Towards smart contract distributed directory based on the uniform description language

Wafa Ben Slama Souei, Chiraz El Hog, Raoudha Ben Djemaa, Layth Sliman, Ikram Amous Ben Amor

Highlights

Towards smart contract distributed directory based on the uniform description language

Wafa Ben Slama Souei, Chiraz El Hog, Raoudha Ben Djemaa, Layth Sliman, Ikram Amous Ben Amor

- Propose a new architecture of a SCs directory called DSD which extends the EbXML registry.
- Allow SC provider to publish, update and retract SCs description on the DSD.
- Propose a queering algorithm to discover SCs according to the user's functional and non-functional preferences, contrariwise to the existing solution that allows just transaction exploration or indexing from the blockchain.
- Allow the SC consumers to retrieve, access and understand SCs before executing them without accessing their source code.

Towards smart contract distributed directory based on the uniform description language

Wafa Ben Slama Souei[a,c], Chiraz El Hog[b,c], Raoudha Ben Djemaa[a,c], Layth Sliman[d], Ikram Amous Ben Amor[e,c]

[a]*University of Sousse, ISITCOM H.Sousse, Sousse, 4011, Tunisia*
[b]*Department of computer science, College of sciences and arts, Unaizah, Saudi Arabia*
[c]*University of Sfax, MIRACL Laboratory, University of Sfax, MIRACL Laboratory, Sfax, 3031, Tunisia*
[d]*Engineering School of Information and Digital Technologies,EFREI Paris, Paris, 94800, Villejuif, France*
[e]*National School of Electronics and Telecommunications of Sfax, University of Sfax, Sfax, 3018, Tunisia*

Abstract

A Smart Contract (SC) is a piece of code executed on the blockchain to automatically trigger transactions upon the occurrence of predefined events. Due to the intrinsic features regarding traceability and data immutability, many companies started using blockchain Smart Contracts to perform collaborative processes. Despite their promising features, there is a lack of Smart Contacts management platforms that enable blockchain participants to describe and publish their smart contacts or "search and match" already deployed ones. In this paper, a new Distributed Smart Directory (DSD) where providers can publish their SCs description is proposed. The SCs descriptions include metadata covering functional, and non-functional properties of the SC. Hence, users can find SCs according to their non-functional preferences, needs, and constraints. The proposed DSD is an extension of the ebXML directory. It was fully implemented on-chain. The SCs descriptions are generated based on the Uniform Description language for SC (UDL-SC). The proposed solution is implemented on the Ethereum blockchain. It was then tested and evaluated.

Keywords: Smart Contract, Distributed Directory, Uniform description language, Solidity, Blockchain.

1. INTRODUCTION

Blockchain technology is a decentralized peer-to-peer network composed of a series of blocks where data is stored. These blocks are linked based on cryptographic hash methods (16). In This distributed ledger, where consumers are identified using pseudonyms, transactions are validated and verified based using cryptography mechanisms (11). More precisely, each blockchain network agrees on a consensus mechanism that guarantees trust, transparency, and security, between users. Blockchain applications are not limited to cryptocurrency transaction execution (7). With the integration of the Smart Contract concept into the blockchain, blockchain is adopted in many applications of various domains (12) . SCs were introduced by Nick Szabo about twenty-five years ago (9). This concept was adopted to automate the execution of decentralized applications on blockchain networks. The integration of SCs enables the implementation and enforcement of real agreements between blockchain participants. A Smart Contract is a piece of code that encapsulates agreement logic stored on a blockchain. SCs run automatically when predetermined conditions are met, without any intermediary's involvement, so that all participants can be certain of the outcome. Many blockchains use SCs and provide an adaptable platform to code and test them, such as Ethereum[1] and Hyperledger[2] Contrariwise, other blockchains, such as bitcoin[3] and Ripple [4], are still using dedicated internal smart contacts only for cryptocurrency transactions.
The number and types of SC are constantly growing and expanding (13). However, there are no available solutions that allow the user to access, read and compare SCs before using them without accessing their source code. Moreover, SC use is costly and time-consuming. Consequently, to attract and convince users to use a contract, a high level of transparency must be established. Trust and transparency can be ensured through the provision of contractual metadata on an adapted platform that enables the consumers to view and discover Smart Contracts' metadata before using them. This requirement can be met using a SC directory that holds SCs information. From a conceptual standpoint, a SC can be seen as a service (6) offered

[1]https://ethereum.org/fr/
[2]https://www.hyperledger.org/
[3]https://bitcoin.org/fr/
[4]https://ripple.com/

to the consumer. This service can automatically execute the business logic implemented in the code (23).

Numerous studies (24) (22) suggest that Smart Contracts and services share many features. This has led to the creation of a new blockchain architecture that leverages the service orientation of SCs, enabling the publication and discovery of SCs without needing third-party involvement. At the heart of this architecture resides an entity called service registry, which provides the publishing and discovering mechanisms. On the participative web, many service-oriented directories prove their efficiency to simplify the service inventory, IT management, and unified visibility for all services across the web (30)(17). Contrariwise to the web3 environment, the service orientation of the blockchain is still in the preliminary stage (24),(22). In fact, existing solutions try to search, explore, or index SC-produced transactions, or provide tokens to contracts directory (18),(19). Therefore, a new directory for SCs is required to enable the deployment of this service-oriented blockchain architecture.
Many researchers proposed SCs' registry in the literature, on-chain (15), (16) or off-chain(8), (6). Some works suggest storing SCs and their metadata (16),(6), while other works, propose simple solutions where just SCs code are collected and stored(14), (8). In addition, accessing and discovering/searching techniques are different, (14) proposes to download the whole registry locally to read and reuse a contract, whereas other solutions (8),(6),(1) provide searching mechanisms via web interfaces or APIs.

This paper tries to resolve issues related to SCs management and use based on blockchain technology and service-oriented architecture concepts.
Our first goal is to provide a distributed directory where providers can publish easily and efficiently SCs metadata. We intend to provide a directory that holds SC descriptions generated based on the Uniform Description language for SC (UDL-SC)(2), where the provider can indicate functional and non-functional properties of the SCs.

Our second goal is to increase the trust between SC providers and consumers by providing a discovery mechanism that allows searching SCs according to consumers' preferences. We focus on filtering SCs according to their features, security, and legal information to help consumers to find suitable SCs and to verify their integrity and origin.
We can list our main contributions in this paper as follows:

1. a new service-oriented architecture based on blockchain technology using smart contracts, that facilitates the interaction between SC providers and consumers, enables the reuse of SCs, and increases cost-effectiveness.
2. a new distributed smart directory (DSD) based on the EbXML registry structure, where consumer interaction and data flow are managed based on SCs.
3. publishing, updating, and retracting solutions that allow publishing and managing UDL-SC descriptions according to the blockchain mechanism.
4. a discovering solution that allows searching SCs, and filtering them based on consumer preferences (Legal and security metadata).

The remainder of the paper is organized as follows. Sect.2 provides a background of registries on the blockchain. Sect.3, presents and compares the related works and projects of existing SCs directory approaches in the literature. Sect.4 introduces the blockchain service-oriented architecture. Sect.5, gives an overview of the UDL-SC description language proposed in our previous work (2). sect.6 is dedicated to exposing our Distributed Smart directory (DSD) as an approach to tackle the previously mentioned challenges. Sect.7 provides the conceptual overview of the DSD, in addition to the proposed algorithms. Sect. 8 describes the implemented prototype and shows test and evaluation results. Finally, sect.10 concludes our work and gives some perspectives for future improvement.

2. BACKGROUND: REGISTRIES ON THE BLOCKCHAIN

Given the increasing number of digital solutions and users, the tasks of storing, processing, and analyzing data are becoming increasingly difficult. Numerous solutions have been proposed to address these challenges, and emerging technologies are continuously being developed to make these tasks easier and safer. A registry or directory on the blockchain refers to a decentralized database that is maintained on a distributed ledger technology platform. Contrary to the traditional database, where all personal data and property records are stored and controlled by a trusted third party, based blockchain registry or directory is not controlled by any single entity, making it more secure and transparent. The blockchain is considered to be a source of truth. In fact, it is used to store the list of records, such as user identities, assets, transactions, or other types of data, which are verified and recorded in

a tamper-proof and immutable manner using cryptography and consensus algorithms. Storing and managing this data is maintained automatically based on the blockchain's internal mechanisms (5). To verify this data, the user needs access to that closed database or trusts the entity that stores it (5). In addition, the blockchain is an append-only type of database, it prevents data corruption and unauthorized changes. This presents the advantage of reducing centralization and making information always available and secure. Furthermore, based blockchain registries are strong solutions for maintaining the traceability of important and unique products such as diamonds, copyright, or coins such as Non-fungible tokens (NFT).
The records on the based blockchain registry are accessible to anyone with permission to access the network, and they cannot be altered or deleted without consensus from the network participants. Blockchain registries can be classified into two main categories: public and private.(5). A private registry builds a more trusted environment to store important data and sensitive information needs to be kept private. They are restricted to a specific group of participants who are granted permission to access the network. Whereas, public registries are open to anyone to read, write, and access the data on the blockchain. There are two solutions to store data on the blockchain;
On-chain solution: Consists of storing all the information on the block used for data that requires high security, transparency, and immutability. On-chain registries guarantee that the data is tamper-proof and that transactions can be verified and audited at any time. Whereas, it can also be limited by the capacity and scalability of the blockchain, as the data is replicated across all nodes on the network, which can lead to slow transaction times and high costs. But it is considered the most secure solution for small, critical pieces of data that require high levels of security and immutability.
Off-chain solution: Consists of storing only the metadata on-chain while the data is stored off-chain on a secondary network or database that is connected to the main blockchain through smart contracts or other mechanisms. It is a cost-efficient option suitable for large data sets that require flexibility and scalability. Off-chain registries provide greater privacy and confidentiality as data is not replicated on the network. They also enable faster transaction times and lower transaction fees, as transactions can be processed and verified off-chain. However, off-chain registries may not be suitable for critical data that requires high levels of security and immutability as they are stored on third-party networks that may be vulnerable to hacking or data breaches.

Based Blockchain registries/directories are used to store and collect important data or metadata with expensive value. Despite the disadvantages of the two described solutions, the level of security of the blockchain pushes users to sacrifice time and money to get the expected features. In fact, storing/updating data on the blockchain is considered a non-frequent task, but it requires a heightened level of trust and security to attract users.

3. RELATED WORKS AND PROJECTS

In this section, we will present related works and projects that propose a registry solution to store and manage data on blockchain technology.

3.1. Related works

In (6) the authors proposed an off-chain registry for SCs metadata descriptions named SC description language (SCDL) registry. The proposed registry holds SC description metadata designed according to the SCDL description language(6). The proposed SCDL registry allows SC developers to share and publish their SCs' information. It gives the ability for the SC consumers to search and retrieve SC information through the SC locator (6). The work presented in (1) proposed a decentralized solution based on blockchain technology for metadata management and storage. It allows storing services metadata using a struct solidity component in a SC metadata repository and uses Oraclize API to check that the resource is really available. This contract also implements add, replace, and retract operations. Resource search and discovery are currently done in the background by using external blockchain explorers, that read the published data directly and scan it.

3.2. Related projects

In this section, we will present related projects proposing a registry or repository system where the SC and metadata are collected.
In (8), the authors proposed Smart Corpus. It is an organized, reasoned, and up-to-date repository where Solidity source code and other metadata about Ethereum SCs (source code, ABI, and byte code) can easily and systematically be retrieved. Smart corpus creation is based on a pipeline model composed of four essential components; data retrieving, data cleaning, data modeling, and data querying. The essential goal of the proposed smart corpus system is to provide a data set of SCs' source code in a variety of programming languages. Users can query the smart corpus to get useful information

on SCs and their software metrics via a simple user interface.
SC Modular Template (14) is an off-chain open-source repository containing educational materials. It includes SC templates for developers, published on GitHub. It is backed by Blockchain Education Network Hong Kong and Taiwan branches.
Metadata registry(15) is an On-chain IPFS (InterPlanetary File System) directory for Ethereum SCs. It allows deployment keys to submit an IPFS multi-hash containing metadata (in any format the author wishes) that will be associated with their given contract. IPFS is a peer-to-peer protocol for content-addressed sharing of data via a distributed file system.
Metamask corporation proposed an on-chain contract metadata implementation project (16) to broadly accepted metadata to check summed Ethereum SCs addresses. All address keys follow the EIP 55 address checksum format. They use a mapping metadata solidity structure to store simple SC metadata (name, logo, ERC20, symbol, and decimals). This project allows adding SC metadata to the mapping file through a SC function.
(18) presents Etherscan, a leading blockchain explorer, search, API, and analytics platform for Ethereum. The main task of this project is to provide equitable access to blockchain data. Based on the documentation presented in (18), Etherscan allows searching verified SCs, getting their source codes using the addresses, getting contract creators, and creating transactions Hash by addresses.
(19) presents an efficient indexing and querying of blockchain data called the Graph. This project simplifies the accessibility and query on the blockchain-based on GraphQL by providing an efficient and user-friendly way to access data stored on the blockchain. The Graph Protocol achieves this by allowing developers to create and deploy custom data indexes, known as subgraphs, which track specific data points on the blockchain. These subgraphs can index data such as transaction history, account balances, events, or other relevant data for a particular Dapp or application and run the attached mapping handlers. The mapping is a WebAssembly (WASM) module that creates or updates the data entities that Graph Node stores in answer to Ethereum events.

3.3. Synthesis

A variety of solutions has been proposed in the literature that present based blockchain registry, Table 1 summarizes studied works and projects discussed in the previous sections according to the following criteria:

- **Data type**: the type of the stored data on the proposed registry. It can be a SC, simple metadata if the registry holds functional metadata about contracts, or extra metadata if the registry holds functional and non-functional metadata about contracts.
- **Location**: the location of the registry, on-chain means available on the blockchain network. Off-chain means that the registry is located on a centralized and traditional server.
- **Search and discovery mechanisms**: the adopted search and discovery mechanism.

Based on Table 1, we argue that previously presented works and projects suffer from certain weaknesses. On the first hand, only the MetaMask contract-metadata project (16)and the SCDL(6) allowed the storage of SCs source code, simple metadata, and extra metadata about SCs. The rest of the research works deal with SCs code and simple metadata (15), (1), (8) or only SCs source code (14). In addition, stored metadata is very restricted. (1) proposed to use one only attribute called *meta* that allows adding information about a Digital resource. (15) proposes to store an IPFS multi-hash containing SS simple metadata. (8) allowed storing extrinsic metrics related to the number of transactions or tokens produced by the contract and intrinsic metrics related to the source code, such as the number of lines of code, modifiers, and payable. (16) proposed a checksummed Ethereum contract addresses to metadata, like names, and images of their logos. (6) proposed to store SC generic information such as description, name, version, author, and access-oriented attributes like the blockchain type, version, and internal address for internal consumers and the address for external consumers. There is no proposed solution that allows storing SC non-functional properties such as gas consummation, legal, pricing, or security information. On the second hand, architecturally, only (1), (16), and (15), proposed an on-chain registry implemented based on the Ethereum platform, despite its promising advantages. Finally, retrieving mechanisms are not considered in almost all of these works. (16), (14) proposed to import the needed contract, or to download and reuse it without having additional information on its source code. While (8) provides a retrieving mechanism that allows users to search for SCs and SCs metadata through a web interface where the searching criteria are restricted by the authors. Otherwise, (18) allows searching validated SCs

Table 1: Summary of related Works and Projects

Solution	**SC**	**Simple metadata**	**Extra metadata**	**Location**	**Search and discovery**	**Platform**
Smart Corpus (8)	*	*	-	Off-chain	Yes	E
Modular Template (14)	*	-	-	Off-chain	No	-
Metadata Registry (15)	-	*	-	On-chain	No	E
Metamask (16)	*	*	*	On chain	No	E
SCDL(6)	*	*	*	Off-chain	yes	-
Metadata Repository(1)	-	-	-	On-chain	yes	E
Etherscan(18)	*	-	-	Off-chain	yes	E
The Graph(19)	-	-	-	Off-chain	yes	E
DSD	-	*	*	On-chain	yes	E

(**Platform** E: Ethereum, - : non specified) (**Stored data** *: is supported) (-: is not supported)

by addresses, while (19) allows indexing SC and traces their event-emitted transactions on the blockchain. Based on these works, we notice that no proposed approach or project provides a complete on-chain system registry to manage and retrieve SCs easily based on users' preferences. Therefore, the last entry in Table 1 presents our proposed smart contract directory, named DSD, which serves as the key contribution of this research article. The DSD enables the storage of smart contracts metadata (simple and Extra) directly on the blockchain. Moreover, it facilitates the discovery and retrieval of these smart contracts based on users' non-functional preferences.

4. THE BLOCKCHAIN SERVICE ORIENTATED ARCHITECTURE

Blockchain technology is merged in many domains, and with smart contracts, the implementation and integration of based blockchain applications called "DAPP" become more and more easy and flexible.
By analyzing a set of implemented smart contracts code[5], we notice that a SC is a block of code; a class, or an object; deployed in the distributed network, has a defined interface where we found methods signatures, name, inputs, and outputs. Also, it has a standard invocation method that enables its execution by any node on the network. These properties are similar to Web services characteristics. Authors in (22) analyzed web services characteristics such as types, interaction styles, interaction protocols, and data formats, and then discussed the corresponding of these characteristics for smart contracts, to demonstrate the ability to propose a service-oriented perspective on blockchain smart contracts. As well, the work presented by (23) introduces the new concept Blockchain-as-a-Service and demonstrates based on a comparative study between SCs, services, and micro-services, that the integration between service-oriented context and blockchain technologies presents promising prospects. Moreover, the work presented by (24) described a full service-oriented architecture using smart contracts in Heterogeneous Blockchain platforms, by proposing a SC description language (SCDL) (6), and SC Invocation Protocol (SCIP) (25) which support the applicability of the service-oriented concept on the blockchain to facilitate its utilization, to foster reuse and increase cost-effectiveness.
After this review, we assume that the service-oriented architecture for the blockchain represents a promising approach in which a smart contract is viewed as a service. This architecture offers several advantages. Firstly, it empowers consumers of smart contracts to discover, access, review, and

[5]https://github.com/wafa2011/blockchainProjectPortfelio

comprehend the purpose of contracts before using them, without the need to access the source code. Secondly, it enables smart contract providers to publish their contracts on a distributed infrastructure. The blockchain service-oriented architecture is composed of three essential components;

1. The directory: Where smart contracts information are stored. This directory provides adaptable mechanisms for publishing and discovering SCs.
2. The smart contract consumer(SCC): it is an internal node that intends to execute a smart contract either to utilize its enclosed functions or to integrate it into business applications for its benefit.
3. The smart contract provider (SCP): it is an internal node that owns a SC and intends to publish it on the directory.

Figure 1 depicts our vision of the blockchain service-oriented architecture established over three essentials such as the directory, the UDL-SC, and the blockchain interaction protocol.

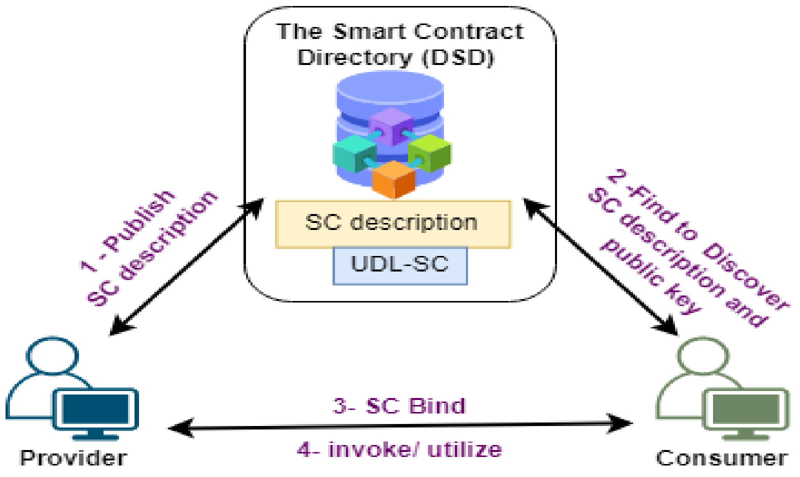

The Smart Contract Directory (DSD)
SC description
UDL-SC
1 - Publish SC description
2 -Find to Discover SC description and public key
3- SC Bind
4- invoke/ utilize
Provider
Consumer

Figure 1: The blockchain service-oriented architecture

The directory is called distributed Smart Directory (DSD) and represents the purpose of this research work. It is a new approach for publishing and discovering UDL-SC description metadata demonstrated carefully in sect.6. The UDL-SC description language is the uniform description language of SCs demonstrated in sect.5 and our recently published work (2). Finally, the blockchain interaction protocol allows interaction between the SCP and

SCC based on the interaction protocol of the blockchain where the DSD is deployed.

5. The UDL-SC description meta-model

The Uniform Description language for SC (UDL-SC) is a description language that extends the Uniform service description language (USDL) (28). It is proposed in our previous work (2) in 2021 to obtain a simple and rich Sc description file, in our case, a JSON file, supported by almost all blockchain-based platforms. UDL-SC draws its power from two main architectural principles: its capability to describe the functional properties of a SC, and its capability to describe non-functional properties. According to USDL, the non-functional properties are divided into business and technical perspectives. Figure 2 exposes the designed meta-model of the UDL-SC description language.

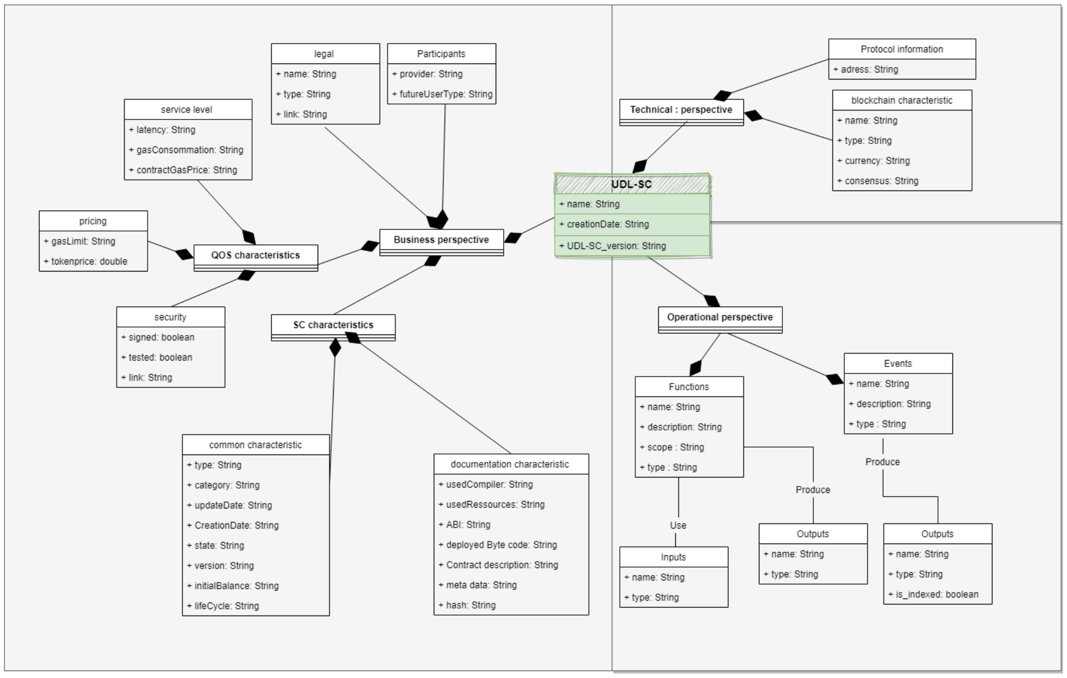


Figure 2: The Uniform Description language for SC (UDL-SC) Meta-model (2)

The meta-model shows description properties that can be associated with a SC and the relationships among them. The description of functional elements covers functions, events, and input/output types. The description of non-functional properties covers Qos, price, legal, and blockchain information which are considered as extra metadata. The blockchain information such as the "name", "type", "currency", and "consensus" are inserted into the UDL-SC description to identify SCs' blockchain information in the case of a federated blockchain platform. The UDL-SC has implemented Extra metadata to enhance the level of detail, expressiveness, and clarity in descriptions of SCs. By doing so, it promotes easier comprehension of the SCs, thereby motivating users to employ and run them. The UDL-SC description language aims to simplify the interaction with SCs, minimize their ambiguity, and help users to understand their functionality and their internal mechanism without accessing their source code. In addition, this description language helps SC providers to highlight SC properties so that they can get more profits by attracting users and increasing the accessibility to their contracts. The meta-model transformation process is performed using JSON rules based on a model-driven approach (MDA). A SC description generator (SCDG) is proposed in (2) to help developers to generate JSON UDL-SC description files automatically.

6. PROPOSAL: DISTRIBUTED SMART DIRECTORY

In this section, we present our Distributed Smart Directory called DSD. The DSD is an extension of the EbXml registry (3) and serves to store and retrieve SCs on the blockchain. Figure 3 exposes the architecture of our proposed directory.
The DSD directory architecture is based on a layered architecture that maintains separation of concerns advantages (10). The DSD is established based on the EbXml registry architecture (4)(3) by using the smart directory service (SDS) and the smart directory client (SDC) components. Furthermore, it is extended by two new components called smart directory data (SDD) and smart directory security (SDSEC) as illustrated by figure 3 (pink rectangle). We choose to extend the ebXML registry because its architecture is light and simple (4)(3), and can be adapted to blockchain technology easily using SCs. The stored SCs metadata is structured according to the Uniform Description language for SCs named UDL-SC presented in sect.5.
Our work's innovation relies on combining functional, business, and technical information of smart contracts into a directory to increase transparency.

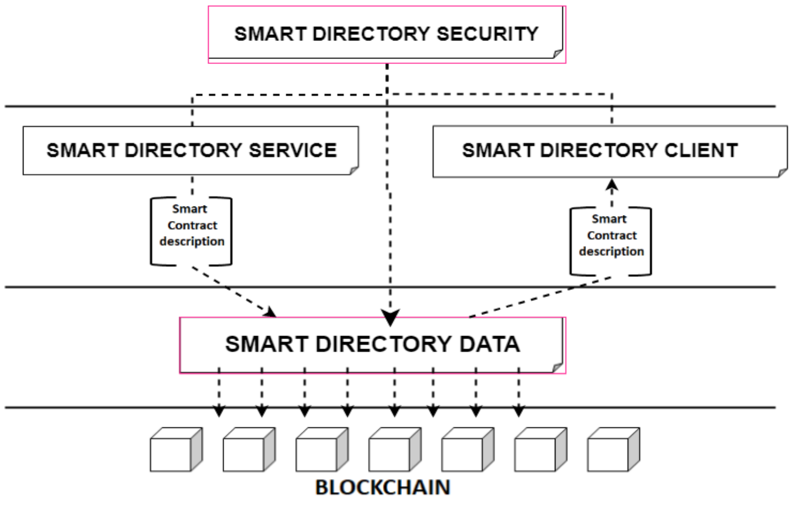


Figure 3: The Distributed Smart directory Architecture

Additionally, we have proposed a discovery system that efficiently retrieves Scs based on consumers' preferences. This will lead to enhanced trust between smart contract providers and consumers and improve accessibility to published smart contracts. Our proposed directory is an on-chain public solution available for all blockchain users. It enables the on-chain storage of Scs metadata to maintain immutability, security, and availability.

6.1. The smart directory service (SDS)

The smart directory service is the provider entry point to the DSD. It is the SC that allows collecting SCs descriptions from providers to make them available to general users. The smart directory service implements the **provider-actions interface** where a set of allowed interactions are described, such as publish function that allows SCP to publish a UDL-SC description, update function that allows SCP to update a UDL-SC description, and retract function that allows deleting a published description from the registry. These functions will be implemented in the SDS.
The DSD directory relies on existing consensus protocols of the blockchain that implements it. The designed system collects information's metadata from providers' transactions (payable transactions) and stores it on the blockchain according to the smart directory data (SDD) data structure.

6.2. The smart directory security (SDSEC)

The Smart directory security is implemented following the Identity Access Control methodology (27) used to manage and control access to resources within an organization or systems. The SDSEC involves several mechanisms;

- **Password-based authentication**, each provider has to authenticate to publish and manage his SCs on the DSD. This process allows for verifying the identity of the provider attempting to access the DSD. The goal of authentication is to ensure that only authorized providers are granted access to published SCs, and to prevent unauthorized access.

- **Role-Based Access Control (RBAC)** It is a security model that controls access to resources based on roles. In fact, access rights are assigned to roles, and users are assigned to those roles. Access to DSD features is then granted based on the roles that the user has been assigned to, based on their public key. On the DSD we have two roles, provider and consumer. We have assigned the "Add, Update, Retract, Querying" actions to the provider role, and the "Querying" action to the consumer role. Therefore, each time a new Provider is registered on the DSD, its public address is assigned to the provider role. The benefits of RBAC include increased security, improved compliance, simplified administration, and greater flexibility in managing access rights.

- **Identities-based authorization**: it is a security model that grants or restricts access to a SC description based on a user's identity (public key). This mechanism is adopted to prevent other providers from updating or retracting a not owned SC metadata. Specifically, it limits the "Updates/Retracts" access right to the provider who created the contract, as determined from the SC description metadata, by the public key value of the provider. In addition, to avoid fake metadata publishing on the DSD, this mechanism, forces that, the SC Provider is one of the parties of the smart contract agreement, and his address is used to send and receive tokens during the execution of the SCs. This will be performed based on the UDL-SC description metadata extracted from the participant description sub-class.

In addition, by its foundation on the blockchain, the DSD improves the trust between the provider and the consumer of the SCs. In fact, by uploading SCs and their metadata on-chain on the same blockchain platform, the DSD will be more trusted by nodes, and the SC consumer will be sure that the published transactions are verified by the same consensus mechanism that all nodes already trust.

6.3. The smart directory data (SDD)

The SDD manages DSD data flow. Figure 4 presents the DSD data flow model.

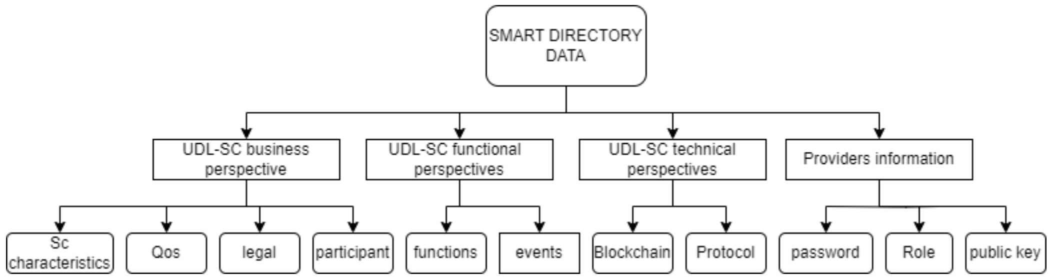


Figure 4: The DSD Data information model

It includes SC description metadata and providers metadata. These information are persistently stored in the blockchain network. The SC description metadata is structured according to the UDL-SC meta-model(2). It is divided into four essential structures. The first structure encapsulates the business perspectives including; QOS, legal, security, and pricing information. The second and third structures encapsulate technical and functional perspectives. Finally, the fourth structure encapsulates the provider's authentication information and the role used by the SDSEC.

6.4. The smart directory client (SDC)

The smart directory client is the consumer entry point to the DSD. This contract implements **the request-actions interface** that describes discovery and retrieval methods. The discovery and retrieval processes are performed based on the UDL-SC metadata deployed on the DSD by the provider. The SC consumer specifies the searching criterion through a web user interface and sends a search request, so the SDC undertakes to seek the suitable contract description to retrieve it for the SC consumer.

7. CONCEPTUAL OVERVIEW OF THE DSD

In this section, we will present the conceptual overview of our proposed DSD platforms. Figure 5 exposes the component diagram where the physical aspects of our solution are presented. According to this diagram, the four essential components introduced in sect.6 are designed to be implemented using SCs. These SCs will be deployed on the blockchain and their public

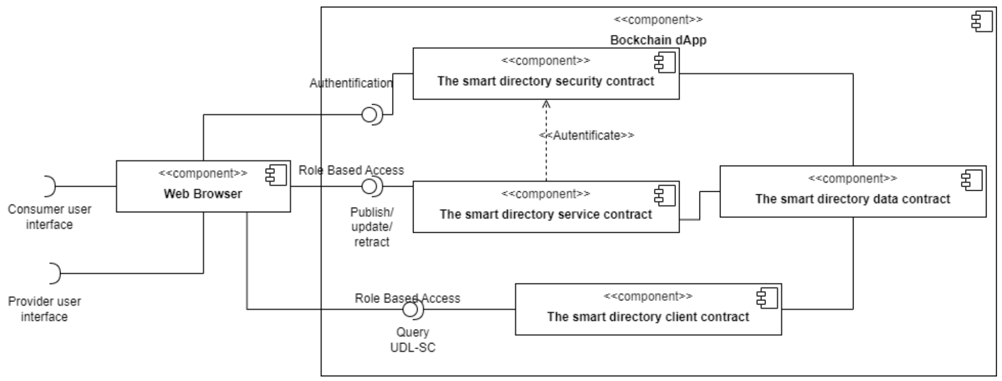


Figure 5: The Distributed Smart Directory component diagram

addresses will be used as a reference to invoke their encapsulated features. To invoke a function within the smart contract, a transaction must be executed. This transition includes the contract address, the function to be called, and any required parameters for that function. These SCs receive data from a web browser component and send data to the blockchain component. Based on this diagram, the DSD software architecture follows the 3-tier architecture principle, where;

- **The presentation tier**: is a DAPP application, that provides two web interfaces, the first is for the SCP that enables the sign-in, log-in, publish, update, and retract functions. The second is for SCC which enables smart contract discovery and retrieval functions.

- **The data tier**: presented essentially by the smart directory data (SDD)that represents the data structure used to manage and organize information about SCs and providers on the directory. They are persistently stored in the blockchain network according to the information model of the smart directory data presented in sect.6.3.

- **The application tier**: is composed of the two interfaces that describe user and provider actions and protocols, such as; *the provider-actions* interface implementation by the SDESC and the SDS. And the *request-actions interface* implementation by the SDC.

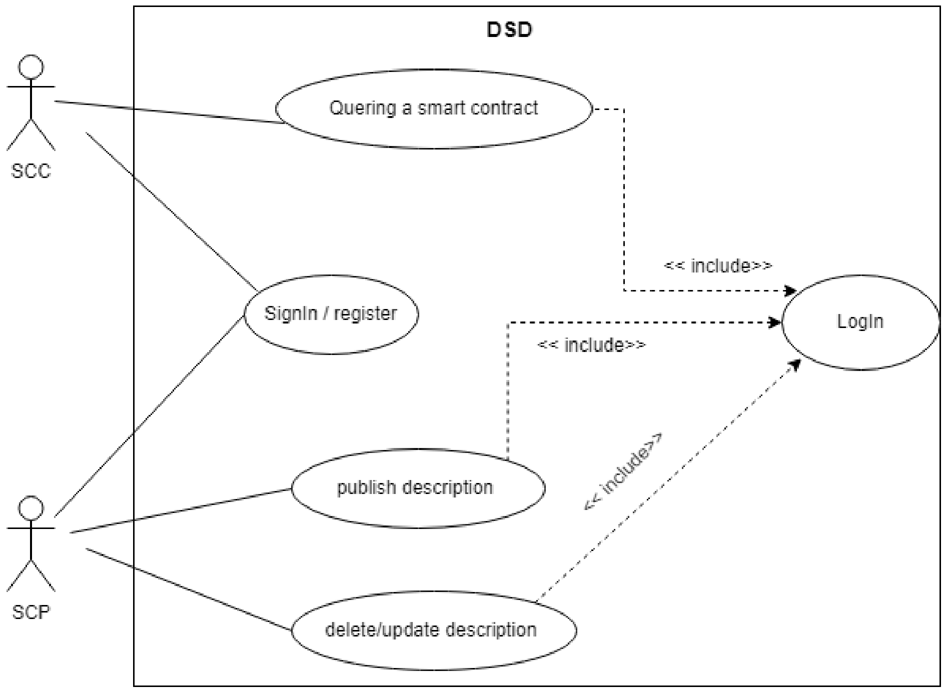


Figure 6: Use case diagram of the Distributed Smart Directory

We choose to use SC technology in the implementation of the DSD to provide a fully distributed solution on all nodes of the Blockchain networks. The goal is to achieve the non-repudiation, immutability, and high availability of the DSD component based on blockchain technology‘s data sharing and replication proprieties. In addition, SCs are used to prevent data modification (contract agreements) and to maintain a high level of trust.
To facilitate synchronization and accessibility to the deployed SCs, the proposed DSD is deployed on the same blockchain where the described SCs are deployed and will be available for all blockchain participants.

Figure 6 exposes the use case diagram that demonstrates the different ways that an actor might interact with our system. This diagram shows two involved participants, the SC provider (SCP) and the SC consumer (SCC).

The SCP must sign up to register on the DSD system. After that, he can log in, publish, update, or retract SC descriptions. The SCC can only query SC descriptions. The SC provider must exist on the same blockchain where the SC and the SC description are deployed because it is one of the SC parties, and its public address is used to transfer and receive tokens.

7.1. The provider-actions interface

The provider actions interface is implemented by the SDS and the SDSEC components. This interface groups SCP-authorized methods signatures such as publish, update, retract, sign-up and sign-in. The class diagram exposed in

Algorithm 1 Publish a SC description on the DSD

```
input : a UDL-SC description **UDL-SCdesc**, and SCP public key **PPK**
output : event message
Begin
if dontExist(UDL-SCdesc, PPK) then
    Store the UDL-SCdesc metadata in the SDD object on the blockchain.
    emit descriptionPublished("description is published successfully!").
else { emit descriptionExist("SC description already Exist!").}
end if
End algorithm.
```

Figure 7 illustrates the structure of the DSD by exposing the SCP SCs, their attributes, operations, and the relationships among the SDS, the SDSEC, and SDD smart contracts. The SDS implements the publish, update, and

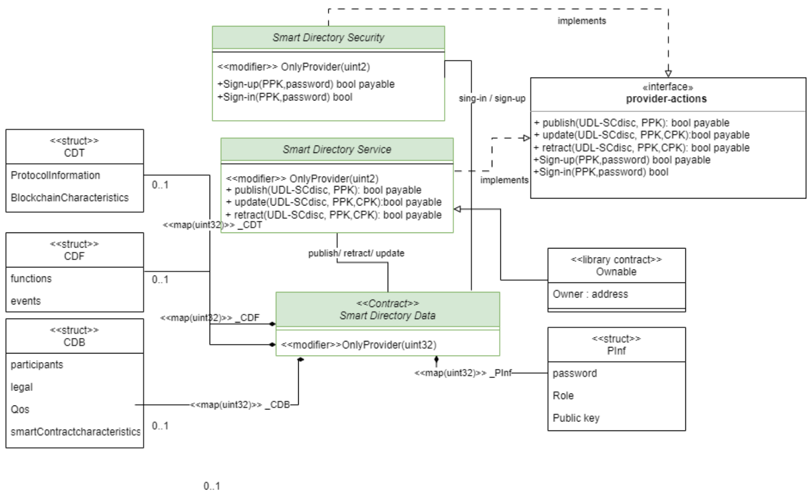

implements
Smart Directory Security
<<modifier>> OnlyProvider(uint2)
+Sign-up(PPK,password) bool payable
+Sign-in(PPK,password) bool
sing-in / sign-up
«interface»
provider-actions
+ publish(UDL-SCdisc, PPK): bool payable
+ update(UDL-SCdisc, PPK,CPK):bool payable
+ retract(UDL-SCdisc, PPK,CPK): bool payable
+Sign-up(PPK,password) bool payable
+Sign-in(PPK,password) bool
<<struct>>
CDT
ProtocolInformation
BlockchainCharacteristics
0..1
Smart Directory Service
<<modifier>> OnlyProvider(uint2)
+ publish(UDL-SCdisc, PPK): bool payable
+ update(UDL-SCdisc, PPK,CPK):bool payable
+ retract(UDL-SCdisc, PPK,CPK): bool payable
implements
<<map(uint32)>> _CDT
<<struct>>
CDF
functions
events
0..1
publish/ retract/ update
<<library contract>>
Ownable
Owner : address
<<map(uint32)>> _CDF
<<Contract>>
Smart Directory Data
<<modifier>>OnlyProvider(uint32)
<<struct>>
CDB
participants
legal
Qos
smartContractcharacteristics
<<map(uint32)>> _CDB
0..1
<<map(uint32)>> _PInf
<<struct>>
PInf
password
Role
Public key
0..1

Figure 7: SC Class diagram for the SDS, the SDSEC, and SDD SCs

retract functions, that allow managing UDL-SC descriptions on the DSD.

The algorithm 1 called "Publish a SC description on the DSD" is invoked by the constructor of the SDS when a SCP wants to publish a SC description on the DSD. This algorithm takes as input the SC UDL-SC description and the SCP public key and emits an event message that informs the SCP that his metadata is published or not on the DSD. One UDL-SC description for each contract is authorized based on the public address of the contract. Figure

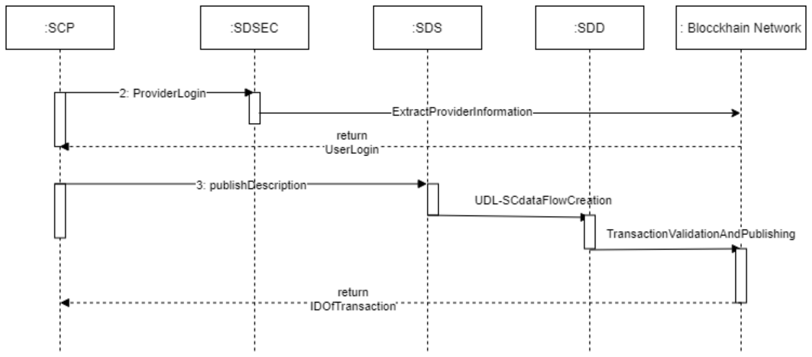


Figure 8: Sequence diagram of publishing a SC description on the DSD

8 exposes the sequence diagram of publishing metadata on the DSD. The sequence diagram illustrates the interactions between actors and the system in chronological order.

The algorithm 2 called "Retract and update a SC description on the DSD" is invoked by the constructor of the SDS when an SCP wants to update or retract a SC description from the DSD. For the update, this algorithm takes as input the SC public key, the SCP public key, and a UDL-SC description. In case of retraction, it takes the same cited input, but with an empty UDL-SC description. As output, this algorithm emits an event message that informs the SCP whether his description is updated or not. When the updating or retracting process is performed, a new SC description for the same SC is published on the DSD using the Algorithm 1. The older SC description transaction will not be dropped from the DSD directory because of the immutability nature of the blockchain, but, when a user or a provider searches it, its latest published version will be retrieved. The older Sc description will be assigned as an invalid transaction by the network users. The SDSEC implements the sign-up and sign-in functions, that allow the registering and authentication of an SCP on the DSD. The algorithm 3 called "SCP sign-up and sign-in" is invoked by the constructor of the SDSEC when a blockchain

Algorithm 2 Retract and update a SC description on the DSD

```
input : a UDL-SC description UDL-SCdesc, the SC and SCP public key PPK, CPK
output : event message
Begin
if search(UDL-SCdesc, PPK, CPK) then
    Retrieve the UDL-SCdesc older version and assign it as invalid.
    Store the UDL-SCdesc metadata in the SDD object on the blockchain using the algorithm 1.
    emit descriptionUpdated("description is Updated successfully!").
else { emit descriptionDontExist("SC description don't Exist!"). }
end if
End algorithm.
```

user wants to publish a UDL-SC description on the DSD. This algorithm takes as input the SCP public key and the password. As output, this algorithm emits an event message that informs the SCP whether he is successfully logged in or is registered on the DSD. If the SCP public key is already stored on the SDD, and the user writes a correct password, he will be redirected to the home page where he can visualize only the list of his published contracts. Else, this algorithm will transfer mining fees from the SCP balance to the miner balance to store its information on the DSD and him to the empty home page.

Algorithm 3 SCP sign-up and sign-in

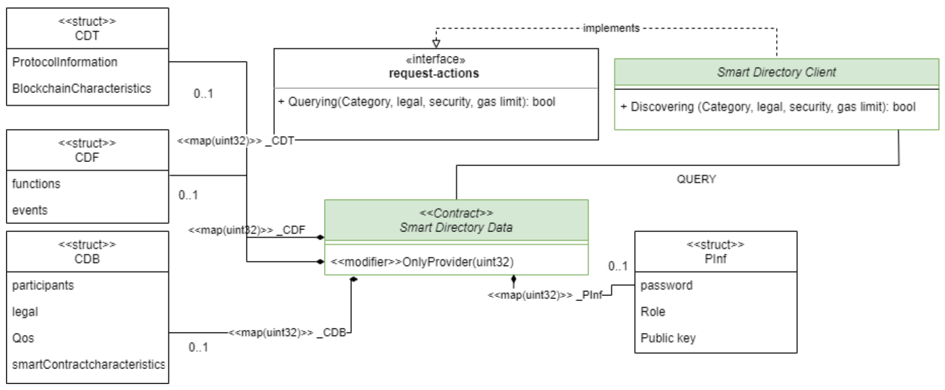
```
input : SCP public key PPK, and password PSW
output : event message, redirection link
Begin
if search(PPK, PSW) then
    Redirect the SCP to his home page if correct(PSW).

    emit providerlogIn("Login successfully!").
else {
(1)Store the provider metadata in the SDD object on the blockchain.
(2)Redirect the SCP to his home page.
(3) Save this user as provider
(4) emit providerSignedUp("The provider is registered successfully!").}
end if.
End algorithm.
```

The publish, update, retract, and sign-up algorithms executions are payable, so the SCP should pay mining fees, otherwise, if he doesn't have the adequate balance, the invocation fails. To avoid unauthorized publishing, updates, and retracting, the Modifier function is used. Modifier functions are used to modify the behavior of a function or to add a prerequisite. onlyProvider Modifier guarantees that only the SCP can invoke publish, update and retract functions. These functions fail if the Modifier **require** instruction fails.

7.2. The request-actions interface

The request-actions interface is implemented by the SDC component. In this interface, we defined the signature of the querying method. The class



<<struct>> CDT
ProtocolInformation
BlockchainCharacteristics
<<struct>> CDF
functions
events
<<struct>> CDB
participants
legal
Qos
smartContractcharacteristics
implements
«interface» request-actions
+ Querying(Category, legal, security, gas limit): bool
Smart Directory Client
+ Discovering (Category, legal, security, gas limit): bool
QUERY
<<Contract>> Smart Directory Data
<<modifier>>OnlyProvider(uint32)
<<struct>> PInf
password
Role
Public key
0..1
<<map(uint32)>> _CDT
<<map(uint32)>> _CDF
<<map(uint32)>> _CDB
<<map(uint32)>> _PInf

Figure 9: SC Class diagram for the SDC Sc.

diagram exposed in Figure 9 illustrates the structure of the DSD by showing the system's SDC Sc, its attributes, operations, and the relationships among the SDC, and SDD components.

The algorithm 4 called "Discovering Smart contracts" is invoked by the constructor of the SDC when a blockchain user wants to query a UDL-SC description from the DSD. SCC can query on the combination of four parameters (Category, legal, security, gas limit) and return from the SDD any UDL-SC description that matches this query. It effectuates a linear search (29) through a collection of SCs descriptions in a linear manner until target SCs are found. First, this algorithm 4 extracts all contracts according to the chosen category. After that, three filters will be executed on the retrieved SCs to find SCs that meet user preferences, respectively;

- security filter: Retrieves contracts with non-empty security information, these contracts are secured and tested by a SC audit service. An audit service executes the SC automatically whenever a network tries to access it for a transaction requested by a user to discover cyberattacks and potential vulnerabilities and correct them.
- legal filter: Retrieves contracts with non-empty legal information, these contracts are legally binding and enforceable if they comply with the contract law of the government.
- gas filter: allows retrieving the smart contracts based on the chosen gas consumption thresholds.

We choose to use the linear search known as sequential search because (29) it represents a good choice for small collections, in our case SCs metadata, as it requires minimal processing overhead. The algorithm 4 returns as output a message if the SDD is empty, else it will return the list of smart contracts that meet the SCC query.

Algorithm 4 Discovering Smart contracts

```
input : Category , legal, security, gas limit
output : Set of SCs or message
Begin
results: an empty Set of contract addresses.
if IsEmpty(SDD then
    emit Empty("directory is empty!").
else
    results <- Extract contracts description based on the category param.
    results <- Extract contract addresses based on (legal, security) parameter values.
    results <- Eliminate SC address out of the requested (gas limit) parameter value.
end if.
return results.
End algorithm.
```

8. Implementation and Evaluation

To evaluate the feasibility of the proposed directory, we implemented the DSD as a full DAPP application on the Ethereum platform [6]. We used RemixIDE as a development environment to code Solidity smart contracts, and Metamask as a blockchain provider to publish smart contracts and sign transactions sent. The implementation source code can be inspected online [7].

The four smart contracts; SDS, SDC, SDSEC, SDD; introduced in sect.6 are implemented in Solidity language (20) and contain all the algorithms introduced in sect.7. To support the upgrade-ability of our solution in case of future development or bug fixing, we have deployed our smart contracts using the OpenZeppelin Upgrades Plugins [8]. By choosing this option, we can upgrade our smart contracts by making changes to their code while maintaining their balance, address, and state.

For the front end of our DApp, we used the React.js Javascript library, HTML5, CSS3, and JavaScript. The front end is composed of two user interfaces according to user types. The SCC user interface allows the discovery and retrieval of SC descriptions based on a set of Discovering parameters.

[6]https://dsd-iota.vercel.app/#

[7]https://github.com/wafa2011/DSD

[8]https://docs.openzeppelin.com/upgrades-plugins/1.x/

The SCP user interface allows the publishing/updating/retracting SCs meta-data, in addition to the sign-up and sign-in services.
To connect our Scs with the front end we used Web3.js and then we deployed them on the polygon L2 side chain networks. We have used the polygonscan website [9] to find users' wallets by their public address and visualize the transactions sent. The SCC user interface is connected to the SDC presented in sect.6.4, and the SCP user interface is connected to the SDS and SDSEC presented in sect.6.2 and sect.6.1. We will use the blockchain to store DSD transactions according to the SDD Data information model presented in sect.6.3. This contract is implemented using Solidity where we define a multi-mapping structure that stores the state information to be shared among the blockchain networks. In the data structures of DSD, we define the representation of the UDL-SC description meta-model and provider information. As illustrated in Figure 4, each struct represents a record of the UDL-SC and provider descriptor metadata. To associate the representation structs to an entity, four elements similar to hash tables, called mapping, are created. Mapping allows efficient key-value lookup. The three first mappings are used to store the UDL-SC functional, business, and technical description meta-data struct called respectively *CDF, CDT, and CDB*, where we assign to each entry a unique integer ID to serve as the mapping key. To store UDL-SC metadata, the three mappings will be updated at the same time. So, they generate the same ID for a SC UDL-SC record. The fourth mapping is used to store the provider meta-data struct called *PInf* (provider information) and assign a unique integer ID to serve as the mapping key to each entry. In the SCP web application, the publish/update functions store smart contracts information on the *CDF, CDT, and CDB* mapping struct. The retract function allows deleting a SC description from these three mapping structs based on the mapping key (ID). The SignIn function stores provider information in *PInf* mapping struct. All the proposed methods in the SCP application use the hash function **Keccak32** (21) available in the Ethereum platform to encode transactions sent. To retrieve these transactions and decrypt their hash's, we used the **abi-decoder library** [10].
For evaluation purposes, the Solidity smart contracts were hosted on the polygon L2 side chain, we get test MATIC tokens from the Polygon Faucet web-

[9]https://mumbai.polygonscan.com/
[10]https://github.com/ConsenSys/abi-decoder

site [11]. According to CoinMarketCap [12] inspected during our experiments, 1 Matic is equal to 0,81637 USD. The experimentation is conducted on a SCs description data set [13] called "UDL-SCdescriptions". UDL-SCdescriptions is composed of 400 JSON files [14]. Each description file contains metadata about one SC generated by the UDL-SC description generator proposed by (2). The deployed smart contracts were experimented upon by imitating the SCP and SCC actions processes. Table 2 shows the gas cost sent for each method invocation of smart contracts.

Table 2: Experiment results for method invocations

DSD Solidity Smart Contract Methods		
Functions	**Transaction Fee (MATIC)**	**Transcation Fee (ETH)**
Publish	2,197311	0.00160415
Update	1.035802	0.00075619
Retract	0.318916	0.00023501
Sign in	0.087023	0.00006412
Log in	0,054176	0.00003955
Discovery	1,025835	0.00075591

Through a careful examination of the results, the methods 'Publish', 'Update', 'Retract' and 'Sign in' require an amount of gas to process bigger than the 'Login' and 'Discovery' methods. In fact, 'Publish', 'Update', 'Retract' and 'Sign in' methods change the state of the blockchain by registering new data, which requires big transaction fees. Whereas, 'Login' and 'Discovery' methods enable the querying on the blockchain without any change of the data. We notice that the methods 'Publish' and 'Update' take much more gas than 'Retract', and 'Sign in' because the stored data by the two first methods is bigger than the stored by the seconds. In addition, when compared, the 'Publish' function execution costs are higher, as during the execution, a new SC description data record is created, as opposed to the 'Update' when only

[11]https://faucet.polygon.technology/

[12]https://coinmarketcap.com/fr/

[13]https://github.com/wafa2011/UDL-SCdescription-data-set

[14]We were unable to obtain enough resources to run with more descriptions metadata at the time of writing.

the specific SC description data record values are updated. The method 'Login' consumes less gas compared to the 'Discovery' method because it allows only extracting from the *PInf* struct where providers' information is stored, which is considered smaller than the *CDF, CDT, and CDB* structs where the SC descriptions are stored. In conclusion, throughout the workflow execution, this implementation displayed reliable behavior and recorded the data efficiently on the blockchain.
For the performance experiment of the DSD system, we tested the transaction throughput of the proposed publishing algorithm by creating a stress insertion test of 400 records and measuring the memory, and time required for the insertion algorithm. A server with an Intel Core i7-7500U CPU featuring 8 GB of RAM with data stored on the IDE storage was used to host this experiment. Figure 10 displays the results of the publishing performance test of the DSD on the polygon L2 side chain. Upon conducting a comprehensive analysis of the data displayed in Figure 10, it was observed that the insertion time for new records into the DSD increases linearly as the number of records in the structure grows. This linear growth in insertion time provides a favorable balance between performance and efficiency, making it a beneficial attribute for the DSD. For example, the algorithm inserts 150 records in 400ms and 350 records approximately in 900ms. The insertion time on the blockchain can be affected by factors such as network congestion, computational resources available to the network, and the size of the blockchain itself. Moreover, upon examining the outcomes of memory usage, it was observed that the DSD insertion algorithm consumes approximately 900 MB of memory for each record that is inserted. These findings indicate that the algorithm's memory usage falls within an acceptable performance value.

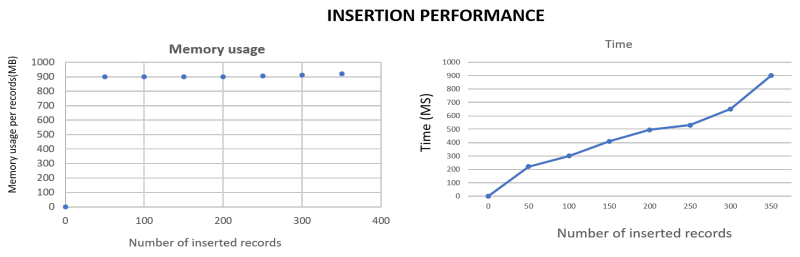


Figure 10: Publishing performance test of the DSD on polygon L2 side chain

For the performance experiments of the distributed discovering algorithm, we tested the time to execute four field queries (category, legal, security, and gas) in the DSD with several entries for 50 random queries. The category parameter is a text field that enables the SCU to input keywords describing the desired SC, whereas the legal and security parameters are represented as checkboxes, allowing users to indicate whether they require a legally-endorsed and secured SC. Lastly, the gas parameter is presented as an option button, enabling users to select one gas threshold option from a predefined set that specifies the maximum gas consumption for the required SC. Figure 11 exposes the recorded discovery execution time results.

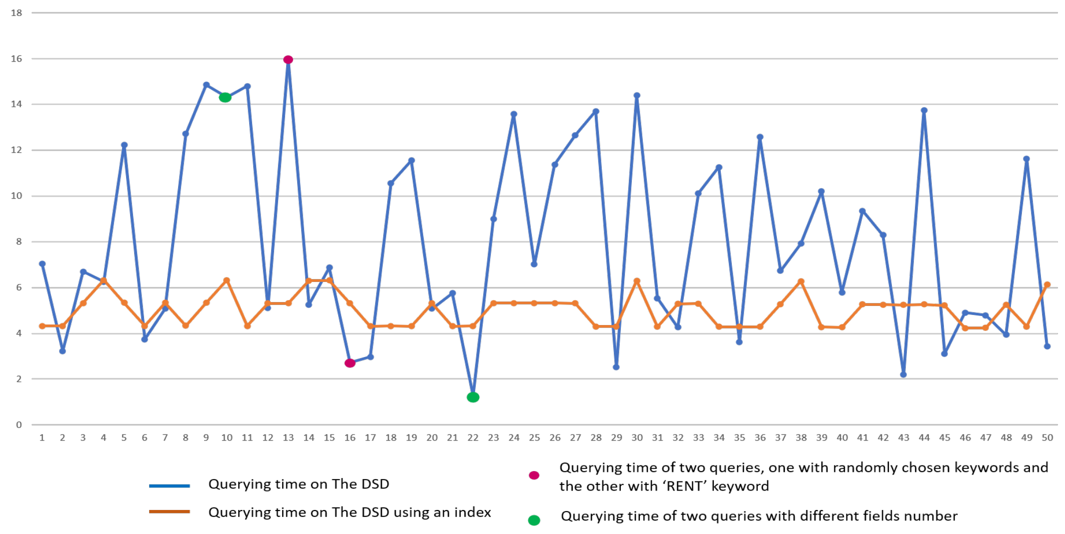
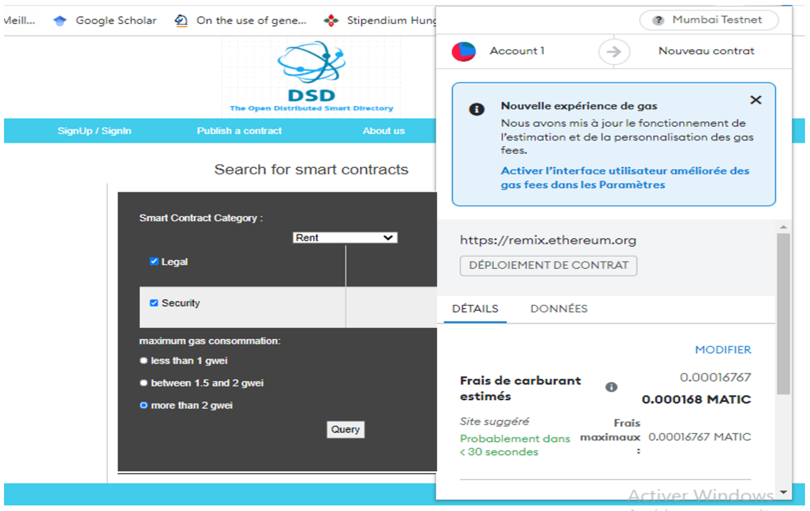


Figure 11: Discovery execution time for 50 random queries

The blue curve in Figure 11 depicts the time for each discovery method invocation of 50 queries. Upon analyzing the computed time, it was observed that the querying time is influenced by multiple factors. First, it depends on the size of the query, more precisely, as the number of non-empty query fields increases the time to retrieve SCs increases. Green points in Figure 11 show the time variation when executing two query types. When running a query with one field; just the category field; the time is equal to 1.6 ms. Whereas, when executing a query with four fields(category, legal, security, and gas), the time is equal to 14.2ms.

Second, the used keywords in the category text field can impact querying time. In fact, the utilization of inaccurate, or excessively lengthy keywords,

will lead to a prolonged search time. Consequently, the algorithm will have to scan through all the description records on the DSD to match the query, but may not be able to find any relevant results. Purple points in Figure 11 show the time variation when executing two similar-size queries using only the category field. The first query uses an inaccurate and very long keyword and takes time equal to 16.2 ms, whereas the second query uses the 'RENT' keyword and takes time equal to 2.9 ms.
Finally, the indexing of the data set impacts the querying time. We have executed the same queries on the same data set after indexing it based on the "category" attribute. The category attribute exists in the UDL-SC description under the smart contract common characteristic class. We found eight smart contract categories ('Rent',' Bank ',' Sale',' 'Copyright',' NFT',' Voting',' Math',' Financial '). In the SDD SC, we have added a new struct called index, and for each category, we have added a new sub-struct where we have stored all SCs public key that belongs to this category. To associate this sub-struct to an entity, we used in the code a new mapping called index, where we assign to each public key an ID. This ID is also extracted from the SDD SC presented in sect.6.3 from the three mappings where we store the complete UDL-SC description metadata. Figure 12 exposes the web interface developed for the interaction with the DSD.



Smart Contract Category : Rent
☑ Legal
☑ Security
maximum gas consommation:
● less than 1 gwei
● between 1.5 and 2 gwei
○ more than 2 gwei

Nouvelle expérience de gas
Nous avons mis à jour le fonctionnement de l'estimation et de la personnalisation des gas fees.
Activer l'interface utilisateur améliorée des gas fees dans les Paramètres

Frais de carburant estimés 0.00016767 0.000168 MATIC
Site suggéré
Probablement dans < 30 secondes
Frais maximaux : 0.00016767 MATIC

Figure 12: Web interface developed for the interaction with the DSD

At the same time we have changed the text box of the category field by a select box option illustrated in figure 12. This new component provides the list of categories from the index automatically, and will dynamically be updated if a new category is added to the index. Figure 11 displays the querying time for 50 random queries based on the new index, depicted by the orange curve, and demonstrates how the index decreases the querying time. Compared to the sequential searching approach, Mapping elements allow efficient and fast key-value lookup. Practically, we have extracted the SCs that belong to the needed category, after that, we have used their key "ID" to extract their metadata from the other three mappings presented in sect.6.3. Finally, we have filtered the set of retrieved Scs using security, gas, and legal filters. Using the index and the selection box features reduces the time required for discovery, while also preventing wasteful searches due to the use of incorrect keywords. To evaluate the quality of the results of the discovery of Scs, we have calculated the Precision, Recall, and F1 score of four different queries chosen randomly. Table 3 exposes the four different queries in addition to the retrieved Scs numbers.

Table 3: SC discovery based on 4 different queries

Query	**SCs**	F.P	T.P	F.N	T.N
Q1["category"="Math" "security"="true" "legal" = "true" "maximumGas"="O"]	70	20	50	40	310
Q2["category"="Financial" "security"="true" "legal" = "false" "maximumGas"="2 gwei"]	130	60	70	20	270
Q3["category"="Voting" "security"="true" "legal" = "true" "maximumGas"="1gwei"]	40	10	30	20	360
Q4 ["category"="NFT" "security"="true" "legal" = "false" "maximumGas"="O"]	30	10	20	0	370

F : false T: true P: positive N: negative

The definition of precision pertains to the ratio of relevant and accurate Scs to all Scs returned by the discovery algorithm. The Recall is defined by the proportion of the relevant Scs of all the existing Scs that have been published on the DSD. Finally, the F1 score is a measure of accuracy that considers both its precision and recall. It is the harmonic mean of precision and recall, with a value between 0 and 1. A high F1 score indicates that the model has both high precision and high recall, whereas a low F1 score indicates that either precision or recall is low. Figure 13 exposes the experimental results conducted for effectuated tests.

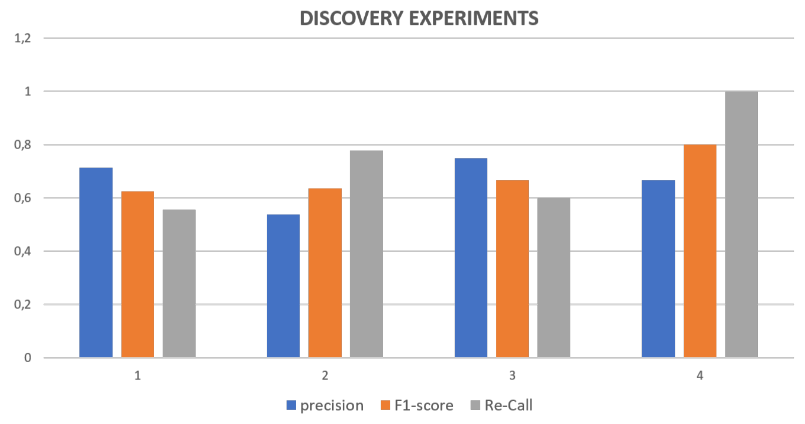


Figure 13: recall, precision, and F1 score results of the Discovering algorithm execution for four queries

After a thorough investigation of the results, we notice that the proposed algorithm retrieves Scs with an acceptable precision score which indicates that it makes few false positive predictions, while the calculated recall score indicates that it is good at identifying actual positive cases. These results are reflected in the F1 score results, especially on the fourth query where the F1 score value is equal to 0.8.

9. Discussion

Based on performance test results, the DSD proves the importance of the proposed functionalities and shows that it improves the accessibility to SCs by encouraging SCC and SCP to use the proposed algorithms to publish, retrieve and discover SCs. In fact, the publishing/updating of SC descriptions is a not frequent task, as we see in the web service context. More precisely, the provider will publish once a SC description, so its expenses will not be very high compared to the profit that he will earn after the execution of the SC by a user.
Since its distributed property, the proposed DSD resolves the limits of existing systems presented in sect.3.3 because ;

- It is a full on-chain solution,
- It allows storing SC metadata,

- The stored metadata is very expressive because it includes functional and non-functional properties of the SC,
- It increases information availability by enabling publishing, searching, and retrieving SCs via a simple user interface.
- It achieves synchronization because each node in the network has a replication of the DSD and all its data.

Besides, with its foundation on blockchain technology, the proposed DSD resolves the limits of centralized registry systems such as;

- **Records vulnerability and risk of loss:** The DSD relies on the consensus mechanisms of the blockchain where it is deployed to assure the verification and the validation of any requested transaction (a record).
- **Delay in bringing records and single point of failure:** this problem is resolved by the data sharing and replication in each blockchain network node.
- **Very difficult to maintain secret and confidential transaction:**this problem is resolved by encrypting all transactions using a cryptographic algorithm and hash function.
- **Records falsification:** this problem is resolved by the immutability nature of the blockchain, defined as the ability to remain unaltered, unchanged, and indelible. Accordingly, published information on the DSD is ensured from fraud and data violence.
- **Member loyalty and delay in decision-making problems:** this problem is resolved by implementing the logic on SCs that enforce and automate the execution of any agreement without depending on a third party member.

We conclude that our proposed DSD keeps the advantages of decentralization, promotes trust between parties, reinforces data security, and achieves records synchronization.

10. CONCLUSION

In this paper, we have sketched a fully on-chain distributed directory called DSD for SC metadata. This directory will help providers to share information on SCs with blockchain users to facilitate its utilization. So, they can understand SC functionalities without accessing the source code. The proposed DSD is an extension of the ebXML directory that enables many properties such as publishing UDL-SC metadata and searching for UDL-SC releases and instances. The evaluation of the DSD directory shows that using blockchain technology improves the availability of information, and supports transparency between parties. By its foundation on the unified SC description language called UDL-SC, and the EbXML registry standard, the adoption of this solution to other blockchain platforms supporting smart contracts is feasible through customization of the UDL-SC meta-model and the implementation of the DSD according to the chosen platform requirements and coding languages. This would enhance the applicability and interoperability of the proposed solution across different blockchain ecosystems. Our next step is to build a new platform for blockchain clients that allow them to search and retrieve SCs using different techniques inspired by what we see in the literature review. Furthermore, our intention is to place greater emphasis on the examination of functional properties of the smart contract throughout the process of discovery. Functional properties include function name, input, output, data, processing logic, and exception management. Also, we intend to implement new mechanisms for the SCs selection, recommendation, and comparison based on SCU preferences. Furthermore, we intend to implement an SLA management system (26) to ensure the outlined contract terms and identify when parties breach the contract based on the UDL-SC description file to prevent fake metadata insertion problems. Finally, we propose to improve the transaction processing capabilities of the DSD system by implementing mechanisms such as sharding or parallel processing of the publishing algorithm. These techniques will allow transactions to be processed concurrently across multiple nodes, thereby distributing the processing workload and reducing the risk of overload on any single node.

References

[1] GARCÍA-BARRIOCANAL, Elena, SÁNCHEZ-ALONSO, Salvador, et SICILIA, Miguel-Angel. Deploying metadata on blockchain technologies.

In : Metadata and Semantic Research: 11th International Conference, MTSR 2017, Tallinn, Estonia, November 28–December 1, 2017, Proceedings 11. Springer International Publishing, 2017. p. 38-49.

[2] SOUEI, Wafa Ben Slama, EL HOG, Chiraz, SLIMAN, Layth, et al. Towards a Uniform Description Language for Smart Contract. In : 2021 IEEE 30th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE). IEEE, 2021. p. 57-62.

[3] KOTOK, Alan et WEBBER, David RR. ebXML: the new global standard for doing business over the internet. Sams Publishing, 2001.

[4] HOFREITER, Birgit, HUEMER, Christian, et KLAS, Wolfgang. ebXML: Status, research issues, and obstacles. In : Proceedings Twelfth International Workshop on Research Issues in Data Engineering: Engineering E-Commerce/E-Business Systems RIDE-2EC 2002. IEEE, 2002. p. 7-16.

[5] KONASHEVYCH, Oleksii. Cross-blockchain protocol for public registries. International Journal of Web Information Systems, 2020, vol. 16, no 5, p. 571-610.

[6] LAMPARELLI, Andrea, FALAZI, Ghareeb, BREITENBÜCHER, Uwe, et al. Smart contract locator (scl) and smart contract description language (scdl). In : Service-Oriented Computing–ICSOC 2019 Workshops: WESOACS, ASOCA, ISYCC, TBCE, and STRAPS, Toulouse, France, October 28–31, 2019, Revised Selected Papers 17. Springer International Publishing, 2020. p. 195-210.

[7] SANKA, Abdurrashid Ibrahim, IRFAN, Muhammad, HUANG, Ian, et al. A survey of breakthrough in blockchain technology: Adoptions, applications, challenges and future research. Computer Communications, 2021, vol. 169, p. 179-201.

[8] PIERRO, Giuseppe Antonio, TONELLI, Roberto, et MARCHESI, Michele. An organized repository of ethereum SCs source codes and metrics. Future internet, 2020, vol. 12, no 11, p. 197.

[9] SZABO, Nick. The idea of SCs. Nick Szabos papers and concise tutorials, 1997, vol. 6, no 1, p. 199.

[10] RICHARDS, Mark. Software architecture patterns. 1005 Gravenstein Highway North, Sebastopol, CA 95472 : O'Reilly Media, Incorporated, 2015.

[11] ABDELHAMID, Mohamed Moetez, SLIMAN, Layth, BEN DJEMAA, Raoudha, et al. ABISchain: Towards a Secure and Scalable Blockchain Using Swarm-based Pruning. In : Proceedings of the 2023 Australasian Computer Science Week. 2023. p. 28-35.

[12] SUNNY, Farhana Akter, HAJEK, Petr, MUNK, Michal, et al. A systematic review of blockchain applications. IEEE Access, 2022.

[13] IWATA, Kotono et OMOTE, Kazumasa. Automatic Monitoring System for Security Using IoT Devices and Smart Contracts. In : Advanced Information Networking and Applications: Proceedings of the 36th International Conference on Advanced Information Networking and Applications (AINA-2022), Volume 1. Cham : Springer International Publishing, 2022. p. 205-216.

[14] H.U.Y.-C. (2020, April 6). GitHub - Turing-Chain/Smart-Contract-Modular-Template: the open-source repository contains educational materials including SC templates for developers. GitHub. Retrieved February 1, 2022, from https://github.com/Turing-Chain/Smart-Contract-Modular-Template

[15] C. (2020a, September 28). GitHub corydickson/ethmetadataregistry: On-chain metadata registry for Ethereum SCs. GitHub. Retrieved March 7, 2022, from https://github.com/corydickson/eth-metadata-registry

[16] M. (2022, May 10). GitHub - MetaMask/contract-metadata: A mapping of ethereum contract addresses to broadly accepted icons for those addresses. GitHub. Retrieved June 21, 2022, from https://github.com/MetaMask/contract-metadata

[17] EL HOG, Chiraz, DJEMAA, Raoudha Ben, et AMOUS, Ikram. Adaptable web service registry for publishing profile annotation description. In : 2013 IEEE 10th International conference on ubiquitous intelligence and computing and 2013 IEEE 10th International conference on autonomic and trusted computing. IEEE, 2013. p. 533-538.

[18] EtherScan Documentation (Contracts) (2022) Etherscan. Available at: https://docs.etherscan.io/api-endpoints/contracts (Accessed: November 6, 2022).

[19] Get started with the graph documentation (2022) The Graph Docs. Available at: https://thegraph.com/docs/en/ (Accessed: November 6, 2022).

[20] DANNEN, Chris. Introducing Ethereum and solidity. Berkeley : Apress, 2017.

[21] KURTULMUS, A. Besir et DANIEL, Kenny. Trustless machine learning contracts; evaluating and exchanging machine learning models on the ethereum blockchain. arXiv preprint arXiv:1802.10185, 2018.

[22] DANIEL, Florian et GUIDA, Luca. A service-oriented perspective on blockchain smart contracts. IEEE Internet Computing, 2019, vol. 23, no 1, p. 46-53.

[23] WEBER, Ingo. Blockchain and Services–Exploring the Links: Keynote Paper. In : Service Research and Innovation: 7th Australian Symposium, ASSRI 2018, Sydney, NSW, Australia, September 6, 2018, and Wollongong, NSW, Australia, December 14, 2018, Revised Selected Papers 7. Springer International Publishing, 2019. p. 13-21.

[24] FALAZI, Ghareeb, LAMPARELLI, Andrea, BREITENBUECHER, Uwe, et al. Unified integration of smart contracts through service orientation. IEEE Software, 2020, vol. 37, no 5, p. 60-66.

[25] FALAZI, Ghareeb, BREITENBÜCHER, Uwe, DANIEL, Florian, et al. Smart contract invocation protocol (SCIP): A protocol for the uniform integration of heterogeneous blockchain smart contracts. In : Advanced Information Systems Engineering: 32nd International Conference, CAiSE 2020, Grenoble, France, June 8–12, 2020, Proceedings 32. Springer International Publishing, 2020. p. 134-149.

[26] HAMDI, Nawel, EL HOG, Chiraz, BEN DJEMAA, Raoudha, et al. A Survey on SLA Management Using Blockchain Based Smart Contracts. In : Intelligent Systems Design and Applications: 21st International Conference on Intelligent Systems Design and Applications (ISDA 2021)

Held During December 13–15, 2021. Cham : Springer International Publishing, 2022. p. 1425-1433.

[27] FERRAIOLO, David, FELDMAN, Larry, et WITTE, Greg. Exploring the next generation of access control methodologies. 2016.

[28] CARDOSO, Jorge, WINKLER, Matthias, et VOIGT, Konrad. A service description language for the internet of services. In : Proceedings of ISSS. 2009.

[29] GLASS, H. et COOPER, L. Sequential search: A method for solving constrained optimization problems. Journal of the ACM (JACM), 1965, vol. 12, no 1, p. 71-82.

[30] NABLI, Hajer, DJEMAA, Raoudha Ben, et AMOR, Ikram Amous Ben. Efficient cloud service discovery approach based on LDA topic modeling. Journal of Systems and Software, 2018, vol. 146, p. 233-248.