A Multi-Perspective Framework for Smart Contract Search in Decentralised Applications Design

Ada Bagozi[a], Devis Bianchini[b], Valeria De Antonellis[c], Massimiliano Garda[d] and Michele Melchiori[e]

Dept. of Information Engineering, University of Brescia, Via Branze 38, 25123, Brescia, Italy

Keywords: Multi-Perspective Model, Blockchain, Decentralised Applications, Smart Contracts, Semantic.

Abstract: With the advent of blockchain technology, many interorganisational collaborative processes that demand trust requirements (e.g., food supply chain, smart grid energy distribution and clinical trials) are being implemented as decentralised applications (DApps). Indeed, blockchain technology provides decentralised control and immutable transaction history, thereby improving security and accountability between parties. In this vision paper, we consider cooperative processes where a subject, which acts as a regulator of the process, promotes the use of blockchain for increasing transparency, while reducing the burden in controlling trustworthiness among participants. To the scope, the regulator provides a registry of basic smart contracts, including both actual deployed ones and code templates, that can be used and extended by the process stakeholders (e.g., retailers, energy providers, researchers) to build up DApps. The adoption of a blockchain and the definition of the registry favour the compliance with best practices and obligations demanded by the regulator, as well as that all relevant information and documents cannot be tampered. To support semantic-based smart contract search in the registry, we propose a multi-perspective framework that, in addition to classification and technical characteristics of smart contracts, takes into account the past experience of developers who have used smart contracts of the registry to develop DApps.

1 INTRODUCTION

With the advent of blockchain technology, many interorganisational collaborative processes demanding trust requirements (e.g., food supply chain, smart grid energy distribution and clinical trials) are being implemented as decentralised applications (DApps) (Cai et al., 2018). DApps are conceived as web applications that orchestrate smart contracts (SCs) to implement the business logic of the applications and also provide a Graphical User Interface to ease interactions among participants. Leveraging the blockchain, a DApp provides decentralised control and immutable transaction history, thereby improving security and accountability between parties. Smart contracts, along with distributed ledger technologies, have the potential to enforce automated negotiations and agreements between parties without the direct involvement and intermediation of central authorities, by providing public and trusted functionalities executed on top of the blockchain.

In this vision paper, we consider cooperative processes occurring in domains, e.g., clinical trials, where a subject, which acts as a regulator (i.e., that carries out regulatory activities in a given domain) of the process, promotes the use of blockchain for increasing transparency, while improving the trustworthiness among participants. In these processes, the regulator is in charge of overseeing the fulfilment of the necessary requirements for safety, quality and efficacy of the assets being supplied, providing also rules, authorisations and, possibly, technological frameworks to the involved parties.

To the purpose, the regulator subject may provide a registry of basic SCs that can be used and extended by stakeholders to set up DApps. Actually, in order to develop real-world DApps, reuse of SCs published in open access registries has become a common practice, as investigated in recent efforts (Tran et al., 2019; He et al., 2020).

[a] https://orcid.org/0000-0002-0193-6500
[b] https://orcid.org/0000-0002-7709-3706
[c] https://orcid.org/0000-0002-3634-0213
[d] https://orcid.org/0009-0006-5823-6595
[e] https://orcid.org/0000-0001-8649-4192

Bagozi, A., Bianchini, D., De Antonellis, V., Garda, M. and Melchiori, M.
A Multi-Perspective Framework for Smart Contract Search in Decentralised Applications Design.
DOI: 10.5220/0011992000003467
In *Proceedings of the 25th International Conference on Enterprise Information Systems (ICEIS 2023) - Volume 1*, pages 229-236
ISBN: 978-989-758-648-4; ISSN: 2184-4992

In this setting, we propose a multi-perspective framework to support search and reuse of SCs that, in addition to classification and technical characteristics of SCs, takes into account the past experience of developers who have used SCs from the registry to develop DApps. The framework supports semantic-based search (Bianchini et al., 2009) and ranking of SCs according to three search scenarios apt to support development of DApps at different phases.

The paper is organised as follows. Section 2 presents a motivating scenario and Section 3 describes the multi-perspective DApp model. SC search scenarios and ranking are illustrated in Section 4. Section 5 emphasises the cutting-edge features of our approach, with respect to the state of the art. Finally, Section 6 closes the paper, sketching future research directions.

2 MOTIVATING SCENARIO

In this paper, we consider a reference scenario from the healthcare domain, regarding clinical trial processes. A clinical trial is articulated over different phases, aimed at introducting a New Chemical Entity (NCE) to the market, initially administered to a limited number of volunteers. Participants in a clinical study may incur in research-related injuries, which should be minimised by researchers steering the trial. In such cases, individuals would be compensated if the injury descends from their participation in the trial. As also discussed in recent research (Wong et al., 2019) published on *Nature Communications*, the use of blockchain in this scenario is relevant and promising. It improves traceability and auditing of data to the benefit of processes, like the compensation one, that may occur within a clinical trial (Omar et al., 2021). In particular, we focus on the following challenges that, without loss of generality, can be extended to other application contexts.

Trustworthy inspection of exchanged data. Normally, for compensation to be recognised, data related to the clinical trial (e.g., protocol setup and registration, individuals enrolment, data collection methods) must ensure a transparent and trustworthy inspection by: (i) the trial regulator subject, which oversees and monitors whether the steps are adhering to the demanded standards and requirements; (ii) Health Insurance Organisations (HIOs), which are involved in the clinical trial as responsible for corresponding the refund to individuals when the conditions specified in the health contract subscribed by individuals participating in the trial are met. Exploiting notarisation on-chain of data exchanged by the parties helps reducing verification time for those events necessitating mandatory inspection by the regulator (e.g., in the case of injuries regarding adverse events).

Automated compliance to standards and rules. In order to comply with rules and/or standards established and promoted by the regulator, the DApps used in the context of the process should reuse, and possibly extend, SCs provided by the regulator and made available by means of the registry. For example, according to this approach, the implementation of a compensation process as a DApp would favour the automated compliance to rules and standards promoted by the regulator.

Assisted DApp design. In order to reduce coding time and effort, a developer should be supported in: (a) specifying the features of the SCs to search from the registry; (b) developing with a proactive support, i.e., specifying the characteristics of the DApp under development and then being suggested with SCs from the registry that can be included into the DApp. For example, through a proactive support, the developer may be proposed with the following two SCs, whose adoption permits to comply with the best practices in the development of a compensation DApp: a SC providing a mechanism to temporarily disable compensation (e.g., in the case the HIO has to modify the compensation policies) and a SC to record data related to the occurred injuries with the required level of detail for later inspection (e.g., in the case of complaints).

3 THE MULTI-PERSPECTIVE DApp MODEL

In the following, we describe the multi-perspective framework aimed at supporting SC search for DApps development. As anticipated in the previous section, SCs are described in a registry, containing both SCs specific for the application domain and general purpose SCs (e.g., from third party registries, such as OpenZeppelin[1]). In the application context considered in this paper, the registry is populated and maintained over time by a group of expert developers belonging to/affiliated with the regulator subject. SCs are aimed at addressing common issues a developer may encounter while creating a DApp compliant with the standards and procedures envisaged by the collaborative process. The model (Figure 1) is

[1] http : / / github.com / OpenZeppelin / openzeppelin - contracts

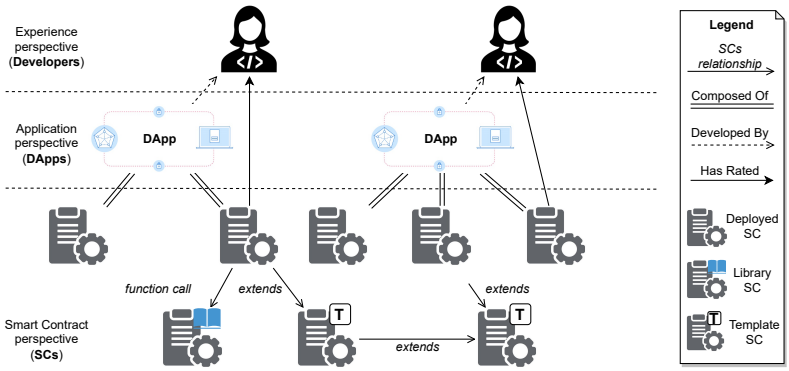

Figure 1: Multi-perspective DApp model.

organised according to three perspectives, describing: (i) smart contracts; (ii) DApps and (iii) developers.

Smart Contract Perspective. Let $\mathcal{SC}$ be the set of SCs described in the registry. The types of SCs and their features are represented considering the conceptual model in Figure 2, described in detail in the following, and which considers as reference the widely adopted Ethereum blockchain.

Smart Contracts Model. In our model we use the term SC in a general sense and hence a SC can be either a `TemplateContract` or a `DeployedContract`. A `TemplateContract` is conceived as a kind of SC used as a base to build other SCs. In particular, an `AbstractSC` is a `TemplateContract` representing a template code pattern (i.e., a partially implemented SC or a SC containing at least one function without any implementation). A `TemplateContract` is usually part of a registry publicly available on the Web (e.g., the OpenZeppelin registry for the Ethereum blockchain) for which `Documentation` is also provided. One or more `TemplateContract` may be employed to build (expressed through the `extends` relationship in the model) a `DeployedContract`. As well, a `TemplateContract` can be also extended from one or more `TemplateContract`. As the name suggests, a `DeployedContract` resides on the blockchain and, thus, it is endowed with `Endpoint` deployment information (i.e, address, chain ID, network ID for the contract, to locate it on the blockchain) and with an ABI (Application Binary Interface), that is metadata describing the SC interface. A `DeployedContract` can be either a `ConcreteSC` or a `LibraryContract`, the latter conceived as a SC containing only callable functions, with no state variables. The functions contained in a `LibraryContract` can be called by a `ConcreteSC` (`function call` relationship). Oracle contracts (either `Abstract` or `Concrete`) are used to access data from the world outside the blockchain.

Semantic Tags. To provide a semantic characterisation for SCs, semantic tags are fostered to tackle both polisemy and homonymy issues of traditional tagging when searching SCs from the registry. Semantic tagging is performed by those developers who add the SC to the registry. During the assignment of tags, sense disambiguation techniques based on WordNet[2] lexical database are applied. In WordNet, terms with the same meaning are grouped into *synsets*. Amongst the others, WordNet contains *hyponymy* and *hypernymy* relationships between synsets, used to represent specialisation and generalisation between two terms, respectively. A synset has a human readable definition and a set of *synonyms*. When the developer assigns a tag, all synsets containing that term are retrieved from WordNet. Each semantic tag is composed of: (i) the term extracted from WordNet; (ii) the set of terms in the same synset; (iii) the human readable description.

In our model, SCs are associated with a *descriptor*, which contains information enabling each contract to be searched by a developer, for subsequent employment within the DApp under construction. The descriptor has *classification features* (the type of SC and the semantic tags) and *technical features* (e.g., the coding language). Depending on the type of SC, also *contract-specific features* may be available (the attributes belonging to the sub-classes of `SmartContract` of the conceptual model in Figure 2). The SC descriptor is formalised as follows.

Smart Contract Descriptor. The descriptor of a SC

[2]https://wordnet.princeton.edu/

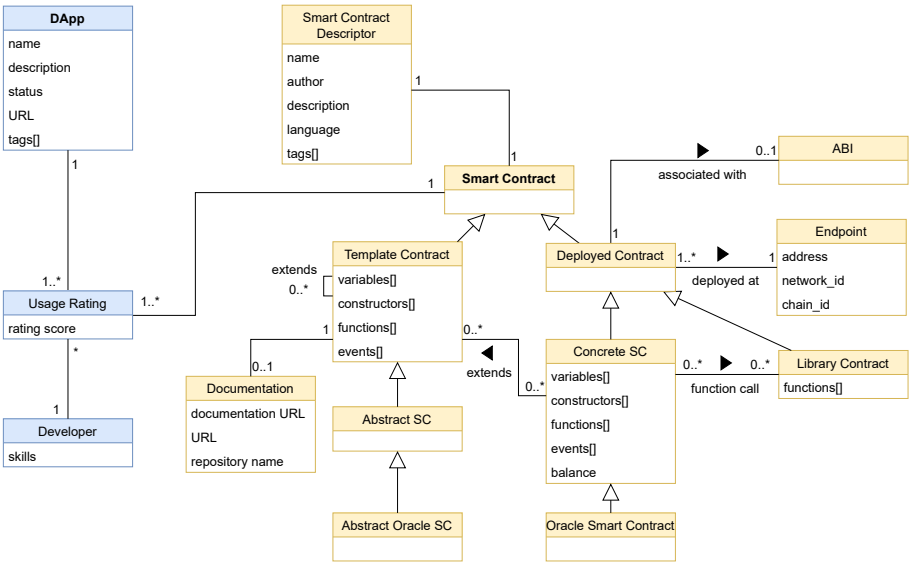

Figure 2: Smart contracts conceptual model for DApps development.

$C_i \in \mathcal{SC}$ is denoted by a tuple

$$ds_{C_i} = \langle type_{C_i}, \{t^j_{C_i}\}, name_{C_i}, lang_{C_i}, desc_{C_i}, CF_{C_i} \rangle$$

where: (i) $type_{C_i}$ is the type of the SC; (ii) $\{t^j_{C_i}\}$ is a set of semantic tags; (iii) $name_{C_i}$ is the name of the SC; (iv) $lang_{C_i}$ is the coding language and (v) $desc_{C_i}$ is a textual description for the SC. Contract-specific features, depending on $type_{C_i}$, are included in the set CF_{C_i} (if any) and represented as pairs $\{feature_j : \{value^k_j\}\}$.

Example. Let us consider Alice, a developer in charge of designing the compensation DApp introduced in Section 2, on behalf of a HIO, to deploy the compensation process on-chain. Specifically, Alice decides to structure her DApp according to two core SCs providing the functionalities to: (i) retain information regarding the policy terms of health contracts signed by individuals at the beginning of the trial with the HIO; (ii) encapsulate all the rules related to compensation logic. To develop the two former SCs, Alice resorts to the registry containing, amongst the others the following two abstract SCs: (a) `HealthPolicyContract` (in brief, `HPC`) which defines the minimum policy terms each health contract must hold to be eligible within the clinical trial, as demanded by the regulator subject; (b) `RecordInjuriesDetailsContract` (in brief, `RID`), providing the functions to record data related to the occurred injuries on the blockchain. An excerpt of the two descriptors for `HPC` and `RID` is reported in the following:

$ds_{HPC} = \langle type_{HPC} : \texttt{AbstractSC};$
$t^1_{HPC} : \langle \texttt{health}, \{\}, \text{“the general condition of body and mind”} \rangle;$
$t^2_{HPC} : \langle \texttt{policy}, \{\texttt{insurance}\}, \text{“written contract or certificate} \rangle;$
of insurance”;
$desc_{HPC}$: “The Health Policy SC specifies minimal data which is mandatory to be recorded for health contracts”;
[...]
$lang_{HPC} : \texttt{Solidity};$
$CF_{HPC} = \{variables : \{\texttt{dateStart}, \texttt{dateEnd}, \texttt{liabilityLimit}\}, \ldots\}\rangle$

$ds_{RID} = \langle type_{RID} : \texttt{AbstractSC};$
$t^1_{RID} : \langle \texttt{health}, \{\}, \text{“the general condition of body and ...”} \rangle;$
$t^2_{RID} : \langle \texttt{injury}, \{\texttt{hurt}, \texttt{harm}, \texttt{trauma}\}, \text{“any physical damage to the body ...”} \rangle;$
$desc_{RID}$: “The Record Injuries Details SC provides base functions to record injuries-related data”;
[...]
$lang_{RID} : \texttt{Solidity};$
$CF_{RID} = \{functions : \{\texttt{storeInjuryDetails}, \texttt{storeInjuryType}\}, \ldots\}\rangle$

In the example, contract-specific features for the two former `AbstractSC` are enlisted (e.g., contract variables and functions).

Application Perspective. Let $\mathcal{DA}$ be the set of DApps built using SCs from the set $\mathcal{SC}$. In our model, a DApp $DA_i \in \mathcal{DA}$: (i) contains the set of SCs from the registry $\{C_{DA_i}\} \subset \mathcal{SC}$ used to build the DApp; (ii) has a set $\{t^j_{DA_i}\}$ of semantic tags and (iii) is associated with several technical features like the deployment status $status_{DA_i}$ (e.g., prototype, live), the description $desc_{DA_i}$ and the URL URL_{DA_i} where the DApp is published (if it is not a prototype).

Example. The DApp for the compensation process from Section 2 is represented as:
DA_{Comp} = [
t^1_{Comp} : ⟨`insurance`, {`indemnity`}, "protection against future loss";⟩;
t^2_{Comp} : ⟨`payment`, {}, "a sum of money paid or a claim discharged";⟩;
t^3_{Comp} : ⟨`refund`, {`return`, `repay`, `give back`}, "pay back"⟩;
$status_{Comp}$: `prototype`;
$\{C_{Comp}\}$: {`IndividualsHCContract` : (`HealthPolicyContract`), `HIOCompensationContract` : (`AccessControlContract`, `PausableContract`, `RecordInjuriesDetailsContract`)}]
SCs enclosed by round brackets are the ones coming from the registry provided by the trial regulator subject, used for creating the core ones, and constitute the content of the set $\{C_{Comp}\}$. For instance, `PausableContract` and `AccessControlContract` are base SCs made available from the OpenZeppelin SCs web registry. SCs from the registry are leveraged by Alice to create the core SCs of her DApp: `IndividualsHCContract` retains information regarding the policy terms of health contracts, whereas `HIOCompensationContract` implements the compensation logic.

Experience Perspective. Each developer $d_i \in \mathcal{D}_C$, who uses the SCs from the registry to build her DApps, is modelled through: (i) the skill σ_i for developing DApps; (ii) a set of usage ratings $\langle C_j, DA_k, \mu_{jk} \rangle$ expressing that the developer rated with a score $\mu_{jk} \in [0,1]$ the SC C_j when used within the DApp DA_k, to consider the fact that a SC could be suitable to be used only in specific DApps. Developer's skill is asked during the registration to the system according to a discrete set of values (one among `expert`, `high confidence`, `medium confidence`, `low confidence`, `unexperienced` corresponding to a value of 1.0, 0.8, 0.5, 0.3, 0.0, respectively). The score μ_{jk} is selected according to a scoring system with nine rating options[3], to increase reliability and consistency, ranging from `poor` (score = 0.2) to `exceptional` (score = 1.0).

Example. The following example reports the ratings assigned by the developer Alice from Example 3 to a subset of SCs from the registry, used in DA_{Comp}:

d_{Alice} = ⟨0.8 (`high confidence`), {⟨`HealthPolicyContract`, DA_{Comp}, 0.8 (`excellent`)⟩, ⟨`RecordInjuriesDetailsContract`, DA_{Comp}, 0.7 (`very good`)⟩}.

[3] https://grants.nih.gov/grants/policy/review/rev_prep/scoring.htm

4 SMART CONTRACT SEARCH AND RANKING

To develop a DApp with the support of our framework, a developer performs three main steps: (i) assignment of a set of semantic tags to describe the functionalities of the DApp; (ii) search of basic SCs from the registry and composition of these SCs in the DApp; (iii) assignment of a rating to the SCs from the registry used for the DApp. Among the three steps, the challenge we focus on regards the second step, for which we conceive iterative and progressive search scenarios for SCs, to support DApps developers, who start by selecting a single SC and proceeds composing incrementally the DApp. Indeed, search scenarios could be fostered either for finding already deployed SCs (e.g., to perform a call to a function in a deployed library) or for the reuse of template SCs. In a search scenario, we distinguish between the *target* and the *modality*, summarised in Table 1. In particular, the choice of invoking a search scenarios is tightly coupled with the development phase of the DApp, as explained in the following.

4.1 Search Scenarios

The **target** of the search task could be a *single selection* (e.g., when the developer starts the design of a new DApp) or *completion* (e.g., looking for SCs to complete the DApp). Instead, the **modality** qualifies how the search is performed:

- *simple Search*: the developer looks for a SC by specifying only a set of semantic tags and it is meant for DApps in their early stage of development;
- *Advanced Search*: it is a variant of the simple search, fostered by a developer having in mind also the DApp where the SC will be used, specified by a set of semantic tags and, possibly, with the set of SCs already included in the DApp. Advanced search is suitable for DApps already gathering more than one SC;
- *Proactive Search*: the developer specifies only the semantic tags of a DApp and the set of SCs from the registry included in the DApp, expecting the system to provide suggestions about SCs to be added to the DApp, given similar DApps developed in the past by other developers. Also proactive search is suitable for DApps already gathering more than one SC.

For advanced and proactive search, there is also the possibility for a developer to specify the $type^r_C$ of the desired SCs, thus resulting in a *typified* search

Table 1: Specification of the request C^r depending on the search scenario.

	Target		
	Single Selection	*Completion*	
Simple	$\langle\{t^r_C\}\rangle$	$n.a.$	**Search Modality**
Advanced (Typified)	$\langle\{t^r_C\},\{t^r_{DA}\}\rangle$ $(\langle type^r_C,\{t^r_C\},\{t^r_{DA}\}\rangle)$	$\langle\{t^r_C\},\{t^r_{DA}\},\{C^r_{DA}\}\rangle$ $(\langle type^r_C,\{t^r_C\},\{t^r_{DA}\},\{C^r_{DA}\}\rangle)$	
Proactive (Typified)	$n.a.$	$\langle\{t^r_{DA}\},\{C^r_{DA}\}\rangle$ $(\langle type^r_C,\{t^r_{DA}\},\{C^r_{DA}\}\rangle)$	

(these variants are conceived for developers with advanced skill, having a clear idea about the type of SC needed). Search scenarios are guided through a set of similarity metrics detailed in the next sections.

Smart Contract Search Request. The general structure of a search request for SC is defined as the following tuple:

$$C^r = \langle type^r_C, \{t^r_C\}, \{t^r_{DA}\}, \{C^r_{DA}\}\rangle \quad (1)$$

where: (i) $type^r_C$ is the type of requested SC (only for typified search variants); (ii) $\{t^r_C\}$ is the set of semantic tags for the searched SC; (iii) $\{t^r_{DA}\}$ is the set of semantic tags denoting the DApp *DA* and (iv) $\{C^r_{DA}\}$ is the set of SCs already part of the DApp *DA* that is being developed, if any. Elements in Equation (1) are not all mandatory, as they depend on the type of search scenario the developer will undertake (Table 1).

Example. If Alice from Example 3 wants to find a SC aimed at handling exchange rate issues when corresponding the compensation amount for trial individuals, she may issue a request compliant with the *completion target* and *advanced search*: $\langle\{t^r_C\} = \{\langle$`rate`$,\{$`charge_per_unit`$\},$"amount of a charge [...]", $\{t_{DA_{Comp}}\},\{C_{DA_{Comp}}\}\rangle\}$.

4.2 Similarity Metrics

SC search leverages the combination of different metrics, grounded on the elements of the request C^r and the features retained in the SCs descriptors. In particular, we consider two distinct metrics, the *semantic Tag Similarity* and the *DApp Composition Similarity*, which are in turn exploited to compute the contract and DApp similarity, as explained in the following.

Semantic Tag Similarity. The semantic similarity between two tags t_1 and t_2 is assessed according to WordNet, relying on hyponymy/hypernymy relationships between synsets. In this respect, the similarity between two tags t_1 and t_2, denoted with $Sim_t(t_1,t_2)$, is calculated according to the widely adopted Wu-Palmer semantic similarity score, belonging to the range $(0,1]$ and based on the WordNet taxonomies (the rationale behind the score calculation and further details can be found in (Wu and Palmer, 1994)). A value of $Sim_t()$ close to 1 means high semantic relatedness between the two tags. For instance, $Sim_t($`refund`$,$`payment`$) = 0.94$ whereas $Sim_t($`rate`$,$`payment`$) = 0.46$. Overall, the similarity between two sets of tags $\mathcal{T}_1$ and $\mathcal{T}_2$, denoted with $Sim_{tag}(\mathcal{T}_1,\mathcal{T}_2) \in (0,1]$, can be obtained by calculating the pairwise similarity between a tag belonging to $\mathcal{T}_1$ and a tag belonging to $\mathcal{T}_2$, applying the formula:

$$Sim_{tag}(\mathcal{T}_1,\mathcal{T}_2) = \frac{2\cdot\sum_{t_1\in\mathcal{T}_1,t_2\in\mathcal{T}_2} Sim_t(t_1,t_2)}{|\mathcal{T}_1|+|\mathcal{T}_2|} \quad (2)$$

Pairs of tags to be considered for Sim_{tag} are selected in order to maximise the overall Sim_{tag}, thus ensuring that each tag from the first set participates in at most one pair with one of the tags from the second set and vice versa (i.e., applying the Kuhn's Hungarian algorithm). Hence, if we have the two sets $\mathcal{T}_1 =$ {`refund`,`payment`} and $\mathcal{T}_2 =$ {`payment`,`charge`} then $Sim_{tag}(\mathcal{T}_1,\mathcal{T}_2) = 2\cdot[max(Sim_t(t^1_1,t^1_2),Sim_t(t^1_1,t^2_2)) + max(Sim_t(t^2_1,t^1_2),Sim_t(t^2_1,t^2_2))]/4 = 0.97$.

DApp Composition Similarity. To evaluate the similarity between the composition of two DApps, the related sets of SCs, drawn from the registry, are considered. In particular, the similarity between a DApp composed of a set of SCs $\{C^r_{DA}\}$ and another DApp composed of a set $\{C_{DA_k}\}$ considers the number of common registry SCs between the two sets. The similarity $Sim_{comp}() \in [0,1]$ is assessed with the Dice's coefficient over the sets of SCs of the two DApps.

$$Sim_{comp}(\{C^r_{DA}\},\{C_{DA_k}\}) = \frac{2\cdot|\{C^r_{DA}\}\cap\{C_{DA_k}\}|}{|\{C^r_{DA}\}|+|\{C_{DA_k}\}|} \quad (3)$$

Contract Similarity and DApp Similarity. Semantic tags and DApp composition similarity are fostered to compute two metrics: (i) $ContractSim() \in (0,1]$ for evaluating the similarity between the request C^r and a SC from the set $\mathcal{SC}$, according to

the Smart Contract perspective; (ii) $DAppSim() \in (0,1]$ for evaluating the similarity between the request and a SC in the scope of DApps where the SC has been used, according to the Application perspective. $ContractSim(C^r, C)$ considers only SCs semantic tags, that is $ContractSim() = Sim_{tag}(\{t_C^r\}, \{t_C\})$. Instead, the similarity between the request and a SC in the scope of a DApp is denoted as $DAppSim(C^r, C, DA)$ and is obtained as follows:

$$w_1 \cdot Sim_{tag}(\{t_{DA}^r\}, \{t_{DA}\}) + w_2 \cdot Sim_{comp}(\{C_{DA}^r\}, \{C_{DA}\}) \quad (4)$$

where $C \in \{C_{DA}\}$ and the set $\{C_{DA}\}$ is the set of SCs of the DApp DA. For the weights, it holds that $w_{1,2} \in [0,1]$, $w_1 + w_2 = 1$. In single selection target, no information regarding the SCs used by the DApp is provided, as a consequence $w_2 = 0$. Otherwise, to balance equally the two terms, $w_1 = w_2 = 0.5$.

The overall similarity measure between the request C^r and each SC C (denoted with $Sim(C^r, C) \in (0,1]$) is obtained as:

$$w_3 \cdot ContractSim(C^r, C) + \frac{(1-w_3)}{|\mathcal{D}_C|} \cdot \sum_{i=1}^{|\mathcal{D}_C|} \left(\frac{\sum_{k=1}^{|\mathcal{DA}|} (\sigma_i \cdot DAppSim(C^r, C, DA_k)}{|\mathcal{DA}|} \right) \quad (5)$$

where $C \in DA_k$ and the term multiplied by the factor $(1-w_3)$, with $w_3 \in [0,1]$, considers the fact that the SC C has been adopted in different DApps by developers with different development skill σ_i, in order to ensure that past experiences of more expert developers have a higher impact on $Sim()$ calculation. Intuitively, the closest the σ_i and $DAppSim()$ values to 1 (maximum value) for all the developers $d_i \in \mathcal{D}_C$, the closest the second term in Equation (5) to 1.0. The weight w_3 in Equation (5) is set according to the search modality; if a simple search is performed $w_3 = 1$ (no information regarding DApp) while $w_3 = 0$ for a proactive search (only information regarding the target DApp is provided). For all the other cases, to balance equally the two aspects, $w_3 = 0.5$.

Finally, to limit the number of SCs to be included in the results, denoted as $\mathcal{R}(C^r)$, a developer may set a threshold $\tau \in (0,1]$.

4.3 Smart Contract Ranking

The results $\mathcal{R}(C^r)$ of a SC search are ranked according to a function $\rho : \mathcal{R}(C^r) \rightarrow [0,1]$ which considers both the scores given by developers who used the SCs in their DApps and the technical features of SCs. The ranking function is defined as follows:

$$\rho(\mathcal{R}(C^r)) = \alpha \cdot \rho_1(\mathcal{R}(C^r)) + \sum_{i=1}^{3} \gamma_i \cdot \hat{\rho}_i(\mathcal{R}(C^r)) \quad (6)$$

where $\alpha \geq 0$, $\gamma_{1,\ldots,3} \leq 1$ and $\alpha + \sum_{i=1}^{3} \gamma_i = 1$ assume all an equal weight in Equation (6). The computation of $\rho_1 \in [0,1]$ adheres to the same rationale of the second term of Equation (5), considering ratings as more important if given by more expert developers:

$$\rho_1(\mathcal{R}(C^r)) = \frac{1}{|\mathcal{D}_C|} \cdot \sum_{i=1}^{|\mathcal{D}_C|} \left(\frac{\sum_{k=1}^{|\mathcal{DA}|} (\sigma_i \cdot \mu_{ik} \cdot DAppSim(C^r, C', DA_k)}{|\mathcal{DA}|} \right) \quad (7)$$

Equation (7) considers the ratings μ_{ik} of developers $d_i \in \mathcal{D}_C$, who used the SC C' in DApps from $\mathcal{DA}$. The $DAppSim()$ similarity is included in the equation only for completion search target whereas for a simple search the value of $DAppSim()$ cannot be calculated, and thus it is replaced with 1. Concerning the calculation of $\hat{\rho}_i(\mathcal{R}(C^r))$, a subset on technical features of SCs and DApps are considered. In our approach, we consider the following: (i) $\hat{\rho_1}$ is based on the popularity of the coding language (i.e., the number of SCs coded with such languages); (ii) $\hat{\rho_{2,3}}$ are based on the reuse frequency (i.e., the number of DApps the SC is included in) for live ($\hat{\rho_2}$) and prototype ($\hat{\rho_3}$) DApps.

5 RELATED WORK

In the literature, recent research efforts have proposed frameworks for modelling SCs features and enabling their subsequent search for DApps deployment. The reuse of existing SCs code for the development of real-world DApps has been investigated in (He et al., 2020), where similarity techniques based on SCs code comparison are set up to find clone SCs in a given set. The approach in (Guida and Daniel, 2019) devises a conceptual model to foster the reuse of SCs in a model-driven development environment. With the aim of understanding functionality and internal mechanism of SCs, a Uniform Description Language (named UDL-SC) is proposed in (Souei et al., 2021); the underlying meta-model focuses on three perspectives: (i) operational, for modelling operations and events provided by a SC; (ii) technical, regarding the information of the target blockchain; (iii) business, including quality of service details and legal information. To support collaborative development in blockchain-oriented software engineering, He et al. (He et al., 2018) present a specification language for SCs based on a model taking into account parties, terms (obligations/rights associated with a party) and

properties (regarding the object of the contract). To help users checking their SCs by referencing existing SCs created and saved in a blockchain platform, the search engine in (Tran et al., 2019) treats SCs as textual documents applying Information Retrieval techniques on contracts code to compute a similarity score between the SC being created and already deployed SCs. Description and invocation of SCs, in a way that is independent of the specific blockchain platform, is discussed in (Falazi et al., 2020). This work defines a protocol for the integration of heterogeneous smart contracts into application. In their approach, SC descriptions are provided by a dedicated registry.

From a general point of view, our approach shares some issues with (Guida and Daniel, 2019; He et al., 2020; Souei et al., 2021; Falazi et al., 2020). However, with respect to (Guida and Daniel, 2019; Falazi et al., 2020; Souei et al., 2021), the multi-perspective framework presented in this paper also includes a semantic and social characterisation of SCs and DApps, and it provides examples of possible applications in real search contexts. Additionally, we conceive several search scenarios with different types and modalities of search, to cope with developers needs and supporting them in finding candidate SCs to be used for DApps development.

6 CONCLUDING REMARKS

In this paper, we proposed a framework to search for smart contracts to develop distributed applications (DApps). The considered context is the one of collaborative processes where a subject, like a regulatory subject, has an interest in stimulating the use of blockchain to increase transparency and accountability among participants, while reducing the burden of the regulator in controlling trustworthiness among participants and the compliance with the process standards. To the scope, the regulator provides a registry of basic smart contracts that can be used and extended by stakeholders to set up DApps. The framework, in addition to classification and technical characteristics of smart contracts, takes into account the experience of developers who have used smart contracts to develop DApps. A preliminary implementation and evaluation of the proposed framework is in progress. Future research efforts regard the enrichment of the SCs model, including also additional features and statistics related to DApps (for instance, inclusion of KPIs suitable for evaluating the transaction volume about a specific DApp or other usage statistics). Moreover, experiments will be conducted, comparing the performance of different variants of the searching procedure, including other sense disambiguation systems (for instance, DBPedia or Babelfy) for tags.

REFERENCES

Bianchini, D., De Antonellis, V., and Melchiori, M. (2009). Service-based semantic search in P2P systems. In *ECOWS'09 - 7th IEEE European Conference on Web Services*, pages 7–16.

Cai, W., Wang, Z., Ernst, J. B., Hong, Z., Feng, C., and Leung, V. C. (2018). Decentralized Applications: The Blockchain-Empowered Software System. *IEEE Access*, 6:53019–53033.

Falazi, G., Breitenbücher, U., Daniel, F., Lamparelli, A., Leymann, F., and Yussupov, V. (2020). 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*, pages 134–149. Springer.

Guida, L. and Daniel, F. (2019). Supporting reuse of smart contracts through service orientation and assisted development. In *2019 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPCON)*, pages 59–68.

He, N., Wu, L., Wang, H., Guo, Y., and Jiang, X. (2020). Characterizing Code Clones in the Ethereum Smart Contract Ecosystem. In *International Conference on Financial Cryptography and Data Security*, pages 654–675.

He, X., Qin, B., Zhu, Y., Chen, X., and Liu, Y. (2018). SPESC: A specification language for smart contracts. In *2018 IEEE 42nd Annual computer software and applications conference (COMPSAC)*, volume 1, pages 132–137. IEEE.

Omar, I. A., Jayaraman, R., Salah, K., Yaqoob, I., and Ellahham, S. (2021). Applications of Blockchain Technology in Clinical Trials: Review and Open Challenges. *Arabian Journal for Science and Engineering*, 46(4):3001–3015.

Souei, W. B. S., El Hog, C., Sliman, L., Djemaa, R. B., and Amor, I. A. B. (2021). Towards a Uniform Description Language for Smart Contract. In *2021 IEEE 30th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pages 57–62.

Tran, H., Menouer, T., Darmon, P., Doucoure, A., and Binder, F. (2019). Smart Contracts Search Engine in Blockchain. In *Proceedings of the 3rd International Conference on Future Networks and Distributed Systems*, pages 1–5.

Wong, D. R., Bhattacharya, S., and Butte, A. J. (2019). Prototype of running clinical trials in an untrustworthy environment using blockchain. *Nature Communications*, 10(1):1–8.

Wu, Z. and Palmer, M. S. (1994). Verb Semantics and Lexical Selection. In *Proceedings of the 32nd Annual meeting of the Associations for Computational Linguistics (ACL)*, pages 133–138.