A blockchain ontology for the pattern-based design of decentralized software applications

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 an ontology of blockchain-based design patterns. That was created based on a previous systematic literature review of existing blockchain patterns. A tool is proposed along with the ontology to assist practitioners in finding blockchain-based patterns that fit their needs. A survey is performed with software architects and blockchain experts to assess the ontology usability. The survey has resulted in positive results, as participants were able to leverage the tool in the context of survey case studies.

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 [19]. The addition of new blocks is ruled by a network of peers, so-called nodes, that each holds a copy of the blockchain. Each block stores a list of transactions, that represent interactions between a user and the blockchain. First-generation blockchains, such as Bitcoin, only allow users to exchange cryptocurrencies with each other through transactions. Since 2015, with the release of Ethereum, several blockchain solutions started to support smart contracts. 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.

As the internal workings of blockchain differ from traditional technologies, it has many unique qualities. First, blockchain is decentralized: there is no central actor in charge of the network. A consensus algorithm executed by each peer in the network rules the addition of new blocks. Then, blockchain guarantees

the immutability of data[1], as it is impossible to alter blocks after their addition. Finally, blockchain is transparent: as every node has a copy of the blockchain, any user that has access to a node can see previous transactions and stored data.

The unique capabilities of blockchain unlock a new range of innovative decentralized applications. For example, supply-chain applications take profit of blockchain to enable trusted traceability of goods [2]. Healthcare applications use blockchain as a central trusted third party for medical data sharing [12]. Because of the specific properties of blockchain, a growing number of software architects attempt to build blockchain-based applications, but have found it to be a tedious task. Blockchain applications can suffer from their own qualities in certain contexts. For instance, transparency and immutability can be a burden when coping 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 software patterns to address design issues in the software design phase. A pattern is a solution to a commonly occurring problem in a given context. In the blockchain field, blockchain-based software patterns can be a solution to the difficulty of ensuring the quality of newly designed decentralized applications. For instance, one of the best-known patterns is the *Oracle* [20]: as a smart contract cannot query data from outside the blockchain, using this pattern consists in sending fresh data to a smart contract when needed. Up to now, only a few patterns have already been identified, regrouped in collections (e.g., [20]) or proposed standalone (e.g., [15]). Moreover, these patterns are scattered across the literature, making the task of identifying adequate patterns for a specific design difficult. To the best of our knowledge, there is no centralized and structured collection of blockchain-based patterns from diverse domains.

To address the aforementioned issues, this paper proposes an ontology of blockchain-based software patterns. This ontology is built over a systematic literature study in which 160+ blockchain-based patterns were collected from the literature. A problem ontology is also included to link pattern categories with the problems they address in the software design phase. Problems are linked to user story sentences to bridge between patterns and the practitioner's requirements. To explore the ontology, an open-source tool[2] is also proposed to graphically explore the ontology. We formally validated our ontology (a) through the verification of its fulfillment of the requirements and (b) through a qualitative survey conducted with 7 blockchain and software architect experts to evaluate the relevance of the produced ontology by leveraging our tool.

The rest of the paper is organized as the following: section 2 introduces existing blockchain-based pattern collections and software pattern ontologies, 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,

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

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

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 [9], Kampffmeyer et al. propose an ontology derived from GoF [4] for design patterns. 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 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 [5]. This ontology encompasses not only design patterns but also architectural patterns and idioms. A pattern is described using different attributes (such as *Problem*, *Context*, *Solution*, ...), and can be linked to other patterns through a pattern system and specific relations (e.g., require, use).

A similar metamodel for software patterns is proposed in [8]. Some differences can be mentioned, such as the possibility to specify that two patterns conflict with each other and cannot be applied at the same time, or the *seeAlso* relationship to indicate other patterns related to a specific pattern.

In addition, [10] 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 group 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. Our contribution follows a similar path to that taken by the aforementioned pattern ontologies by structuring a set of patterns of a specific domain, in our case blockchain-based patterns.

Some ontologies have been proposed for modeling the blockchain domain, such as that proposed by De Kruijff and Weigand [3], that of Ugarte-Rojas and Chullo-Llave [7], and that of Glaser [6] (though Glaser provides no formalization of his ontology) that models the technology itself and its components. Another work by Seebacher and Maleshkova [11] focuses on modeling the characteristics of blockchains within corporate networks and their use. To the best of our knowledge, no work has been conducted on exploring formal ontologies for software patterns within the blockchain domain.

3 Methodological Approach

The proper design of an ontology relies on the usage of a reliable and proven method. For the construction of the blockchain-based pattern ontology, the NeOn method [16] has been chosen due to its inherent flexibility and focus on the reuse

of both ontological and non-ontological sources in a structured manner. It does not force rigid guidelines upon its users: a set of scenarios is given and the user is free to select, and if needed, 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 ontologies covering blockchain patterns and the inability of existing software pattern ontologies to adequately capture the results of the literature review upon which we base our 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 the blockchain-based pattern ontology will be primarily based on the reuse of previous results obtained through a systematic literature review. 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 (Figure 1).

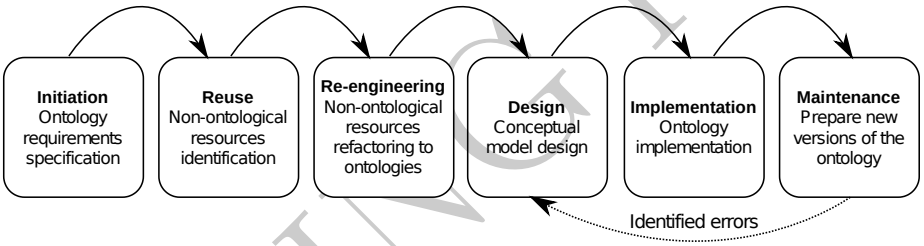

Fig. 1: NeOn framework workflow.

3.1 Ontology requirements specification

One important step in the construction of a sound ontology is the specification of requirements through an ORSD (Ontology Requirement Specification Document) [16] that serves as an agreement on what 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 [16]. Competency questions (CQs) encode the functional requirements of an ontology; their coverage, ideally in a generalizable manner, allows one to consider the ontology functionally complete. For the sake of brevity, only the CQs are detailed in this paper, listed in Table 1. However, more information about the ontology's purpose can be found in the introduction or in the full ORSD, available on GitHub[3].

The set of competency questions is divided in three categories. The first, *Proposed patterns information*, relates to obtaining information on a specific pattern. The main elements of the ontology are the patterns themselves. However,

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

Table 1: Blockchain-based software patterns ontology competency questions.

CGQ1 - Proposed patterns information	
CQ1	What are the context and problem linked with the Oracle pattern in [20]?
CQ2	What is the solution proposed by the Oracle pattern in [20]?
CGQ2 - Relations between patterns	
CQ3	What are the linked patterns of the Contract Registry pattern in [20]?.
CQ4	What are the patterns proposed in the literature for the Oracle pattern?
CQ5	What is the pattern targeted proposed in [20] under the name Oracle?
CGQ3 - Pattern recommendation	
CQ6	What patterns can I use to improve the security of a smart contract?

a distinction is made between software patterns that are proposed in different sources (so-called proposal), and software patterns that are the result of merging all sources into one common definition (so-called pattern). This distinction is important as a pattern within a source is proposed in relation to a specific context (domain, technology, language, ...). The second, *Relations between patterns*, relates to dependency links between pairs of patterns. As an example, the *Tokenization pattern* requires the *Address mapping pattern*, as it is necessary to establish a mapping between users and owned tokens to enable their usage. Being able to answer those questions using the ontology is important to extend the capabilities of recommending relevant patterns in a given case later. Indeed, the usage of some patterns is conditioned on the usage of others. Two more questions (CQ4 and CQ5) address the relations between a proposed pattern and its corresponding class. The ontology must allow the user to get all proposed patterns that are linked to a specific pattern class, but also the opposite. Finally, the third, *Pattern Recommendation*, relates to the connection between patterns and the problems they address. Answering CQ6 returns a set of proposed patterns that address a high-level problem that have multiple patterns as an answer.

The process outlined in [16] 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 ontology is to formalize the knowledge of a previous systematic literature review, the ontology incorporates knowledge from two different non-ontological resources that can both be found on GitHub[4]. The first is a collection of 160 patterns that were found during the literature review within 20 different papers; out of which 120 unique patterns have been derived.

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

Each of the collected patterns is described by a set of attributes, e.g., a *Name*, a *Context and Problem*, and a *Solution*. 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 (the Solidity smart contract language[5]), or in the context of a specific domain (e.g., patterns to enable decentralized identity on blockchain). Also, different types of relations between patterns were identified throughout the study: *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 in 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. This taxonomy emerges from the categorization of the results in the literature review, and is comprised of 4 main categories and 14 subcategories. The blockchain-based software pattern ontology reuses all collected knowledge from the literature review. More details are given in the introduction of the ontology conceptual model in the results in Section 4.

4 Results

The application of the NeOn method resulted in a blockchain-based software pattern ontology, and a querying tool that can be used to leverage the ontology through different ways of retrieving then selecting blockchain-based patterns.

4.1 Blockchain-based software pattern ontology

The primary outcome is the creation of the blockchain-based software pattern ontology, stands in the conceptual model[6] presented in Figure 2. As the figure shows, the driving idea of the ontology is the explicit distinctions between: (a) patterns and pattern proposals, (b) proposals and descriptions of proposals in source papers, and (c) design problems outside the scope of patterns.

The central element of this model is the *Proposal* class. A proposal is a pattern introduced within an academic paper. Each proposal instance is linked to concrete instances of the *Pattern* and *Source* classes. In the current form of the ontology, all sources are academic papers, but this class includes other types of sources such as technical reports. As an example, the *Oracle* pattern

[5] https://docs.soliditylang.org/

[6] For the sake of clarity, subclasses of *Domain*, *Blockchain*, *Pattern*, and *Design problem* were omitted in the figure.

proposed by Xu et al. [20] is an individual of *Proposal* as it is proposed in the Xu et al. [20] paper, and attached to the *Oracle* pattern, an individual of the *Pattern* class. The distinction between a pattern and the proposals it results from is important as in some cases multiple papers proposed the same pattern using different words, templates, and for different domains or blockchains. In the previous example, the *Oracle pattern* has multiple proposals from different papers. This is represented in the ontology by one-to-many relations between *Proposal*, *Domain*, *Language*, or *Blockchain* technology. The usage of *Proposal* entity to map a paper and its proposed patterns also allows extensibility, as papers can be abstracted as documents, regrouping other types of literature (industrial reports, documentation, ...).

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 introduce the different elements composing the pattern solution. This structure for pattern description is derived from the two main pattern formats (GoF pattern format and Alexandrian form [17]), usually used by researchers and practitioners 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. *Proposals* and *Patterns* can also be linked as pairs, using 5 different relation types that were identified from the systematic literature review: *Created from* for a pattern that directly takes its sources in another, *Variant of* when a pattern is a variant of another, *Requires* and *Benefits from* when a pattern might or have to use another to perform well once implemented, and *Related to* to identify a weak relation between a pattern and another (e.g., "see also").

The subclasses of the *Pattern* class emanate from the reused taxonomy for blockchain-based patterns, built in its related systematic literature review. For instance, the *Oracle* proposal from [20] is linked to the *Oracle* pattern class, that inherits from the *Data exchange pattern*, then *On-chain pattern*, *Design pattern*,

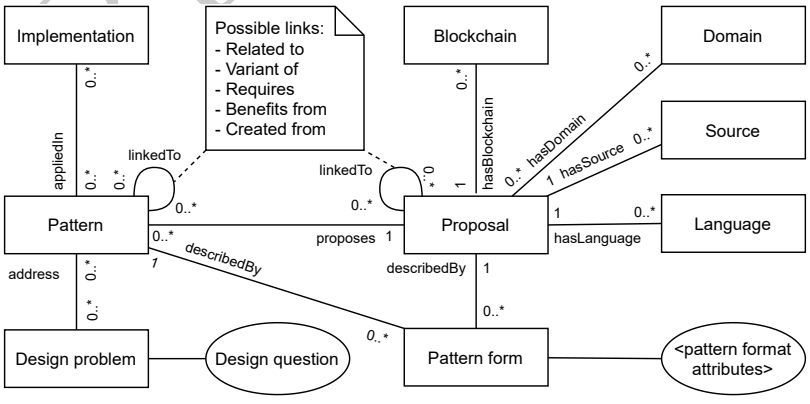

Fig. 2: Ontology conceptual model.

and finally *Pattern*. To further refine this part of the ontology, each *Pattern* addresses a specific *Design problem*. By extension, each subclass of *Pattern* address 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 use smart contracts in the design of my blockchain application". Such an affirmation can be thus presented as a question to the user and answered positively or negatively, to guide pattern recommendation.

4.2 Ontology Querying Tool

In parallel with the ontology, a tool was designed to leverage the blockchain-based pattern ontology without having to query directly the ontology through SPARQL requests. This tool has two main features. The first one is the explorer feature, which allows one to dive into the blockchain-based pattern 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 and goals). Indeed, any user reading available patterns descriptions might find some that suit their goals. In the application, each pattern is displayed by its name but also the number of proposals linked to the pattern. By clicking on the pattern card, proposals can be consulted in detail: the context, problem, and solution are given to the user, along with a list of linked patterns following the same notation as the ontology. Patterns can also be filtered out using the proposal respective domains, blockchains, and languages, as filters. 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 feature, any user can leverage the recommender to pass from the problem domain (a set of questions asked by the user), to the solution domain (a set of patterns matching given answers). In the recommender, the user must answer a set of questions linked to design problems, as presented in Subsection 4.1.

The recommender must be able to convert the answers from the questions to a set of scores attributed to each pattern. An illustrative scheme of this process is shown in Figure 3. The set of questions is browsed as a tree structure: the questionnaire begins with a high-level question (e.g. "I want to use design patterns in my application") and if the user answer "Yes", direct subquestions are asked to the user. Also, the question is assigned a score of 1. The user can also answer "No": in this case, the question and every subquestion below will be assigned a score of -1, and subquestions will be automatically skipped. A third possible case is answering "I don't know". In this case, the question will simply be skipped and its score will be of 0. Once the questionnaire is filled, recommendations are generated.

As the set of questions forms a tree, and as patterns are grouped under leaves categories from the taxonomy, the tool can compute the score of a pattern by summing the score of all tree questions, then dividing it by the length of the branch. This step is essential as some branches are shorter than others, thus the sum would be lower although the user answered yes to all questions of those branches. After this scoring phase, the recommendation process is concluded by a presentation of the patterns to the user, ordered by score. To facilitate the understanding of the pattern recommendation level among others, this score is displayed as a label. 5 labels have been created, from "Not recommended" to "Extremely recommended". As patterns are scored between 0 and 1, each label corresponds to an interval of 0.2.

5 Validation

To evaluate whether the ontology addresses the initial requirements, and if the implemented tool is capable of leveraging the ontology, a two-fold validation was conducted. First, it has been verified that the ontology indeed covers the set of requirements as set out in the ORSD and is thus functionally complete. Second, a survey is performed with experts in the software development and blockchain fields, to validate the usability of the tool designed over the ontology. Validating the tool usability is, by extension, validating the ontology soundness.

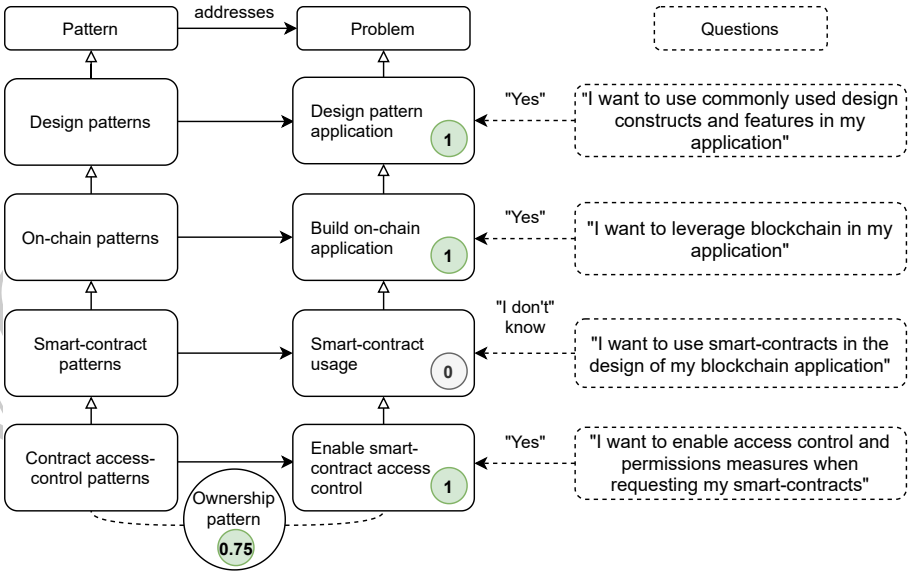

Fig. 3: Pattern scoring based on patterns/problem categories.

5.1 Protocol

In this survey, we surveyed a panel of 7 experts from different backgrounds (academia, industry) and positions (engineers, manager) 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+

This survey is divided into three parts. The first two parts aim at assessing whether the users were able to exploit the ontology through the tool, either for pattern recommendations or consultation. Then we assess the overall acceptance of the tool for the design of blockchain applications. For that, 3 validation hypotheses were proposed:

- 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).
- H_3: A practitioner can leverage the tool to design blockchain applications.

In other terms, if the tool fulfills those three hypotheses, a user will be able either to assess if the proposed software patterns fit its goals and requirements, but also to get a selection of recommended patterns from the tool recommender system, and use the tool to design blockchain applications.

To address H_1, 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 in the survey within the given timeframe (30 minutes for both parts). Organizers proposed 5 patterns $P^j_{H_1}$ $(0 < j < 5)$ for each expert n, 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 of pattern 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 but with full knowledge of the patterns, having analyzed the associated papers beforehand $\tilde{R}(P^j_{H_1})$. Finally, 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 organizers score as shown in equation (1).

H_2 was addressed differently: after reading a second case study, each participant n had to answer questions formulated by the recommender system, then refer to the generated pattern recommendation to select 5 patterns $P^{j,n}_{H_2}$, $0 < j < 5$ fitting the case study. Then, each participant's selected pattern was graded on the same 0-4 scale by survey organizers $\tilde{R}(P^{j,n}_{H_2})$. We then computed

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

normalized score $S_n^{H_2}$ based on the average pattern score for each expert n as show in equation (2).

$$S_n^{H_1} = \frac{\sum_{j=0}^{\#P_{H_1}} 4 - |R_n(P_{H_1}^j) - \tilde{R}(P_{H_1}^j)|}{\#P_{H_1}} \quad (1) \qquad S_n^{H_2} = \frac{\sum_{j=0}^{\#P_{H_2}} \tilde{R}(P_{H_2}^{j,n})}{\#P_{H_2}} \quad (2)$$

The finality of this score is the assessment that participants successfully managed to take profit of the tool to build a set of patterns that suits the use case needs. For each hypothesis, a score of 4 would mean a perfect agreement between participants and organizers and a score of 0 would mean a perfect disagreement.

Finally, for H_3, the validation was conducted through a questionnaire sent to the panel to gather their structured feedback on the tool after the session. To design the questionnaire, we adapted 12 questions from the Unified Theory of Acceptance and Use of Technology (UTAUT) [18], that is widely used for assessing artifact acceptance. Each panel member provided answers on a Likert scale[8] from a blockchain software architect perspective. We grouped our 12 questions following 4 different UTAUT categories. We report the mean value for all the questions for each category to get feedback on the tool usability and utility. The following list presents those categories, with question references in parenthesis:

- **Performance Expectancy (U6, RA1)**: the level to which an individual believes that using the system will improve its job performance. (ie. the user is able to identify adequate patterns for designing blockchain applications).
- **Effort Expectancy (EOU3, EOU5, EOU6, EU4)**: the degree of ease associated with the use of the system. (ie. users should understand the tool easily to find adequate patterns).
- **Facilitating Conditions (PBC2, PBC3)**: The impact of the environment that makes something easy to do, such as technical or technical infrastructure. (ie. user has the technical knowledge or resources to use the tool.)
- **Self-Efficacy (SE1, SE4, SE6, SE7)**: the degree of which the user is able to complete the task by himself (with or without help).

5.2 Results and Discussion

Figure 4 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 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. Strong prior blockchain experience is not necessarily a good predictor for successfully judging patterns, since the most experienced participant has the lowest score. The most junior profiles having a score of 3, have used the tool effectively despite their lack of proficiency in blockchain application design.

Figure 4 scores put forth an even higher ability to navigate the problem space thanks to the recommender system, with an average score of 3.5/4. Several participants (E2, E6) selected only highly relevant patterns from the list, while E1

[8] Strongly disagree (0), disagree (1), neutral (2), agree (3), and strongly agree (4)

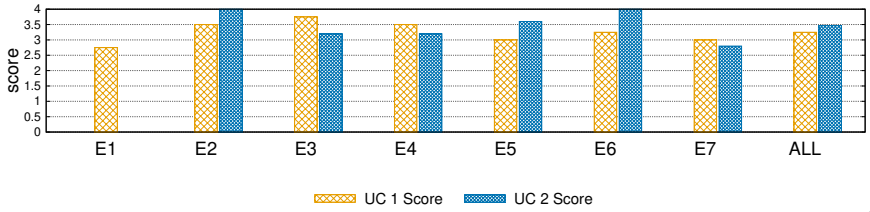

Fig. 4: Panel Usecase Score $S_n^{H_i}$

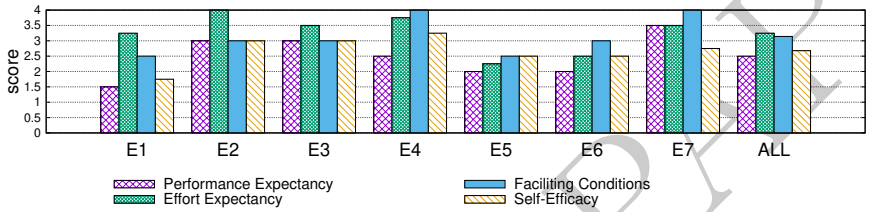

Fig. 5: Tool acceptance Evaluation

was not able to use the recommender system at all due to his lack of comprehension of the system. Again, blockchain experience is not a good predictor for a high score for H_2, which strengthens the idea that blockchain beginners have a high benefit of using the tool.

Finally, we see in Figure 5 that tool acceptance shows more nuanced results. Performance Expectancy is weakly positive, due to E1 not being able to fully use the recommender system, while most junior profiles, E5 and E6 are neutral about the tool improving their work performance. This may be explained by the fact that using the tool is not enough for them to be confident in their design choices, and external expertise is still envisaged for them to produce blockchain software. The tool, however, is judged easy to use (Effort Expectancy score is 3.25/4) and provides enough resources to work on (Facilitating Conditions = 3.14/4) by most participants. Juniors (E5, E6) tend to lower the average for those two metrics. The mean score of Self-efficacy (2.68/4) is neutral to positive indicating that even if the tool can be used autonomously, it learning curve should be considered for improvements.

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 from 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 usability.

6 Threats to Validity

Four classical threats to validity framework were used by the authors to analyze this work (internal, external, construction, and conclusion). Regarding internal threats to validity, the survey method can be mentioned. Although UTAUT was used to design part of the method, an ad hoc method has been proposed for the evaluation of the tool functionalities. Nonetheless, the design of this method has been carefully made to assess the different hypotheses mentioned in the results section. Using this method also helped to identify a second internal threat, that is the difficulty for users to leverage the recommendation section. Indeed, most of the participants have found the recommendations too coarse to make accurate decisions. This issue will be addressed in future works, yet this does not invalidate the usability of the ontology itself as the explore part returned positive results. The method used to build the ontology can also be a threat to validity. This study uses NeOn as its primary method for the construction of the ontology, yet applying another method could have resulted in a different ontology. However, the possible difference of results is mitigated by the fact that the blockchain-based software pattern ontology can address all of the competency questions designed to guide the construction of the ontology.

The main external threat to validity 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 this ontology is inspired by another ontology from the Design Pattern Intent ontology [9], to bind design patterns (by extension, software patterns) with design problems. Patterns are also expressed using a shortened pattern format, similar to the GoF pattern format or the Alexandrian form. 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 ontologies, such as [3], a blockchain domain ontology.

Regarding conclusions threat to validity, the scores obtained during the third part (H_3) of the survey can be discussed. We observed a lower score in this part than the others, pointing out potential usability issues in the tool. Two main issues were identified from participants' feedback: too many patterns were recommended by the tool, and it was impossible to use filters to restrict the number of displayed patterns. This has caused many participants to be overwhelmed by the number of proposed patterns, making difficult the selection of adequate patterns. Nevertheless, those issues are more related to the tool interface than the ontology itself, as the scores obtained for the first two parts were satisfying.

7 Conclusion and Future Work

This paper proposes an ontology to store, classify, and reason about blockchain-based software patterns. The ontology has been built over previous results obtained by performing a systematic literature review 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 120 different software patterns identified. Also, those patterns have been classified using a taxonomy reused from the systematic literature review 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.

A survey was conducted among 7 practitioners in the blockchain software engineering field to evaluate the tool usability, and by extension, the ontology soundness. It has yielded positive results. During the first part, participants were successfully able to rate the applicability of a list of patterns for a specific case study, both proposed in the context of the survey. In the second part, participants managed to select 5 patterns from a list of recommended patterns fitting another case study. However, we identified some improvement opportunities, notably in the second and third parts. Participants have pointed out that too many patterns were proposed by the recommender, thus making difficult the selection. Also, the lack of filters in the recommender hinders participants to refine the selection into a more desirable set of recommended patterns. However, those problems are more related to the tool than the ontology itself, as it was highlighted in the usability part of the survey (part 3).

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[9], a semi-automated framework for the design and implementation of blockchain applications [13]. 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 [14]. 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. Indeed, the blockchain-based pattern ontology could be generalized to all software patterns, thus allowing its reuse for the collection of software patterns in other domains, such as Internet-of-Things (IoT) or microservices.

The recommender within the tool proposed in this paper will also be improved through future works, by taking into account user feedback collected during the survey. The main objective will be to increase the recommendation precision, notably with the addition of filters and improvement in the recommendation process itself. 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),

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

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. De Kruijff, J., Weigand, H.: Understanding the blockchain using enterprise ontology. In: International Conference on Advanced Information Systems Engineering. pp. 29–43. Springer (2017)
4. Gamma, E., Helm, R., Johnson, R., Vlissides, J., Patterns, D.: Elements of reusable object-oriented software, vol. 99. Addison-Wesley Reading, Massachusetts (1995)
5. 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)
6. Glaser, F.: Pervasive decentralisation of digital infrastructures: a framework for blockchain enabled system and use case analysis (2017)
7. Hector, U.R., Boris, C.L.: Blondie: Blockchain ontology with dynamic extensibility. arXiv preprint arXiv:2008.09518 (2020)
8. Henninger, S., Ashokkumar, P.: An ontology-based metamodel for software patterns. CSE Technical reports p. 55 (2006)
9. 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)
10. 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)
11. 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)
12. Shen, B., Guo, J., Yang, Y.: Medchain: Efficient healthcare data sharing via blockchain. Applied sciences **9**(6), 1207 (2019)
13. Six, N.: Decision process for blockchain architectures based on requirements. CAiSE (Doctoral Consortium) pp. 53–61 (2021)
14. Six, N., Herbaut, N., Salinesi, C.: Blade: Un outil daide à la décision automatique pour guider le choix de technologie blockchain. Revue ouverte dingénierie des systèmes dinformation **2**(1) (2021)
15. 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)
16. 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)
17. Tešanovic, A.: What is a pattern. Dr. ing. course DT8100 (prev. 78901/45942/DIF8901) Object-oriented Systems (2005)
18. Venkatesh, V., Morris, M.G., Davis, G.B., Davis, F.D.: User acceptance of information technology: Toward a unified view. MIS quarterly pp. 425–478 (2003)
19. Wüst, K., Gervais, A.: Do you need a blockchain? In: 2018 Crypto Valley Conference on Blockchain Technology (CVCBT). pp. 45–54. IEEE (2018)
20. 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)