

# Multi-Perspective Description and Search of Smart Contracts for DApp Design

(Discussion Paper)

Ada Bagozi<sup>1</sup>, Devis Bianchini<sup>1</sup>, Valeria De Antonellis<sup>1</sup>, Massimiliano Garda<sup>1</sup> and Michele Melchiori<sup>1</sup>

<sup>1</sup>University of Brescia, Dept. of Information Engineering  
Via Branze 38, 25123 - Brescia (Italy)

## Abstract

With the advent of blockchain technology, interorganisational collaborative processes that demand trust requirements (e.g., food supply chain, smart grid energy distribution and clinical trials) can be implemented as decentralised applications (DApps) taking advantage of blockchain technology, which provides decentralised control and immutable transaction history, thereby improving security and accountability between parties. In this discussion 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 searchable registry of basic smart contracts (i.e., deployed ones and code templates), that can be adopted and possibly extended by the participants of the process to build up DApps. To support semantic-based 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.

## Keywords

multi-perspective model, blockchain, decentralised applications, smart contracts, semantic search

## 1. Introduction

With the advent of blockchain technology (BT), interorganisational collaborative processes demanding trust requirements (e.g., food supply chain, smart grid energy distribution and clinical trials) can be implemented as decentralised applications (DApps) [1]. DApps are conceived as web applications that orchestrate smart contracts (SCs) to implement the business logic of the applications and provide a Graphical User Interface to ease interactions between participants and SCs. 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.

In this discussion paper, we consider a cooperative process occurring in a domain, e.g., clinical trials, where a subject that 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


---

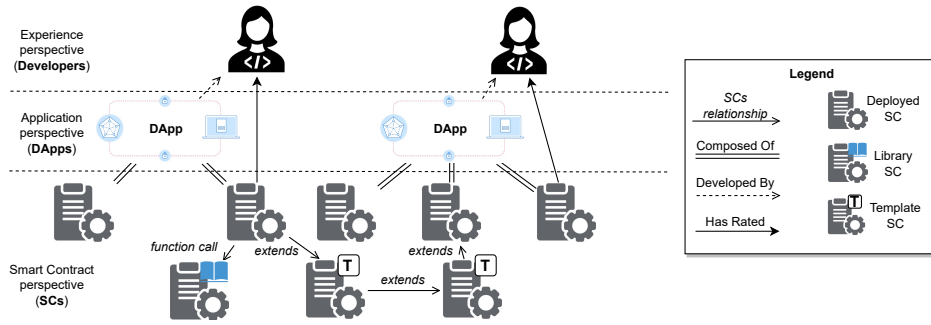
SEBD 2023: 31st Symposium on Advanced Database System, July 02–05, 2023, Galzignano Terme, Padua, Italy

✉ ada.bagozi@unibs.it (A. Bagozi); devis.bianchini@unibs.it (D. Bianchini); valeria.deantonellis@unibs.it (V. De Antonellis); massimiliano.garda@unibs.it (M. Garda); michele.melchiori@unibs.it (M. Melchiori)



© 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)



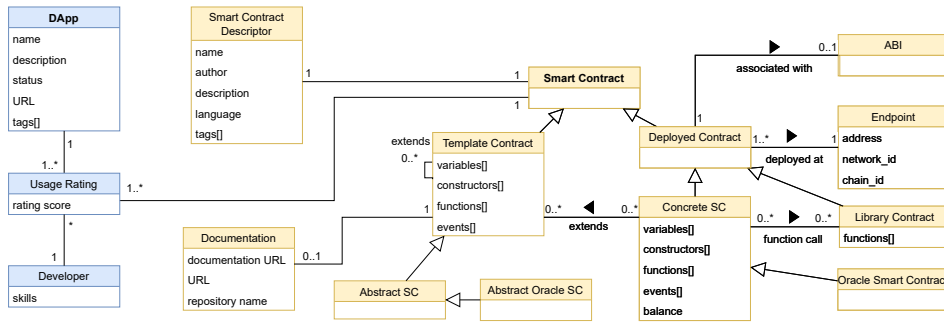
**Figure 1:** Multi-perspective DApp model.

improving the trustworthiness among the process participants. In this type of 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 SCs, which can be used and extended by any participant to set up DApps for automating process activities, where a DApp implements a business logic compliant with the rules and policies established by the regulator. Actually, developing real-world DApps is not a trivial task, hence the reuse of SCs published in open access registries has become a common practice [2, 3].

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 and ranking of SCs according to three search scenarios apt to support development of DApps at different phases. An extended version of this work has been presented in [4]. 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

We consider a motivating scenario from the healthcare domain, regarding clinical trials. A clinical trial (CT) is articulated over different phases and it is aimed at introducing a New Chemical Entity (NCE), initially administered to volunteers, to some pharmaceutical markets. Volunteers may incur in research-related injuries, which should be minimised by researchers steering the trial. In such cases, individuals would be compensated. As also discussed in a recent research [5], the use of blockchain in this scenario is relevant and promising. In fact, even if CTs are subject to a centralized regulatory authority, a blockchain provides interesting features such as relieving the regulatory authority from verifying detailed patient and clinical investigator interactions and offering better protection to the identity of patients. Moreover, it improves data traceability and auditing to the benefit of the compensation process [6]. We



**Figure 2:** Smart contracts conceptual model for DApps development.

focus on the following challenges.

**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, for example, a Health Insurance Organisation (HIO), which is responsible for corresponding compensations.

**Automated compliance to rules and standards.** 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 a 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.

**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.

### 3. The multi-perspective DApp model

In the following, we describe the multi-perspective framework aimed at supporting SC search for DApps development. The framework (Figure 1) is organised according to three perspectives, describing: (i) smart contracts; (ii) DApps and (iii) developers. 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 [7]). The registry is populated and maintained over time by a group of expert developers belonging to/affiliated with the regulator subject.

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

- *Smart contracts model.* In the model, a SC can be either a `TemplateContract` or a `DeployedContract`. A `TemplateContract` is conceived 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 with no 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

$ds_{HPC} = \langle type_{HPC} : \text{AbstractSC};$ $lang_{HPC} : \text{Solidity};$ $t_{HPC}^1 : \langle \text{health}, \{\}, \text{"the general condition of"};$ $\text{body and mind"} \rangle;$ $t_{HPC}^2 : \langle \text{policy}, \{\text{insurance}\}, \text{"written contract or"};$ $\text{certificate of insurance"} \rangle;$ $desc_{HPC} : \text{"The Health Policy SC specifies minimal data}$ $\text{which is mandatory to be recorded for health contracts"};$ $[...]$ $CF_{HPC} = \{variables : \{\text{dateStart}, \text{dateEnd},$ $\text{liabilityLimit}, \dots\}\}$	$ds_{RID} = \langle type_{RID} : \text{AbstractSC};$ $lang_{RID} : \text{Solidity};$ $t_{RID}^1 : \langle \text{health}, \{\}, \text{"the general condition of body and ..."} \rangle;$ $t_{RID}^2 : \langle \text{injury}, \{\text{hurt, harm, trauma}\}, \text{"any physical}$ $\text{damage to the body ..."} \rangle;$ $desc_{RID} : \text{"The Record Injuries Details SC provides base}$ $\text{functions to record injuries-related data"};$ $[...]$ $CF_{RID} = \{functions : \{\text{storeInjuryDetails},$ $\text{storeInjuryType}, \dots\}\}$
---	---

**Table 1**

Example: descriptors for HPC and RID abstract Smart Contracts; features specific to a contract (e.g., contract variables and functions) are also shown.

TemplateContract may be employed to build (expressed through the extends relationship in the model) a DeployedContract. As well, a TemplateContract can also be extended from one or more TemplateContract. As the name suggests, a DeployedContract resides on the blockchain. 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).

- *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 [8] lexical database are applied. Each semantic tag is composed of: (i) the term extracted from WordNet; (ii) the set of terms in the same synset (i.e., a group of terms with the same meaning); (iii) the human readable description.

- *Smart contract descriptor.* In the model, SCs are associated with a *descriptor*. 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.

A tuple  $ds_{C_i} = \langle type_{C_i}, \{t_{C_i}^j\}, name_{C_i}, lang_{C_i}, desc_{C_i}, CF_{C_i} \rangle$  represents a SC  $C_i \in SC$ , where: (i)  $type_{C_i}$  is the type of the SC; (ii)  $\{t_{C_i}^j\}$  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  $\{feature_j : \{value_j^k\}\}$ .

*Example.* Let us consider Alice, a developer in charge of designing a DApp for the compensation process 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 (called IndividualSHCContract); (ii) encapsulate all the rules related to compensation logic (called HIOCompensationContract). To develop the two former SCs, Alice resorts to the following two SCs already available in

```

DAComp = [SCComp =
{IndividualsHCCContract : (HealthPolicyContract),
HICompensationContract : (AccessControlContract,
PausableContract, RecordInjuriesDetailsContract)}
tComp1 : ( insurance, {indemnity}, "protection against
future loss");
tComp2 : ( payment, {}, "a sum of money paid or a claim
discharged");
tComp3 : (refund, {return, repay, give back}, "pay back");
statusComp : prototype]

```

(a)

```

dAlice = (0.8 (high confidence),
{(HealthPolicyContract,
DAComp, 0.8 (excellent)),
(RecordInjuriesDetailsContract,
DAComp, 0.7 (very good))}.

```

(b)

**Table 2**

Description of the compensation DApp(a); ratings assigned to the HPC and RID SCs (b).

the registry: (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 Table 1.

**Application Perspective.** Let  $\mathcal{DA}$  be the set of DApps built using SCs from the set  $\mathcal{SC}$ . In the model, a DApp  $DA_i \in \mathcal{DA}$ : (i) contains the set  $SC_{DA_i}$  used to implement the DApp; (ii) has a set  $\{t_{DA_i}^j\}$  of semantic tags and (iii) is associated with 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 accessible (if it is not a prototype).

*Example.* A DApp  $DA_{Comp}$  for the compensation process from Section 2 is described as in Table 2(a). SCs enclosed by round brackets are the ones coming from the registry provided by the trial regulator subject and are used for creating the two core ones. For instance, PausableContract and AccessControlContract are base SCs made available from the OpenZeppelin SCs web registry and are leveraged by Alice for developing her DApp in addition to the HPC and RID ones.

**Experience Perspective.** Each developer  $d_i \in \mathcal{DC}$ , who uses the SCs from the registry to build her DApps, is modelled through: (i) the self-assessed skill  $\sigma_i$  in 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 the rate evaluates the use of  $C_j$  in a specific DApp. 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  $\mu_{jk}$  is selected according to a scoring system with nine rating options [9] to increase reliability and consistency, ranging from poor (score = 0.2) to exceptional (score = 1.0).

*Example.* Table 2(b) reports the ratings assigned by Alice to two of SCs from the registry, used for  $DA_{Comp}$ .

## 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

	<i>Single Selection Target</i>	<i>Completion Target</i>	
<i>Simple</i>	$\langle \{t_C^r\} \rangle$	n.a.	<b>Search Modality</b>
<i>Advanced (Typified)</i>	$\langle \langle \{t_C^r\}, \{t_{DA}^r\} \rangle \rangle$ $\langle type_C^r, \{t_C^r\}, \{t_{DA}^r\} \rangle$	$\langle \langle \{t_C^r\}, \{t_{DA}^r\}, \{C_{DA}^r\} \rangle \rangle$ $\langle \langle type_C^r, \{t_C^r\}, \{t_{DA}^r\}, \{C_{DA}^r\} \rangle \rangle$	
<i>Proactive (Typified)</i>	n.a.	$\langle \langle \{t_{DA}^r\}, \{C_{DA}^r\} \rangle \rangle$ $\langle \langle type_C^r, \{t_{DA}^r\}, \{C_{DA}^r\} \rangle \rangle$	

**Table 3**

Specification of the request  $C^r$  depending on the search scenario.

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 compose incrementally the DApp. In a search scenario, we distinguish between the *target* and the *modality*, summarised in Table 3, as explained in the following.

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:

- (i) *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;
- (ii) *advanced search*: it is a variant of the simple search, fostered by a developer having in mind also the DApp, specified by a set of semantic tags, where the SC will be used and, possibly, a set of SCs already selected for the DApp.
- (iii) *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. Proactive search is suitable for DApps already gathering at least one SC.

For advanced and proactive search, there is also the possibility for a developer to specify the  $type_C^r$  of the desired SCs, thus resulting in a *typified* search (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 as explained in the following.

The general structure of a Smart Contract search request is defined as:

$$C^r = \langle type_C^r, \{t_C^r\}, \{t_{DA}^r\}, \{C_{DA}^r\} \rangle \quad (1)$$

where: (i)  $type_C^r$  is the type of requested SC (only for typified search variants); (ii)  $\{t_C^r\}$  is the set of semantic tags for the searched SC; (iii)  $\{t_{DA}^r\}$  is the set of semantic tags denoting the DApp  $DA$  and (iv)  $\{C_{DA}^r\}$  is the set of SCs already part of the DApp  $DA$  that is being developed, if any. The actual presence in a search request the of elements of Equation (1) depends on the type of search scenario (Table 3).

*Example.* If Alice from wants to find a SC aimed at handling exchange rate issues when corresponding the compensation amount for trial individuals in order to add this functionality to the compensation DApp, she may issue a request compliant with the *completion target* and *advanced search*:  $\langle \{t_C^r\} = \{ \langle rate, \{charge\_per\_unit\}, "amount\ of\ a\ charge\ [...]" \rangle, \{t_{DAComp}^r\}, \{C_{DAComp}^r\} \rangle$ .

**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 and calculated according to the widely adopted Wu-Palmer semantic similarity score. Based on this, a more general 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 defined [4]. A value of similarity close to 1 means high semantic relatedness.

*DApp composition similarity.* To evaluate the similarity between the composition of two DApps, their respective sets of SCs, drawn from the registry, are considered. In particular, the similarity between a DApp composed of a set of SCs  $\{C_{DA}^r\}$  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.

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

*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.  $ContractSim()$  considers only SCs semantic tags, that is  $ContractSim(C^r, C) = Sim_{tag}(\{t_C^r\}, \{t_C\})$ . 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:

$$w_1 \cdot Sim_{tag}(\{t_{DA}^r\}, \{t_{DA}\}) + w_2 \cdot Sim_{comp}(\{C_{DA}^r\}, \{C_{DA}\}) \quad (3)$$

where  $C \in \{C_{DA}\}$  and  $\{C_{DA}\}$  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, we set  $w_1 = w_2 = 0.5$ .

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:

$$w_3 \cdot ContractSim(C^r, C) + \frac{(1 - w_3)}{|\mathcal{D}_C|} \cdot \sum_{i=1}^{|\mathcal{D}_C|} \left( \frac{\sum_{k=1}^{|\mathcal{DA}|} (\sigma_i \cdot DAppSim(C^r, C, DA_k))}{|\mathcal{DA}|} \right) \quad (4)$$

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  $\sigma_i$ , in order to ensure that past experiences of more expert developers have a higher impact on  $Sim()$  calculation. Intuitively, the closest the  $\sigma_i$  and  $DAppSim()$  values to 1 (maximum value) for all the developers  $d_i \in \mathcal{D}_C$ , the closest the second term in Equation (4) to

1.0. The weight  $w_3$  in Equation (4) is set according to the search modality. Otherwise, for all the other cases,  $w_3 = 0.5$  to balance equally the two aspects. Possibly, to limit the number of SCs to be included in the results, denoted as  $\mathcal{R}(C^r)$ , a developer may set a threshold  $\tau \in (0, 1]$ .

**Smart contract ranking.** The results  $\mathcal{R}(C^r)$  of a search request 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 (i.e., the popularity of the used coding language and the number of DApps the ranked SC is included in). Please refer to [4] for details.

## 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 [3], where similarity techniques based on SCs code comparison are set up to find clone SCs. The approach in [10] 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 [11]; the underlying meta-model focuses on three perspectives (i.e., operational, technical, and business). To support collaborative development in blockchain-oriented software engineering, He et al. [12] 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, a search engine is proposed in [2]. Description and invocation of SCs, in a way that is independent of the specific blockchain platform, is discussed in [13].

From a general point of view, our approach shares some issues with [3, 10, 11, 13]. However, with respect to [10, 11, 13], the multi-perspective framework presented in this paper also includes a semantic and experience-based characterisation of SCs and DApps, and it provides examples of possible applications in real contexts. Additionally, we conceive various search scenarios with different types and modalities of search, to cope in a flexible way with developers needs to identify candidate SCs 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 regulatory subject, has an interest in stimulating the use of blockchain among the process participants. To this purpose, the regulator provides a searchable registry of basic smart contracts that can be used and extended to set up DApps. The framework, in addition to classification and technical features of smart contracts, takes into account the experience of developers who have already used the smart contracts of the registry 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. Moreover, experiments will be conducted,



comparing the performance of different variants of the searching procedure, including other sense disambiguation systems (for instance, DBpedia or Babelify) for tags.

## References

- [1] W. Cai, Z. Wang, J. B. Ernst, Z. Hong, C. Feng, V. C. Leung, Decentralized Applications: The Blockchain-Empowered Software System, *IEEE Access* 6 (2018) 53019–53033.
- [2] H. Tran, T. Menouer, P. Darmon, A. Doucoure, F. Binder, Smart Contracts Search Engine in Blockchain, in: *Proceedings of the 3rd International Conference on Future Networks and Distributed Systems*, 2019, pp. 1–5.
- [3] N. He, L. Wu, H. Wang, Y. Guo, X. Jiang, Characterizing Code Clones in the Ethereum Smart Contract Ecosystem, in: *International Conference on Financial Cryptography and Data Security*, 2020, pp. 654–675.
- [4] A. Bagozi, D. Bianchini, V. De Antonellis, M. Garda, M. Melchiori, A Multi-perspective Framework for Smart Contract Search in Decentralised Applications Design., in: *Proceedings of the 25th International Conference on Enterprise Information Systems (ICEIS) - Volume 1*, 2023, pp. 229–236.
- [5] D. R. Wong, S. Bhattacharya, A. J. Butte, Prototype of running clinical trials in an untrustworthy environment using blockchain, *Nature Communications* 10 (2019) 1–8.
- [6] I. A. Omar, R. Jayaraman, K. Salah, I. Yaqoob, S. Ellahham, Applications of Blockchain Technology in Clinical Trials: Review and Open Challenges, *Arabian Journal for Science and Engineering* 46 (2021) 3001–3015.
- [7] OpenZeppelin Contracts Library, 2023. URL: <https://github.com/OpenZeppelin/openzeppelin-contracts>, Accessed on April 2023.
- [8] G. A. Miller, WordNet: a lexical database for English, *Communications of the ACM* 38 (1995) 39–41.
- [9] NIH Scoring System, 2023. URL: [https://grants.nih.gov/grants/policy/review/rev\\_prep/scoring.htm](https://grants.nih.gov/grants/policy/review/rev_prep/scoring.htm), Accessed on April 2023.
- [10] L. Guida, F. Daniel, Supporting reuse of smart contracts through service orientation and assisted development, in: *2019 IEEE Proceedings of International Conference on Decentralized Applications and Infrastructures (DAPPCON)*, 2019, pp. 59–68.
- [11] W. B. S. Souei, C. El Hog, L. Sliman, R. B. Djemaa, I. A. B. Amor, Towards a Uniform Description Language for Smart Contract, in: *2021 IEEE 30th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, 2021, pp. 57–62.
- [12] X. He, B. Qin, Y. Zhu, X. Chen, Y. Liu, SPESC: A specification language for smart contracts, in: *2018 IEEE Proceedings of 42nd Annual computer software and applications conference (COMPSAC)*, volume 1, IEEE, 2018, pp. 132–137.
- [13] G. Falazi, U. Breitenbücher, F. Daniel, A. Lamparelli, F. Leymann, V. Yussupov, Smart Contract Invocation Protocol (SCIP): A protocol for the uniform integration of heterogeneous blockchain smart contracts, in: *Proceedings of 32nd International Conference on Advanced Information Systems Engineering (CAiSE)*, 2020, pp. 134–149.