=formatString(rule).replace("size", "lenght");
                        evalResult+="\t\tif("+rule+"){\n";
                    }
                }
                count++;
            }

            (...)
```

Figura 46. Extracto del código de la clase Transformacion2

NOTA: Código disponible en GitHub.

V.4. Resumen del Capítulo

En este Capítulo se ha presentado nuestra propuesta para obtener las pruebas funcionales a partir de un modelo de smart contract, es decir, se han detallado cómo se realizan las transformaciones especificadas para generar las pruebas funcionales de un smart contract.

Adicionalmente, para hacer posible la utilización práctica de nuestra propuesta teórica, se han indicado las pautas seguidas para su implementación en EA para construir y automatizar las reglas de transformación definidas. Con ello, se ha hecho posible la aplicación práctica y efectiva de todo el marco de trabajo teórico presentado hasta ahora.

Capítulo VI.
Casos prácticos y validación

"The good ideas will survive."
- Quentin Tarantino –

En este Capítulo se presenta un caso práctico donde se han aplicado los resultados obtenidos en esta Tesis Doctoral en el contexto del testing temprano para los smart contracts y que ha servido de marco de validación de nuestra propuesta.

Tras seguir las pautas propuestas, el equipo de trabajo ha constatado que, siguiendo este enfoque, se han podido detectar bugs y errores en los smart contract antes de que fuesen desplegados en una red blockchain y que, en general, la calidad del software desarrollado ha mejorado sustancialmente.

A continuación, se expone un caso práctico abordado por el grupo ES3 recientemente, donde se ha aplicado esta propuesta.

VI.1. Proyecto SmartAuditor

VI.1.1. Referencia

El proyecto SmartAuditor - Testeo temprano de smart contracts para elegir socios en el emprendimiento - (Ref. P20_00644), es un proyecto financiado por la Consejería de Economía, Conocimiento, Empresas y Universidad - Junta de Andalucía -, en la convocatoria de "Ayudas a proyectos de I+D+i (PAIDI) para universidades y entidades públicas de investigación del Sistema Andaluz del Conocimiento". Presupuesto concedido: 124.600€ (2021- 2022).

VI.1.2. Visión global del proyecto

El objetivo principal del proyecto SmartAuditor (ver Figura 47) fue desarrollar una propuesta metodológica, acompañada de herramientas que dieran soporte al aseguramiento temprano de la calidad de los smart contracts (blockchain) y, por otro, que dicha propuesta se instancie en una solución tecnológica que facilite la búsqueda inteligente de socios en entornos de emprendimiento.

Figura 47. Pantalla principal de SmartAuditor

Los objetivos específicos del proyecto eran los siguientes:

OBJ 1. Definir los mecanismos para definir y diseñar blockchain smart contracts para garantizar su testeo temprano.

OBJ 2. Definir el proceso a desarrollar y las técnicas a aplicar para la sistematización del aseguramiento de la calidad, partiendo de los blockchain smart contracts.

OBJ 3. Implementar las herramientas necesarias para dar soporte a la aplicación de esta propuesta metodológica.

OBJ 4. Desarrollar una solución en el entorno del emprendimiento basada en esta propuesta metodológica.

OBJ 5. Validar y transferir a entornos empresariales.

OBJ 6. Obtener resultados transferibles a entornos docentes.

En este proyecto se tenía claro que no se podía avanzar de manera competitiva en la implantación de la tecnología blockchain (o cualquier otra) si no se avanza en mecanismos para el aseguramiento de la calidad de la misma. Para ello, en el proyecto SmartAuditor se aplicaron los principios y técnicas del testing temprano, ya que esto permitía asegurar la calidad de los desarrollos de smart contracts en entornos blockchain, es decir, se siguieron las pautas y herramientas propuestas en la presente Tesis.

El proyecto SmartAuditor es un proyecto que ofrece dos importantes retos. Por un lado, presenta la necesidad de desarrollar un contexto de pruebas de blockchain, poniendo en marcha todo lo relativo al Capítulo V de este trabajo de Tesis. Pero, además, requería que la solución propuesta se presentase sobre un entorno funcional complejo. Para ello, se trabajó en colaboración con el grupo SEJ230- Planificación y Análisis Económico y se abordó una solución orientada a definir un sistema inteligente de búsqueda de socios

empresariales. La solución a desarrollar debía implementar una serie de test, desarrollados por un grupo multidisciplinar de profesores de la Facultad de Turismo y Económicas de la Universidad de Sevilla. Estos test, analizan las competencias de emprendedores y la herramienta obtenida es capaz de encontrar sinergias empresariales para detectar potenciales socios empresariales.

En cuanto a la arquitectura tecnológica blockchain, en el proyecto se usó Hyperledger Fabric, tal y como se muestra en la Figura 48.

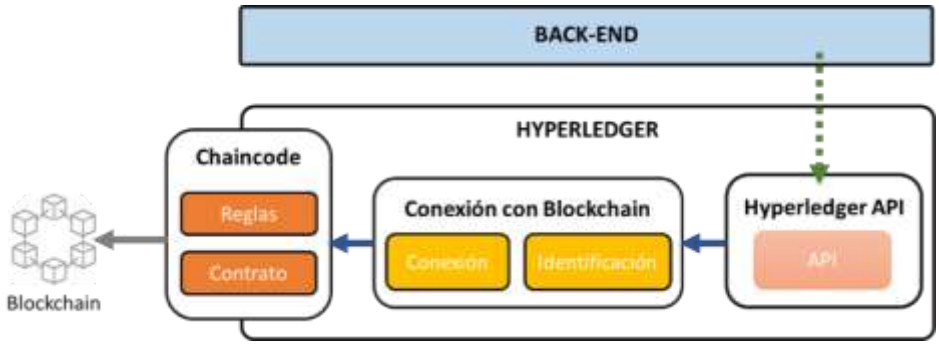


Figura 48. Arquitectura capa blockchain (proyecto SmartAuditor)

En esta arquitectura, como se puede apreciar, el enlace de comunicación se realiza mediante el uso del kit de desarrollo Fabric-SDK, que permite crear una conexión con la red blockchain, Hyperledger Fabric en este caso, y gestionar las transacciones hacia ella. El flujo de comunicación es a través del elemento Conexión. Éste se comunica con el elemento Chaincode, que son los smart contracts de la blockchain Hyperledger.

VI.1.3. Aplicación de la propuesta

Se sale del contexto de la presente Tesis presentar todos los casos de uso (ver Figura *49*) y todos los requisitos funcionales desarrollados en el desarrollo del proyecto.

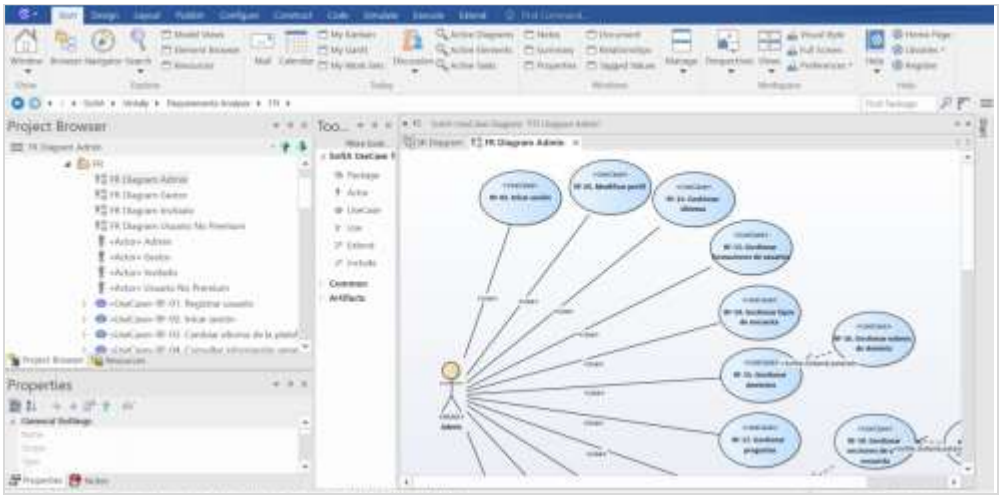


Figura 49. Interfaz del Enterprise Architect con los plugins instalados

Sin embargo, para poder dar una visión general de cómo los resultados de esta Tesis se han validado en este contexto, vamos a mostrar algún Caso de Uso sencillo que nos permita mostrar cómo se ha ido aplicando nuestra propuesta.

Antes de seguir avanzando, es importante indicar que dado que el proyecto SmartAuditor, al ser un proyecto funcionalmente complejo - como ya se ha comentado -, se decidió involucrar a la empresa Dinngo, consultores especializados en Design Thinking[11] que, de manera externa, entrevistó a más de 40 usuarios/as al objeto de disponer de un diseño de alto nivel del producto software a desarrollar y una definición de las diferentes historias de usuario de la solución objetivo.

Una vez finalizado este trabajo inicial y, teniendo ya instalados los metamodelos y perfiles - descritos del Capítulo IV - en Enterprise Architect, así como las transformaciones implementadas como extensiones (plugin) en Enterprise Architect - las descritas en el Capítulo V -, en el proyecto

[11] Design Thinking es un marco metodológico de resolución de problemas centrado en el humano. Se utiliza para abordar de manera creativa y colaborativa desafíos complejos en diversos campos, como diseño de productos, desarrollo de servicios, innovación empresarial, entre otros.
Fue popularizado por la firma de diseño IDEO y la Escuela de Diseño de la Universidad de Stanford en la década de 1990, pero desde entonces se ha extendido a muchas otras industrias y disciplinas.

SmartAuditor se siguió el procedimiento de generación de pruebas funciones, descritas en la Figura 22 de la presente Tesis.

En concreto, los pasos seguidos fueron los siguientes (ver Figura 50):

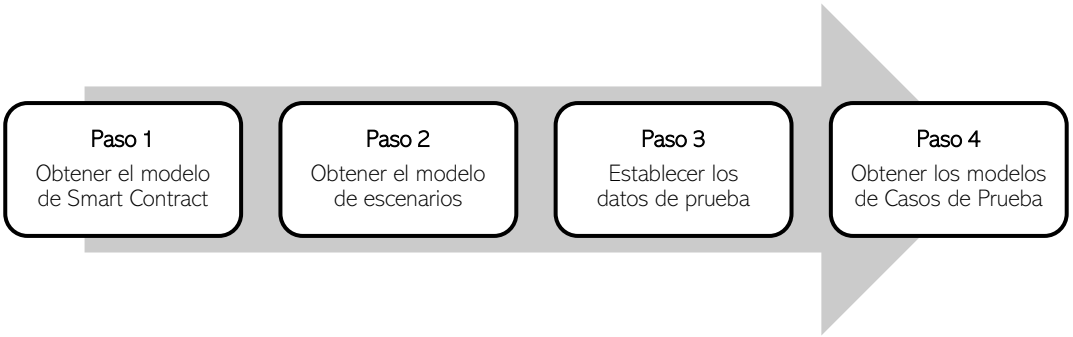


Figura 50. *Procedimiento de generación de pruebas funcionales (pasos)*

PASO 1. Obtener el modelo de smart contract, a partir de la sintaxis concreta de los requisitos funcionales y los acuerdos, condiciones, reglas...

Figura 51. Pautas seguidas en el proyecto
(Proyecto SmartAuditor)

El primer paso era la definición de todos los casos de uso y diagramas asociados. En concreto, en el proyecto se siguió el esquema recogido en la Figura 51, al objeto de modelar los requisitos funcionales del proyecto global y las especificaciones de los smart contract a implementar. En concreto, se utilizaron los diagramas de Casos de Uso y los diagramas de comportamiento mediante diagramas de Actividad, para especificar los requisitos funcionales, los diagramas de Clase para diseñar los datos y para las especificaciones del comportamiento de los smart contract los diagrama DMN que nos permite indicar condiciones y reglas, tal y como se plantea en nuestra propuesta. Todos ellos se modelaron a través de la herramienta EA y, más concretamente, mediante las extensiones implementadas en SofIA, que ya incluye los resultados de nuestro trabajo de Tesis.

Adicionalmente, como indica la Figura 51, en el proyecto se propuso la utilización de plantillas de código para la obtención de pequeños esqueletos de código ya estructurado con las partes que componen un smart contract: funciones, constructores, modificadores y eventos.

Para exponer de forma amigable este trabajo, en la Figura 52 se indica a modo de ejemplo un Caso de Uso. Éste se utilizará como punto de partida para exponer, de forma ilustrativa, cómo se ha validado nuestra propuesta en el proyecto SmartAuditor.

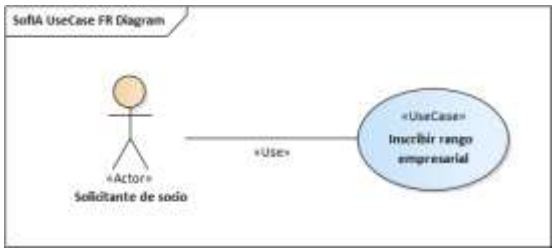


Figura 52. Ejemplo de un Caso de Uso
(Proyecto SmartAuditor)

Siguiendo el marco metodológico SofIA, la definición del comportamiento de este Caso de Uso se apoyará en un diagrama de Actividad (ver Figura *53*) y,

como se puede apreciar, existe en el modelo de Smart Contract un paso con un icono en forma de tabla que indica que tiene asociado un diagrama DMN – extensión implementada en SofIA -.

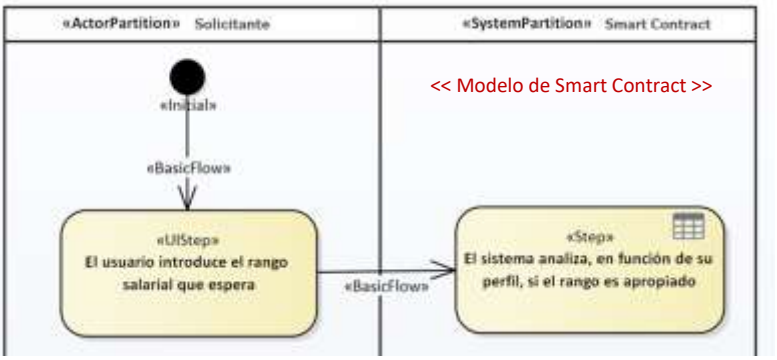


Figura 53. Extracto del diagrama de Actividad
(Proyecto SmartAuditor)

Este diagrama DMN (ver Figura *54*) permite identificar de forma visual todos los atributos a tener en cuenta en la tabla de decisión (Experiencia, Educación, …). Incluso, como se puede apreciar en la Figura, uno de los atributos puede ser el resultado de otra tabla de decisión.

Figura 54. Extracto de diagrama DMN
(Proyecto SmartAuditor)

Las condiciones y reglas a cumplir estarían indicadas en cada una de estas tablas de decisión. Si nos centramos, en la tabla de decisión "Determinar el rango salarial", estos mismos atributos que aparece en el diagrama son los que se muestran en la cabecera de la tabla de decisión (ver Figura 55).

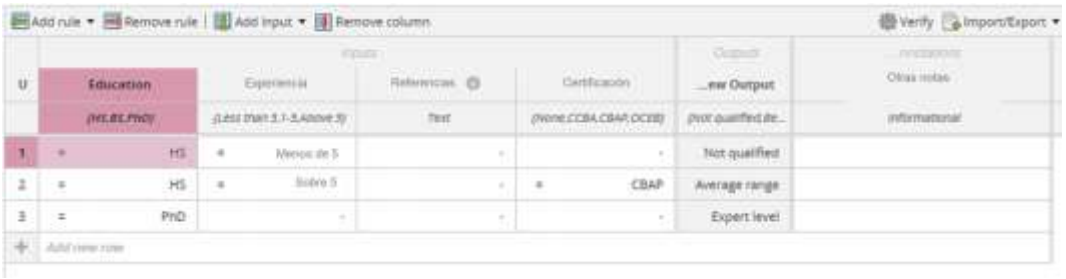


Figura 55. Ejemplo de tabla de decisión utilizada
(Proyecto SmartAuditor)

Como se puede apreciar, en cada una de las filas de la tabla se definen las distintas casuísticas o combinaciones que tienen que darse para que la solución tome una decisión u otra. Por ejemplo si su educación es HS (*High School*) y tiene menos de 5 años de experiencia, el candidato no está cualificado, es decir, no aplica.

Toda esta modelización, debe ser enriquecida con el diagrama de Clases y, opcionalmente, se pueden incluir plantillas de código, si se pretendiera obtener esqueletos de los smart contract – tal y como se hizo a modo de pruebas en este proyecto -.

- En la Figura 56 se muestra un extracto de diagrama de Clases que se corresponde con nuestro ejemplo. Nótese que este diagrama es mucho más amplio, de hecho, se han puesto puntos suspensivos en los atributos para aumentar la elegibilidad del ejemplo.

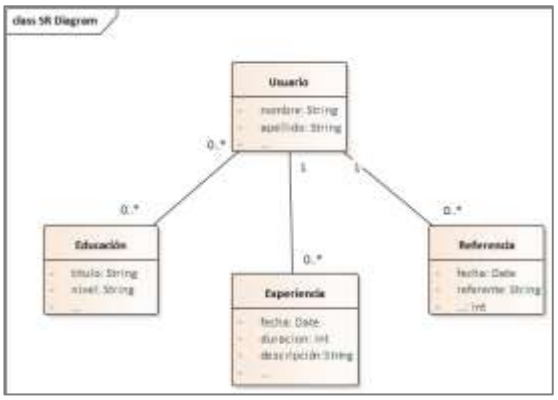


Figura 56. Extracto del diagrama de Clases
(Proyecto SmartAuditor)

- Siguiendo con el ejemplo, en la Figura 57 se muestra como a través de EA se podría importar/exportar plantillas de código. Este ejemplo muestra, además, un extracto de una de las plantillas utilizadas, a modo de prueba, en este proyecto.

Figura 57. Importación/Exportación de las plantillas de código
(Proyecto SmartAuditor)

PASO 2. Obtener el modelo de escenarios de Casos de Uso

Una vez introducidos todos los elementos anteriores, mediante la Transformación T1 – implementada como un plugin de SofIA -, se puede generar un primer modelo de escenarios de Casos de Uso (ver Figura *58*).

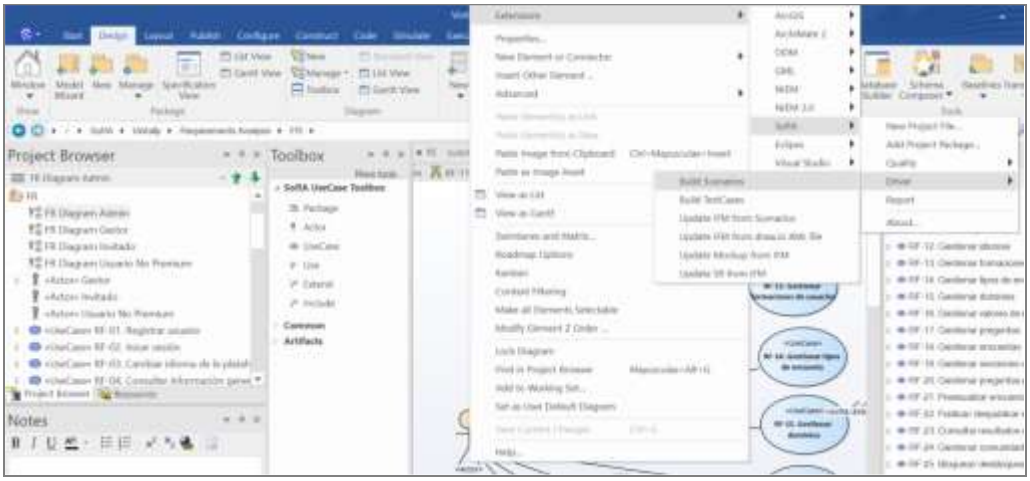


Figura 58. Interfaz del Enterprise Architect para generar escenarios
(Proyecto SmartAuditor)

Este escenario se puede enriquecerse manualmente con las restricciones y ampliaciones que se estimen oportunas. Así, para el diagrama de Actividad esbozado en la Figura *53*, el modelo resultante sería el que se muestra en la Figura 59.

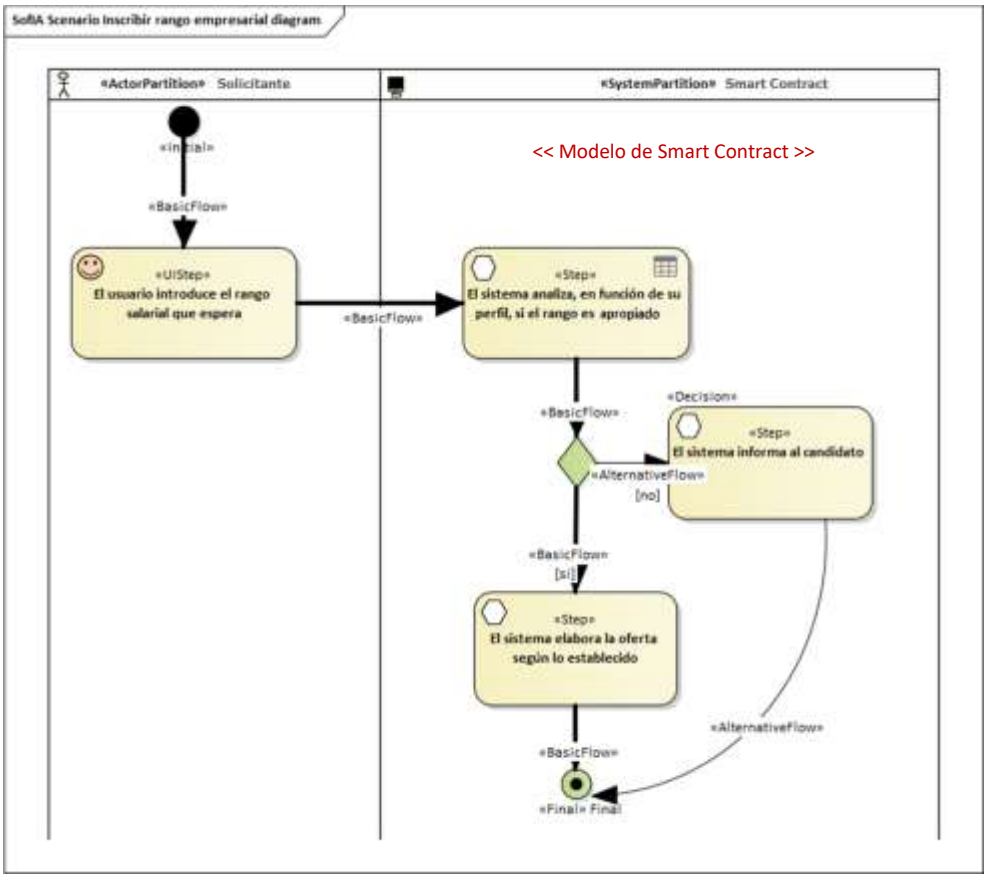


Figura 59. Ejemplo de escenario de Caso de Uso
(Proyecto SmartAuditor)

Este diagrama establece que un usuario, desde el Front-End, establece sus condiciones económicas "introduciendo el rango salarial que espera". Tras este paso, el sistema llama al smart contract y este "analiza, en función de su perfil, si el rango introducido es adecuado", mediante las condiciones y reglas establecidas en el paso que contiene un diagrama DMN y la tabla de decisión en cuestión. Si la decisión es [no], el sistema emite un mensaje al

usuario - un evento - y finaliza. Si es [si], la oferta – la transacción en curso - quedaría registrada en la blockchain.

PASO 3. Establecer los datos de prueba.

Este se establece a partir del diagrama de clases previamente establecido y que se ha mostrado en la Figura 56. Respecto al conjunto de datos (dataset) concreto que se utilizará para poder probar el Caso de Prueba en cuestión, es importante indicar que la elección de este dataset depende en gran medida del propósito y los requisitos del Caso de Uso específico, así como del contexto del sistema o aplicación que se está desarrollando. En este sentido, el equipo de trabajo tiene dos formas de obtener el dataset adecuado (esto es una decisión que debe tomar el equipo de trabajo):

- Generación sintética. Si el caso de uso es simple, es posible generar datos sintéticos o aleatorios, teniendo en cuenta el tipo de dato en cuestión.
- Recopilación de datos existentes (si es posible). Si el Caso de Uso implica datos del mundo real, puede ser necesario recopilar datos reales de diferentes fuentes, registros de eventos o, incluso, de la misma base de datos.

PASO 4. Obtener los modelos de Casos de Prueba

Llegados a este punto, se puede ejecutar la Transformación T2 – implementada también como un plugin de SofIA -, que permitirá generar automáticamente los Casos de Prueba. En las Figura siguiente se muestran los diagramas que, la extensión de SofIA, genera automáticamente en nuestro ejemplo. En concreto, se generan dos casos de prueba (ver Figura 60), donde se diferencia claramente los pasos a probar en el Smart Contract.

Uno de los casos de prueba sería para el camino en el que se acepta el rango salarial (BasicPath) y otro en el que se muestra el camino cuando no se acepta (Alternative), en los cuales se diferencia claramente la parte de pruebas asociadas al smart contract.

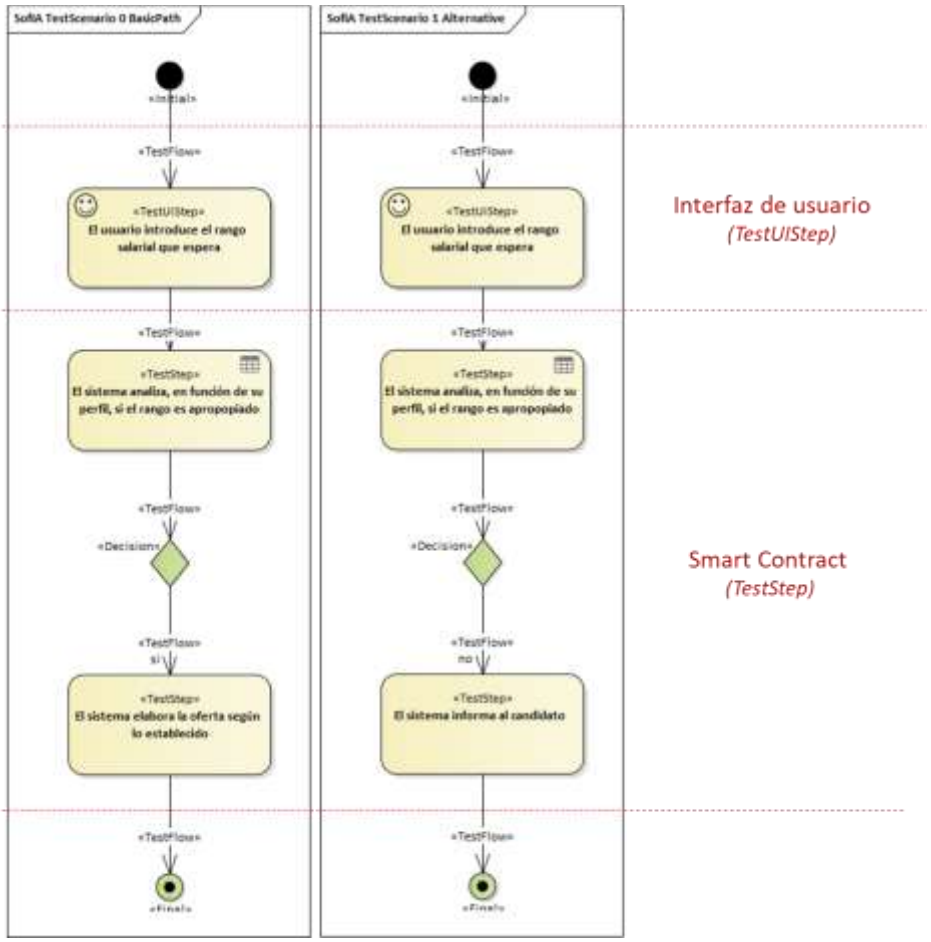


Figura 60. Modelos de Casos de Prueba generados
(Proyecto SmartAuditor)

Llegados a este punto, lo que sería necesario es que, con los datasets de prueba establecidos, el equipo de testing ejecute estas pruebas funcionales y valide los resultados obtenidos una vez esté implementado el código.

No obstante, desde las etapas iniciales del desarrollo de este proyecto, las pruebas se han tenido en cuenta, ya que se han ido obteniendo escenarios de Caso de Uso y los casos de prueba de forma sistemática, aplicando los principios del testing temprano o el early testing.

VI.1.4. Conclusiones del proyecto

En el ejemplo que se ha esbozado de SmartAuditor, solo se ve una mínima parte de la magnitud del proyecto, pero nos ha permitido visibilizar de forma simple y amigable cómo nuestra propuesta se ha llevado a cabo.

El proyecto SmartAuditor, es un proyecto complejo donde se abordaron unos 30 casos de uso con su correspondiente diagrama de Clases (15 clases). La aplicación de nuestra propuesta utilizaba tres smart contracts y generó un total de 124 Casos de Prueba. Estas pruebas fueron ejecutadas por el equipo de trabajo y, posteriormente, por el área usuaria, y se generaron independientemente del proceso de desarrollo, siguiendo así las pautas establecidas por la norma ISO 29119.

La aplicación de los trabajos de la presente Tesis Doctoral, en este proyecto basado en la tecnología blockchain, han supuesto una validación real del trabajo realizado, pero también nos permitió mejorar la calidad el proyecto resultante.

Gracias a la aplicación de nuestra propuesta, se pudo mejorar sustancialmente la fase de pruebas funcionales debido a las siguientes razones:

- Automatización de la generación de pruebas funcionales. No solo se han generado pruebas a partir de las especificaciones funcionales de la solución, sino también de las especificaciones de los smart contract. En concreto, se han podido generar automáticamente numerosos casos de prueba (124 en total), cubriendo diferentes escenarios,

situaciones y comportamientos tanto de la solución global como de los smart contract a implementar, pudiendo tener en cuenta las condiciones y reglas establecidas en los mismos, lo que ha llevado a un aumento significativo en la cantidad de pruebas funcionales.

- Cobertura amplia de escenarios. Detallar mejor los escenarios ha permitido una mejor comprensión y especificación del sistema y, sobre todo, de las especificaciones de los smart contract a través de sus modelos. Además, al haber podido traducir los modelos de smart contract en casos de pruebas funcionales, de forma sistemática, se ha logrado una cobertura más exhaustiva de los diferentes escenarios de Caso de Uso y situaciones específicas que pueden surgir en el software. Esto ha llevado a una mayor cantidad de pruebas funcionales que abarcan una gran variedad de casuística.

- Detección temprana de errores. Gracias a haber diseñado y analizado el sistema global, así como las especificidades de los smart contracts antes de la implementación. Se ha podido detectar errores o problemas potenciales en etapas tempranas del proceso de desarrollo, lo que ha permitido corregir problemas antes de que se conviertan en defectos reales. Esto ha conducido a poder hacer más pruebas durante las primeras fases del ciclo de vida del desarrollo.

- Reutilización de modelos. Partes de un modelo de smart contract ha podido ser reutilizado en otros modelos, lo que ha permitido ahorrar tiempo y esfuerzo en la creación de nueva casuística desde cero.

- Mayor rapidez en el proceso de pruebas. La automatización y la generación sistemática de pruebas funcionales a partir de modelos, ha permitido que el equipo de trabajo realizar pruebas funcionales de manera más rápida y de forma más eficiente que las pruebas manuales tradicionales. Esto ha permitido aumentar la frecuencia con la que se pueden ejecutar pruebas, lo que ha conducido a un mayor

número de pruebas funcionales realizadas en el mismo período de tiempo.

En resumen, la aplicación de los resultados obtenidos en la presente Tesis en este proyecto ha permitido mejorar la realización de las pruebas funcionales en el proyecto SmartAuditor basado en blockchain. Sobre todo, ha permitido probar de forma exhaustiva las especificaciones de los smart contract, gracias a la generación automática de pruebas funcionales y de comportamiento de los mismos. En definitiva, ha permitido disponer de una mayor cobertura de escenarios, realizar la detección temprana de errores y posibles defectos en los smart contracts, reutilizar modelos y, todo ello, se ha traducido en un proceso de pruebas más efectivo y eficiente, con una mayor cantidad de pruebas funcionales y en un software de mayor calidad.

VI.2. Otros trabajos

La presente Tesis, como se ha venido comentando, tiene un marcado carácter de aplicación en el entorno industrial, de hecho, opta a dicha mención. Por ello, y gracias al entorno en el que ha sido desarrollado y, especialmente a disponerse de una herramienta que sistematiza y automatiza gran parte del proceso, nuestra propuesta ha podido ser aplicada con éxito en otro proyecto empresarial ya finalizado.

Este otro proyecto muestra la capacidad que tiene el trabajo realizado, los beneficios que aporta y la aplicabilidad e interés que despiertan en el tejido empresarial. A continuación, se comenta esta otra validación que no se muestra en detalle, pero que es necesario referenciar pues también ha sido un contexto real de aplicación y validación que avalan los resultados del trabajo.

En concreto, hablamos del proyecto Trabis - TRAzabilidad de Muestras BIológicaS de reproducción humana asistida (EXP 00130378 / IDI-20210090). Proyecto financiado por el Centro de Desarrollo Tecnológico e Industrial

(CDTI) del Ministerio de Ciencia e Innovación - Gobierno de España -, a través de las "subvenciones para proyectos de desarrollo tecnológico I+D+i en grandes empresas y Pymes". El proyecto fue adjudicado a la empresa G7Innovation Solutions, en colaboración con el grupo ES3. La responsable del proyecto es Dña. Mª José Escalona Cuaresma, directora del grupo ES3 y Catedrática de la Universidad de Sevilla.

El proyecto, con un presupuesto total superior a los 200.000,00 €, comenzó a mediados de 2020 y finalizó a primeros de 2022. Este proyecto se enmarca, donde más auge está teniendo la tecnología blockchain, la trazabilidad, así como en el testing del software desde etapas tempranas del desarrollo.

El objetivo general de este proyecto era el diseño y desarrollo de una solución tecnológica que realice una monitorización de la ejecución de procesos de reproducción humana asistida, de manera integrada con dispositivos físicos de laboratorio, al objeto de mejorar el control, salvaguarda y trazabilidad de muestras biológicas de paciente. Para alcanzar este objetivo, dadas sus características intrínsecas, se ha empleado como sustento la tecnología blockchain y los smart contracts.

El proceso de trabajo de la solución tecnológica, como se muestra en la Figura 61, permite realizar un emparejamiento e identificación unívoca y segura de cada paciente-muestra durante la manipulación de muestras, llevando un exhaustivo control del proceso. Además, el sistema aporta un registro inmutable, transparente y seguro de las evidencias, en tiempo real, de cada actuación llevada a cabo, proporcionando notificaciones y avisos - en tiempo real - al profesional sanitario que esté realizando la manipulación de muestras, al objeto de evitar los posibles errores humanos.

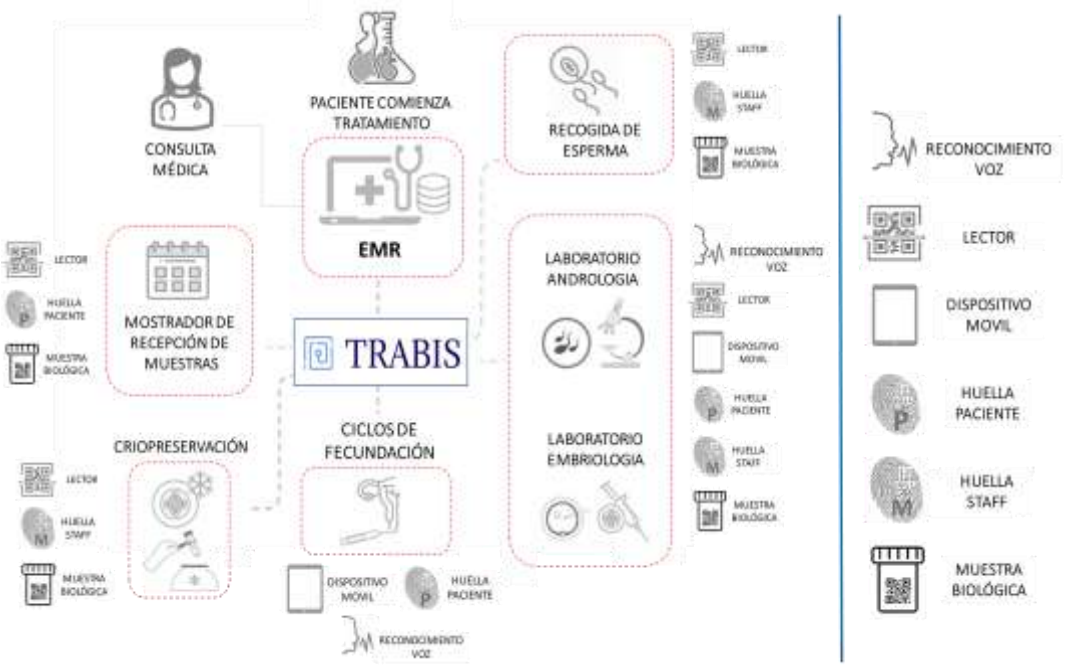


Figura 61. Procesos de trabajo global
(Proyecto TRABIS)

En el proyecto Trabis se aplicó con éxito también el proceso seguido en el proyecto SmartAuditor (ver Figura *50*). Su complejidad era alta, concretamente se definieron 16 casos de uso que, a su vez, generaron 36 escenarios de Casos de Uso.

VI.3. Resumen del Capítulo

La propuesta de esta Tesis Doctoral, como se ha venido comentando, pretendía ser llevada al mercado y convertirse en una propuesta que pudiese mejorar el entorno de las pruebas de proyectos reales en los que se aplique la tecnología blockchain.

En este Capítulo se ha esbozado cómo se aplicó nuestra propuesta en el contexto del proyecto SmartAuditor, proyecto adjudicado al grupo ES3 y financiado por la Consejería de Economía, Conocimiento, Empresas y

Universidad (Junta de Andalucía). Se ha mostrado de manera detallada, sobre un pequeño ejemplo del mismo, como aplicar todo lo expuesto en la Tesis. Además, se han aportado importantes conclusiones resultantes de la aplicación de nuestra propuesta en el proyecto. Adicionalmente, se ha mencionado otro proyecto más, Trabis (proyecto financiado por el CDTI), ejecutado también por el grupo ES3 en colaboración con la empresa G7Innovation Solutions, en el que también se ha aplicado este trabajo de Tesis.

Estos proyectos ya han finalizado y han resultado escenarios claves para validar nuestra propuesta. Sin embargo, estos proyectos no son los únicos en los que la propuesta se ha usado. Gracias a las decisiones tomadas en la elaboración de este trabajo de Tesis que han permitido incorporar nuestras herramientas al entorno de SofIA, son varios los proyectos reales en los que podemos aplicar nuestro trabajo.

Capítulo VII. Resultados, trabajo futuro y conclusiones

" The gratification comes in the doing, not in the results"
– James Dean –

En los Capítulos anteriores, se ha evidenciado que el testing temprano y su aplicación en el ciclo de vida de desarrollo de smart contracts permite minimizar los posibles errores de funcionamiento de estos programas informáticos.

Esta Tesis Doctoral, además, ha descrito una serie de metamodelos y transformaciones para resolver el problema identificado, desde una perspectiva de ingeniería basada en modelos. Como parte de este trabajo, se ha diseñado y desarrollado una herramienta o utilidad como marco teórico centrado en dichos metamodelos y reglas de transformación y se ha evaluado la misma, mediante verificaciones y validaciones en un contexto industrial.

En las secciones de este último Capítulo se realizará un resumen de los resultados obtenidos, se avanza en posibles líneas de trabajo futuras y se establecen las conclusiones de los aspectos más importantes de la presente Tesis Doctoral.

VII.1. Aportaciones de este trabajo de Tesis

Una vez descrito y documentado el trabajo realizado durante la realización de esta presente Tesis Doctoral y expuesto el marco de trabajo en el que se ha llevado a cabo, así como el caso práctico donde se ha validado nuestra propuesta, es momento para recapitular las principales aportaciones obtenidas como resultado de la misma.

Como se puede apreciar, cada una de las aportaciones, que se indican a continuación, hace referencia al objetivo que al comienzo del trabajo se planteó y evidencian que todos ellos han sido cubiertos.

- Se ha identificado el problema a resolver, se ha valorado la importancia del problema y se han identificado estudios disponibles en la literatura que han identificado o experimentado previamente este problema. En concreto, se ha abordado un estudio del estado del arte en el contexto del ciclo de vida de desarrollo de los smart contracts, al objeto de analizar si hay estudios primarios que aborden el paradigma de la ingeniería dirigida por modelos u otras buenas prácticas para analizar, diseñar e implementar smart contracts.

  Después de dar por finalizado este estudio, se vio de forma clara la necesidad de explorar la aplicación del paradigma de la ingeniería dirigida por modelos ya que éste podría facilitar los mecanismos necesarios para verificar y validar los smart contracts, ya que allanaba el camino para aplicar técnicas de prueba tempranas antes de que los smart contracts fueran desplegados en una red blockchain.

- Se han identificado los artefactos que dan solución al problema expuesto (metamodelo de smart contract y metamodelo de pruebas funcionales) para, posteriormente, plantear una estructura que permite modelar los smart contracts a partir de especificaciones funcionales. En concreto, se consideró que aplicar técnicas de

ingeniería software y técnicas de modelado y transformaciones, permitirían instanciar casos de prueba funcionales a partir de una especificación desarrollada mediante requisitos funcionales y de los acuerdos, condiciones y reglas de negocio establecidos entre las partes.

- Se han definido los mecanismos que permiten generar las pruebas funcionales, a partir de los artefactos propuestos, y de la manera más automática posible. En concreto, se ha detallado cómo se podría hacer la automatización de la generación de pruebas funcionales a partir de las especificaciones de estos requisitos, los acuerdos, las condiciones y las reglas de negocio.

- Se ha implementado una herramienta, integrada en el marco de trabajo de SofIA, que facilita la representación de la solución y artefactos propuestos y, a su vez, permite la generación automática de pruebas funcionales, de forma sistemática.

- Se ha evaluado la herramienta propuesta, mediante verificaciones y validaciones en el contexto industrial y se ha mostrado un ejemplo de dicha aplicación.

VII.2. Trabajos futuros

Una vez recapituladas las principales aportaciones realizadas con el desarrollo de este trabajo de Tesis, es importante poner en valor que dicho trabajo no es algo estanco, sino que sus resultados, en confluencia con los otros trabajos y/o proyectos del grupo ES3, están propiciando la apertura de nuevas ideas, áreas de mejora y líneas de trabajo para avanzar en esta temática. En concreto, como futuros trabajos, tendríamos las siguientes actuaciones:

- En el contexto de las pruebas de software, seguir mejorando las pautas y técnicas de trabajo propuestas con el objetivo de que éstas sean lo más generalistas posibles. Además, nuestra idea no es solo

mejorar los mecanismos actuales, sino incorporar, por un lado, la priorización de pruebas, no sólo su generación. Es decir, no solo se debe poder generar las pruebas funcionales a partir de las especificaciones de los requisitos, garantizando su correspondencia con los mismos, sino también priorizarlas para que se pudieran generar las pruebas según la priorización establecida. Por otro lado, incorporar nuevos elementos visuales para identificar de forma clara su funcionalidad. Por ejemplo, si es un evento indicar con distintos iconos si se comunica con otro smart contract, con un Oráculo o vuelve a la interfaz de usuario.

- Habilitar mecanismos análogos a los descritos en la presente Tesis, que permitan la generación automáticamente del código de los smart contracts o al menos los esqueletos de estos scripts, ya que la generación a partir de diagramas UML no es una funcionalidad estándar a día de hoy.

  En concreto, desde nuestro punto de vista y siguiendo unas pautas análogas, sería posible utilizar herramientas y enfoques específicos para lograr cierto grado de automatización en el proceso de generación de código. Es decir, mediante la ingeniería guiada por modelos se podría definir un modelo UML y, luego, mediante transformaciones se podría generar el código del smart contract en el lenguaje de programación deseado, lo que permitiría potenciar la reutilización de código ya probado.

  No obstante, habría que tener en cuenta que la generación automática de código a partir de UML, puede ser útil para la creación de esqueletos básicos de smart contracts o para acelerar el proceso inicial de desarrollo. Sin embargo, es posible que aún sea necesario personalizar y agregar lógica específica para adaptar el smart contract a los requisitos y la lógica empresarial en particular. Además, el desarrollo de los smart contracts es un proceso crítico y se deben

tener en cuenta aspectos como la seguridad, la eficiencia y las mejores prácticas de la plataforma blockchain a utilizar.

VII.3. Conclusiones

A lo largo de toda la presente Tesis, se ha presentado el trabajo realizado en los diferentes Capítulos que la componen, desde la definición del problema, pasando por los resultados obtenidos en la revisión sistemática de literatura, hasta la propuesta de solución, inicialmente desde un punto de vista más teórico, para llegar a un enfoque totalmente práctico con la herramienta de soporte y la manifestación de su utilidad mediante la exposición de los casos prácticos abordados.

Esta Tesis Doctoral está dentro del contexto de trabajo del grupo ES3, lo que ha influido notablemente en los enfoques establecidos durante la ejecución del presente trabajo. Además de las principales aportaciones realizadas por esta Tesis Doctoral, dentro de un determinado ámbito de trabajo, se han abierto un conjunto de ideas de trabajo futuro que complementarían los resultados aquí alcanzados.

Todo esto, ha permitido que desde enero de 2023, el grupo ES3 esté inmerso en un nuevo proyecto denominado Proyecto GRIP (Global Rights Management Platform) - *Desarrollo de una plataforma global de gestión de derechos de publicación basada en Deep Learning, Procesamiento del Lenguaje Natural y Blockchain* - , donde uno de los objetivos del proyecto es "realizar una gestión de derechos de autor innovadora que permita llegar a acuerdos de compraventa a través de procesos automatizados o semiautomatizados, donde la intervención de terceras partes no sea necesaria o sea mínima".

La empresa que lidera el proyecto, Lantia Publishing, conocedora de nuestros trabajos de investigación y, sobre todo, de los resultados obtenidos en la presente Tesis, ha seguido apostando por el uso de la solución propuesta como punto de partida para dar un paso más, promover no solo las pruebas funcionales de los smart contracts, sino la generación automática del código.

VII.4. Conclusions

Throughout this thesis, the work carried out has been presented in the different chapters that comprise it, from the definition of the problem, through the results obtained in the systematic review of the literature, to the proposed solution, initially from a more theoretical point of view, to reach a totally practical approach with the support tool and the demonstration of its usefulness through the presentation of the practical cases addressed.

This Doctoral Thesis is within the working context of the ES3 group, which has notably influenced the approaches established during the execution of the present work. In addition to the main contributions made by this Doctoral Thesis, within a given field of work, a set of ideas for future work have been opened that would complement the results achieved here.

All this, has allowed that since January 2023, the ES3 group is immersed in a new project called GRIP Project (Global Rights Management Platform) - Development of a global publishing rights management platform based on Deep Learning, Natural Language Processing and Blockchain - , where one of the objectives of the project is "to carry out an innovative copyright management that allows reaching sale and purchase agreements through automated or semi-automated processes, where the intervention of third parties is not necessary or is minimal".

The company leading the project, Lantia Publishing, aware of our research work and, above all, of the results obtained in this thesis, has continued to support the use of the proposed solution as a starting point to go a step further, promoting not only the functional testing of smart contracts, but also the automatic generation of the code.

Glosario de términos

A continuación, se recogen de forma abreviada y ordenada todos los términos usados en el presente documento o necesarios para la comprensión de éste, al objeto de tener una referencia rápida de los mismos.

A

Activo Digital. Bien transferible de manera electrónica, intangible, con valor de mercado. En el caso de blockchain, los activos digitales abarcan tanto a las criptomonedas como a los tokens (representaciones de valor basados en blockchain).

AI (Artificial Intelligence). Es la combinación de algoritmos planteados con el propósito de crear máquinas que presenten las mismas capacidades que el ser humano. Una tecnología que desde hace unos años está presente en nuestro día a día.

Algoritmo de consenso. Éstos se puede definir como el proceso mediante el cual se consigue un acuerdo sobre el valor de un dato en un sistema distribuido. Son diseñados para conseguir confianza en una red donde están involucrados múltiples nodos.

API (Application Programming Interface). Es una pieza de código que permite a diferentes aplicaciones comunicarse entre sí y compartir información y funcionalidad. Una API es un intermediario entre dos o más sistemas, que permite que una aplicación se comunique con otras y pida datos o acciones específicas.

Asset. En español se traduce como activo. En el mundo de las criptomonedas hace referencia a un tipo de token que puede representar de forma única lo que sea (físico o no) y se transfiere de forma digital.

Árbol de Merkle (Merkel tree). Nombre que recibe la estructura que relaciona todas las transacciones de un bloque y las agrupa entre pares con el objetivo de obtener un hash que actúe de identificador único (llamado Root Hash) para todas esas transacciones.

B

Bitcoin (en minúscula). Es la primera moneda digital que utilizó la tecnología blockchain, surgiendo con el lanzamiento del libro blanco (Whitepaper) de bitcoin en 2009 por Satoshi Nakamoto. Un bitcoin es divisible en 100 millones de trozos, denominados satoshis. Bitcoin utiliza el algoritmo de Prueba de trabajo (PoW) para alcanzar un consenso en la red.

Hoy en día, Bitcoin se está utilizando para pagos de igual a igual en todo el mundo. Sin embargo, más que eso, está liderando el camino hacia un futuro en el que la tecnología financiera es confiada, segura y resistente a la censura.

BITCOIN (en mayúsculas). Protocolo y red de pagos entre usuarios abierta y libre, no propiedad de ninguna empresa ni gobierno, que gestiona un libro de contabilidad descentralizado llamado blockchain a través de matemáticas avanzadas (criptografía).

Blockchain. Es el primer tipo de red distribuida P2P, basada en criptografía en la cual la información se almacena en un conjunto de bloques entrelazados entre sí. Permite la validación de información y el intercambio de valor entre pares sin autoridad central emisora, ni administrador central.

- **Blockchain con permisos** (Permissioned Ledger). Es una blockchain privada donde sus nodos deben ser autorizados previamente por una entidad central. Las transacciones incorporadas al libro mayor, se realizada una prueba de consenso limitado y se realiza por participantes de confianza, siendo más sencillos de mantener y más rápidas que las redes de acceso libre.

- **Blockchain sin permisos** (permissionless blockchains). En una blockchain donde no existen restricciones de acceso, a la hora de procesar transacciones y crear bloques.

- **Blockchain privada** (Private Ledger). Blockchain que tiene acceso restringido y todos los nodos se conocen entre sí, ya que solo están abiertas a un consorcio o grupo de personas u organizaciones que ha decidido compartir el libro mayor entre ellos

- **Blockchain pública** (Public Ledger). Blockchain públicas donde cualquier persona puede formar parte de ella y nadie la controla. Son resistentes a la censura, ya que se impide que ninguna entidad central puedan evitar que una transacción ocurra.

Bloque. Elemento fundamental de la blockchain que crean los mineros y que permite vincular las transacciones realizadas en una red. Los bloques se crean en intervalos de tiempo y vinculan las transacciones nuevas con las ya existentes en la blockchain. Si podemos decir que la blockchain es como un libro contable digital, cada bloque sería cada una de las páginas de ese libro mayor.

- **Bloque Génesis**. El primer bloque que se generó en la red Bitcoin. Fue minado por Satoshi Nakamoto en enero de 2009. Este término también se utiliza para hablar del primer bloque de una blockchain.

- Bloque Huérfano. Son bloques que han sido resueltos correctamente, pero que por diferentes motivos el resto de los nodos de la red, no lo acepta por consenso.

BPMN (Business Process Model and Notation). Modelo y Notación de Procesos de Negocio, en español, es una notación gráfica estandarizada que permite el modelado de procesos de negocio, en un formato de flujo de trabajo (workflow).

C

Clave privada. Conjunto de caracteres de cualquier tipo que se generan de manera aleatoria y que tienen la función de contraseña única e intransferible. Se genera en base a un algoritmo matemático y siempre va acompañado de otro texto llamado clave pública. A diferencia de la clave pública, la clave privada no se debe revelar, dar o perder jamás.

Clave pública. Identificador personal basado en nuestra clave privada que podemos compartir sin miedo para que otras personas. En las criptomonedas se usan para generar las direcciones a las cuales otras personas podrán mandar criptomonedas.

Código abierto (Open source). Modelo de desarrollo de software basado en la colaboración abierta. Permite modificar el código fuente del programa sin restricciones de licencia. Los programadores pueden leer, modificar y redistribuir el código fuente de un programa de forma que éste evoluciona, se desarrolla y mejora. Los usuarios lo adaptan a sus necesidades, corrigen sus errores con un tiempo de espera menor a la aplicada en el desarrollo de software convencional o cerrado, dando como resultado la producción de un mejor software.

Código no confiable. En aplicaciones o plataformas de código abierto, cualquiera puede escribir o modificar el código, creando nuevo código. Si este nuevo código no ha sido testeado por nadie, se dice que es 'código no confiable'.

Confirmación. La consecución de resolver el hash de una transacción y añadirla a la blockchain. Se obtiene cuando la transacción realizada en la blockchain ha sido verificada por la red. Esto sucede mediante un proceso llamado minado que implica la resolución de un problema matemático. Una vez confirmada la transacción es irreversible para evitar el problema del doble gasto (duplicidad). Cuantas más confirmaciones tiene una transacción, más difícil es realizar un ataque de doble gasto.

Consenso. El consenso se obtiene cuando todos los participantes de la red coinciden en la validez de las transacciones, garantizando que todos los libros distribuidos son copias exactas de los mismos. El consenso, en blockchain, es precisamente la pieza clave que permite que todos los participantes en el mismo puedan confiar en la información que tiene depositada.

Contrato inteligente (Smart contract): Programa informático que corre sobre una blockchain de forma descentralizada. Son aplicaciones que se ejecutan exactamente como se programaron sin posibilidad de tiempo de inactividad, censura, fraude o interferencia de terceros. Funciona como una sentencia if-then (si-entonces) de cualquier otro programa de ordenador con la diferencia de que se realiza para interactuar con activos reales. Cuando se dispara una condición preprogramada, no sujeta a ningún tipo de valoración humana, el contrato inteligente ejecuta la cláusula contractual correspondiente. Pueden interactuar con otros contratos, tomar decisiones, almacenar datos y enviar criptomonedas o tokens. Existirán y serán ejecutables mientras exista toda la red, solo desaparecerán si fueron programados para autodestruirse.

Criptografía. Técnicas de cifrado o codificado destinadas a alterar las representaciones lingüísticas de ciertos mensajes con el fin de hacerlos ininteligibles a receptores no autorizados. Uno de algoritmos criptográficos recurrentes cuando se estudia el protocolo Bitcoin es SHA-256.

- **Criptografía asimétrica.** También denominada criptografía de clave pública se utiliza para enviar mensajes en base a un par de claves, una es la clave pública y la otra la clave privada. Ambas participan en el cifrado de la información trasferida, así como en la verificación de que la información original no ha sufrido alteraciones.

- **Criptografía simétrica.** Denominada criptografía de clave secreta o de una clave, que sirve para cifrar y descifrar el mensaje en el emisor y receptor, quienes se han puesto de acuerdo con la clave de usar para cifrar el mensaje enviado por el emisor y descifrarlo por parte del receptor.

Criptomoneda. Es un tipo de token criptográfico basado en la tecnología blockchain que actúa como activo monetario ya que permite la transferencia y reserva de valor. El valor de la criptomoneda depende de las condiciones del mercado, siendo una de las criptomonedas más populares el bitcoin (primera moneda digital, creada en 2009).

D

DApp. Son aplicaciones de carácter descentralizado que se ejecutan de manera autónoma, almacenando los datos dentro la blockchain y que opera según los parámetros establecidos.

DAO (Decentralized Autonomous Organization). La traducción en español es Organismo Autónomo Independiente. DAO es una organización descentralizada que aprovecha los smart contracts y otras características de la tecnología blockchain. Puede ejecutarse por sí solo y no tiene una administración jerárquica asociada. El código está pre escrito y procesa las diferentes características y funcionalidades que DAO tiene para ofrecer. Se opera con la ayuda de protocolos de consenso distribuidos. Ethereum y EOS admiten DAO y cualquiera puede definir una regla y construir un DAO que funcione de forma independiente.

Descentralización. Elemento fundamental en una criptomoneda y una de las propiedades básicas. Se fundamenta en que son los usuarios y no empresas privadas, estados u otros organismos centralizados, quienes validan las operaciones y determinan el valor de una moneda. Donde descentralizado sería el término utilizado para indicar que detrás de una criptomoneda y su red blockchain, no existe empresa privada, organismo, código de un tercero o Estado que la regule, controle o manipule.

Dificultad. Es el valor que indica el grado de complejidad del problema o acertijo que ha de resolverse en una Proof-of-Work, por ejemplo. Es variable y su valor depende de la potencia de la red y de la potencia del minero, ajustándose de manera automática según el estado de la red. En la red Bitcoin, la dificultad de minado se ajusta cada 2016 bloques con la finalidad de mantener el tiempo de verificación de cada bloque en 10 minutos.

Dirección. Las direcciones (Cryptocurrency addresses) se utilizan para recibir y enviar transacciones en la red. La dirección es una cadena de caracteres alfanuméricos, pero también se puede representar como un código QR escaneable.

- **Dirección Bitcoin.** Es un identificador que contiene entre 27 y 34 caracteres alfanuméricos y normalmente empieza por 1 o 3. Se generan de forma sencilla mediante un tipo de programa llamado wallet o monedero. Bitcoin es un sistema basado en criptografía asimétrica, por tanto, cuando se genera una dirección Bitcoin, se generan dos claves: pública y privada. Una dirección Bitcoin es simplemente la clave pública, la que usas para recibir el dinero y mostrar lo que sería el "número de cuenta" Bitcoin.

- **Dirección de cambio (Change Address).** Son direcciones implementadas en las wallet de Bitcoin que permiten que se devuelve al usuario el 'cambio'. Este cambio es similar al que recibimos cuando vamos a comprar y se nos devuelve la diferencia entre la cantidad dada y lo cobrado por la tienda.

DMN (Decision Model and Notation). Modelo de Decisión y Anotación, en español, es un estándar que se aplica específicamente para describir, crear y visualizar la toma de decisiones en una organización con el objetivo de hacerlas comprensibles para todos.

Doble gasto. Operación fraudulenta propia del dinero de naturaleza digital que se basa en usar dos veces las mismas monedas para comprar o pagar algo. Este tipo de falsificación es uno de los principales problemas que Bitcoin resuelve.

E

ERC-20 / ERC-223 / ERC-721 / ERC-777, son las denominaciones de diferentes estándares para la creación de tokens criptográficos basados en la red Ethereum

Escrow. Son contratos de depósitos de garantía donde el dinero queda en fase de reserva mediante un tercero que garantiza el cumplimiento de las partes involucradas.

Estereotipos. Son el mecanismo de extensibilidad incorporado más utilizado dentro de UML. Este representa una distinción de uso. Puede ser aplicado a cualquier elemento de modelado, incluyendo clases, paquetes, relaciones de herencia, etc. Por ejemplo, una clase con estereotipo 'actor' es una clase usada como un agente externo en el modelado de negocio. Una clase patrón es modelada como una clase con estereotipo parametrizado, lo que significa que puede contener parámetros.

Ether. Unidad de cuenta o token de Ethereum. Es un elemento necesario, un combustible, para operar Ethereum. Es una forma de pago realizada por los clientes de la plataforma a las máquinas que ejecutan las operaciones solicitadas. Es el incentivo que asegura que los desarrolladores escriban aplicaciones de calidad, y que la red se mantenga saludable ya que las personas reciben una compensación por sus recursos aportados.

Ethereum. Es una tecnología blockchain de código abierto y permite a los desarrolladores utilizar dicha tecnología para crear aplicaciones descentralizadas que también se conocen como DApps. La diferencia técnica entre bitcoin y Ethereum es significativa, ya que Ethereum permite a los desarrolladores alojar nuevas aplicaciones en su blockchain con la ayuda de smart contracts y DApps. La automatización también es una parte importante de la red Ethereum. Ethereum utiliza el algoritmo de consenso de Prueba de trabajo (PoW), sin embargo, ya está planteando el uso de la Prueba de participación (PoS). Cuando comparamos Ethereum con Bitcoin, cae bajo la segunda generación de blockchains, es decir, utiliza smart contracts para automatizar tareas.

Ethereum Virtual Machine (EVM). Máquina virtual Turing completa que tiene como misión ejecutar el código de la EVM. Los nodos de Ethereum operan dentro de la EVM con la finalidad de mantener el consenso de la blockchain.

Explorador de bloques (Block Explorer). Sitio web donde puedes ver la información y el estado de las transacciones de una red blockchain pública, y por tanto verificar el estado de una transacción realizada con una criptomoneda determinada.

F

Fee. Comisión que se cobra cuando se realiza cualquier transacción dentro de una blockchain

Fintech o tecnología financiera es una industria financiera que aplica nuevas tecnologías a actividades financieras y de inversión.

Firma digital. Código generado mediante la encriptación de clave pública y que se adjunta a un documento electrónico para verificar su contenido y la identidad del remitente.

Fork. Un fork crea una versión alternativa de la blockchain, resultando en dos cadenas bifurcadas operando en la red de manera simultánea.

- Hard fork. Tipo de fork que valida transacciones invalidadas previamente y viceversa. Este tipo de fork requiere que todos los nodos y usuarios se actualicen a la última versión del protocolo de software.

- Soft fork. Actualización menor del código del software de una red blockchain que es compatible con las versiones anteriores y no provoca que la red se bifurque dando lugar a lo que se denominaría hard fork.

Full node (nodo completo). Son todos los nodos que verifican de manera completa todas las reglas de una criptomoneda. En Bitcoin, el software para nodos completos se denomina Bitcoin Core.

G

GO. Es un lenguaje de programación, desarrollado por Google, concurrente y compilado, imperativo, estructurado, orientado a objetos —de una manera bastante especial— y con recolector de basura, inspirado en la sintaxis de C.

Gas. Con el fin de evitar bucles infinitos accidentales, hostiles, u otro desperdicio computacional en el código, cada transacción es obligada a establecer un límite al número de pasos computacionales de ejecución de código que ella puede utilizar. La unidad fundamental de computación es "gas". Por lo general, un paso computacional cuesta 1 gas, pero algunas operaciones cuestan cantidades más altas de gas porque son más costosas computacionalmente, o porque aumentan la cantidad de datos que deben ser almacenados como parte del estado. También hay una tarifa de 5 gas por cada byte en los datos de transacción. La intención del sistema de comisiones es obligar a un atacante a pagar proporcionalmente por cada recurso que consume, incluyendo computación, ancho de banda y almacenamiento. Por lo tanto, cualquier operación que conduzca a la red a consumir una mayor cantidad de cualquiera de estos recursos debe tener una comisión de gas más o menos proporcional al incremento.

- Gas limit, de una transacción, indica el valor máximo de Gas que una transacción puede llegar a consumir para ser válida. Cuando nos referimos al límite de Gas de un bloque hace referencia a la cantidad máxima de Gas que puede ser usado entre todas las transacciones que se añadan a ese bloque.

- Gas used, de una transacción, indica el Gas que ha necesitado la transacción para que se llevase a cabo. En el caso de referirnos al Gas usado de un bloque hace referencia al Gas usado por todas las transacciones que están añadidas a ese bloque.

- Gas Price, en una transacción, señala el precio de cada unidad de Gas (en Gwei, una submedida del Ether). La comisión de una transacción sale de multiplicar la cantidad de Gas usado por el precio del Gas. Cuando sumamos las comisiones de todas las transacciones procesadas en un bloque y añadimos la recompensa que se genera por defecto al validar un bloque, obtenemos el valor de 'Block reward' o Recompensa del bloque. Esta recompensa es obtenida por el minero o grupo de mineros que validaron el bloque.

H

Hash. Es el resultado de una operación criptográfica que genera identificadores únicos e irrepetibles a partir de una información dada. En concreto, es un algoritmo matemático que transforma cualquier bloque arbitrario de datos en una nueva serie de caracteres con una longitud fija, es marcado temporalmente para conocer la fecha y hora de cuando se procesó dicho bloque y contiene información sobre las transacciones realizadas. Es más, nos permite saber si dicha cadena original ha sido alterada.

Hyperledger. Es una colaboración de código abierto de blockchain para avanzar en esta tecnología y facilitar el desarrollo del marco de blockchain de registro de clase empresarial. Tanto los gigantes tecnológicos como las nuevas empresas están participando en este ecosistema para garantizar que blockchain tenga un marco de nivel empresarial con el que trabajar. Para asegurarse de que no pierdan el enfoque, no lanzarán ninguna criptomoneda adjunta al proyecto.

I

ICO (Initial Coin Offering). Traducido al español como oferta inicial de moneda, es como se denomina a la creación de un token como medio de financiación de un proyecto basado en red blockchain en su fase de desarrollo.

Identidad Digital. Identidad adoptada online o en un sistema de intercambio digital en el ciberespacio por una persona física, organización, o dispositivo electrónico.

I+D+i (Investigación, desarrollo e innovación). Es un concepto de aparición reciente en el contexto de los estudios de ciencia, tecnología y sociedad; como superación del anterior concepto de investigación y desarrollo.

J

Java. Es un lenguaje de programación de propósito general, concurrente, orientado a objetos, que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo.

JavaScript. Lenguaje de programación para hacer páginas web interactivas. Desde actualizar fuentes de redes sociales a mostrar animaciones y mapas interactivos, las funciones de JavaScript pueden mejorar la experiencia del usuario de un sitio web.

L

Libro mayor (Ledger). Libro mayor. Registro contable. Archivo o base de datos de registros inmutables.

- Libro mayor autorizado (Permissioned ledger). Existen dos tipos de ledgers con permiso: públicos y privados. Son registros o libros con uno o muchos propietarios, donde un número limitado de participantes tienen el poder de aprobar los datos nuevos que se van añadiendo.
    - En el caso de los permissioned private ledgers (ledgers privados autorizados), sólo las entidades autorizadas pueden leer el contenido del ledger y escribir en el ledger; por ejemplo, Corda de R3. Los ledgers privados autorizados pueden tener uno o muchos propietarios. Cuando se agrega un nuevo registro, se comprueba la integridad del libro mayor mediante un proceso de consenso limitado. Esto es llevado a cabo por agentes de confianza como departamentos gubernamentales o bancos. Este proceso hace que la entrada y verificación de datos sea más rápida y eficiente cuando se compara con el proceso de consenso de libros libres de permisos. Además, el uso de firmas digitales por nodos en la cadena también crea conjuntos de datos altamente verificables.
    - En los permissioned public ledgers (ledgers públicos autorizados), sólo las entidades autorizadas pueden escribir en el ledger, pero cualquiera puede ver el contenido del libro mayor, por ejemplo, Ripple. Un ledger autorizado puede tener algunos aspectos 'sin permiso' en circunstancias donde las entidades 'no autorizadas' pueden tener acceso restringido para ver conjuntos de datos parciales. Sin embargo, invariablemente no tendrán derechos de edición en ese blockchain
- Libro mayor sin permisos (Permissionless ledgers). Es una DLT sin propietario único, como la utilizada en Bitcoin. Estos ledgers descentralizados públicos son accesibles para todos los usuarios de Internet. Esto permite que cualquier persona aporte datos y que todos tengan exactamente la misma copia del registro, de tal manera que nadie puede impedir que se agreguen transacciones.
- Libro mayor distribuido (Distributed ledger). Registro distribuido en múltiples sitios, países o instituciones. Los registros se almacenan uno tras otro en un libro mayor continuo. La consulta es universal pero los privilegios de los participantes para modificarla varían en función de si se trata de una blockchain pública o privada.

Licencia MIT. Una licencia gratuita que permite que cualquiera obtenga una copia de este software y archivos de documentación asociados. Permite tratar el software sin restricciones, incluyendo derechos de uso, copia, modificación y fusión, publicación, distribución, sublicenciar y/o vender copias del software.

Locktime. Es el tiempo que una transacción de Bitcoin debe esperar para que un minero pueda agregar la transacción a su hash de la raíz de Merkle para conformar el próximo bloque que formará parte de la blockchain.

M

Master Private Key. Este tipo de claves privadas se generan en las wallet jerárquicas deterministas y se basa en datos derivados de la semilla raíz.

Master Public Key. Permite crear tantas direcciones públicas como se quiera de una wallet de Bitcoin en base a una Master Private Key, la cual implementa un compromiso que impide a un atacante gastar lo almacenado en esta wallet. Se suele usar para habilitar el almacenamiento y gasto fuera de línea o lo que es lo mismo, las operaciones se realizan con un equipo no conectado a la red y el resultado se transporta en un USB a un equipo conectado a la red. Es el sistema que utilizan las wallet físicas o de hardware.

Masternode. Es un tipo de nodo que se encargan de procesar las transacciones de la blockchain y reciben una recompensa cuando se mina un bloque. Se caracterizan porque para correr un masternode en una red debes tener congeladas una importante cantidad de criptomonedas de esa red.

Mecanismos de consenso. Son protocolos que garantizan que todos los nodos de una blockchain estén sincronizados entre sí y que establezcan un acuerdo sobre qué transacciones se pueden realizar y agregar a la cadena de bloques.

Mempool. Es la abreviatura de Memory Pool. Conjunto de transacciones no confirmadas en una blockchain. Cada vez que se realiza una transacción entra directamente al mempool, después los mineros van cogiendo grupos de transacciones para construir los bloques.

Minería. La minería de criptomonedas es el proceso de resolución de un problema matemático para dar seguridad a una red distribuida. Minar está incentivado económicamente: el minero recibe nuevas criptomonedas recién emitidas por el programa además de las comisiones de las transacciones que añada al bloque (**Minero.** Cualquier ordenador que intente solucionar el reto matemático de una red blockchain basada en Proof Of Work).

Mineros. Los nodos mineros utilizan algoritmos matemáticos para convertir la información de un bloque en un código alfanumérico o hash que enlace al hash del bloque anterior y encadenar los bloques entre sí.

Por cada bloque añadido a la cadena, el nodo minero percibe una remuneración en criptomonedas o una participación en el negocio objeto de la transacción. La participación de los nodos mineros sigue las reglas definidas por cada plataforma relativas al mecanismo de consenso, el cual determina en gran medida la seguridad, fiabilidad, velocidad y coste computacional y energético del proceso.

MOF (Meta-Object Facility) es un estándar que provee de un marco de trabajo de gestión de metadatos y de un conjunto de servicios para permitir el desarrollo de la interoperabilidad de sistemas dirigidos por modelos y metadatos. En la arquitectura de modelado, MOF estaría situado en el nivel de meta-metamodelo (M3) y, con MOF, es posible desarrollar nuevos lenguajes de modelado.

- MOFM2T (MOF Model to Text Transformation Language) es una aproximación basada en plantillas que permite definir transformaciones desde un modelo a código ejecutable.

- MOFScript. Framework que permite desarrollar transformaciones entre modelos y textos (M2T), soportando la generación de código ejecutable desde modelos gráficos.

Moneda fíat: Dinero de curso legal. Únicamente los bancos centrales tienen el poder de emitir dinero fiat, pero los bancos comerciales lo pueden crear a través de préstamos.

Multifirma (multisignature). Las direcciones multifirma añaden una capa extra de seguridad mediante la utilización de más de una clave para autorizar una transacción.

N

Nodo. Ordenador conectado a la red blockchain que utiliza un cliente que lleva a cabo la tarea de validar transacciones. Cada nodo mantiene una copia íntegra y actualizada del libro mayor de la blockchain.

Nonce. Significa 'umber that only used once' (número que solo se usa una vez) y tiene una importancia vital junto al hash en la verificación de los datos de la red blockchain de Bitcoin. El "Nonce" de un bloque Bitcoin es un campo de 32 bits (4 bytes) cuyo valor se establece de modo que el hash del bloque contenga una ristra de ceros. El resto de los campos no se pueden cambiar, ya que tienen un significado definido. Cualquier cambio en el bloque de datos (como pueden ser el Nonce) hará que el hash del bloque sea completamente diferente. Dado que es imposible predecir qué combinación de bits se traducirá en el hash correcto, se intentan muchos Nonce diferentes y el hash se vuelve a calcular para cada valor hasta que se encuentre un hash que contenga el número necesario de bits a cero. Como este cálculo iterativo requiere tiempo y recursos, la presentación del bloque con el valor Nonce correcta constituye una prueba de trabajo.

O

OCL (Object Constraint Language). Es un lenguaje formal usado para describir restricciones sobre modelos UML y realizar consultas sobre los objetos descritos en ellos.

OMG (Object Management Group). Es un consorcio dedicado al cuidado y el establecimiento de diversos estándares de tecnologías orientadas a objetos, tales como UML, XMI y BPMN.

Oráculo. Los oráculos son sistemas automatizados que obtienen información de diferentes medios o usuarios para introducirlos en la red blockchain que se utilizara en los smart contracts. Es algo aportado por la red Ethereum.

P

Participante (stakeholder). Actor que tiene acceso al libro mayor y puede consultar y añadir registros.

Perfil UML. Un perfil en el lenguaje de modelado unificado (UML) proporciona un mecanismo de extensión genérico para personalizar modelos UML para dominios y plataformas particulares. Los mecanismos de extensión permiten refinar la semántica estándar de manera estrictamente aditiva, evitando que contradiga la semántica estándar

Pool (minería). Combinación de recursos de varios mineros para obtener una potencia de minado mayor y así conseguir mayores recompensas por la apertura de bloques. Los hay que son públicos y los hay que son privados.

Programación orientada a objetos (Object Oriented Programming, OOP). Es un modelo de programación informática que organiza el diseño de software en torno a datos y objetos, en lugar de funciones y lógica. Un objeto se puede definir como un conjunto de datos que tiene atributos y un código (funcionamiento) en forma de métodos.

Protocolo de consenso. Las criptomonedas acostumbran a utilizar el protocolo de consenso Proof of Wook (PoW) que utiliza Bitcoin o también Proof of Stake (PoS). Sin embargo, algunos proyectos como Ripple ha desarrollado su propio consenso y su nombre es Algoritmo de Consenso del Protocolo Ripple, o Ripple Protocol Consensus Algorithm (RPCA).

- **Proof-of-Stake (PoS).** En español se puede traducir por 'prueba de participación' y es un sistema de validación de las transacciones de una red basada en una serie de master-nodos que almacenan criptomonedas en una wallet o cartera.

- **Proof-of-Work (PoW).** En español 'prueba de trabajo' es un sistema de validación de las transacciones de una red mediante la resolución de operaciones matemáticas a través de equipos informáticos especializados.

Punto de Consenso. Un punto – bien en el tiempo o definido en términos de un número determinado o volumen de registros a añadir al libro mayor- en el que los pares logran un acuerdo en cuanto al estado del libro.

Python. Es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional.

R

Red descentralizada. Sistema basado en nodos donde la información se distribuye en un esquema árbol. Los nodos centrales distribuyen información hacia los nodos intermedios y éstos a su vez pueden decidir si distribuyen o no la información (la red centralizada es aquella que la información se distribuye desde un único punto).

Red distribuida. Red donde el poder computacional y la información almacenada están repartidos en nodos en lugar de en un servidor de datos centralizado.

Ricardiano (Contrato). Es un contrato digital que define los términos y condiciones de una interacción, entre dos o más partes, que está firmado y verificado criptográficamente. Es importante destacar que es legible tanto para los humanos como para las máquinas. (Grigg, 2004) presentó el triple contrato ricardiano de "prosa, parámetros y código".

Ripple. El proyecto de Ripple es una red de proveedores de pagos institucionales, como bancos y empresas, que utilizan las soluciones de la compañía Ripple, para ofrecer una experiencia de envío de dinero, a nivel mundial, sencilla y a bajo coste. La tecnología que utiliza Ripple para llevar a cabo sus funciones es DLT (Ledger Distribuido) en conjunto de un protocolo de consenso propio llamado Algoritmo de Consenso del Protocolo Ripple (RPCA). El ledger distribuido de Ripple utiliza su propio token, XRP, con el fin de salvaguardar todos los datos autorizados de las aplicaciones y operaciones realizadas. Mediante este mecanismo, se pueden utilizar los tokens nativos de la red, es decir la criptomoneda XRP, para procesar pagos de forma internacional, con bajas comisiones y de forma rápida.

RPA (Robotic Process Automation). La automatización robótica de procesos, en español, es una forma naciente de automatización de los procesos de negocio que replica las acciones de un ser humano interactuando con la interfaz de usuario de un sistema informático, liberándose de la dependencia de APIs de programación.

S

Satoshi. Un satoshi es la unidad mínima en la que se puede dividir un bitcoin y equivale a 0.00000001 Bitcoin (BTC) Debe su denominación al nombre del creador de Bitcoin, Satoshi Nakamoto.

Satoshi Nakamoto. Pseudónimo de la persona (o grupo de personas) que desarrolló y difundió la idea original de Bitcoin y la primera red blockchain del mundo. Se desconoce su nombre verdadero o si es una persona física, conjunto de personas o empresas.

Script. En informática, es una secuencia de comandos. Es un término informal que se usa para designar un lenguaje de programación que se utiliza para manipular, personalizar y automatizar las instalaciones de un sistema existente.

SDK. Son las siglas de kit de desarrollo de software que es generalmente un conjunto de herramientas de desarrollo de software que le permite al programador o desarrollador de software crear una aplicación informática para un sistema concreto, por ejemplo, ciertos paquetes de software, entornos de trabajo, plataformas de hardware, computadoras, videoconsolas, sistemas operativos, etcétera.

Sellado de tiempo confiable. Proceso de llevar, de manera segura, la cuenta del tiempo tanto de la creación como de la modificación de un documento electrónico. Nadie puede cambiarlo una vez que ha sido guardado.

SHA-256. Es el algoritmo criptográfico que se usa en la red Bitcoin para el minado de esta criptomoneda y la creación de sus direcciones. SHA son las siglas de 'Secure Hash Algorithm', concepto desarrollado por la Agencia de Seguridad Nacional (NSA) de EE.UU

Solidity. Es un lenguaje de programación orientado a objetos utilizado, principalmente en Ethereum, para la programación de los smart contracts.

T

Tecnología de Contabilidad Distribuida (DLT, acrónimo de Distributed Ledger Technology). Es un término que engloba la categoría completa de redes descentralizadas o sistemas de consenso distribuido que existen. La categoría de 'redes DLT' comparten la característica de que no necesitan una base de datos ni una entidad centrales de toma de decisión. La primera DLT funcional y operativa fue la red blockchain de Bitcoin, la cual lleva funcionando sin pausa desde 2009.

Testing Temprano (o pruebas tempranas). Es un conjunto de procesos, herramientas, técnicas, etc. que persiguen incrementar la calidad del software mediante la detección eficaz de errores en fases tempranas del ciclo de vida de su desarrollo, disminuyendo asimismo el impacto que estos errores tendrían sobre el desarrollo si se detectaran con posterioridad. Se ha observado que mientras más avanza el ciclo de vida de desarrollo de un software, más se tarda en identificar los errores en el sistema y mayor es el coste asociado para subsanarlos. Es evidente, que las pruebas de software deben comenzar lo más pronto posible, para detectar y evitar todos los defectos posibles en fases tempranas del desarrollo.

Testnet. Es una blockchain simulada, utilizada por los desarrolladores de la comunidad Ethereum, para testear cualquier tipo de código.

Timestamp. Es una marca de tiempo que se calcula según diferentes parámetros y contribuye a la verificación de información en la red.

Token. En el mundo de las criptomonedas un token es la representación digital del valor de un activo (físico o no). Existen una serie de estándares para crearlos y actualmente la red Ethereum es la que alberga mayor porcentaje de los tokens existentes. La tokenización de activos es la forma en que cualquier activo del mundo real, tangible o intangible, se digitaliza y luego se divide en partes más pequeñas que toman la forma de tokens.

Transacción. Son las operaciones que se realizan para agregar información a una blockchain. Su contenido puede ser muy variado y generalmente depende del tipo de red que se esté operando. Por ejemplo, hay plataformas blockchain que por medio de transacciones permiten subir archivos digitales al registro.

- Transaction block. Transacciones en la red agrupadas en un bloque que puede ser hasheado y añadido a la blockchain.

- Transaction fee. Tarifa de transacción. Estas tarifas se suman a la recompensa que obtiene el minero al minar un bloque.

Turing completo. Capacidad de una máquina de realizar cálculos que cualquier otra computadora programable es capaz de realizar. Un ejemplo es la EVM de Ethereum.

U

UML. Lenguaje gráfico, adoptado como estándar por el OMG, que sirve para especificar, visualizar, construir y documentar los artefactos de los sistemas software, así como para el modelado del negocio y otros sistemas no software. Está situado en el nivel 2 (M2) – Metamodelo - de la arquitectura de modelado y dispone de un mecanismo de extensión mediante perfiles, que permite crear lenguajes basados en él para modelar dominios concretos.

V

Vyper. Lenguaje de programación basado en Python y orientado a smart contracts que trabaja sobre la Máquina Virtual Ethereum (EVM). Basado en los principios de seguridad, simplicidad, auditabilidad, Vyper busca a convertirse en una forma sencilla, natural y segura de escribir smart contracts para Ethereum.

W

Wallet (cartera o monedero). Software que almacena las claves privadas que se necesitan para acceder a las criptomonedas registradas en una dirección o clave pública para gastarlos. Hay varios tipos de monederos dependiendo de la forma en que se almacena la clave privada. Algunos utilizan las casas de cambio como monederos online (Coinbase, Blockchain.info, Kraken, etc.) y otros utilizan monederos físicos (Trezor, Ledger Nano, papel), siendo estos últimos los más seguros. Normalmente incluye un cliente tipo web o App que permite consultar y crear transacciones en la blockchain específica para la que se ha diseñado. Incluso existen wallets físicas denominadas cold wallets.

Por tanto, el término wallet hace referencia a una cartera, billetera o monedero virtual en el que podemos gestionar nuestros activos digitales y es un software (o hardware) diseñado para almacenar y gestionar las claves públicas y claves privadas.

- Wallet determinista jerárquica (HD Wallet). Son wallets con una jerarquía determinista que se pueden compartir de manera parcial o total entre diferentes sistemas, cada uno con o sin la capacidad de gastar monedas. Este tipo de wallets no generan su clave pública a raíz de su clave privada, sino mediante un algoritmo matemático de curva elíptica.

- Wallet Física (Hardware Wallet). Están dentro de la categoría Cold Wallet y son normalmente unidades de formato USB que se conectan al ordenador y permiten almacenar nuestras criptomonedas de manera segura y las claves privadas. Suelen ofrecer la opción de añadir un PIN para desbloquear la unidad.

- Wallet Móvil (Smartphone/Tablet). Aplicaciones para smartphone de Android, iOS u otro sistema operativo para terminales portátiles que permite almacenar y gestionar nuestras criptomonedas, así como hacer pagos de manera rápida y sencilla.

- Wallet Online. Son wallets directamente en páginas web donde se nos ofrece una clave pública y la clave privada se almacena en la web de la compañía o de la organización. Un ejemplo serían las que nos ofrecen las diferentes exchange. Hay proyectos que también ofrecen la posibilidad de disponer de una wallet online.

- Wallet en Papel (Paper Wallet). Proceso por el cual está la clave privada de una dirección Bitcoin (o de cualquier otra criptomoneda) en un papel. De esta forma, puedes mandar fondos a la dirección asociada, sin correr el riesgo de que la clave privada esté en un programa con conexión a Internet. A su vez, este proceso también es un Cold Wallet (Monedero frío) o Cold Storage (Almacenamiento en frío). Cuando se quiere recuperar el control de los fondos se debe introducir dicha clave en un cliente (de la criptomoneda que corresponda) con conexión a Internet.

- Wallet PC o de escritorio. Son programas informáticos especiales pensados para almacenar nuestras criptomonedas y que nos ofrecen una clave pública y se protegen mediante una clave privada. Tienen la característica de ser muy seguras y permiten ver todo el historial de transacciones. Donde Light Wallet son carteras para Bitcoin u otras criptomonedas que se instalan en nuestro equipo y que ocupan menos espacio en nuestro disco duro y permiten una sincronización más rápida, ya que no es necesario descargar la información completa de la blockchain para usarla adecuadamente.

Whisper. Protocolo de comunicación para las DApps desplegadas sobre la blockchain de Ethereum.

Whitepaper. Documento técnico que describe las principales características o propiedades de un proyecto basado en tecnología blockchain y su correspondiente criptomoneda.

X

XMI (XML Metadata Interchange) es otro estándar OMG que permite el intercambio de modelos definidos mediante MOF, utilizando para ello una representación basada en XML.

XML. Siglas en inglés de eXtensible Markup Language, traducido como 'Lenguaje de Marcado Extensible' o 'Lenguaje de Marcas Extensible', es un metalenguaje que permite definir lenguajes de marcas desarrollado por el World Wide Web Consortium utilizado para almacenar datos en forma legible.

Referencias

Abrahäo S. et al. (2022). Calidad y sostenibilidad de sistemas de información en la práctica. Editorial RA-MA. Available online at: https://www.ra-ma.es/libro/calidad-y-sostenibilidad-de-sistemas-de-informacion-en-la-practica_136633/

Alharby, M., Aldweesh, A., and van Moorsel, A. (2018). blockchain-based smart contracts: A systematic mapping study of academic research (2018). In 2018 International Conference on Cloud Computing, Big Data and blockchain (ICCBB) (pp. 1-6). IEEE.

Alharby, M., and Van Moorsel, A. (2017). Blockchain-based smart contracts: A systematic mapping study. arXiv preprint arXiv:1710.06372.

Amsyar, I., Christopher, E., Dithi, A., Khan, A. N., and Maulana, S. (2020). The Challenge of Cryptocurrency in the Era of the Digital Revolution: A Review of Systematic Literature. Aptisi Transactions on Technopreneurship (ATT), 2(2), 153-159.

Angelis, J., and Da Silva, E. R. (2019). blockchain adoption: A value driver perspective. Business Horizons, 62(3), 307-314.

Argañaraz, Á. A., Mazzuchelli, A., Albanese, D., and López, M. D. L. Á. (2019). Blockchain: un nuevo desafío para la contabilidad y auditoría

Baker, P., Dai, Z. R., Grabowski, J., Schieferdecker, I., and Williams, C. (2007). Model-driven testing: Using the UML testing profile. Springer Science and Business Media.

Batubara, F. R., Ubacht, J., and Janssen, M. (2018). Challenges of blockchain technology adoption for e-government: a systematic literature review. In Proceedings of the 19th Annual International Conference on Digital Government Research: Governance in the Data Age (pp. 1-9).

Baudry, B., Nebut, C., and Le Traon, Y. (2007). Model-driven engineering for requirements analysis. In 11th IEEE international enterprise distributed object computing conference (EDOC 2007) (pp. 459-459). IEEE.

Bézivin, J. (2012). History and Context of MDE. Principles and Applications of Model Driven.

Bialy, M., Pantelic, V., Jaskolka, J., Schaap, A., Patcas, L., Lawford, M., and Wassyng, A. (2017). Software engineering for model-based development by domain experts. In Handbook of System Safety and Security (pp. 39-64). Syngress.

Blanco, R., Enríquez, J. G., Domínguez-Mayo, F. J., Escalona, M. J., and Tuya, J. (2018). Early integration testing for entity reconciliation in the context of heterogeneous data sources. IEEE Transactions on Reliability, 67(2), 538-556.

Buterin, V. (2014). A next-generation smart contract and decentralized application platform. white paper, 3(37).

Conrad, M., Fey, I., and Sadeghipour, S. (2005). Systematic model-based testing of embedded automotive software. Electronic Notes in Theoretical Computer Science, 111, 13-26.

Crosby, M., Pattanayak, P., Verma, S., and Kalyanaraman, V. (2016). blockchain technology: Beyond bitcoin. Applied Innovation, 2(6-10), 71.

Dhaiouir, S., and Assar, S. (2020). A systematic literature review of blockchain-enabled smart contracts: platforms, languages, consensus, applications and choice criteria. In International Conference on Research Challenges in Information Science (pp. 249-266). Springer, Cham.

Drave, I., Hillemacher, S., Greifenberg, T., Kriebel, S., Kusmenko, E., Markthaler, M., ..., and Wortmann, A. (2019). SMArDT modeling for automotive software testing. Software: Practice and Experience, 49(2), 301-328.

Dresch, A., Lacerda, D. P., and Antunes, J. A. V. (2015). Design science research. In Design science research (pp. 67-102). Springer, Cham.

Elallaoui, M., Nafil, K., Touahni, R., and Messoussi, R. (2016). Automated model driven testing using AndroMDA and UML2 testing profile in scrum process. Procedia Computer Science, 83, 221-228.

Enríquez, J. G., Blanco, R., Domínguez-Mayo, F. J., Tuya, J., and Escalona, M. J. (2016). Towards an MDE-based approach to test entity reconciliation applications. In Proceedings of the 7th International Workshop on Automating Test Case Design, Selection, and Evaluation (pp. 74-77).

Escalona, M. J., and Aragón, G. (2008). NDT. A model-driven approach for web requirements. IEEE Transactions on software engineering, 34(3), 377-390.

Escalona, M. J., Gutiérrez, J. J., Mejías, M., Aragón, G., Ramos, I., Torres, J., and Domínguez, F. J. (2011). An overview on test generation from functional requirements. Journal of Systems and Software, 84(8), 1379-1393.

Escalona, M.J., Jiménez Ramírez, A., González Enríquez, J., Sánchez-Gómez, N. , García-Borgoñón, L. and López, G. (2022). SofIA. Model-Driven Software TOOL.

Escalona, M. J., García-Borgoñon, L., García-García, J., López-Nicolás, G., and Koch, N. (2023). Choose your Preferred Life Cycle and SofIA will do the Rest. In International Conference on Web Engineering (pp. 359-362). Cham: Springer Nature Switzerland.

Ethereum community, The (2022). Available online at: https://ethereum.org/en/developers/docs/programming-languages/

Feyer, S., Siebert, S., Gipp, B., Aizawa, A., and Beel, J. (2017). Integration of the scientific recommender system Mr. DLib into the reference manager JabRef. In European Conference on Information Retrieval (pp. 770-774). Springer, Cham.

Forward, A., and Lethbridge, T. (2008). Problems and opportunities for model-centric versus code-centric software development: A survey of software professionals. International Workshop on Models in Software Engineering.

García-García, J. A., Escalona, M. J., Domínguez-Mayo, F. J., and Salido, A. (2014). NDT-suite: a methodological tool solution in the model-driven engineering paradigm. Journal of Software Engineering and Applications, 7(04), 206.

Götz, S., Fehn, A., Rohde, F., and Kühn, T. (2020). Model-driven software engineering for construction engineering: quo vadis?. J. Object Technol., 19(2), 2-1.

Grigg, I. (2004). The ricardian contract. In Proceedings. First IEEE International Workshop on Electronic Contracting, 2004. (pp. 25-31). Available online at: http://iang.org/papers/ricardian_contract.html

Gutiérrez, J. J., Nebut, C., Escalona, M. J., Mejías, M., and Ramos, I. M. (2008). Visualization of use cases through automatically generated activity diagrams. In International Conference on Model Driven Engineering Languages and Systems (pp. 83-96). Springer, Berlin, Heidelberg.

Gutiérrez, J. J., Escalona, M. J., and Mejías, M. (2015). A model-driven approach for functional test case generation. Journal of Systems and Software, 109, 214-228.

Hidayanto, A. N., and Prabowo, H. (2019). The latest adoption blockchain technology in supply chain management: A systematic literature review. ICIC Express Letters, 13(10), 913-920

Homès, B. (2013). Fundamentals of software testing. John Wiley and Sons.

Hsain, Y. A., Laaz, N., and Mbarki, S. (2021). Ethereum's smart contracts construction and development using model driven engineering technologies: a review. Procedia computer science, 184, 785-790.

Hu, K., Zhu, J., Ding, Y., Bai, X., and Huang, J. (2020). Smart Contract Engineering. Electronics, 9(12), 2042

Huang, F. (2017). Human Error Analysis in Software Engineering. Available online at: https://doi.org/10.5772/intechopen.68392.

Hyperledger Project (2022). Available online at: https://www.hyperledger.org

Jain, S., and Joshi, H. (2016). Impact of early testing on cost, reliability, and release time. In 2016 5th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO) (pp. 318-322). IEEE.

Johannesson, P., and Perjons, E. (2014). An introduction to design science (Vol. 10, pp. 978-3). Cham: Springer.

Jurgelaitis, M., and Butkienė, R. (2022). Solidity Code Generation From UML State Machines in Model-Driven Smart Contract Development. IEEE Access, 10, 33465-33481.

Karafiloski, E., and Anastas M. (2017). Blockchain solutions for big data challenges: A literature review. IEEE EUROCON 2017 -17th International Conference on Smart Technologies: 763-768.

Khalid, K.S., and Kunz, R. (2003). Systematic Reviews to Support Evidence-based, Medicine, Springer.

Kitchenham, B., and Brereton, P. (2013). A systematic review of systematic review process research in software engineering. Information and software technology, 55(12), 2049-2075.

Kitchenham, B., Pretorius, R., Budgen, D., Brereton, O. P., Turner, M., Niazi, M., and Linkman, S. (2010). Systematic literature reviews in software engineering–a tertiary study. Information and software technology, 52(8), 792-805.

Kitchenham, B.A., and Charters, S. (2007). Guidelines for Performing Systematic Literature Reviews in Software Engineering Technical Report, EBSE-2007-01, 2007.

Kuo, T. T., Zavaleta Rojas, H., and Ohno-Machado, L. (2019). Comparison of blockchain platforms: a systematic review and healthcare examples. Journal of the American Medical Informatics Association, 26(5), 462-478.

Ladleif, J. (2021). Enforceability aspects of smart contracts on blockchain networks (Doctoral dissertation, Universität Potsdam).

Ladleif, Jan and Weske, Mathias. (2019). A Unifying Model of Legal Smart Contracts. 323-337. 10.1007/978-3-030-33223-5_27.

Legerén-Molina, A. (2018). Los contratos inteligentes en España (La disciplina de los smart contracts) / Smart contracts in Spain; the regulation of smart contracts. Revista de Derecho civil, 5(2), 193-241.

Leka, E., Selimi, B., and Lamani, L. (2019). Systematic literature review of blockchain applications: Smart contracts. In 2019 International Conference on Information Technologies (InfoTech) (pp. 1-3). IEEE.

Lo, S. K., Liu, Y., Chia, S. Y., Xu, X., Lu, Q., Zhu, L., and Ning, H. (2019). Analysis of blockchain solutions for IoT: A systematic literature review. IEEE Access, 7, 58822-58835.

Lu, Q, Weber, I, and Staples, M. (2018). Why Model-Driven Engineering Fits the Needs for blockchain Application Development. IEEE blockchain Technical Briefs.

Lu, Q., Binh Tran, A., Weber, I., O'Connor, H., Rimba, P., Xu, X., ... and Jeffery, R. (2021). Integrated model-driven engineering of blockchain applications for business processes and asset management. Software: Practice and Experience, 51(5), 1059-1079.

Luu, L., Chu, D. H., Olickel, H., Saxena, P., and Hobor, A. (2016). Making smart contracts smarter. In Proceedings of the 2016 ACM SIGSAC conference on computer and communications security (pp. 254-269).

Macrinici, D., Cartofeanu, C., and Gao, S. (2018). Smart contract applications within blockchain technology: A systematic mapping study. Telematics and Informatics, 35(8), 2337-2354.

Merkle, R. C. (1988). A digital signature based on a conventional encryption function. In Advances in Cryptology—CRYPTO'87: Proceedings 7 (pp. 369-378). Springer Berlin Heidelberg.

Miraz, M. H., and Ali, M. (2020). Blockchain enabled smart contract-based applications: Deficiencies with the software development life cycle models. arXiv preprint arXiv:2001.10589.

Morales, G. (2021). La ingeniería informática como disciplina. Revista de Ingeniería Informática, vol 1. núm 1.

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. Decentralized Business Review, 21260.

Nanayakkara, S., Rodrigo, M. N. N., Perera, S., Weerasuriya, G. T., and Hijazi, A. A. (2021). A methodology for selection of a Blockchain platform to develop an enterprise system. Journal of Industrial Information Integration, 23, 100215

Ngai, E. W., Hu, Y., Wong, Y. H., Chen, Y., and Sun, X. (2011). The application of data mining techniques in financial fraud detection: A classification framework and an academic review of literature. Decision support systems, 50(3), 559-569.

Olea N. and Glenn Tjon G. (2019). Tecnologías disruptivas: Descubra el potencial de blockchain. Available online at: https://assets.kpmg/content/dam/kpmg/pa/delineandoestrategias/DE-Tecnologia-disruptiva-Descubra-el-potencial-de-blockchain-SECURED-updated.pdf

OMG (2006). Systems Modeling Language. Available online at: https://www.omg.org/spec/SysML

OMG (2011). UML 2.4.1. Available online at: https://www.omg.org/spec/UML/2.4.1/About-UML

OMG (2015). MDA–the architecture of choice for a changing world. Available online at: https://www.omg.org/mda/

OMG (2016). Meta Object Facility (MOF), Version 2.5.1. Available online at: https://www.omg.org/spec/MOF/

Pankov, K. N. (2020). Testing, verification and validation of distributed ledger systems. In 2020 Systems of Signals Generating and Processing in the Field of on-Board Communications (pp. 1-9). IEEE.

Petersen, K., Feldt, R., Mujtaba, S., and Mattsson, M. (2008). Systematic mapping studies in software engineering. In 12th International Conference on Evaluation and Assessment in Software Engineering (EASE) 12 (pp. 1-10).

Petticrew, M., and Roberts, H. (2005). Systematic Reviews in the Social Sciences: A Practical Guide, Blackwell Publishing.

Pohl, K., Hönninger, H., Achatz, R., and Broy, M. (Eds.). (2012). Model-based engineering of embedded systems: The SPES 2020 methodology (pp. 978-3642346132). Heidelberg: Springer.

Pranto, S., Jardim, L., Oliveira, T., and Ruivo, P. (2019). Literature review on blockchain with focus on supply chain.

Preukschat, A. (2017). blockchain: la revolución industrial de internet. Gestión 2000.

Ramos, A. L., Ferreira, J. V., and Barceló, J. (2011). Model-based systems engineering: An emerging approach for modern systems. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 42(1), 101-111.

Ratanasopitkul, P. (2018). blockchain–Revolutionize Green Energy Management. In 2018 International Conference and Utility Exhibition on Green Energy for Sustainable Development (ICUE) (pp. 1-6). IEEE.

Royce, W. W. (1987, March). Managing the development of large software systems: concepts and techniques. In Proceedings of the 9th international conference on Software Engineering (pp. 328-338).

Sánchez-Gómez, N., Mejías Risoto, M., Ramos Cueli, J. M., Wojdynsky, T., Lizcano, D., and Torres Valderrama, J. (2020c). The current limitations of blockchain traceability: Challenges from industry. In WEBIST 2020: 16th International Conference on Web Information Systems and Technologies (2020), pp. 373-380. SciTePress.

Sánchez-Gómez, N., Morales Trujillo, L., and Torres Valderrama, J. (2019). Towards an approach for applying early testing to smart contracts. In WEBIST 2019: 15th International Conference on Web Information Systems and Technologies (2019), pp. 445-453. SciTePress.

Sánchez-Gómez, N., Morales-Trujillo, L., Gutiérrez, J. J., and Torres-Valderrama, J. (2020a). The importance of testing in the early stages of smart contract development life cycle. Journal of Web Engineering, 215-242.

Sánchez-Gómez, N., Torres-Valderrama, J., García-García, J. A., Gutiérrez, J. J., and Escalona, M. J. (2020b). Model-based software design and testing in blockchain smart contracts: A systematic literature review. IEEE Access, 8, 164556-164569.

Sánchez-Gómez, N., Torres-Valderrama, J., Mejías Risoto, M., and Garrido, A. (2021). Blockchain smart contract meta-modeling. Journal of Web Engineering, vol. 20.

Schmucker, C. M., Blümle, A., Schell, L. K., Schwarzer, G., Oeller, P., Cabrera, L., ... and OPEN consortium. (2017). Systematic review finds that study data not published in full text articles have unclear impact on meta-analyses results in medical research. PloS one, 12(4), e0176210.

*Shailak, J. (2020). Smart Contracts: Building Blocks for Digital Transformation.
Available online at: https://doi.org/10.13140/RG.2.2.33316.83847*

Sheikh, H., Azmathullah, R. M., and Rizwan, F. (2019). Smart contract development, adoption, and challenges: the powered blockchain. International Research Journal of Advanced Engineering and Science, 4(2), 321-324.

Skotnica, M., Klicpera, J., and Pergl, R. (2020). Towards model-driven smart contract systems–code generation and improving expressivity of smart contract modeling. In Proc. EEWC (pp. 1-15).

Sparx System (2022). UML 2 Tutorial. Available online at:
https://sparxsystems.com/resources/tutorials/uml2/index.html

Sparx Systems (2023). Enterprise Architect.
Available online at: https://sparxsystems.com/

Steiu, M. F. (2020). blockchain in education: Opportunities, applications, and challenges. First Monday.

Sultan, K., Ruhi, U., and Lakhani, R. (2018). Conceptualizing blockchains: characteristics and applications. arXiv preprint arXiv:1806.03693.

Swan, M. (2015). blockchain: Blueprint for a new economy. " O'Reilly Media, Inc.".

Szabo, N. (1994). Smart contracts. 1994. Virtual School.

Tapscott, D., and Tapscott, A. (2017). La revolución blockchain. Descubre cómo esta nueva tecnología transformará la economía global. ediciones deusco. Séptima edición.

Tuya, J. (2009). Nuevo estándar de pruebas del software: ISO IEC 29119. Available online at: https://in2test.lsi.uniovi.es/gt26/presentations/ISO-29119-Javier-Tuya-SoloPruebas2009.pdf

Vacca, A., Di Sorbo, A., Visaggio, C. A., and Canfora, G. (2021). A systematic literature review of blockchain and smart contract development: Techniques, tools, and open challenges. Journal of Systems and Software, 174, 110891.

Valerievitch, T. A., Vladimirovitch, T. I., Alexandrovitch, K. J., and Andreevitch, B. D. (2019). Overview of the languages for safe smart contract programming. Proceedings of the Institute for System Programming of the RAS

Van Der Straeten, R., Mens, T., and Van Baelen, S. (2008). Challenges in model-driven software engineering. In International Conference on Model Driven Engineering Languages and Systems (pp. 35-47). Springer, Berlin, Heidelberg.

Vandenbogaerde, B. (2019). A graph-based framework for analysing the design of smart contracts. In Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (pp. 1220-1222).

Viriyasitavat, W., and Hoonsopon, D. (2019). Blockchain characteristics and consensus in modern business processes. Journal of Industrial Information Integration, 13, 32–39.

Werbach, K., and Cornell, N. (2017). Contracts ex machina. Duke LJ, 67, 313.

Wieringa, R. J. (2014). Design science methodology for information systems and software engineering. Springer.

Westerkamp, M., Victor, F., and Kupper, A. (2020). Tracing manufacturing processes using blockchain-based token compositions. Digital Communications and Networks, 6(2):167—176.

Whittle, J., Hutchinson, J., and Rouncefield, M. (2013). The state of practice in model-driven engineering. IEEE software, 31(3), 79-85.

Wohlin C., and Prikladniki, R. (2013). Systematic literature reviews in software engineering. Inf. Softw. Technol., vol. 55, pp. 919–920.

Xu, X., Pautasso, C., Zhu, L., Lu, Q., and Weber, I. (2018, July). A pattern collection for blockchain-based applications. In Proceedings of the 23rd European Conference on Pattern Languages of Programs (pp. 1-20).

Yadav, V. S., and Singh, A. R. (2019). A systematic literature review of blockchain technology in agriculture. In Proceedings of the International Conference on Industrial Engineering and Operations Management (pp. 973-981).

Yaqoob, I., Salah, K., Jayaraman, R., and Al-Hammadi, Y. (2021). blockchain for healthcare data management: Opportunities, challenges, and future recommendations. Neural Computing and Applications, 1-16.

Producción y actividad investigadora

A continuación se recogen las publicaciones que se han realizado sobre el trabajo presentado en la presente Tesis y otros trabajos relevantes adicionales, así como los proyectos de I+D+i – en convocatorias competitivas - en los que he participado durante la realización de la presente Tesis.

Producción investigadora

Sánchez-Gómez, N., Gutiérrez, J.J, Parrilla, E.E., and García-García, J.A. Mecanismo para la generación sistemática de pruebas funcionales de smart contracts en sistemas de gestión de publicaciones digitales. Congreso IBERSID 2023. Zaragoza del 2 al 4 de octubre de 2023.

Sánchez-Gómez, N., Gutiérrez, J.J, Parrilla, E.E., García-García, J. A., De Acuña Garrido, M.A., and Escalona, M.J. (2023). Mechanism for the systematic generation of functional tests of smart contracts in digital publication management systems. Chapter of IGI Global eEditorial Discovery.

García-García, J. A., Ramírez de Verger, C., and Sánchez-Gómez, N. (2022). La Calidad del Software Como Mecanismo de Éxito en Proyectos Multidisciplinares: Proyecto Imedea y Meet2care. Pag. 433-447. Calidad y Sostenibilidad en Sistemas de Información en la Práctica. Ra-Ma. 2022. ISBN 978-84-1897-160-0

Sánchez-Gómez, N., Torres-Valderrama, J., Mejías Risoto, M., and Garrido, A.(2021). Blockchain smart contract meta-modeling. Journal of Web Engineering, 20.
Este trabajo fue realizado durante mi estancia pre-doctoral en el Centro LIFIA. Facultad de Informática de la Universidad de La Plata (Argentina), desde el 8 de marzo de 2021 hasta el 7 de septiembre de 2021.
Métrica SJR : Categoría Computer Networks and Communications. Cuartil Q3

Domínguez, D., Morales Trujillo, L., Sánchez-Gómez, N., and Navarro Pando, J.M. (2021). IoMT-driven eHealth: a technological innovation proposal based on smart speakers. Comunicación en congreso. International Work-Conference on Bioinformatics and Biomedical Engineering. Granada.

García-García, J.A. , Sánchez Gómez, N., Ramírez de Verger, C., Bautista , M.J., and Navarro Pando, J.M. . Reflection on Telemedicine Competencies in Spanish Medicine Degrees After the COVID-19 Pandemic. Comunicación en congreso. 31st Medical Informatics Europe Conference. Online. 2021

Sánchez-Gómez, N., Mejías Risoto, M., Ramos Cueli, J. M., Wojdynsky, T., Lizcano, D., and Torres-Valderrama, J. (2020). The current limitations of blockchain traceability: Challenges from industry. In *WEBIST 2020: 16th International Conference on Web Information Systems and Technologies (2020), pp. 373-380.* SciTePress.

Este trabajo fue realizado durante mi estancia pre-doctoral en la Universidad a Distancia de Madrid (UDIMA), desde el 7 de septiembre de 2020 hasta el 6 de septiembre de 2021.

Sánchez-Gómez, N., Torres-Valderrama, J., García-García, J. A., Gutiérrez, J. J., and Escalona, M. J. (2020). Model-based software design and testing in blockchain smart contracts: A systematic literature review. *IEEE Access*, *8*, 164556-164569.
Métrica SJR: Categoría Computer Science (miscellaneous. Cuartil Q1.

Sánchez-Gómez, N., Morales-Trujillo, L., Gutiérrez, J. J., and Torres-Valderrama, J. (2020). The importance of testing in the early stages of smart contract development life cycle. *Journal of Web Engineering*, 215-242.
Métrica CiteScore: Categoría Computer Networks and Communications. Cuartil Q3

Garcia-Garcia, J. A., Sánchez-Gómez, N., Lizcano, D., Escalona, M. J., and Wojdyński, T (2020.. Using Blockchain to Improve Collaborative Business Process Management: Systematic. IEEE Access, 8, 142312-142336.
Métrica SJR - Computer Science (miscellaneous). Cuartil Q1.

Sánchez-Gómez, N., Morales Trujillo, L., and Torres Valderrama, J. (2019). Towards an approach for applying early testing to smart contracts. In WEBIST 2019: 15th

International Conference on Web Information Systems and Technologies (2019), pp. 445-453.. SciTePress.

Escalona, M. J., Domínguez Mayo, F. J., García-García, J. A., Sánchez-Gómez, N., and Ponce González, J. (2015). Evaluating enterprise content management tools in a real context. Journal of Software Engineering and Applications, 8(08), 431.

Sánchez Begines, J.M., Domínguez Mayo, F.J., Escalona, M.J., Mejías Risoto, M., and Sánchez-Gómez, N. (2015). A framework to evaluate software developer's productivity. VALORTIA project. Poster en Congreso. International Joint Conference on Software Technologies 2015. Colmar, Alsace, France.

Alba, M., Ramírez, M., Pavon, I., Sánchez-Gómez, N., and García-García, J.A. (2013). Comparativa de Herramientas ECM en el marco de la e-administración. Congreso Internacional de Computación y Telecomunicaciones. Lima (Perú).

A continuación, gracias a *Google Scholar Citations,* se muestran las citas recibidas, así como el índice h e i10-index[12] (a 30 de septiembre de 2023).

Figura 62. Citas recibidas (a fecha 30/septiembre/2023)
Fuente: Google Scholar Citations.

[12] El índice h indica que h publicaciones se han citado al menos h veces y el índice i10. indica las publicaciones que se han citado al menos 10 veces.

Proyectos de I+D+i en convocatorias competitivas

A continuación, se muestra la Tabla 35 con los proyectos I+D+i en convocatorias competitivas donde he participado.

Fechas inicio-fin	Rol	Denominación	Agencia financiadora	Importe concedido (€)
05/10/2021-31/12/2022	Investigador	SmartAuditor: Testeo temprano de Contratos Inteligentes en entornos de Cadenas de Bloques para elegir socios en el emprendimiento (P20_00644)	Consejería de Economía, Conocimiento, Empresas y Universidad (Autonómico)	124.600,00
01/06/2020-31/05/2023	Investigador	NICO: Nuevas Iniciativas para el Aseguramiento Temprano de la Calidad Funcional y no Funcional en Procesos y Productos Software Orientados al Usuario (PID2019-105455GB-C31)	Ministerio de Ciencia, Innovación y Universidades (Nacional)	125.114,00
01/02/2020-30/04/2022	Investigador	GISPARQ. Gestión Inteligente y Sostenible del Patrimonio ARQuitectónico: innovación digital y gestión integrada para la conservación, el turismo y la economía de la cultura. Aplicación al Real Alcázar de Sevilla (US-1263780)	Junta de Andalucía (Consejería de Economía y Conocimiento) (Autonómico)	30.000,00
30/12/2016-29/12/2020	Investigador	Explorando Soluciones Guiadas para Sistematizar el Aseguramiento Temprano de la Calidad del Software (TIN2016-76956-C3-2-R)	Ministerio de Economía y Competitividad (Nacional)	219.252,00
01/09/2016-31/12/2019	Contratado	IDE4ICDS: Diseño de un marco de trabajo basado en herramientas para mejorar la gestión de guisas clínicas y procesos asistenciales (RTC-2016-5824-1)	Ministerio de Economía y Competitividad (Nacional)	150.380,00

Tabla 35. Proyectos de I+D+i en convocatorias competitivas

Anexo. Características clave de la tecnología blockchain

La blockchain es una tecnología conceptualmente segura gracias a su naturaleza distribuida, a la irreversibilidad de las transacciones y al uso intensivo del cifrado. En concreto, esta tecnología presenta una serie de características que permiten gestionar la información de manera segura y trazable, proporcionando transparencia y privacidad a los usuarios.

Estas características resultan muy interesantes en el entorno empresarial, ya que permite a las empresas y organizaciones mejorar sus procesos de negocio, además de definir nuevos modelos de negocio basados en entornos colaborativos soportados por la tecnología blockchain. A continuación se exponen, de forma más detallada, las principales características asociadas a esta tecnología.

A.1. Aspectos generales

A.1.1. Distribuida (o descentralizada)

La tecnología blockchain, más que descentralizada, es una tecnología distribuida. Ésta funciona sin la necesidad de un ente central que controle el tráfico de datos y la información no está únicamente en manos de una persona o autoridad central, sino que tal información es controlada por muchos usuarios (nodos), todos ellos con una copia de los diferentes bloques que componen la cadena. Se elimina, por tanto, la necesidad de un tercero

que actúe como garante e intermediario, posición que ocupan en condiciones de igualdad[13] los distintos usuarios (nodos).

Según el creador de Ethereum, Vitalik Buterin: *"Las blockchain están políticamente descentralizadas (no hay nadie que las controle) y arquitectónicamente descentralizadas (no hay punto central de fallo de infraestructura), pero están lógicamente centralizadas (hay un estado comúnmente acordado y el sistema se comporta como una sola computadora)".*

A.1.2. Segura

La criptografía es un elemento esencial en esta tecnología, ya que proporciona seguridad sobre la información que se almacena en la blockchain y en la información compartida entre los nodos de la red. Para poder operar en una red blockchain determinada es necesario disponer, además, de un conjunto de claves asimétricas válidas para operar.

Aunque no todas las blockchain usen el mismo formato de claves asimétricas, normalmente todas las transacciones van firmadas por una clave privada[14] del emisor. Dentro de la transacción, además, se incluye la clave pública[15], que permite verificar el contenido de la transacción, detectando si ésta ha sido manipulada.

Otro elemento que proporciona seguridad a la blockchain es la función criptográfica hash. Ésta permite generar identificadores únicos del contenido de los bloques. Éstos son utilizados para interconectar los bloques, ofreciendo un mecanismo que permite identificar alteraciones en la cadena. La seguridad

[13] Todos los nodos de la red blockchain son iguales (P2P) al tener una copia de toda la información. En concreto, cada usuario que forma parte de la red tiene una copia íntegra y actualizada de toda la blockchain y, según se vaya añadiendo información, también se irá sincronizando en todos los ordenadores conectados.

[14] La clave privada es una de las claves que se generan durante el procedimiento de generación de claves del sistema criptográfico asimétrico.

[15] La clave pública es un identificador que podemos compartir ya que es una de las dos partes que forman el conjunto de claves creadas por la criptografía asimétrica para compartir "secretos" de forma segura.

radica en la capacidad que tienen los nodos para detectar modificaciones de los datos rápidamente, rechazando la transacción o el bloque.

A.1.3. Confiable

Como se ha indicado anteriormente, una consecuencia de que la blockchain sea distribuida es que la confianza no recae únicamente sobre un determinado usuario, sino en una pluralidad de usuarios (nodos). La confianza, en el funcionamiento de la blockchain, es esa característica que hace que dos o más actores que no confían entre sí puedan realizar una transacción en la blockchain gracias al consenso.

Los usuarios (nodos) de una red blockchain pueden adoptar tres roles: (i) usuarios con derecho a tener y consultar una copia de la base de datos distribuida (accessors), (ii) participantes con derecho a realizar transacciones (participants) y (iii) usuarios encargados de validar transacciones y crear bloques (miners), tras resolver algoritmos de consenso que verifican la legitimidad de la transacción.

Por tanto, los nodos mineros (miners) se especializan en validar la transacción y escribirla cifrada en el bloque, encadenándolo a los preexistentes una vez completado. Antes de que un nuevo bloque pueda ser agregado a la blockchain, su autenticidad ha de ser verificada por un proceso de validación por consenso. Este mecanismo de consenso asegura que todas las copias del libro mayor comparten el mismo estado.

Además, una vez validada la transacción, los mineros actualizan la base de datos distribuida mediante la agregación de la transacción al bloque de transacciones en curso (todavía no está definitivamente registrado). Cuando este bloque alcanza un número dado de transacciones validadas, los nodos mineros proceden a sellarlo e incorporar el bloque a la cadena, quedando estas transacciones registradas de forma cronológica.

A.1.4. Transparente (y con privacidad)

Partiendo de la base de que todos los usuarios de una red blockchain tienen acceso al libro mayor, eso implicaría que todos tienen la información sobre las transacciones que se efectúan, aunque lo escrito en ella solamente es inteligible por quien sepa descifrarlo. Esta transparencia también se potencia publicando las reglas con las que se define el funcionamiento de la blockchain. Esto se logra haciendo público el código del software necesario para ejecutar la blockchain y generando una comunidad de nodos y desarrolladores que siguen este principio de transparencia a través del uso de código abierto.

Como se ha indicado anteriormente, esta tecnología emplea mecanismos criptográficos de seguridad para acceder, cifrar y firmar las transacciones, los bloques y su encadenado. Las claves privadas, a emplear por los usuarios de la red, pueden estar vinculadas a la identidad de los usuarios o pueden estar vinculadas a elementos intermedios como, por ejemplo, las "wallet" con las que se ofrece anonimato en las operaciones.

En concreto, el proceso que permite generar estas claves y la dirección de un bloque no requiere de ningún dato personal, por lo que estos datos no van asociados a la identidad de la persona o entidad que crea la dirección. Luego este mecanismo proporciona privacidad a la hora de operar dentro de la red blockchain, ya que las transacciones van asociadas a una dirección y firmadas con una clave que no tienen asociados datos sobre la identidad de la persona o entidad que realiza la misma.

A.1.5. Trazable

La blockchain permite recorrer la cadena de bloques y trazar todas las operaciones que se han realizado sobre una determinada dirección[16], o retroceder en el tiempo y revisar las transacciones que se hicieron en una

[16] En el caso de Bitcoin, una *dirección* es una *identificación única* que te permite enviar y recibir criptomonedas de forma rápida, segura y sencilla.

fecha determinada, explorando todos los bloques generados en la fecha indicada.

Al enlazar la secuencia de transacciones e incorporar una marca de tiempo, esta tecnología ofrece transparencia y trazabilidad a las operaciones, sin por ello quebrantar a priori la privacidad de los usuarios. Esta característica hace posible que se puedan consultar todas las operaciones realizadas.

Además, los mecanismos criptográficos requeridos para el procesamiento de las transacciones permiten atribuir con certeza absoluta la autoría de la información que se genera y se transmite, y hace extremadamente difícil cambiarla a posteriori, eliminando la posibilidad de alteraciones.

A.1.6. Irrevocable e inmutable

Todas las características indicadas hace que la manipulación del contenido de una blockchain resulte muy difícil e incluso imposible, ofreciendo irreversibilidad, es decir, una vez registrado un dato, éste no puede ser modificado o borrado.

Mejor dicho, es extremadamente fácil darse cuenta de que alguien intenta modificar alguna información. Es más, cualquier intento de alterar la información es detectado rápidamente por el resto de los nodos y la cadena o el bloque con la información alterada es rechazada. Se suele decir, por ello, que esta tecnología es inmutable (e irreversible), o lo que es lo mismo, que no existe ninguna forma práctica de cambiar o editar cualquier dato que se haya grabado en la red blockchain (o de volver atrás).

Todo ello es consecuencia del encadenamiento sucesivo de los bloques basado en la criptografía. Si un nodo decide cambiar el contenido de la cadena de bloques alterando una transacción ya realizada e incluida en un bloque, provocará que el contenido de su versión del libro mayor varíe. Un cambio que será fácilmente identificable por el resto de los nodos. Por lo tanto, a la hora de someter a aprobación una nueva transacción, éstos no aceptarán su versión del registro, puesto que el contenido será distinto.

A.2. Aplicaciones de esta tecnología

La tecnología blockchain está en una situación de adaptación y crecimiento continuo, cada vez más organismos, entidades y empresas aplican esta tecnología a sus productos, servicios u operaciones internas para que éstas sean más dinámicas y eficientes.

Esta tecnología, sin embargo, se encuentra en el inicio del proceso, como Internet en los años 90, aunque ya haya conseguido calar en una infinidad de sectores, donde destaca sobre todo el sector financiero.

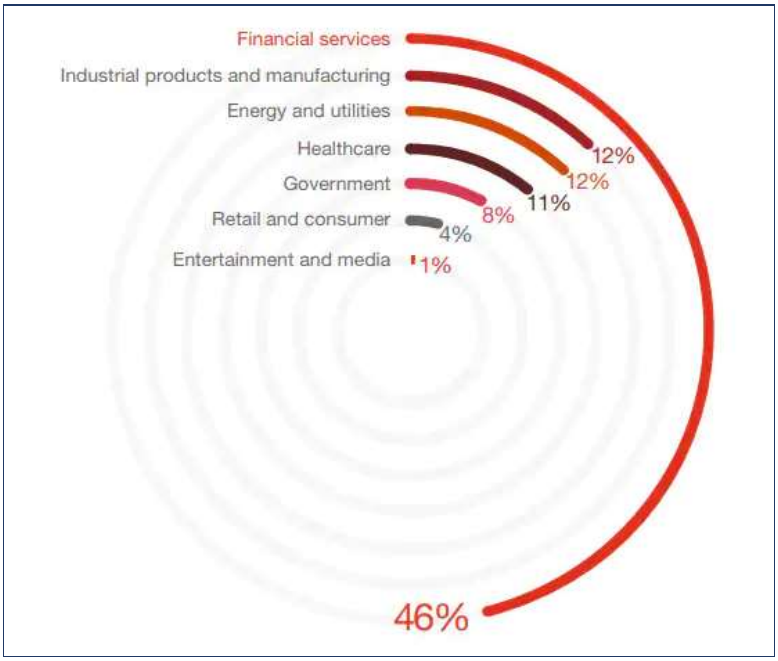


Figura 63. Sectores liderando la blockchain
(Fuente: https://www.pwc.es)

Pero, para tener una visión general, también es necesario hacer referencia a los países que están liderando esta tecnología, así como las perspectivas de los mismos a corto plazo.

Figura 64. Países que están liderando la blockchain – hoy y mañana
(Fuente: https://www.pwc.es)

Aunque ya destacan algunos sectores, hay otros en los que parecía imposible realizar actividades basadas en esta tecnología y, al final, ya han surgido numerosas iniciativas y casos de éxito. Por tanto, resulta de interés realizar un breve repaso sobre el uso que se le puede dar a esta tecnología en los diferentes sectores:

Sector financiero

Esta tecnología parece propiciar el desarrollo de nuevos modelos de negocio en este sector e, incluso, posibilitar que la banca se redefina a medio-largo plazo en el contexto Fintech (https://www.fin-tech.es). Según el Foro Económico Mundial, en los próximos años seremos testigos de una importante transformación en la que la blockchain acabará convirtiéndose en el "corazón" del futuro sistema financiero mundial, siendo algunas de sus aplicaciones más inmediatas las siguientes:

- La gestión de pagos, transferencias y el envío de remesas (abaratando sus costes).

- La detección del fraudes y gestión de riesgos, ya que esta tecnología permitiría verificar la autenticidad de los clientes, de sus contratos, detectando posibles actuaciones de sujetos bajo sospecha.

- El mercado de valores donde, por ejemplo, Nasdaq ya utiliza esta tecnología.

Las entidades financieras, de prácticamente cualquier país, ya están inmersas en algún proyecto relacionado con esta tecnología, pero es la banca de la R.D. China la que destaca actualmente.

Sector asegurador

Ya han aparecido las Insurtech, término análogo al del sector financiero, y se han impulsado para afianzar su foco en el cliente, la digitalización, los seguros de salud y los micro-seguros. Según los expertos, esta tecnología podría permitir que las aseguradoras trabajasen "de oficio" ante cualquier incidente. Por ejemplo, si un turista tiene un billete de avión con seguro de reembolso y no puede embarcar en su vuelo, éste podría recibir automáticamente la cantidad acordada con su aseguradora.

Este sector también se podría ver beneficiado por la simplificación del proceso de intermediación ya que, según algunos estudios, el uso de smart contracts podría suponer una importante reducción del precio en el seguro de automóvil, al desaparecer estos intermediarios.

Por otro lado, la unión de esta tecnología con Internet of Things (IoT) también podría revolucionar completamente este sector. Por ejemplo, se podría proporcionar a los usuarios un sistema de gestión de demandas más transparente, responsable e indiscutible, ya que, a medida que los hogares, vehículos, etc. estén más interconectados, se podría detectar automáticamente cualquier incidente, evaluar los daños y, si está cubierto, realizar el pago correspondiente al afectado conforme a los términos del seguro, incluso antes de que el cliente lo solicite expresamente.

Sector sanitario

El enfoque principal de la implementación de esta tecnología, en la Unión Europea, es el Reglamento General de Protección de Datos (GDPR, siglas de *General Data Protection Regulation*). Esta tecnología podría otorgar a las personas un mayor control sobre cómo almacenan, administran y usan sus datos personales generados en línea.

Además, la sanidad podría sufrir una verdadera revolución si una red blockchain sirviera para registrar todo tipo de historiales médicos y, con ello, resolver uno de los problemas clásicos de la gestión de la sanidad: la confidencialidad de la información clínica.

Sector público

Los ciudadanos exigen simplicidad, ubicuidad e inmediatez, demandando un modelo de Administración Pública más transparente, rápida, eficiente e integrada en la vida cotidiana de la ciudadanía. Teniendo en cuenta las características de esta tecnología, la administración pública podría hacer posible estas demandas al propiciar que los ciudadanos, pero también las organizaciones y empresas, puedan acceder a información donde la transparencia y la confianza sean elementos clave.

Algunos ejemplos donde se podría aplicar esta tecnología es la Identidad digital, las contrataciones públicas, los registros públicos y privados, la gestión del alumbrado público, de los residuos sólidos urbanos, del agua y alcantarillado, etc. Ya existen gobiernos pioneros en el uso de esta tecnología, a pesar de la resistencia propia del sector público a los grandes cambios (y su tendencia a aguardar un caso de éxito concreto que sea replicable fácilmente). En este sentido destaca Estonia, la sociedad digital más avanzada del mundo según los expertos (https://forbes.es), que emplea esta tecnología en sus registros fiscales y empresariales, y de ahí la ha extendido a los registros sanitarios electrónicos de sus ciudadanos.

Sector logístico

En las empresas de logística y transporte, especialmente, esta tecnología aportaría dos elementos esenciales para la transformación: más confianza y menos fricción, a la vez que pone el foco sobre la privacidad de la información y el rendimiento. Incluso determinados estudios hablan de que esta tecnología tendría un rol destacado en la trazabilidad, la transparencia y la confidencialidad que aportaría a las nuevas estrategias de gestión de las cadenas de suministro, tanto desde un enfoque más global de importaciones y exportaciones, o relaciones entre cliente y proveedor, hasta el reparto final en la distribución urbana de mercancías.

Sector energético

Lo habitual, hoy en día, es que un operador abastezca de electricidad o gas a cada casa, empresa o edificio público a cambio de unas tarifas. Sin embargo, en el caso de la electricidad, cada vez son más las casas o edificios que pueden generar su propia electricidad con sistemas de energías renovables. En estos casos, para cuando hay excedentes, algunos países han instaurado un sistema de compensación, entre los vatios aportados a la red y los consumidos, para realizar el cálculo de la factura energética. Utilizando la blockchain, las casas y edificios, conectados entre sí a través de una red distribuida, podrían comprar energía a la red o vender sus excedentes dependiendo de sus necesidades en cada momento, sin necesidad de que ningún intermediario lleve el control. Todas las transacciones de pagos e intercambios de energía quedarían almacenadas en la blockchain y serían verificadas por los miembros de la red.

Sector turístico y cultural

Por un lado, la posibilidad de crear una Identificación Digital (ID) de los viajeros evitaría el uso de pasaportes o datos de pago, entre otros y facilitaría la acreditación del turista. Combinando la Identificación Digital con IoT también permitirá abordar otras iniciativas, como abrir la habitación del hotel con el móvil, agilizar la gestión de equipajes, etc.

Por otro lado, La aplicación de esta tecnología en la música, el cine o la industria editorial, etc. podrían suponer mejoras revolucionarias. Los artistas podrían tener el control sobre sus obras y hacer un seguimiento de las mismas. Se podría tener el registro del recorrido completo de un cuadro, una escultura o cualquier objeto valioso, desde la creación hasta su destino final. Se podrían digitalizar las galerías de arte donde los espacios físicos podrían dejar de tener sentido ya que se podría tener un mercado online descentralizado y transparente, operando sin intermediarios gracias a la blockchain.

Sector de la defensa y la seguridad

Actualmente hay aplicaciones prácticas en Defensa como, por ejemplo, el bloqueo o desbloqueo automático de armas o vehículos militares dependiendo de quién trate de manejarlos, pudiendo IoT junto con la blockchain mejorar la seguridad, reforzar la privacidad y el intercambio de datos. De esta forma, al tratarse de una tecnología inmutable, se consigue una base de datos distribuida fácilmente verificable y difícil de modificar por actores maliciosos.

La combinación de la Identidad Digital (ID) e IoT permitiría, además, crear sistemas de seguridad automatizado que garanticen o impidan el acceso no deseado, de forma completamente automatizada. Según los expertos, esta especie de ID basado en la blockchain reemplazará pronto a las llaves y a las credenciales (usuario y clave), ya que se podrá utilizar nuestra identidad blockchain para acceder a un coche o a sitios web.

Sector automovilístico

La blockchain ya está en la hoja de ruta de este sector al existir ya iniciativas que proponen utilizar esta tecnología en los vehículos autónomos para pagar los impuestos de peaje, o para disminuir drásticamente los accidentes de tráfico. También existen iniciativas para facilitar el repostado de los vehículos eléctricos mediante el uso de smart contracts.

Sector de apuestas, juegos de azar e industria del entretenimiento

Ya han aparecido plataformas como Gnosis, Augur o Stox, que se han centrado en los mercados predictivos, pero que también dan soporte a las apuestas sobre resultados deportivos, procesos electorales, etc. En la Industria del entretenimiento ya existen diversos proyectos relacionados con la transmisión de video y el desarrollo de juegos, lo que ha puesto ya, sobre el mantel, nuevos productos y servicios.

A.3. Funcionamiento de la red blockchain

La construcción y el funcionamiento de la red blockchain depende de una serie de elementos que examinaremos a continuación:

A.3.1. Bloques

Un **bloque** es el conjunto de transacciones confirmadas - incluidas en un momento dado -, e información adicional que se ha incluido en la blockchain. Cada bloque que forma parte de la cadena (excepto el bloque génesis o bloque cero, que es el bloque que inicia la cadena) está formado por:

- **Versión**. Se trata del número que indica el nivel de actualización del software empleado en el momento en que el bloque fue minado (es empleado para leer el contenido de cada bloque de manera correcta).
- **Hash del bloque anterior** (valor alfanumérico). Este valor permite que los bloques queden vinculados secuencialmente formando una cadena.

- Target de dificultad. Es el ajuste de la dificultad de minería y depende de la programación y de los protocolos de funcionamiento de cada blockchain.

- Timestamp (Marca de tiempo). Esta marca permite identificar el instante en el que fue creado el bloque. Por ejemplo, en Bitcoin, se pone el número de segundos pasados desde enero de 1970.

- Nonce (Número arbitrario). Es el valor encontrado por fuerza bruta en el proceso de minado. Éste debe cumplir el Target de dificultad, si éste se ha establecido, y es el número que los mineros utilizan para encontrar un hash válido para el bloque.

- Árbol de Merkle, donde el valor raíz es un "resumen" de todos los valores hoja, como se comentará más detalladamente a continuación.

- Transacciones, siendo ahora mismo entre 1.500 – 2.500 la lista de transacciones por cada bloque.

- Hash del bloque, código alfanumérico que servirá de enlace con el siguiente bloque.

De forma ilustrativa, las partes básicas de un bloque serían (ver Figura 65):

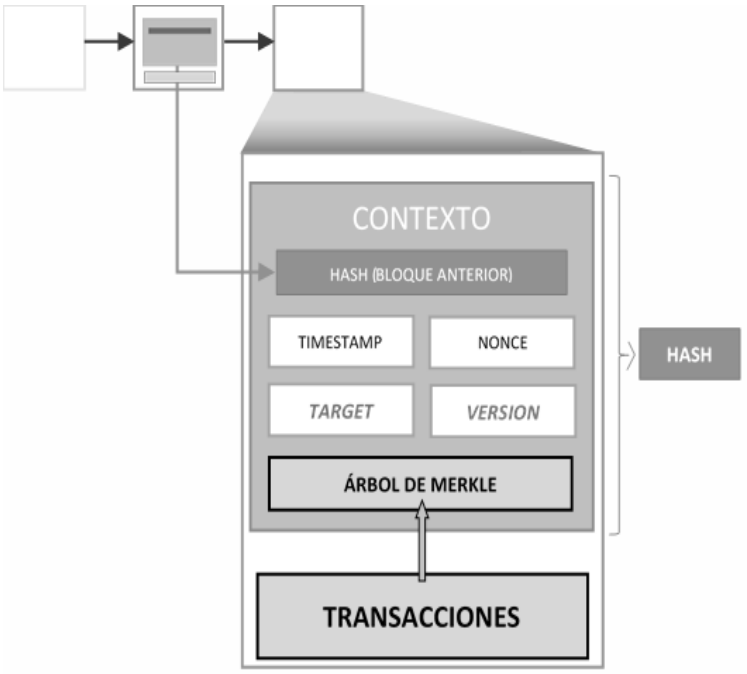


Figura 65. Partes de un bloque de la blockchain

Como se puede apreciar en la Figura 65, cada bloque alberga la información correspondiente a las transacciones de un período concreto y éstas son almacenadas en una estructura llamada árbol de Merkle[17], en alusión a su creador Ralph Merkle. Si un bloque contiene 512 transacciones, el árbol de Merkle se encargaría de agruparlas en 256 pares, que se reducirían posteriormente a la mitad, luego a 64, de ahí de nuevo a la mitad, después 16, 8, 4, 2 y la última (hash raíz). Por ejemplo, Bitcoin emplea la función hash criptográfica SHA-256, siendo sus hash de un tamaño fijo de 256 bit y,

[17] El uso más frecuente de estos árboles es hacer seguros los bloques de datos recibidos de otros pares en una red P2P, es decir, permiten asegurar que éstos son recibidos sin ser alterados y sin estar dañados, permitiendo, además, que los datos de un bloque puedan ser entregados por partes: un nodo puede descargar solo la cabecera de un bloque (árbol) desde una fuente, y otra pequeña parte del árbol relevante para él, desde otra fuente, y todavía asegurar que los datos son correctos.
El motivo por lo que esto funciona es porque los hashes se propagan hacia arriba: si un usuario con malas intenciones intenta hacer un cambio en una transacción falsa en la parte inferior del árbol, este cambio provocaría un cambio en el nodo superior y seguidamente otro cambio en el nodo por encima de éste, hasta que finalmente, se produzca un cambio en la raíz del árbol y por tanto en el hash del bloque, haciendo que el protocolo tenga que registrarlo como un bloque completamente diferente (y casi con toda seguridad con una prueba de trabajo inválida). Fuente: Wikipedia.

a través del árbol de Merkle (ver Figura 66), aglutina los bloques de información por pares y genera un hash por cada uno de ellos, que vuelven a agruparse en pares y generan un nuevo hash que a su vez se agrupa con otro y se repite hacia la parte superior del árbol, hasta alcanzar un único bloque raíz (hash raíz) y se registra en la dirección del bloque actual (block hash) con el fin de reducir el espacio ocupado por cada bloque.

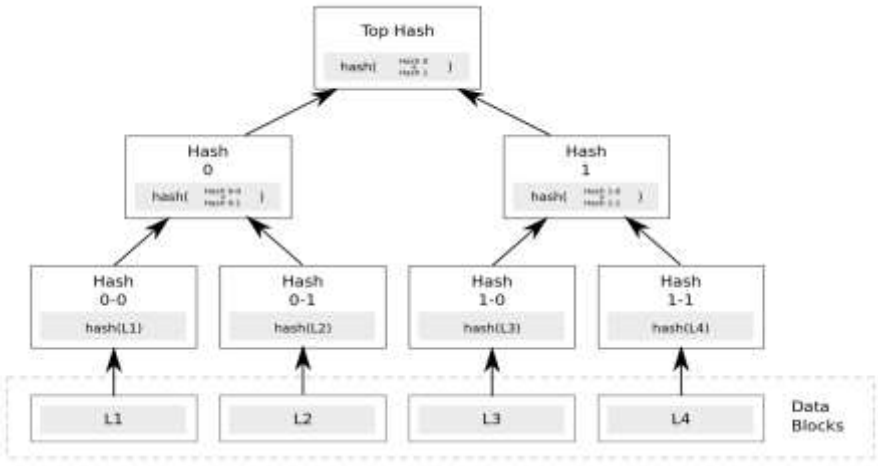


Figura 66. Ejemplo de árbol binario de Merkle
(Fuente: Wikipedia)

En la práctica, el árbol de Merkle lo que busca es poder relacionar una serie de datos separados en un único hash (raíz) para reducir el tiempo y recursos empleados en verificar la integridad de una cantidad de información. Esta estructura relaciona todas las transacciones y las agrupa entre pares para obtener un Root Hash o «dirección maestra», la cual está basada en todos los hashes del árbol. Verificar todas las transacciones de una red sería algo extremadamente lento e ineficiente, por eso se implementó este sistema: si un hash es cambiado, cambiarían todos los demás hasta llegar a la raíz (Root hash). En un Árbol de Merkle los hashes se agrupan en pares en una relación 2n, donde «n», es la cantidad de pares, y no existe un número máximo determinado, pueden ser 2, 4, 8, 16… los límites los establece el tamaño del bloque. De esta forma, validar 10.000 transacciones en la red cuestan lo mismo que validar una única transacción.

El propio Nakamoto explica en Bitcoin whitepaper (Nakamoto, 2009) que una vez que una transacción está enterrada bajo suficientes bloques, las transacciones anteriores pueden ser descartadas para salvar espacio. Para lograrlo sin romper el hash que las asegura y conecta al resto de la blockchain, los viejos bloques se compactan con el árbol de Merkle, mientras que los hashes interiores que formaron la raíz no necesitan ser conservados.

A.3.2. Nodos

Un nodo es un ordenador/chipset conectado a la red blockchain utilizando un software que almacena y distribuye una copia actualizada en tiempo real de la cadena de bloques. Cada vez que un bloque se confirma y se añade a la cadena, se comunica a todos los nodos y éste se añade a la copia que cada uno almacena.

Para formar parte de la misma blockchain, todos los nodos deben poseer el mismo software/protocolo para comunicarse entre sí. La estructura física de esta red sería el conjunto de ordenadores (nodos) que, conectados en red, utilizan un mismo sistema de comunicación (protocolo) con el objetivo de validar y almacenar la misma información registrada en una red P2P. La estructura lógica sería la suma de todos esos elementos que logran que la información recogida no pueda modificarse, ya que los algoritmos criptográficos, sumados a la propia capacidad colectiva de la red, contribuyen a asegurar la irreversibilidad de la información.

Aunque los nodos de una misma red blockchain sean iguales, éstos pueden asumir diferentes roles dependiendo de la funcionalidad que están apoyando. Por ejemplo, en Bitcoin, los nodos se pueden clasificar en tres categorías:

- Nodos que sólo emiten transacciones (*broadcast node*)
- Nodos que retransmiten o propagan y transmiten las transacciones (*relay node*)
- Nodos que emiten, transmiten y minan, es decir, crean nuevos bloques con las transacciones (*mining node*)

De estas categorías, el "*broadcast node*" es el nodo que requiere menos capacidad computacional, siendo el "*mining node*" el que requiere mayor capacidad para resolver algoritmos complejos, entre otros temas. Por ejemplo, el algoritmo de Prueba de Trabajo (Proof-Of-Work) es el utilizado en la minería Bitcoin.

A continuación, para detallar la funcionalidad que realiza cada tipo de nodo se muestra, a modo de ejemplo, los pasos que las transacciones siguen durante la compra de una bebida usando criptomonedas Bitcoin, a través de un dispositivo móvil:

- El nodo de partida, el smartphone (*broadcast node*), crea una transacción que transfiere sus criptomonedas a la dirección Bitcoin del establecimiento en cuestión. Este nodo entonces envía inmediatamente esta transacción a sus pares en la red blockchain, que probablemente sean nodos que transmiten transacciones.
- El nodo que transmite (*relay node*), propaga esta transacción a otros nodos que también la transmiten, permitiendo que la transacción se propague rápidamente a todos los nodos de la red.

  Contado en un párrafo parece un proceso sencillo, pero esta función no es tan simple. Los nodos de transmisión deben desconfiar de las

transacciones maliciosas, es decir, si un nodo que transmite, reenvía cualquier mensaje, la red se colapsaría rápidamente con una saturación de transacciones basura. Por ello, todos estos nodos deben realizar estas pautas:

1. Comprobar que la transacción tiene el formato correcto
2. Deben asegurar que las firmas son válidas y
3. Verificar que las criptomonedas que se está transfiriendo están en la cuenta origen.

- Una vez superados todos los controles, la transacción llega a todos los nodos de la red que realizan labores de minería. Estos nodos (*mining node*) añaden esta transacción a un bloque preliminar, que tratarán de minar satisfactoriamente.

- Si el minado se termina ejecutando correctamente, se emite entonces el bloque recién extraído a través de la red, lo que confirma las transacciones del bloque y el minero recibe su "recompensa" por haber minado esas transacciones.

- Finalmente, cada transacción queda registrada en todos los nodos de la red blockchain.

A.3.3. Fichas (tokens)

Los tokens tienen su origen en el mundo real, ya que originalmente los tokens eran fichas, pseudo-monedas o vales, utilizados como sustitutos de monedas reales. Un claro ejemplo serían las fichas de un Casino. Éstas sustituyen el dinero, sólo son válidas en ese Casino y sólo tienen validez mientras el Casino esté funcionando.

En el mundo blockchain, de forma análoga, un token sería como la representación digital de algo que tiene valor dentro de un contexto, es emitido por una entidad y solo es válido bajo este universo concreto.

Las criptomonedas comparten ciertas similitudes con los tokens: ambos representan valor y pueden usarse en transacciones. Las monedas criptográficas son nativas de la blockchain en cuestión (Bitcoin, Ethereum, etc.), mientras que los tokens funcionan «sobre la base» de una blockchain. Las criptomonedas son esencialmente una representación digital de la moneda en cuestión, pero los tokens tienen una definición mucho más amplia, ya que pueden representar digitalmente cualquier activo de valor.

Por tanto, un token criptográfico puede tener diferentes finalidades: puede servir para realizar un pago, para participar en un evento, puede ser una obligación dentro de una empresa, puede ser una entrada a un juego, un elemento coleccionable, o cualquier cosa que pueda ser representada en el mundo real.

Todos los tokens criptográficos se dividen en dos categorías:

- Tokens fungibles (FT). Se caracterizan porque pueden fraccionarse. También sirven para intercambiarlos por otras cosas, incluso del mundo real. El ejemplo más claro lo tenemos en las criptomonedas que vamos acumulando y, además, permiten la transferencia a otros usuarios.

- Tokens no fungibles (NFT). Se caracterizan porque no pueden dividirse ni intercambiarse entre sí. Los activos de este tipo de token son únicos y sirven, por ejemplo, para registrar una propiedad en una blockchain, para certificar la autenticidad de una obra de arte, etc. Las entradas para un espectáculo cultural también pueden convertirse en un NFT que da acceso al recinto y garantice que no se trata de una falsificación.

A.3.4. DApps (aplicaciones descentralizadas)

Las DApps, de forma resumida, son aplicaciones descentralizadas que utilizan blockchain para que los usuarios se relacionen directamente entre ellos y

cierren acuerdos, sin que exista un ente que centralice la gestione del servicio.

Su funcionamiento es similar a una aplicación de dispositivos móviles o una aplicación web, pero para considerarla como DApp debe ser 100% código abierto, funcionar de forma autónoma (sin que ninguna entidad la controle, dejando todo el poder de decisión sobre la misma en su comunidad de usuarios) y los datos deben ser almacenados de forma criptográfica a través de una blockchain.

Todo usuario, de la misma DApp, sería un nodo de la red blockchain en la que todos los miembros actúan de forma conjunta como un "notario colectivo" de cualquier movimiento que se realiza a través de la red. En la práctica, nadie debe dar su consentimiento de forma expresa, sino que todo funciona de manera automática, ya que es el propio sistema el encargado de corroborar la validez de cada transacción a través de los smart contracts. Cada vez que hay una nueva operación, la información de la plataforma se actualiza en cada nodo, donde queda almacenada una copia de todo el histórico de la DApp, así que cada usuario contribuye a mantener en pie la aplicación con los recursos de su ordenador.

Un ejemplo de DApp, son los Wallets (Monederos). Éstos son necesarios para almacenar criptomonedas y no son más que aplicaciones encargadas de generar transacciones, firmarlas, enviarlas a los nodos para ser validadas y ponerlas en una lista de espera (mempool) a disposición de los mineros.

Los Wallets tienen bastantes semejanzas con lo que sería la cuenta corriente de una entidad bancaria, pero con un Wallet se tiene el control total de las criptomonedas que se guardan y se puede enviar/recibir criptomonedas de forma libre, autónoma y sin intermediarios.

En definitiva, estos monederos permiten custodiar criptomonedas y, además, conectar con la blockchain para autorizar una transacción monetaria, siendo imprescindible para la protección, custodia y transacción de las

criptomonedas dos claves únicas, una pública y otra privada, que el monedero generará de forma aleatoria y automática.

- La clave pública tiene la propiedad de ser una dirección que se puede compartir con quien se quiera, permite la recepción de criptomonedas (análogo al número de cuenta del banco), cualquiera que la tenga puede enviarte dinero, pero no puede acceder a los fondos disponibles.

- La clave pública depende la clave privada, es decir, la clave privada es una contraseña que jamás se debe compartir, ya que da acceso total a las criptomonedas almacenadas (análogo a las credenciales a tu cuenta bancaria).

Las DApps y las Apps tradicionales tienen muchos elementos en común, sin embargo, su diferencia radica en cómo interactúan con dichos elementos. Ambos tipos de aplicaciones tienen tres estructuras básicas que son el Frontend, el Backend y la capa de almacenamiento de datos.

Frontend

Esta capa es la interfaz que el área usuaria utiliza para interactuar con la aplicación. Tanto las DApp como las App tradicionales pueden hacer uso de los inmensos recursos gráficos existentes para ello, desde interfaces web escritas en HTML5 hasta cualquier framework existente. Su finalidad es simplemente dar al usuario la capacidad de interactuar, recibir y enviar información a la aplicación que esté usando.

Backend

Esta capa hace mención a la lógica de negocio de la aplicación. En una aplicación tradicional, esta lógica es centralizada, a diferencia de las DApps en la que está descentralizada. En las DApps, el Backend está vinculado a una plataforma blockchain y a los smart contract que se ejecutan sobre dicha red. De esta forma, los smart contracts tiene cierta lógica de negocio que

garantiza el funcionamiento de la DApp. Además, como en el caso de Ethereum, pueden existir diversas APIs para controlar la interacción del usuario con las capas de almacenamiento o autenticación, por poner algunos ejemplos.

Almacenamiento de datos

En una App tradicional está capa también es centralizada y, normalmente, los datos son almacenados en servidores en mode on-premise o en la nube. En las DApp, el almacenamiento de datos es descentralizado también. Normalmente, cada usuario de la DApp almacena un historial completo de las acciones que se realizan en la red y, adicionalmente, las interacciones son almacenadas en la blockchain dentro de los bloques de la misma. Todo ello de forma criptográficamente segura, impidiendo accesos no autorizados por terceras personas.

A.3.5. Oráculos

La blockchain puede requerir acceso a datos que se mantienen fuera de su red, por ejemplo, los tipos de cambio de moneda o el estado del tiempo. El oráculo es un medio para permitir el acceso a esos datos externos desde una blockchain (Xu, 2018), es decir, es un puente entre la blockchain y el mundo real.

En concreto, un oráculo es un servicio de terceros que permite conectar la blockchain con el mundo exterior, principalmente para alimentarse de información, pero también puede ser al revés. Los oráculos funcionan consultando y autenticando datos de bases de datos u otras fuentes externas, pero un oráculo, en sí mismo, no es la fuente de datos, sino solo un conector entre la blockchain y la fuente de información.

Para ilustrar esta idea, la siguiente Figura muestra el funcionamiento del Oráculo que está accediendo a información que está fuera de la red blockchain (off-chain), información que es accesible mediante servicios API

Rest (mecanismo más utilizado actualmente para intercambiar información, de manera segura, a través de Internet).

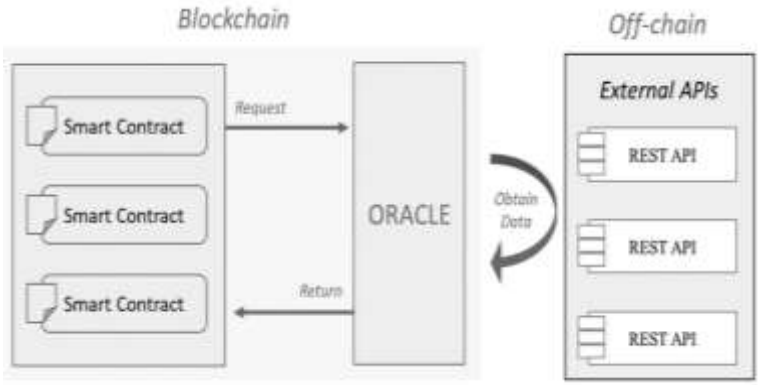


Figura 67. Oráculo de la blockchain
(Sánchez-Gómez, 2020a)

Un oráculo actúa como una API interna a la blockchain (on-chain) y se encarga de recopilar y verificar información del mundo real, es decir, es un tipo de fuente de datos que informa a la red blockchain de los sucesos que ocurrieron en el exterior.

Los oráculos, en su concepción, han sido pensados como una pieza bajo un control centralizado, un modelo que está lejos del modelo descentralizado que aboga la tecnología blockchain. Esto significa que, al usar un servicio de este tipo, confías plenamente en la información de una entidad que puede manipular y corromper algo que en principio ha sido diseñado para que no lo sea. Este es el mayor riesgo y desventaja de los oráculos y para evitarlo nacieron los oráculos descentralizados.