An Ontology for Software Patterns: Application to Blockchain-Based Software Development

Nicolas Six, Camilo Correa, Nicolas Herbaut, Camille Salinesi

An ontology for software patterns: application to blockchain-based software development

Nicolas Six, Camilo Correa-Restrepo, Nicolas Herbaut, and Camille Salinesi

Centre de Recherche en Informatique (CRI)
Université Paris 1 Panthéon-Sorbonne, Paris, France
{first.last}@univ-paris1.fr

Abstract. Ensuring the quality of software design is usually a difficult task. In the blockchain field, the design of an application is particularly important as flaws can lead to critical vulnerabilities and cost overheads. To assist practitioners in this task, software patterns can be used (solutions to repeatable problems in a given context). Some blockchain patterns exist but they are scattered, and described in many different notations and templates. As a result, practitioners can be lost in the selection of adequate blockchain-based patterns. This paper fills the gap by proposing a blockchain-based software pattern ontology. The ontology is composed of two distinct subspaces: first, a set of classes and individuals related to blockchain-based patterns, based on a previous Systematic Literature Review (SLR). It notably reuses a taxonomy of blockchain design patterns, that helps to classify these patterns in relevant categories. Along that, another subspace has been created to further organize the knowledge related to software patterns and allow inferences. A tool is proposed along with the ontology to assist practitioners in finding blockchain-based patterns that fit their needs. An evaluation is performed to assess the usability and the relevancy of the ontology.

Keywords: Blockchain · Software design · Knowledge engineering

1 Introduction

A blockchain is a data structure where each block is linked to the previous one with a cryptographic hash. The addition of new blocks is ruled by a network of peers that each holds a copy of the blockchain (so-called blockchain network). Each block stores a list of transactions, that represent interactions between a user and the blockchain. Following the release of Ethereum in 2015, some blockchain solutions started to support Turing-complete smart contracts, enabling a new range of applications. A smart contract is a computer program that executes predefined actions when certain conditions within the system are met [1]. Hence, users can interact with smart contracts using transactions.

The inner workings of blockchain differ from traditional database technologies, granting blockchain many unique qualities. First, blockchain is decentralized: there is no single central actor in charge of the network. The level of decentralization varies depending on the blockchain type [23]. Then, blockchain

guarantees the immutability of data[1], as it is impossible to alter blocks after their addition. Finally, blockchain is transparent: any user that has access to a node can see past transactions and stored data. Transparency is not absolute: for instance, some blockchain technologies support private smart contracts, where data and execution are restricted to a set of authorized parties.

Blockchain directly addresses use cases that are difficult to solve with mainstream IS (Information System) technologies, such as ERPs. For instance, supply-chain applications benefit from blockchain immutability for goods traceability [2]. In this use case, trust is improved between supply-chain participants, as malicious actions to alter traceability data will be seen by everybody on the network. Blockchain smart contracts might also enable trustable service automation, such as the ones proposed by DeFi (Decentralized Finance) platforms [13].

Because of the specific properties of blockchain, a growing number of software architects undertake designing blockchain-based applications (BBA). A BBA can either be an application using blockchain as an improved platform to serve existing use cases (e.g., supply-chain traceability), or an application made feasible only by using blockchain technologies (e.g., on-chain decentralized cryptocurrency exchanges). However, they found the creation of BBAs to be a tedious task in both cases. Indeed, BBAs can *suffer* from their own qualities in certain contexts. For instance, transparency and immutability can be a burden when dealing with personal data, that might be subject to data protection regulations such as the General Data Protection Regulation (GDPR). Also, blockchain lacks certain native capabilities due to the way it functions, such as the ability to query external data or store large amounts of data.

Practitioners often employ well-known reusable recipes in the software design phase to improve software quality. Such recipes, known as patterns, are a solution to commonly occurring problems in given contexts. For instance, one of the best-known blockchain patterns is the *Oracle* [22]: as a smart contract cannot query data from outside the blockchain, using this pattern consists of sending fresh data to a smart contract when needed. Up to now, only a few patterns have already been proposed, grouped in collections (e.g., [22]) or standalone (e.g., [18]). Moreover, these patterns are scattered across the literature, making the task of identifying adequate patterns for a specific design difficult. Where a previous Systematic Literature Review (SLR) [17] tackled these issues by gathering these patterns in a collection, more work is needed to structure and formalize this knowledge, as well as facilitate its reuse.

In this paper, the following question is investigated: *How to formalize existing knowledge on blockchain-based software patterns to facilitate its reuse by practitioners?* To address this research question, this paper proposes a blockchain-based software pattern ontology, spanning two interrelated subspaces: (1) blockchain software patterns descriptions and (2) pattern provenance. While the former focuses on the classification of blockchain patterns established in the aforementioned SLR, the latter gathered knowledge on the original material proposing the pattern and its relationship with the state of the art, hence distinguishing

[1] This assumption is only valid if the network is not compromised

patterns found in the literature from the concept itself. Using an ontology facilitates the inference between linked patterns, using the knowledge found in studies. To explore the ontology, we created an open-source tool[2] that provides, from a series of questions, personalized pattern recommendations. Our ontology validation methodology is twofold. First, we assess that the ontology fulfills specified requirements. Then, we evaluate our ontology-based recommendations over a selection of papers from the literature. The rest of the paper is organized as the following: Section 2 introduces existing blockchain-based pattern collections and the blockchain-based software pattern ontology, then Section 3 presents the research method employed to build the ontology as well as its requirements. Section 4 describes the resulting ontology, and Section 5 presents the validation phase to assess the relevance of the ontology. Finally, threats of validity are discussed in Section 6, and Section 7 concludes with future works.

2 Related Works

The literature shows that the idea of using ontologies to describe software patterns has already been explored. In [10], Kampffmeyer et al. propose an ontology derived from GoF (Gang-of-Four) design patterns [5]. Each pattern is linked to a set of design problems it solves, along with a tool to help practitioners select patterns without having to write semantic queries. However, their ontology does not bring out any dependency link between the patterns themselves. Our contribution reuses the concept of problem ontology and extends it, as shown in Section 4. Another ontology for software patterns is proposed in [6], that encompasses not only design patterns but also architectural patterns and idioms. A pattern is described using different attributes (such as *Problem*, *Context*, *Solution*, etc.), and can be linked to other patterns through a pattern system and relations (e.g., require, use). A similar metamodel for software patterns is proposed in [9]. Some differences can be mentioned, such as the possibility to specify that two patterns conflict with each other, or the *seeAlso* relationship to indicate other patterns related to a specific pattern. In addition, [11] proposes a design pattern repository taking the form of an ontology. The contribution enlights tedious knowledge management and sharing with traditional pattern collections and argues for a structured ontology format. The proposed ontology groups patterns into pattern containers, where one pattern can belong to many containers. Patterns can also be linked to a set of questions and answers, elicited from expert knowledge, through an *answer relevance* attribute. It indicates how relevant a pattern is in addressing a specific question. This contribution follows a similar path to that taken by our ontology by structuring a set of blockchain-based patterns.

Some ontologies have been proposed for modeling the blockchain domain and its components, such as that proposed by De Kruijff and Weigand [4], that of Ugarte-Rojas and Chullo-Llave [8], and that of Glaser [7]. Another work by Seebacher and Maleshkova [14] focuses on modeling the characteristics of

[2] https://github.com/harmonica-project/blockchain-patterns-ontology

blockchains within corporate networks and their use. However, there is still a gap in the usage of ontologies to store blockchain-based patterns.

Finally, another decision model for the blockchain-based pattern can be mentioned [21]. In this model, a BPNM (Business Process Model and Notation) guides the process of selecting design patterns during the design phase of the software. Each activity in this process is further described using a BPMN-like notation, where the advantages and drawbacks of using a pattern are highlighted on each arrow leading to a pattern. This approach allows fine-grained guidance in the selection of blockchain patterns, but requires more work upfront to include patterns in the model and makes difficult the addition of patterns from outside the scope defined by the main BPMN.

3 Methodological Approach

The proper design of an ontology relies on the usage of a reliable and proven method. To build the ontology, the NeOn method [19] has been chosen due to its inherent flexibility and focus on the reuse of both ontological and non-ontological resources. However, these resources must be handled differently, as they can take multiple forms (e.g. whitepaper, publication, etc.). More information on handling non-ontological resources is given in Subsection 3.2. NeOn does not force rigid guidelines upon the ontology designer: a set of scenarios is given and the designer is free to select and adapt any scenario that suits their needs.

In this study, we base our approach on two of the scenarios envisaged within NeOn. The first scenario mainly concerns ontology construction from the ground up, to produce a new, standalone, ontology. The principal motivation for this choice is the absence of literature on existing ontology covering blockchain-based patterns. There is also the inability of existing software pattern ontologies to adequately capture the results of the SLR upon which we base our pattern proposal ontology, hence our need to produce a standalone ontology to cover our particular domain of interest. The second addresses the specific aspects of reusing non-ontological resources in the construction of ontologies. This is key since our ontology will be primarily based on the reuse of previous results. The NeOn methodology proposes a set of closely related life cycle models linked to the different scenarios it incorporates. In our case, given our need to reuse non-ontological resources, the six-phase waterfall life cycle has been chosen.

3.1 Ontology Requirements Specification

One important step in the construction of a sound ontology is the specification of requirements through an Ontology Requirement Specification Document (ORSD) [19] that serves as an agreement on which requirements the ontology should cover, its scope, implementation language, intended uses and end users. The ORSD facilitates the reuse of existing knowledge-aware resources in the creation of new ontologies [19]. Competency questions (CQs) are a way to introduce the functional requirements (FRs) of an ontology; their coverage, ideally

Table 1: Ontology competency questions.

CQ1	What are the classes of patterns in the blockchain area and how can they be differentiated and characterized?
CQ2	What are the different propositions of patterns in the academic literature and how can their acceptance by others be quantified?
CQ3	How can we differentiate the concept of pattern from their possible descriptions in different sources?
CQ4	What are the types of relations that can connect two patterns together?
CQ5	What are the different problems bound to the design and implementation of blockchain applications?

in a generalizable manner, allows one to consider the ontology functionally complete. The CQs are not formulated as FRs, but rather as questions that can be translated to requirements afterward (e.g. for CQ4, the ontology shall allow the user to retrieve the possible relations between two patterns). For the sake of brevity, only the CQs are detailed in this paper, listed in Table 1. Nevertheless, the full ORSD is available on GitHub (aforementioned in Section 1).

These competency questions define the two main purposes of the ontology. The first purpose is the definition of a sound structure to store software patterns, especially patterns proposed in the academic literature (CQ3). As these patterns might not have been applied enough in real use cases, one objective is to quantify the acceptance by others (CQ2) of a proposed pattern in a study. One possible solution to this problem is the usage of paper citations, described in Section 4.1. The relations between these patterns are also an important topic, as patterns are often used together to address larger-scale problems (CQ4). The second purpose is the storage of blockchain-based software patterns, taking their specificities into account. It notably includes their classification into comprehensive categories (CQ1) to guide the reader in the space of blockchain-based software patterns, as well as the problems they address (CQ5). The process outlined in [19] was followed to validate our requirements specification, within the larger framework of the NeOn methodology. Since the ontology was to be built with extensibility in mind, should new requirements arise, the queries that correspond to the competency questions to act as a test suite that ensures the ontology remains conformant as it evolves.

3.2 Reuse of non-ontological resources

As the purpose of constructing the blockchain-based software pattern ontology is to formalize the knowledge of a previous SLR, the ontology incorporates knowledge from two different non-ontological resources that can both be found on GitHub[3]. The first is a collection of 160 patterns that were identified during the SLR within 20 different papers; out of which 114 unique patterns have been

[3] https://github.com/harmonica-project/blockchain-patterns-collection

derived. Each of the collected patterns is described by a set of attributes, e.g., a *Name*, a *Context and Problem*, and a *Solution*. The citations count for each paper that proposes one or more patterns have also been collected. Thus, the reliability of a pattern can be assessed more easily: a pattern proposed in a paper cited multiple times can be considered, to some extent, to be more trustable than a pattern from a non-cited study. More rationale on the usage of these citations to assess pattern reliability is given in Subsection 4.2. The domain, programming language, implementation examples, and blockchain technology associated with the pattern are also collected if available. Indeed, some patterns may be proposed by paper for a specific programming language (e.g. Solidity), or in the context of a specific domain (e.g., decentralized identity). Also, different types of relations between patterns were identified: *Created from*, *Variant of*, *Requires*, *Benefits from*, and *Related to*. As the application of a specific pattern might require considering other patterns, its relations to others must be made explicit. Further details about these relations are given in Subsection 4.1. Patterns are classified into one of three categories depending on their general purpose: *Architectural patterns* that regroup patterns impacting the general structure of the application (elements, connections); *Design patterns* that are a way to organize modules, classes, or components to solve a problem; and *Idioms*, solutions to a programming language-related problems. The second non-ontological resource is used to extend this classification: design patterns are classified in subcategories derived from a taxonomy, that emerges from the categorization of the results in the SLR, and is comprised of 4 main categories and 14 subcategories. More details are given about the taxonomy in Section 4.

4 Results

4.1 Blockchain-based software pattern ontology

The primary outcome is the creation of the blockchain-based software pattern ontology, depicted in the conceptual model[4] Figure 1. This ontology has been written in Turtle, using the OWL language profile. As the figure shows, the driving idea of the ontology is: (a) the explicit distinctions between patterns, variants, and pattern proposals, (b) proposals and their relations with others, and (c) design problems outside the scope of patterns. Using ontologies as a support to store gathered knowledge about patterns also enables the usage of inference engines to find new relations between entities. Also, using an ontology also allows connecting it with other ontologies. Although it is beyond the scope of this study, it is possible to connect patterns with blockchain technologies expressed in other blockchain ontologies.

The central element of this model is the *Proposal* class, that is a pattern introduced within a particular academic paper. In the current form of the pattern proposal ontology, all sources of patterns are academic papers, thus this class is not implemented yet. Nonetheless, a class named *Source* will merge other types of

[4] For the sake of clarity, some subclasses of the ontology are hidden.

sources such as technical documentation or code repositories in the future. This improves the extensibility of the model, as patterns might be found in many different sources. Each paper is linked to a *Identifier*, which can take the form of a DOI (Digital Object Identifier) or an ArXiv ID. For each paper present in the ontology, its citations have been included as identifiers individuals and linked to the paper using a *references* object relation. This system allows inferring the citations of a specific paper, then making enhanced pattern recommendations by computing a "citation score" for each pattern proposal. More rationale on pattern recommendation is given in Subsection 4.2.

In this model, a proposal is linked to a variant. A variant inherits from a specific `Pattern` and represents one of its possible forms. Indeed, variants are used to express the variability of a pattern: two variants of a pattern might be close enough to address the same problem and solution, but may vary in some aspects (e.g., implementation). As an example, the *Oracle* pattern proposed by Xu et al. [22] is an individual of *Proposal* as it is proposed in the Xu et al. [22] paper, and attached to the *Oracle* variant, an individual of the *Variant* and *Oracle* class. The distinction between the proposals and the resulting pattern is important as in some cases multiple papers proposed the same pattern using different words, templates, and for different domains or blockchains. As such, a *Proposal* inherits from a specific blockchain, domain, or language[5]

[5] For the sake of clarity, subclasses of blockchain system, domains, and language are hidden (e.g., resp. Ethereum, IoT, or Solidity) in the provided conceptual model.

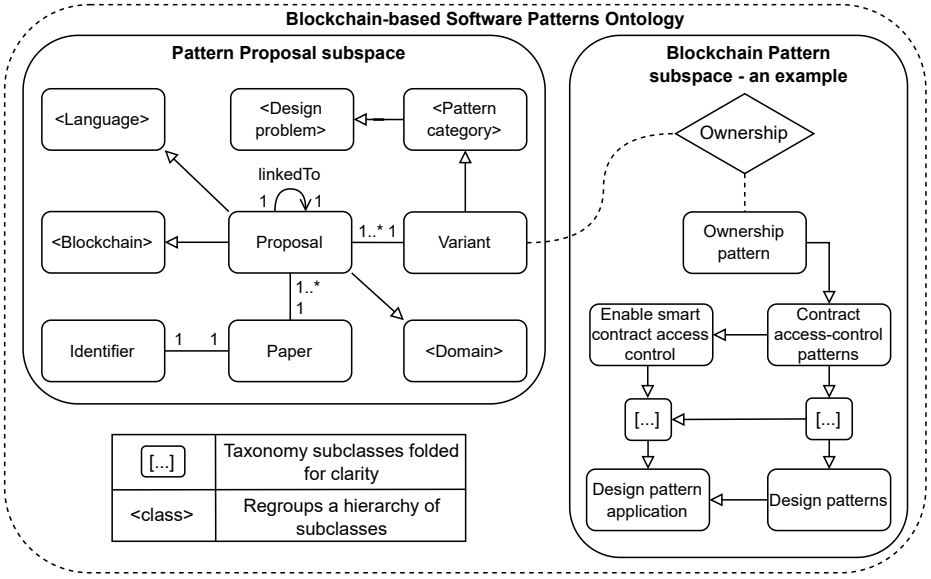

Fig. 1: Blockchain-based software pattern ontology with an exemplified section.

In this conceptual model, a *Proposal* is described by a *Context and Problem*, that gives a rationale for the purpose of the pattern and addressed problems, and a *Solution* field to introduces the elements composing the pattern solution. This structure for pattern description is derived from the two main pattern formats (GoF and Alexandrian pattern formats [20]), usually employed to express software patterns. Because of the lack of standardization across the literature on the description of patterns, only the context, problem, and solution have been kept to describe a pattern in this ontology. Yet, as the ontology references each proposal paper, the reader can refer to the paper of a specific proposal to learn more about it. *Proposals* can also be linked together, using 5 different relation types that were identified from the SLR:

- *Created from* - when a pattern directly takes its sources in another.
- *Variant of* - when a pattern is a variant of another.
- *Requires* - when a pattern requires another to be applied.
- *Benefits from* - when a pattern might benefit from another when applied.
- *Related to* - to identify weak relations between patterns (e.g., "see also").

By using inference, it is possible to translate these relations from proposals to variants, creating new knowledge about possible relations between patterns. SWRL rules have been written for the inference engine to generate such relations. As an example, the following rule translate a *benefitsFrom* object relation from two proposals to their corresponding variant (Equation 1).

$$\begin{aligned} &\forall\ (p_1, p_2) \in P\ and\ (v_1, v_2) \in V, \\ p_1\ benefitsFrom\ p_2\ \cdot\ p_1\ hasVariant\ v_1\ \cdot\ p_2\ hasVariant\ v_2 &\\ &\implies v_1\ benefitsFrom\ v_2 \end{aligned} \tag{1}$$

The subclasses of the *Pattern* class emanate from the reused taxonomy for blockchain-based patterns, built in its related SLR. For instance, the *Oracle* variant from [22] is linked to the *Oracle* pattern class, that inherits from the *Data exchange pattern*, then *On-chain pattern*, *Design pattern*, and finally *Pattern*. Although this hierarchy exists in the blockchain-based software pattern ontology, it is not shown in Figure 1 for clarity. To further refine this part of the ontology, each *Pattern* addresses a specific *Design problem*. By extension, each subclass of *Pattern* addresses a design *Design problem* subclass. Each problem has been assigned an associated literal question, notably used for recommendations. These questions have been designed along the construction of the design problem taxonomy to give a literal sentence of the problem. The question is presented as an affirmation (here, a user story sentence), that can be answered by yes or no. For instance, the question associated with the *Smart contract usage* design problem, solved by the *Smart contract patterns* is "I want to execute part of my application on-chain". Such an affirmation can be thus presented as a question to the user to guide pattern recommendations.

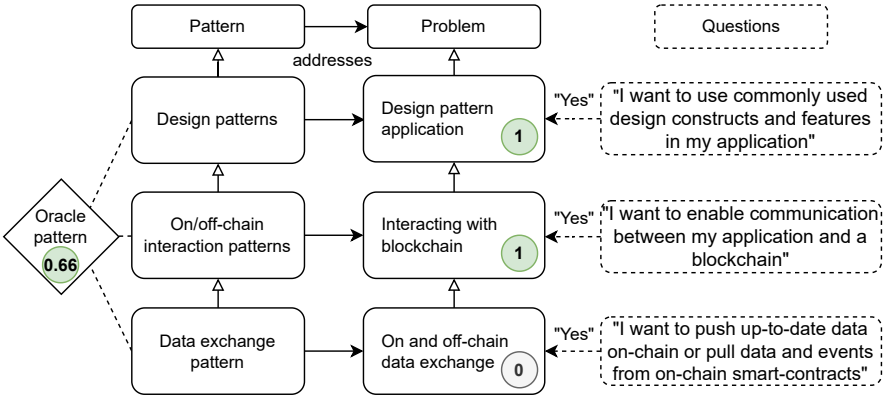

Fig. 2: Pattern scoring based on patterns/problem categories.

4.2 Ontology Querying Tool

In parallel with the ontology, a tool was designed to leverage it without having to use ontology-specific tools (e.g. Protégé). Using this tool eases the access to ontology content for non-experts, but the ontology can also be queried when served using a SPARQL server such as Apache Jena Fuseki[6]. This tool has two main features: the explorer and the recommender. The explorer allows one to dive into the knowledge of the ontology through the presentation of all available patterns in a grid. This section's purpose is to link the solution domain (the list of patterns) to the problem domain (user requirements). Indeed, any user reading available pattern descriptions might find some that suit their goals. The application shows each pattern's name but also the number of linked proposals and variants. By clicking on the pattern card, the user can consult the context, problem, and solution for each pattern variant and proposal. She also has access to a list of linked patterns following the same notation as defined in the pattern proposal ontology. The tool allows filtering patterns out, using the proposal's respective domains, blockchains, and languages. For instance, a user can select Ethereum as the desired blockchain and filter out every non-corresponding pattern. The second part of the tool is the recommender feature. Contrary to the explorer, any user can leverage the recommender to navigate from the problem domain (a set of questions asked by the user) to the solution domain (a set of patterns matching given answers). To personalize pattern recommendations, the user answers a set of questions linked to design problems, as presented in Subsection 4.1. An illustrative scheme of this process is shown in Figure 2.

Questions are organized in a tree structure, traversed by a conditional depth-first algorithm. The questionnaire starts with a high-level question (e.g., "I want to use design patterns in my application") Depending on the user's answers, each node of the tree is assigned a score: 1 for "Yes", -1 for "No" and 0 for "'I don't

[6] https://jena.apache.org/documentation/fuseki2/

know", children of nodes with a negative score being skipped. The tool generates the recommendation once the questionnaire is filled up. Patterns constitute the leaf node of the tree. To compute the score S_q for each pattern, the tool sums the score of every parent node and then normalizes the score using the length of the branch, accounting for branch length differences.

Next, we use three different algorithms to compute pattern rankings based on the scores S_q: `NoCitationsAndQS`, `WeightedCitationAndQS` and `UnWeightedCitationAndQS`. `NoCitationsAndQS` simply orders the patterns based on their score, while the other two also take into account an inferred number of citations. For each pattern, this number of citations is computed by summing the number of citations of all papers that proposes the pattern. As an example, if a pattern is proposed by two papers that respectively have 50 and 150 citations, the pattern is given a number of citations of 200. In `UnWeightedCitationAndQS`, we offset the rank, by multiplying S_q by the ratio of the number of citations of a pattern and the number of citations of the most-cited pattern. For `WeightedCitationAndQS`, instead of using the number of citations directly, we use its logarithm. The rationale for this is the extreme ranking skewness in favor of highly cited patterns in `UnWeightedCitationAndQS`. Note that both `UnWeightedCitationAndQS` and `WeightedCitationAndQS` might discriminate newly proposed patterns that do not have many citations yet.

The user can select one of the three algorithms. When `NoCitationsAndQS` outputs a ranking only impacted by the answers, `WeightedCitationAndQS` and `UnWeightedCitationAndQS` also take into account citations, which serves as an indicator of the pattern adoption in the literature. Also, `UnWeightedCitationAndQS` tends to recommend highly cited patterns, considered more reliable, whereas `NoCitationsAndQS` provides a ranking closer to the questionnaire's answers, to the risk of having newly proposed patterns that lack prior reuse. `WeightedCitationAndQS` is compromising between the two others, as it reduces the impact of citations without discarding them. In conclusion, the decision of using one algorithm instead of another is up to the user, depending on its goals: either using recognized patterns or newly proposed patterns that don't have this recognition yet. Nonetheless, the ranking differences between all of those algorithms are evaluated in Section 5.

5 Evaluation

To evaluate whether the ontology addresses the initial requirements and if the implemented tool is capable of leveraging the ontology, we conducted a threefold validation. Our first method of validation draws from both the ORSD mentioned above and the more general ontology evaluation methods outlined in the work of Raad and Cruz [12]. In particular, we follow their task-based approach, linking the evaluation of the ontology and the tool itself. We demonstrate the ability of the ontology to cover its requirements by, on the one hand, using SPARQL queries in isolation to answer the CQs, and, on the other, by showing its ability to be used as the central knowledge representation mechanism of our tool,

through the validation methods to be covered below. We briefly touch on some of the main evaluation criteria mentioned by Raad and Cruz (cf. [12] for the definition of each concept): **accuracy, completeness, clarity, and conciseness**, though difficult to demonstrate in an absolute sense, are nevertheless covered by the fact that the ontology has been constructed on the basis of an extensive SLR of the field, where care has been taken to isolate only the most relevant aspects; **adaptability** is a consequence of the use of the NeOn methodology and the use of SHACL shapes for automated verification of the ontology and the inferences made thereof, rendering the addition of new patterns to the ontology straightforward; **computational efficiency** is ensured by the compactness of the ontology and the avoidance of recomputing rule-based inferences for every query through pre-compilation of the inferred ontology triples; and, finally, **consistency** is ensured through the use of the aforementioned SHACL shapes for every main class in the knowledge base. As such, SHACL shapes ensure that the current knowledge in the ontology complies with conceptualized rules and that the addition of new knowledge and subsequent OWL reasoning also does.

For the second dimension of our validation scheme, we demonstrate the relevancy of the ontology by addressing the following hypotheses:

- H_1: A practitioner can leverage the tool to navigate from the solution space (blockchain-based patterns) to the problem space (requirements).
- H_2: A practitioner can leverage the tool to navigate from the problem space (requirements) to the solution space (relevant blockchain-based patterns).

Each of these hypotheses will be treated using a specific protocol, both described in the following subsection. For H_1, a survey has been conducted with experts to assess the capability of using the explorer to understand the pattern proposals, and by extension to assess the relevancy of knowledge within our ontology. For H_2, a protocol has been designed to evaluate the recommendations produced by the recommender. Indeed, if the recommendation system is able to suggest adequate patterns, it illustrates the capability of using the ontology to find adequate patterns for specific requirements.

5.1 Protocol

***H*$_1$** - To answer this hypothesis, we surveyed a panel of 7 experts from different backgrounds and positions as shown in Table 2.

Table 2: Panel Description[7].

ID	**E1**	**E2**	**E3**	**E4**	**E5**	**E6**	**E7**
Role	*	§	†	‡	§	†	§
Blockchain Experience (y)	4	4	4	4	2	1	2
Software Design Experience (y)	5+	1	5	5	2	2	5+

[7] §PhD student, *Lead Tech, † Software Engineer, ‡Blockchain Engineer

A custom case study has been designed on a blockchain use case. This case study was short enough to ensure participants had the time to assimilate it within the allocated 30" survey timeframe. Organizers proposed 5 patterns $P^j_{H_1}$ $(0 < j < 5)$ for each expert n, and the objective was to assess if the expert was able to find and understand the patterns well enough to decide if they were applicable to the case study. This applicability j was rated by each participant n from 0 (non applicable) to 4 (must-have) $R_n(P^j_{H_1})$. Then, the survey organizers performed the same exercise. As they worked on the pattern knowledge base and the ontology, they know in detail the patterns presented in the tool and their related papers $\tilde{R}(P^j_{H_1})$. Participants' answers were compared to the organizers' own responses and a normalized score for each participant was calculated $S^{H_1}_n$, the average absolute difference between his score and the organizer's score.

***H*$_2$** - In the second validation step, we aim at evaluating the performance of the various recommender engines, especially those including citation metrics in the ontology. The initial idea was to evaluate the precision and recall of the recommender engine using a set of papers that present blockchain applications. For each paper, the goal was to identify the most suitable patterns $\hat{I}_p$ for the introduced application, then execute the recommendation engine based on the application requirements to retrieve a set of recommended patterns I_p. Using these sets, it is possible to compute the precision and the recall of the recommender system as we respectively assess the proportion of most suitable patterns found in the recommended patterns $\hat{I}_p \in I_p$ over the total amount of most suitable patterns $\hat{I}_p$ and the total number of recommended patterns I_p. However, this approach is very limited for recommender systems: as the set of recommended patterns is very large, the precision will be artificially high. Nevertheless, another approach exists to evaluate the precision and the recall with regards to the ranking, that is precision/recall at cutoff k [3]. Instead of calculating the precision and the recall for all of the recommended patterns, these metrics are computed several times for the k^{th} first patterns, k varying between 1 and the total number of recommended patterns. As a result, it is possible to draw a curve that shows the varying precision and recall depending on k.

To compute these values, the following protocol was undertaken:

- Select a paper p from the literature (n=13), that propose a blockchain-based application, extract the requirements R_p and identify possible patterns $\hat{I}_p$, which represents the golden standard of patterns for p.
- Answer recommender's questions using only the requirements R_p, and retrieve a set of I_p recommended patterns, and their position/score $S_{p,i}, i \in I_p$.
- Compute the precision and the recall at cutoff k, resp. $\frac{\hat{I}_p \in I_p}{k}$ and $\frac{\hat{I}_p \in I_p}{\hat{I}_p}$.

5.2 Results

H_1 - Figure 3 shows the descriptive statistics for the score for each panel participant. The mean values for all the questions range from 2.75 to 3.75 with an

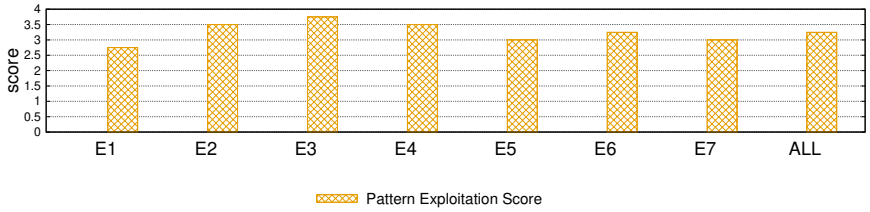

Fig. 3: Panel Usecase Score $S_n^{H_i}$

average of 3.25/4, which indicates that the participants have successfully navigated the solution space and provided adequate options on the relevance of the proposed pattern. It must also be noted that the most junior profiles having a score of 3, have used the tool effectively despite their lack of proficiency in blockchain application design. The expert panel results show positive mean scores for all metrics, our hypothesis can be considered valid w.r.t. our protocol, despite having room for improvements, essentially in its perceived added value. The small sample size, should also prompt further large-scale surveys, including a pre-flight questionnaire to better quantify prior blockchain background for the respondents, and question its impact on the tool's usability.

H_2 - Figure 4 and 5 respectively show the precision and the recall at cutoff k for the three recommender systems considered. To interpret the results, it is required to select an adequate k w.r.t. the usage of the recommendation system. As the web platform displays the first 18 patterns on the first page when executing the recommendation system, a value of $k = 18$ has been chosen. Nonetheless, the selection of a suitable k is a difficult issue, discussed in Section 6. Regarding the precision, the three algorithms are producing similar results except for a cutoff of $k < 20$, where the inclusion of citations increases the precision. For a cutoff of $k = 18$, the precision is 0.2, meaning that on average 20% of the first 18 recommended are relevant for the considered paper. Regarding the recall, the curves are similar for the three different algorithms, with a small advantage to the *NoCitationsAndQS* algorithm. For a cutoff of $k = 18$, on average 57% of the identified relevant patterns in the papers $\hat{I}_p$ are recommended. This number goes up to 80% for a cutoff of $k = 40$. By extension, it indicates that the majority of the most suitable patterns are ranked at the top by the recommender system.

6 Threats to Validity

Internal threat to validity : some aspects of the method used to validate $H2$ can be mentioned. Indeed, the selection of adequate patterns for a given paper has been carried out by two researchers in 13 papers. Although these researchers are experts in blockchain technologies and decentralized applications, it still leaves some space for subjectivity. Several measures have been taken to limit the impact on the results: restricting the selection to the most important patterns, and

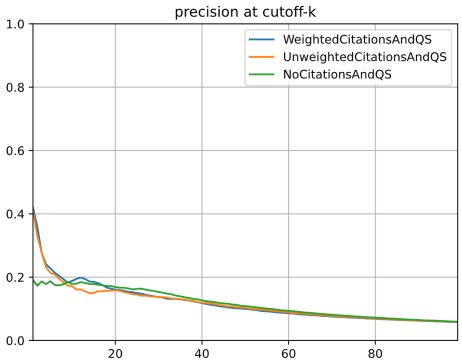

Fig. 4: Average precision at cutoff-k.

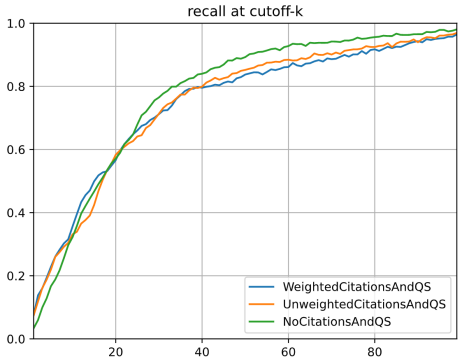

Fig. 5: Average recall at cutoff-k.

comparing the results between the researchers to evaluate possible discrepancies. The method used to build the ontology can also be a threat to validity. Ontologies, by their very nature, have a degree of subjectivity; nevertheless, by using a structured and proven methodology (NeOn), and building upon a SLR, this risk is mitigated, in conjunction with the formal specification of the ORSD and with the evaluation methodology outlined above. Finally, the selection and retrieval of pattern proposals from the literature, that constitutes the core of the ontology, is also subject to be a threat to validity. However, it is mitigated by the strict method followed to perform the literature review (SLR). More details about possible threats and mitigations are given in another study [17].

External threat to validity : the main threat is the generalizability of the ontology. Even if the main purpose of the ontology was its reusability in a tool, careful attention has been made to maximize the ontology reusability. Part of the ontology is inspired by the Design Pattern Intent ontology [10], to bind design patterns (by extension, software patterns) with blockchain design problems. Patterns are also expressed using a shortened pattern format, similar to the GoF pattern format or the Alexandrian form [20]. Future works will refine those patterns to fully comply with one of those two formats. Finally, the ontology has been designed with extensibility in mind. For example, the blockchain class can easily be a connection point between this ontology and other blockchain-related ontologies, such as [4], a blockchain domain ontology.

Conclusion threats to validity : we can mention the difficulty to conclude on the recommendation engine relevancy w.r.t. adequate cutoffs k, as its selection mainly depends on the user behavior. Indeed, some users might only read the first 5 patterns, whereas others might fetch all of the recommendations. In our web platform, the first page of the recommender results displays 18 patterns; thus this number might be a good candidate for k. Nonetheless, this number might change depending on the usage of the recommendation engine in the future.

7 Conclusion and Future Work

This paper proposes a blockchain-based software pattern ontology to store, classify, and reason about blockchain-based patterns. The ontology has been built over previous results obtained by performing a systematic literature review (SLR) of the state-of-the-art of blockchain-based patterns. It is composed of proposals that are patterns formalized in the context of an academic paper. 160 proposals have been stored in the ontology, resulting in 114 different software patterns identified. Also, those patterns have been classified using a taxonomy reused from the SLR mentioned above. To best make use of both the categorization and the ontology, a tool has been built. Using it, practitioners can explore the ontology and its collection of patterns, but also use a recommender to get adequate patterns fulfilling their needs. This tool is also meant to be extendable following ontology evolution and support future works. The ontology can also be leveraged as standalone, using SPARQL queries.

This paper paves the way for future works in assisting practitioners in the design of a blockchain application. The different artifacts will be integrated into the Harmonica project[8], a semi-automated framework for the design and implementation of blockchain applications [15]. The integration will notably be done between the tool presented in this paper and BLADE (BLockchain Automated Decision Engine), a decision-making tool for the selection of a blockchain technology [16]. The combination will allow users to select a blockchain, then adequate patterns that are applicable to the chosen technology. Some extensions of this work could also be envisioned in the software pattern domain. Although the pattern proposal ontology is introduced within the scope of blockchain patterns, it could be generalized to all software patterns, such as Internet-of-Things (IoT) or microservices. Finally, existing software patterns in the ontology might be extended to include a formal description using existing pattern formats.

References

1. Belotti, M., Božić, N., Pujolle, G., Secci, S.: A vademecum on blockchain technologies: When, which, and how. IEEE Communications Surveys & Tutorials **21**(4), 3796–3838 (2019)
2. Casado-Vara, R., Prieto, J., De la Prieta, F., Corchado, J.M.: How blockchain improves the supply chain: Case study alimentary supply chain. Procedia computer science **134**, 393–398 (2018)
3. Croft, W.B., Metzler, D., Strohman, T.: Search engines: Information retrieval in practice, vol. 520. Addison-Wesley Reading (2010)
4. De Kruijff, J., Weigand, H.: Understanding the blockchain using enterprise ontology. In: International Conference on Advanced Information Systems Engineering. pp. 29–43. Springer (2017)
5. Gamma, E., Helm, R., Johnson, R., Vlissides, J., Patterns, D.: Elements of reusable object-oriented software, vol. 99. Addison-Wesley Reading, Massachusetts (1995)

[8] https://github.com/harmonica-project

6. Girardi, R., Lindoso, A.N.: An ontology-based knowledge base for the representation and reuse of software patterns. ACM SIGSOFT Software Engineering Notes **31**(1), 1–6 (2006)
7. Glaser, F.: Pervasive decentralisation of digital infrastructures: a framework for blockchain enabled system and use case analysis (2017)
8. Hector, U.R., Boris, C.L.: Blondie: Blockchain ontology with dynamic extensibility. arXiv preprint arXiv:2008.09518 (2020)
9. Henninger, S., Ashokkumar, P.: An ontology-based metamodel for software patterns. CSE Technical reports p. 55 (2006)
10. Kampffmeyer, H., Zschaler, S.: Finding the pattern you need: The design pattern intent ontology. In: International Conference on Model Driven Engineering Languages and Systems. pp. 211–225. Springer (2007)
11. Pavlic, L., Hericko, M., Podgorelec, V.: Improving design pattern adoption with ontology-based design pattern repository. In: ITI 2008-30th International Conference on Information Technology Interfaces. pp. 649–654. IEEE (2008)
12. Raad, J., Cruz, C.: A survey on ontology evaluation methods. In: Proceedings of the International Conference on Knowledge Engineering and Ontology Development, part of the 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (2015)
13. Schär, F.: Decentralized finance: On blockchain-and smart contract-based financial markets. FRB of St. Louis Review (2021)
14. Seebacher, S., Maleshkova, M.: A model-driven approach for the description of blockchain business networks. In: Proceedings of the 51st Hawaii International Conference on System Sciences (2018)
15. Six, N.: Decision process for blockchain architectures based on requirements. CAiSE (Doctoral Consortium) pp. 53–61 (2021)
16. Six, N., Herbaut, N., Salinesi, C.: Blade: Un outil d'aide à la décision automatique pour guider le choix de technologie blockchain. Revue ouverte d'ingénierie des systèmes d'information **2**(1) (2021)
17. Six, N., Herbaut, N., Salinesi, C.: Blockchain software patterns for the design of decentralized applications: A systematic literature review. Blockchain: Research and Applications p. 100061 (2022)
18. Six, N., Ribalta, C.N., Herbaut, N., Salinesi, C.: A blockchain-based pattern for confidential and pseudo-anonymous contract enforcement. In: 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom). pp. 1965–1970. IEEE (2020)
19. Suárez-Figueroa, M.C., Gómez-Pérez, A., Fernández-López, M.: The neon methodology for ontology engineering. In: Ontology engineering in a networked world, pp. 9–34. Springer (2012)
20. Tešanovic, A.: What is a pattern. Dr. ing. course DT8100 (prev. 78901/45942/DIF8901) Object-oriented Systems (2005)
21. Xu, X., Bandara, H.D., Lu, Q., Weber, I., Bass, L., Zhu, L.: A decision model for choosing patterns in blockchain-based applications. In: 2021 IEEE 18th International Conference on Software Architecture (ICSA). pp. 47–57. IEEE (2021)
22. Xu, X., Pautasso, C., Zhu, L., Lu, Q., Weber, I.: A pattern collection for blockchain-based applications. In: Proceedings of the 23rd European Conference on Pattern Languages of Programs. pp. 1–20 (2018)
23. Xu, X., Weber, I., Staples, M., Zhu, L., Bosch, J., Bass, L., Pautasso, C., Rimba, P.: A taxonomy of blockchain-based systems for architecture design. In: 2017 IEEE international conference on software architecture (ICSA). pp. 243–252. IEEE (2017)